MARCOS A RODRÍGUEZ GARCÍA

MEMORIA PI3 MARRODGAR60

DATOS

CIUDAD

```java
package datos;

public record Ciudad(String nombre,Integer puntuacion) {


    public static Ciudad ofFormat(String[] formato) {
        String nombre = formato[0];
        Integer puntuacion = Integer.parseInt(formato[1].replace("p", "".trim()));


        return new Ciudad(nombre,puntuacion);
    }

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

    @Override
    public String toString() {
        //String nn = this.nombre + "\n" + this.ciudadNacimiento+ " " + this.anyo;
        return this.nombre;
    }

    public String toString2() {
        String res = this.nombre + "\n" + this.puntuacion + " puntos";
        return res;
    }


}
```

PERSONA

```java
package datos;

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


    public static Persona ofFormat(String[] formato) {
        Integer id = Integer.parseInt(formato[0]);
        String nombre = formato[1];
        Integer anyo = Integer.parseInt(formato[2]);
        String ciudadNacimiento= formato[3];
        return new Persona(id,nombre,anyo,ciudadNacimiento);
    }

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

    @Override
    public String toString() {
        //String nn = this.nombre + "\n" + this.ciudadNacimiento+ " " + this.anyo;
        return this.nombre;
    }

    public String toString2() {
        String res = this.nombre + "\n" + this.ciudadNacimiento+ " " + this.anyo;
        return res;
    }


}
```

RELACIÓN

```java
package datos;

public record Relacion(Integer id) {

private static int num =0;

    public static Relacion ofFormat(String[] formato) {
        Integer id = num;
        num++;
        return new Relacion(id);
    }


    @Override
    public String toString() {
        String res = "";
        return res;
    }

}
```

TRAYECTO

```java
package datos;


//Ciudad1,Ciudad2,30euros,45min
public record Trayecto(Integer id,Double euros,Double minutos) {

    private static int num =0;

    public static Trayecto ofFormat(String[] formato) {


        Double euros = Double.parseDouble(formato[2].replace("euros", "").trim());
        Double minutos = Double.parseDouble(formato[3].replace("min", "").trim());
        Integer id = num;
        num++;
        return new Trayecto(id,euros,minutos);
    }

    public static Trayecto of(Double euros) {
        Double euro = euros;
        Double minuto = null;
        Integer id = num;
        num++;
        return new Trayecto(id, euro, minuto);
    }

    @Override
    public String toString() {

        return this.euros+" euros \n"+this.minutos+" minutos" ;
    }
}
```

EJERCICIO 1

```java
public class Ejercicio1 {

    // ================================================================================================
    // APARTADO A
    // ================================================================================================
    /*a. 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.*/


    private static List<Persona> getPadres(Graph<Persona, Relacion> gf, Persona p) {
        Set<Relacion> aristasPadres = gf.incomingEdgesOf(p); // añado las aristas que le llegan al vértice p
        List<Persona> padres = List2.empty();
        for (Relacion a : aristasPadres) {      // recorro las aristas (padres)
            padres.add(gf.getEdgeSource(a)); // añado los vértices de donde sale la arista es decir los padres
        }
        return padres;
    }

    public static Set<Persona> apartadoA(Graph<Persona, Relacion> gf) {
        Set<Persona> res = Set2.empty();
        Set<Persona> personas = gf.vertexSet();         // conjunto con todas las personas del grafo
        for (Persona p : personas) {                    // recorro todas las personas
            List<Persona> padres = getPadres(gf, p);    // obtengo una lista con los padres de cada persona

            if (padres.size() == 2 // si tiene 2 padres y ambos tienen el mismo año y ciudad de nacimiento los
                                   // añado a la lista
                    && padres.get(0).ciudadNacimiento().equals(padres.get(1).ciudadNacimiento())
                    && padres.get(0).anyo().equals(padres.get(1).anyo())) {
                res.add(p);
            }
        }
        return res;

    }


    public static void apartadoAColorear(Graph<Persona, Relacion> gf, Set<Persona> ls, String file) {

        GraphColors.toDot(gf, "resultados/ejercicios/ejercicio1/" + file + ".gv", // nombre del fichero que voy a guardar
                                                                                  // la solución
                p -> p.nombre(),                        // los vertices van a ser el nombre
                r -> "",                                // las aristas son vacias
                p -> GraphColors.color(ls.contains(p) ? // vértices en la lista --> solución (azul)
                        Color.blue : Color.black),
                r -> GraphColors.style(Style.solid));
    }
```

```java
// ============================================================================================
// APARTADO B
// ============================================================================================

/*b. 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 Set<Persona>ancestros(Graph<Persona,Relacion>gf,Persona p,Set<Persona>ls){

    List<Persona>padres = getPadres(gf, p); //obtengo los padres de la persona
    for(Persona padre : padres) { //por cada padre hago una llamada recursiva para obtener sus padres
        ls.add(padre); //añado sus padres a la lista
        ancestros(gf, padre, ls);
    }

    return ls;
}


public static Set<Persona> apartadoB(Graph<Persona, Relacion> gf ,Persona p) {

    //llamo al método ancestros que devuelve todos los ancestros
     Set<Persona> res = ancestros(gf, p, Set2.empty());

    return res;

}
public static void apartadoBColorear(Graph<Persona, Relacion> gf, Set<Persona> ls,Persona persona,String file) {

    GraphColors.toDot(gf, "resultados/ejercicios/ejercicio1/" + file + ".gv",
            p -> p.nombre(),
            r -> "",
            p -> GraphColors.color(p.equals(persona)? //vertice es igual a la persona
                Color.red :                           //coloreo de rojo
                ls.contains(p) ?                      //si en los vertices están en los ancestros(azul)
                Color.blue : Color.black),

            r -> GraphColors.style(Style.solid));
}
```

```java
// APARTADO C
// ================================================================================================

/*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 ninguno de los dos casos. 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 enum PrimosHermanosOtros{
    HERMANOS,PRIMOS,OTROS
}

public static List<Persona>getAbuelos(Graph<Persona,Relacion>gf,Persona p){

    List<Persona>res = List2.empty();
    List<Persona> padres = getPadres(gf, p);
    List<Persona> abuelos = List2.empty();
    for (Persona p0 : padres ) {
        abuelos = getPadres(gf, p0);
        for(Persona abuelo:abuelos) {
            res.add(abuelo);
        }
    }
    return res;
}

public static PrimosHermanosOtros apartadoC(Graph<Persona,Relacion>gf,Persona p1,Persona p2) {


    List<Persona> abuelosP1 = getAbuelos(gf, p1);
    List<Persona> abuelosP2 = getAbuelos(gf,p2);

    List<Persona> padresP1 = getPadres(gf, p1);
    List<Persona> padresP2 = getPadres(gf, p2);

    PrimosHermanosOtros res = PrimosHermanosOtros.OTROS;

    for(Persona p0 : padresP1) {
            if (padresP2.contains(p0)) {
                res= PrimosHermanosOtros.HERMANOS;
                break;
        }
            for(Persona a0 : abuelosP1) {
                if (abuelosP2.contains(a0)) {
                    res= PrimosHermanosOtros.PRIMOS;
                    break;
                }
            }
    }
    return res;
}
```

```java
    // ================================================================================
    // APARTADO D
    // ================================================================================

    /*d. 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.*/

    private static List<Persona> getHijos (Graph<Persona,Relacion>gf,Persona p){
        Set<Relacion> aritasHijos = gf.outgoingEdgesOf(p);
        List<Persona> hijos = List2.empty();
        for(Relacion r : aritasHijos) {
            hijos.add(gf.getEdgeTarget(r));
        }
        return hijos;
    }
    public static Set<Persona> apartadoD (Graph<Persona,Relacion>gf){
        /*Vamos a preguntar a cada persona del grafo quienes son sus hijos y a cada hijo de la
         * persona p vamos a preguntar quienes son sus padres. Sus padres los metemos en un Set de tal formal
         * que cada hijo tiene que tener 2 padres, si hemos preguntado a todos los hijos y uno de ellos tiene un padre
         * diferente en el conjunto que hemos creado habrá 3 padres en vez de dos es decir que esa persona ha tenido
         * un hijo con más de una persona*/

        Set<Persona>res = Set2.empty();
        Set<Persona>personas = gf.vertexSet(); //Todas las personas del grafo
        for (Persona p: personas) {
            List<Persona>hijos = getHijos(gf, p); //hijos de cada persona
            if(!hijos.isEmpty()){
                Set<Persona> padresCadaHijo = Set2.empty(); //Padres de cada hijo
                for(Persona h :hijos) { // a cada hijo le preguntamos cuales son sus padres
                    List<Persona> padres = getPadres(gf, h);
                    for(Persona padre: padres) {
                        padresCadaHijo.add(padre); //añadimos los padres al set
                    }
                }
                if(padresCadaHijo.size()>2) { //si hay mas de dos padres significa que todos los hermanos no tienen los mismos padres
                    res.add(p);
                }
            }
        }
        return res;

    }
    public static void apartadoDColorear(Graph<Persona, Relacion> gf, Set<Persona> ls,String file) {

        GraphColors.toDot(gf, "resultados/ejercicios/ejercicio1/" + file + ".gv",
                p -> p.nombre(),
                r -> "",
                p -> GraphColors.color(
                    ls.contains(p) ?
                    Color.blue : Color.black),

                r -> GraphColors.style(Style.solid));
    }
    // ================================================================================
    // APARTADO E
    // ================================================================================

    /*e. 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 Set<Persona> apartadoE (Graph<Persona,Relacion>gf) {

        GreedyVCImpl vCover = new GreedyVCImpl(gf);
        Set<Persona> res = vCover.getVertexCover();
        return res;
    }

    public static void apartadoEColorear(Graph<Persona, Relacion> gf,Set<Persona>ls,String file) {

        GraphColors.toDot(gf, "resultados/ejercicios/ejercicio1/" + file + ".gv",
                p -> p.nombre(),
                r -> "",
                p -> GraphColors.color(
                    ls.contains(p) ?
                    Color.blue : Color.black),
                r -> GraphColors.style(Style.solid));
    }
}
```

EJERCICIO 1 TEST

```java
public class TestEjercicio1 {


    public static void main(String[] args) {
//------------DATOS DE ENTRADA ------------------------
        DatosDeEntrada("PI3E1A_DatosEntrada");
        DatosDeEntradaB("PI3E1B_DatosEntrada");

//------------APARTADO A-------------------------------
        apartadoA_A("PI3E1A_DatosEntrada");
        apartadoA_B("PI3E1B_DatosEntrada");

//------------APARTADO B-------------------------------

        apartadoB_A("PI3E1A_DatosEntrada");
        apartadoB_B("PI3E1B_DatosEntrada");

//------------APARTADO C-------------------------------

        apartadoC_A("PI3E1A_DatosEntrada");
        apartadoC_B("PI3E1B_DatosEntrada");

//------------APARTADO D-------------------------------

        apartadoD_A("PI3E1A_DatosEntrada");
        apartadoD_B("PI3E1B_DatosEntrada");

//------------APARTADO E-------------------------------

        apartadoE_A("PI3E1A_DatosEntrada");
        apartadoE_B("PI3E1B_DatosEntrada");
    }
```

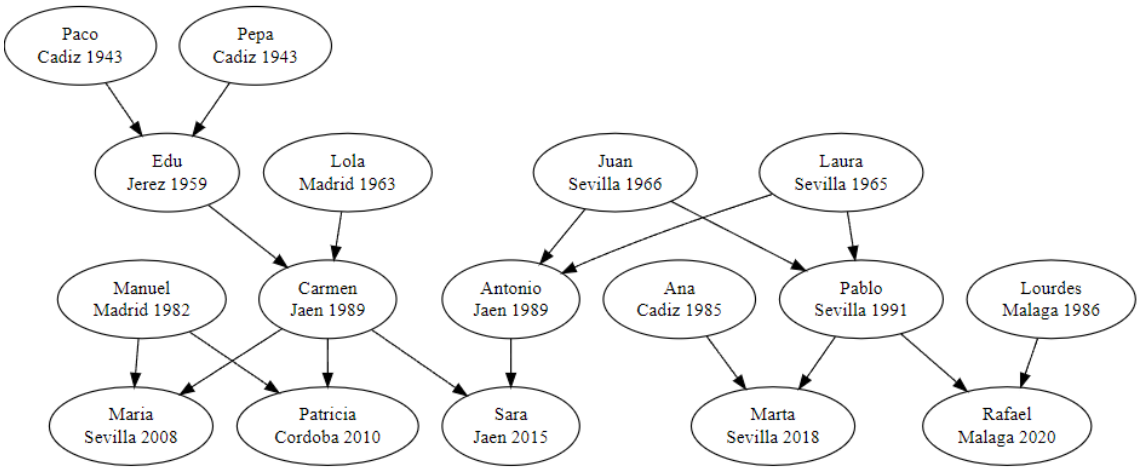
```java
//============================================================
//                   FICHERO ENTRADA A
//============================================================

//------------DATOS DE ENTRADA ------------------------
    public static void DatosDeEntrada(String file) {
        //Leer grafo
        Graph<Persona, Relacion> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleDirectedGraph);

        GraphColors.toDot(gf,
                        "resultados/ejercicios/ejercicio1/" +"DatosDeEntrada_A"+ ".gv",
                        x -> x.toString2(),
                        x -> x.toString(),
                        v -> GraphColors.color(Color.black),
                        e -> GraphColors.color(Color.black));
    }

//------------APARTADO A------------------------------
    public static void apartadoA_A(String file) {
        Graph<Persona, Relacion> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
                Persona::ofFormat,
                Relacion::ofFormat,
                Graphs2::simpleDirectedGraph);

        Set<Persona> s = Ejercicio1.apartadoA(g);

        Ejercicio1.apartadoAColorear(g, s,"Apartado A_A");
        System.out.println("====================APARTADO A_A ================================================");
        System.out.println("Personas cuyos padres aparecen en el grafo y cumplen los requisitos:\n " + s);
        System.out.println("================================================================================");
    }
//              APARTADO B

//------------APARTADO B------------------------------
    public static void apartadoB_A(String file) {
            Graph<Persona, Relacion> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
                    Persona::ofFormat,
                    Relacion::ofFormat,
                    Graphs2::simpleDirectedGraph);

            Persona Maria = Persona.of(13,"Maria",2008,"Sevilla");
            Set<Persona> s = Ejercicio1.apartadoB(g,Maria);

            Ejercicio1.apartadoBColorear(g, s,Maria,"Apartado B_A");
            System.out.println("====================APARTADO B_A ================================================");
            System.out.println("Ancestros de Maria:\n " + s);
            System.out.println("================================================================================");
        }
//------------APARTADO C------------------------------
    public static void apartadoC_A(String file) {
            Graph<Persona, Relacion> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
                    Persona::ofFormat,
                    Relacion::ofFormat,
                    Graphs2::simpleDirectedGraph);

                Persona Maria = Persona.of(13,"Maria",2008,"Sevilla");
                Persona Patricia = Persona.of(12,"Patricia",2010,"Cordoba");
                Persona Rafael = Persona.of(16,"Rafael",2020,"Malaga");
                Persona Sara = Persona.of(14,"Sara",2015,"Jaen");
                Persona Carmen = Persona.of(8,"Carmen",1989,"Jaen");

                PrimosHermanosOtros MariayPatricia = Ejercicio1.apartadoC(g,Maria,Patricia);
                PrimosHermanosOtros RafaelySara= Ejercicio1.apartadoC(g,Rafael,Sara);
                PrimosHermanosOtros CarmenyRafael= Ejercicio1.apartadoC(g,Carmen,Rafael);


                System.out.println("====================APARTADO C_A ================================================");
                System.out.println("Rafael y Sara son: " + RafaelySara);
                System.out.println("Maria y Patricia son: " + MariayPatricia);
                System.out.println("Maria y Patricia son: " + CarmenyRafael);
                System.out.println("================================================================================");
            }
```

```
            ,
//------------APARTADO D------------------------------
    public static void apartadoD_A(String file) {
            Graph<Persona, Relacion> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                    Persona::ofFormat,
                    Relacion::ofFormat,
                    Graphs2::simpleDirectedGraph);

                Set<Persona> s = Ejercicio1.apartadoD(gf);
                Ejercicio1.apartadoDColorear(gf,s,"Apartado D_A");

                System.out.println("====================APARTADO D_A ================================================");
            System.out.println("Personas que tienen hijos/as con distintas personas "+Ejercicio1.apartadoD(gf));
                System.out.println("================================================================================");
            }
//------------APARTADO E------------------------------
    public static void apartadoE_A(String file) {
                Graph<Persona, Relacion> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleGraph);

                    Set<Persona> s = Ejercicio1.apartadoE(gf);

                    System.out.println("====================APARTADO E_A ================================================");
                    Ejercicio1.apartadoEColorear(gf,s,"Apartado E_A");
                    System.out.println(s);
                    System.out.println("================================================================================");
                }



//============================================================
//                  FICHERO ENTRADA B
//============================================================

//------------DATOS DE ENTRADA ------------------------
    public static void DatosDeEntradaB(String file) {
            //Leer grafo
            Graph<Persona, Relacion> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                            Persona::ofFormat,
                            Relacion::ofFormat,
                            Graphs2::simpleDirectedGraph);

            GraphColors.toDot(gf,
                            "resultados/ejercicios/ejercicio1/" +"DatosDeEntrada_B"+ ".gv",
                            x -> x.toString2(),
                            x -> x.toString(),
                            v -> GraphColors.color(Color.black),
                            e -> GraphColors.color(Color.black));
        }

//------------APARTADO A------------------------------
    public static void apartadoA_B(String file) {
            Graph<Persona, Relacion> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
                    Persona::ofFormat,
                    Relacion::ofFormat,
                    Graphs2::simpleDirectedGraph);

            Set<Persona> s = Ejercicio1.apartadoA(g);

            Ejercicio1.apartadoAColorear(g, s,"Apartado A_B");
            System.out.println("====================APARTADO A_B ================================================");
            System.out.println("Personas cuyos padres aparecen en el grafo y cumplen los requisitos:\n " + s);
            System.out.println("================================================================================");
        }
//------------APARTADO B------------------------------
    public static void apartadoB_B(String file) {
                Graph<Persona, Relacion> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleDirectedGraph);

                Persona Raquel = Persona.of(13,"Raquel",1993,"Sevilla");
                Set<Persona> s = Ejercicio1.apartadoB(g,Raquel);

                Ejercicio1.apartadoBColorear(g, s,Raquel,"Apartado B_B");
                System.out.println("====================APARTADO B_B ================================================");
                System.out.println("Ancestros de Raquel:\n " + s);
                System.out.println("================================================================================");
            }
```

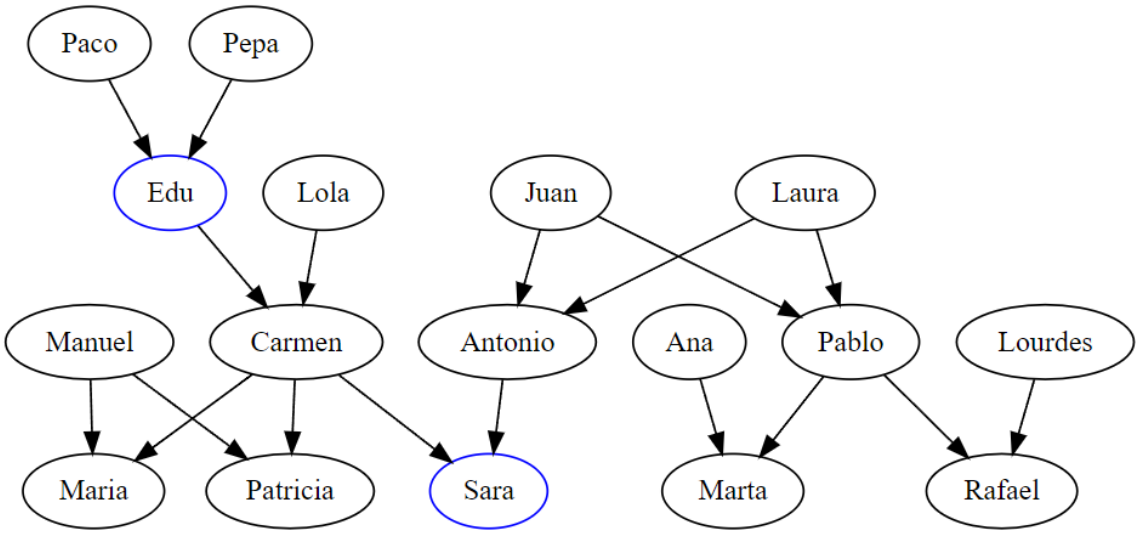
```java
//------------APARTADO C------------------------------
    public static void apartadoC_B(String file) {
                Graph<Persona, Relacion> g = GraphsReader.newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleDirectedGraph);

                    Persona Julia = Persona.of(14,"Julia",1996,"Jaen");
                    Persona Angela = Persona.of(6,"Angela",1997,"Sevilla");
                    Persona Alvaro = Persona.of(15,"Alvaro",2000,"Sevilla");
                    Persona Raquel = Persona.of(13,"Raquel",1993,"Sevilla");
                    Persona Laura = Persona.of(3,"Laura",1965,"Jerez");

                    PrimosHermanosOtros JuliayAngela = Ejercicio1.apartadoC(g,Julia,Angela);
                    PrimosHermanosOtros AlvararoyRaquel= Ejercicio1.apartadoC(g,Alvaro,Raquel);
                    PrimosHermanosOtros LaurayRaquel= Ejercicio1.apartadoC(g,Laura,Raquel);


                    System.out.println("==================APARTADO C_B ================================================");
                    System.out.println("Julia y Angela son: " + JuliayAngela);
                    System.out.println("Alvaro y Raquel son: " + AlvararoyRaquel);
                    System.out.println("Laura y Raquel son: " + LaurayRaquel);
                    System.out.println("================================================================================");
                }
//------------APARTADO D------------------------------
    public static void apartadoD_B(String file) {
                Graph<Persona, Relacion> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleDirectedGraph);

                    Set<Persona> s = Ejercicio1.apartadoD(gf);
                    Ejercicio1.apartadoDColorear(gf,s,"Apartado D_B");

                    System.out.println("==================APARTADO D_B ================================================");
                System.out.println("Personas que tienen hijos/as con distintas personas "+Ejercicio1.apartadoD(gf));
                    System.out.println("================================================================================");
                }
//             ------------

                ,

//------------APARTADO E------------------------------
    public static void apartadoE_B(String file) {
                   Graph<Persona, Relacion> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                           Persona::ofFormat,
                           Relacion::ofFormat,
                           Graphs2::simpleGraph);

                       Set<Persona> s = Ejercicio1.apartadoE(gf);

                       System.out.println("==================APARTADO E_B ================================================");
                       Ejercicio1.apartadoEColorear(gf,s,"Apartado E_B");
                       System.out.println(s);
                       System.out.println("================================================================================");
                   }

//=============================================================
```

```
====================APARTADO A_A ================================================
Personas cuyos padres aparecen en el grafo y cumplen los requisitos:
 [Edu, Sara]
================================================================================
====================APARTADO A_B ================================================
Personas cuyos padres aparecen en el grafo y cumplen los requisitos:
 [Raquel, Angela, Josefina, Julia]
================================================================================
====================APARTADO B_A ================================================
Ancestros de Maria:
 [Manuel, Pepa, Paco, Edu, Lola, Carmen]
================================================================================
====================APARTADO B_B ================================================
Ancestros de Raquel:
 [Irene, Daniel, Encarna, Pedro, Manuela, Ramon, Francisco, Josefina]
================================================================================
====================APARTADO C_A ================================================
Rafael y Sara son: PRIMOS
Maria y Patricia son: HERMANOS
Maria y Patricia son: OTROS
================================================================================
====================APARTADO C_B ================================================
Julia y Angela son: PRIMOS
Alvaro y Raquel son: HERMANOS
Laura y Raquel son: OTROS
================================================================================
====================APARTADO D_A ================================================
Personas que tienen hijos/as con distintas personas [Pablo, Carmen]
================================================================================
====================APARTADO D_B ================================================
Personas que tienen hijos/as con distintas personas [Josefina]
================================================================================
====================APARTADO E_A ================================================
[Carmen, Pablo, Antonio, Edu, Manuel, Ana, Rafael]
================================================================================
====================APARTADO E_B ================================================
[Josefina, Ramon, Laura, Pedro, Marcos, Javier]
================================================================================
```

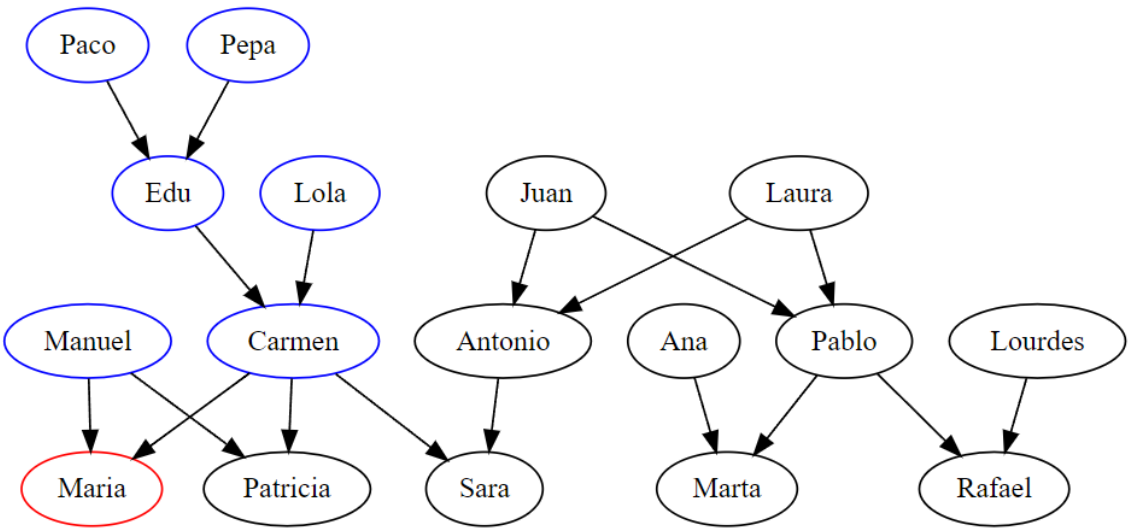
ENTRADA A



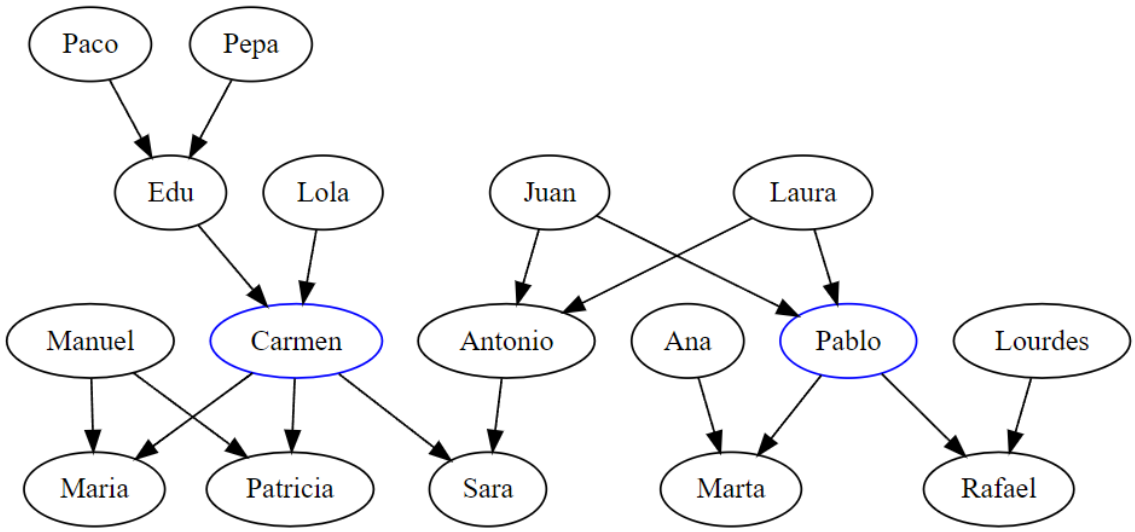
```
==================APARTADO A_A ================================================
Personas cuyos padres aparecen en el grafo y cumplen los requisitos:
 [Edu, Sara]
================================================================================
```



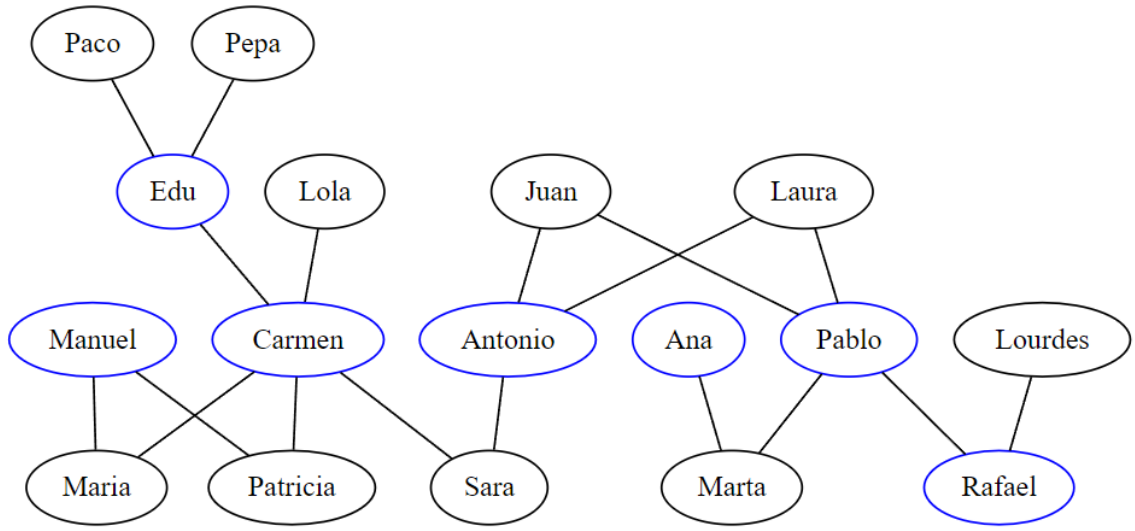

```
==================APARTADO B_A ================================================
Ancestros de Maria:
 [Manuel, Pepa, Paco, Edu, Lola, Carmen]
================================================================================
```



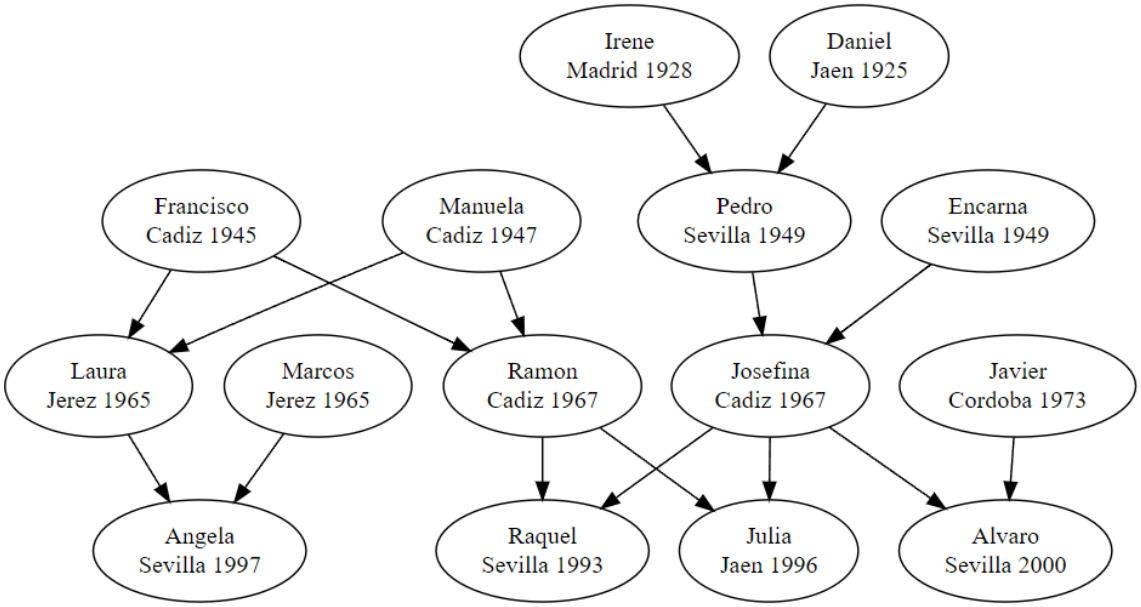

```
==================APARTADO C_A ================================================
Rafael y Sara son: PRIMOS
Maria y Patricia son: HERMANOS
Maria y Patricia son: OTROS
================================================================================

==================APARTADO D_A ================================================
Personas que tienen hijos/as con distintas personas [Pablo, Carmen]
================================================================================
```



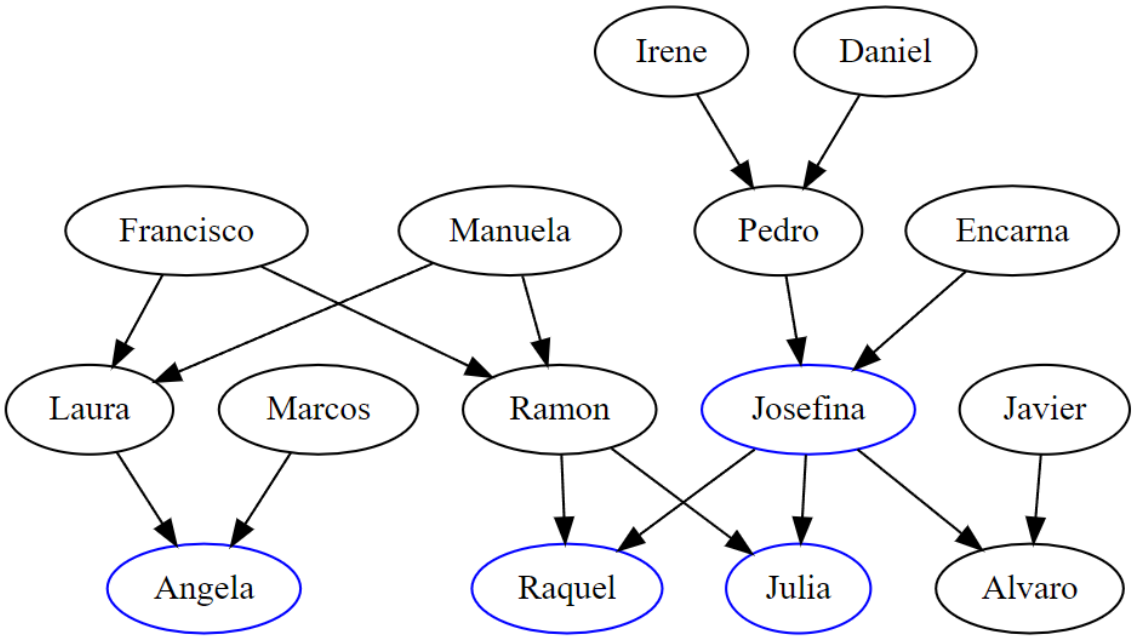

```
====================APARTADO E_A ================================================
[Carmen, Pablo, Antonio, Edu, Manuel, Ana, Rafael]
================================================================================
```



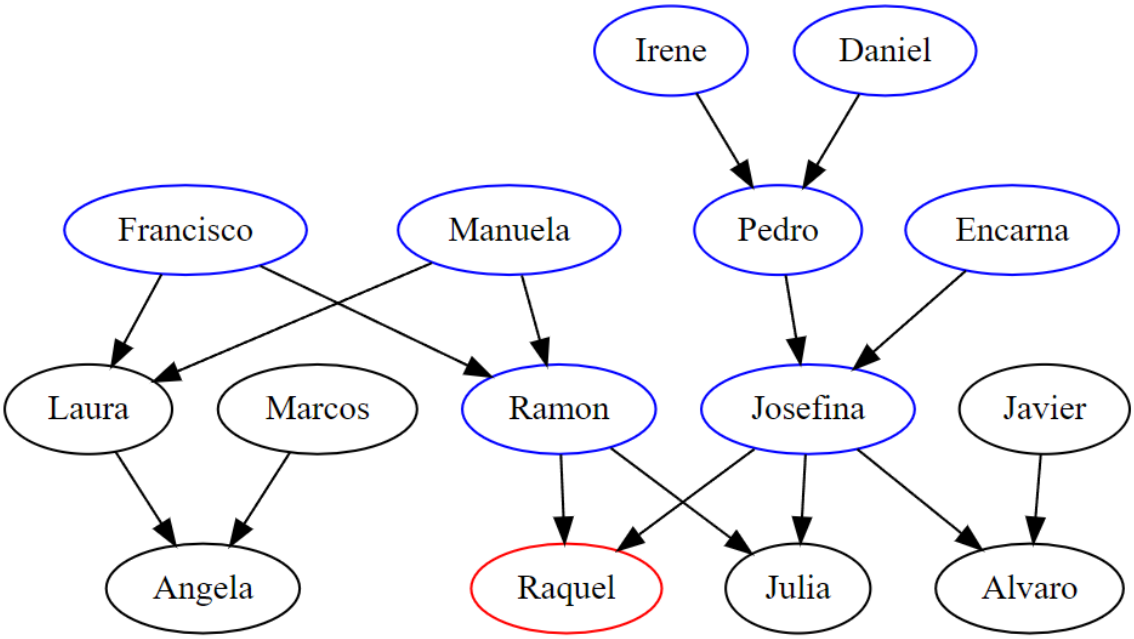
ENTRADA B




```
==================APARTADO A_B ============================================
Personas cuyos padres aparecen en el grafo y cumplen los requisitos:
 [Raquel, Angela, Josefina, Julia]
================================================================================
```

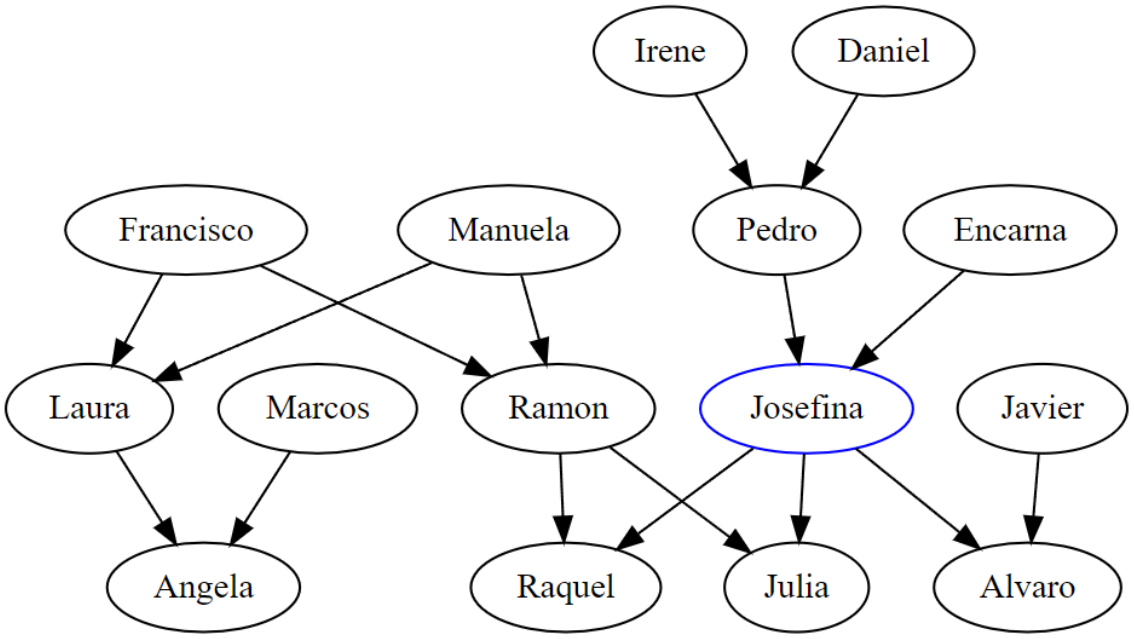



```
==================APARTADO B_B ============================================
Ancestros de Raquel:
 [Irene, Daniel, Encarna, Pedro, Manuela, Ramon, Francisco, Josefina]
================================================================================
```

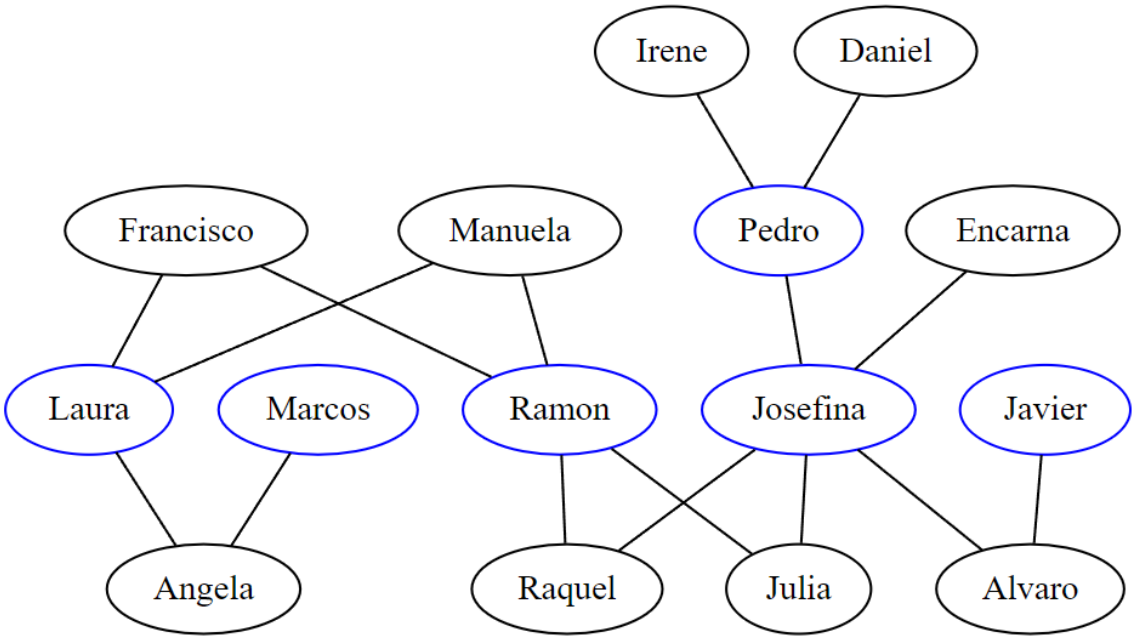


```
==================APARTADO C_B ============================================
Julia y Angela son: PRIMOS
Alvaro y Raquel son: HERMANOS
Laura y Raquel son: OTROS
==========================================================================
```

```
==================APARTADO D_A ============================================
Personas que tienen hijos/as con distintas personas [Pablo, Carmen]
==========================================================================
```



```
====================APARTADO E_B ============================================
[Josefina, Ramon, Laura, Pedro, Marcos, Javier]
================================================================================
```



EJERCICIO 2

```java
public class Ejercicio2 {

    // ============================================================================================
    // APARTADO A
    // ============================================================================================
    /*a. 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.*/

    public static List<Set<Ciudad>> apartadoA(Graph<Ciudad, Trayecto> gf) {
        //Componentes conexas

        ConnectivityInspector alg = new ConnectivityInspector<>(gf);
        List<Set<Ciudad>> ls = alg.connectedSets(); //Listas con las componentes conexas que hay


        Set<Ciudad> vertices0 = ls.get(0); //vertices primera componente
        Set<Trayecto> aristas0 = Set2.empty(); //aristas primera componente
        for(Ciudad v : vertices0) {
            aristas0.addAll(gf.edgesOf(v));
        }
        GraphColors.toDot(gf,
                          "resultados/ejercicios/ejercicio2/" + "Apartado A" + ".gv",
                          x -> x.nombre(),
                          x -> "",
                          v -> GraphColors.color(vertices0.contains(v)?Color.green:Color.yellow),
                          e -> GraphColors.color(aristas0.contains(e)?Color.green:Color.yellow));
        return ls;
    }
```

```java
// ============================================================================================
// APARTADO B
// ============================================================================================
/*b. 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 Set<Ciudad> apartadoB(Graph<Ciudad, Trayecto> gf) {
    //Componentes conexas

    ConnectivityInspector alg = new ConnectivityInspector<>(gf);
    List<Set<Ciudad>> ls = alg.connectedSets();


    Set<Ciudad> vertices0 = ls.get(0);
    Set<Trayecto> aristas0 = Set2.empty();
    for(Ciudad v : vertices0) {
        aristas0.addAll(gf.edgesOf(v));
    }
    Set<Ciudad> vertices1 = ls.get(1);
    Set<Trayecto> aristas1 = Set2.empty();
    for(Ciudad v : vertices1) {
        aristas1.addAll(gf.edgesOf(v));
    }

    Integer puntuacion0=0;  //Puntuacion del primer grupo de ciudades
    for (Ciudad v : vertices0) {
        puntuacion0=puntuacion0+v.puntuacion();
    }
    Integer puntuacion1=0; //Puntuacion del segundo grupo de ciudades
    for (Ciudad v : vertices1) {
        puntuacion1=puntuacion1+v.puntuacion();
    }

    Set<Ciudad>ciudadesPuntacionMasAlta = puntuacion0>puntuacion1? vertices0:vertices1; // nos quedamos con el mas alto
    Set<Trayecto>aristasPuntacionMasAlta = Set2.empty();

    for(Ciudad v : ciudadesPuntacionMasAlta) {
        aristasPuntacionMasAlta.addAll(gf.edgesOf(v));
    }

    GraphColors.toDot(gf,
                    "resultados/ejercicios/ejercicio2/" + "Apartado B" + ".gv",
                    x -> x.nombre() +"\n"+x.puntuacion()+" puntos",
                    x -> "",
                    v -> GraphColors.color(ciudadesPuntacionMasAlta.contains(v)?Color.blue:Color.blank),
                    e -> GraphColors.color(aristasPuntacionMasAlta.contains(e)?Color.blue:Color.blank));

    return ciudadesPuntacionMasAlta;
}
```

```java
// ============================================================================================
    // APARTADO C
    // ============================================================================================
    /*c. 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 List<List<String>>  apartadoC(Graph<Ciudad, Trayecto> gf) {

    // Obtenemos todas las componentes conexas

    ConnectivityInspector alg = new ConnectivityInspector<>(gf);
    List<Set<Ciudad>> ls = alg.connectedSets();

    //Creamos subgrafo componente conexa 0
    Set<Ciudad> vertices0 = ls.get(0);
    Predicate<Ciudad> pv0 = c -> vertices0.contains(c);
    Predicate<Trayecto> pe0 = p -> vertices0.contains(gf.getEdgeSource(p)) && vertices0.contains(gf.getEdgeTarget(p));
    Graph<Ciudad, Trayecto> comp0 = SubGraphView.of(gf,pv0,pe0);


    //Creamos subgrafo componente conexa 1
    Set<Ciudad> vertices1 = ls.get(1);
    Predicate<Ciudad> pv1 = c -> vertices1.contains(c);
    Predicate<Trayecto> pe1 = p -> vertices1.contains(gf.getEdgeSource(p)) && vertices1.contains(gf.getEdgeTarget(p));
    Graph<Ciudad, Trayecto> comp1 = SubGraphView.of(gf,pv1,pe1);

    //comparamos que precio es el menor para visitar todas las ciudades

    Double precioComp0= new HeldKarpTSP().getTour(comp0).getWeight(); //Precio comp0
    List<String> verticesComp0= new HeldKarpTSP().getTour(comp0).getVertexList(); //camino minimo vertices comp0
    List<String> aristasComp0= new HeldKarpTSP().getTour(comp0).getEdgeList();//camino minimo aristas comp0

    Double precioComp1= new HeldKarpTSP().getTour(comp1).getWeight();//Precio comp0
    List<String> verticesComp1= new HeldKarpTSP().getTour(comp1).getVertexList();//camino minimo vertices comp0
    List<String> aristasComp1= new HeldKarpTSP().getTour(comp1).getEdgeList();//camino minimo aristas comp0

    //cogemos el conjunto de ciudades que tenga el menor precio
    List<String> resVertices = precioComp0 > precioComp1? verticesComp1:verticesComp0;
    List<String> resAristas = precioComp0 > precioComp1? aristasComp1:aristasComp0;

    //Nos quedamos con el precio que nos cuestan  recorrer esas ciudades
    Double resPrecio = precioComp0 > precioComp1? precioComp1:precioComp0;
    List<List<String>> res = List.of(resVertices,List.of(resPrecio.toString()));


   GraphColors.toDot(gf,
           "resultados/ejercicios/ejercicio2/" + "Apartado C" + ".gv",
           x -> x.nombre(),
           x -> x.euros().toString() + " euros",
           v -> GraphColors.color(resVertices.contains(v)?Color.blue:Color.blank),
           e -> GraphColors.color(resAristas.contains(e)?Color.blue:Color.blank));


   return res;
}
```

```java
// ============================================================================================
// APARTADO D
// ============================================================================================
/*d. 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.*/

private static Ciudad getVertexOf(Graph<Ciudad, Trayecto> graph, String nombre) {
    return graph.vertexSet().stream().filter(c -> c.nombre().equals(nombre)).findFirst().get();
}

public static Double apartadoD(Graph<Ciudad, Trayecto> gf,String file, String c1, String c2) {
    // Camino mínimo entre origen y destino: Dijkstra
    DijkstraShortestPath<Ciudad, Trayecto> alg = new DijkstraShortestPath<>(gf);

    Ciudad origen = getVertexOf(gf, c1);
    Ciudad destino = getVertexOf(gf, c2);

    GraphPath<Ciudad, Trayecto> path = alg.getPath(origen, destino);
    Double  tiempo  = path.getWeight();

    GraphColors.toDot(gf, "resultados/ejercicios/ejercicio2/" + file + ".gv",
            x -> x.nombre(),
            x -> x.minutos().toString() + " minutos",
            v -> GraphColors.styleIf(Style.bold, path.getVertexList().contains(v)),
            e -> GraphColors.styleIf(Style.bold, path.getEdgeList().contains(e)));

    return tiempo;
}
```

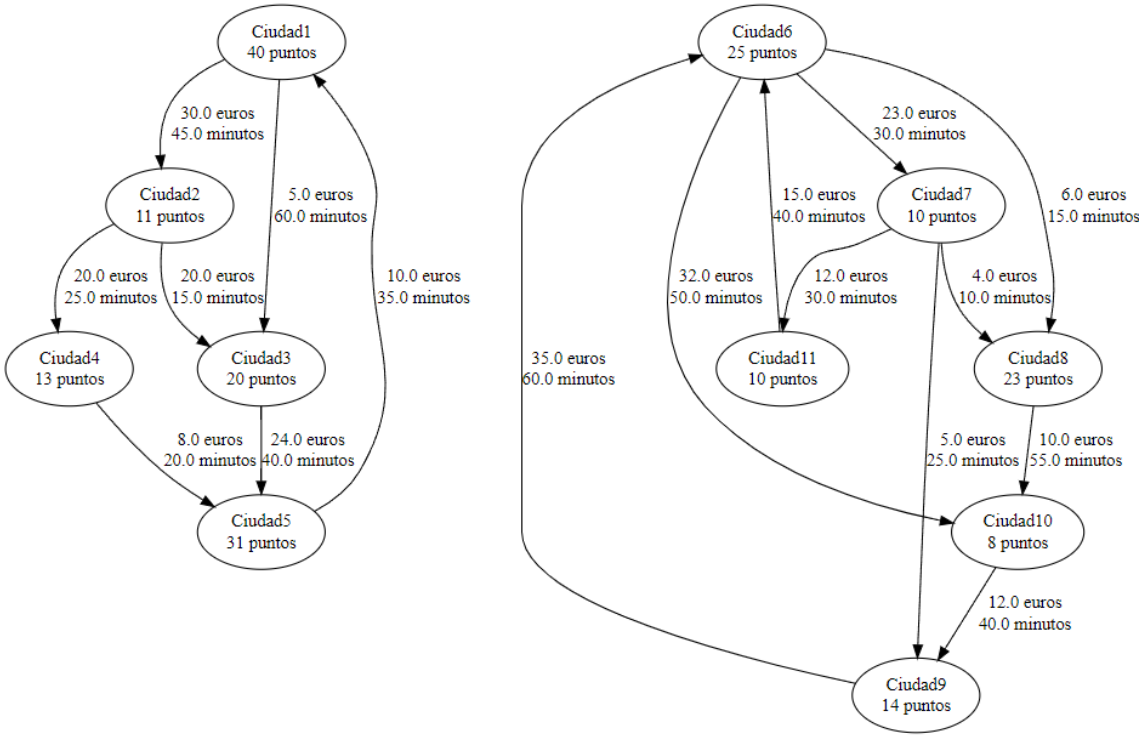
EJERCICIO 2 TEST

```java
    public static void main(String[] args) {
//------------DATOS DE ENTRADA ------------------------


        System.out.println("================================================================================");
        System.out.println("                              EJERCICIO 2                                       ");
        System.out.println("================================================================================ \n");

        DatosDeEntrada("PI3E2_DatosEntrada");
        //------------APARTADO A------------------------------
        TestApartadoA("PI3E2_DatosEntrada");
        //------------APARTADO B------------------------------
        TestApartadoB("PI3E2_DatosEntrada");
        //------------APARTADO C------------------------------
        TestApartadoC("PI3E2_DatosEntrada");
        //------------APARTADO D------------------------------
        TestApartadoD("PI3E2_DatosEntrada");
        }

//------------DATOS DE ENTRADA ------------------------
    public static void DatosDeEntrada(String file) {
        //Leer grafo
        Graph<Ciudad, Trayecto> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
                    Ciudad::ofFormat,
                    Trayecto::ofFormat,
                    Graphs2::simpleDirectedGraph);

            GraphColors.toDot(gf,
                        "resultados/ejercicios/ejercicio2/" +"EntradaEjercicio2"+ ".gv",
                        x -> x.toString2(),
                        x -> x.toString(),
                        v -> GraphColors.color(Color.black),
                        e -> GraphColors.color(Color.black));
        }
```

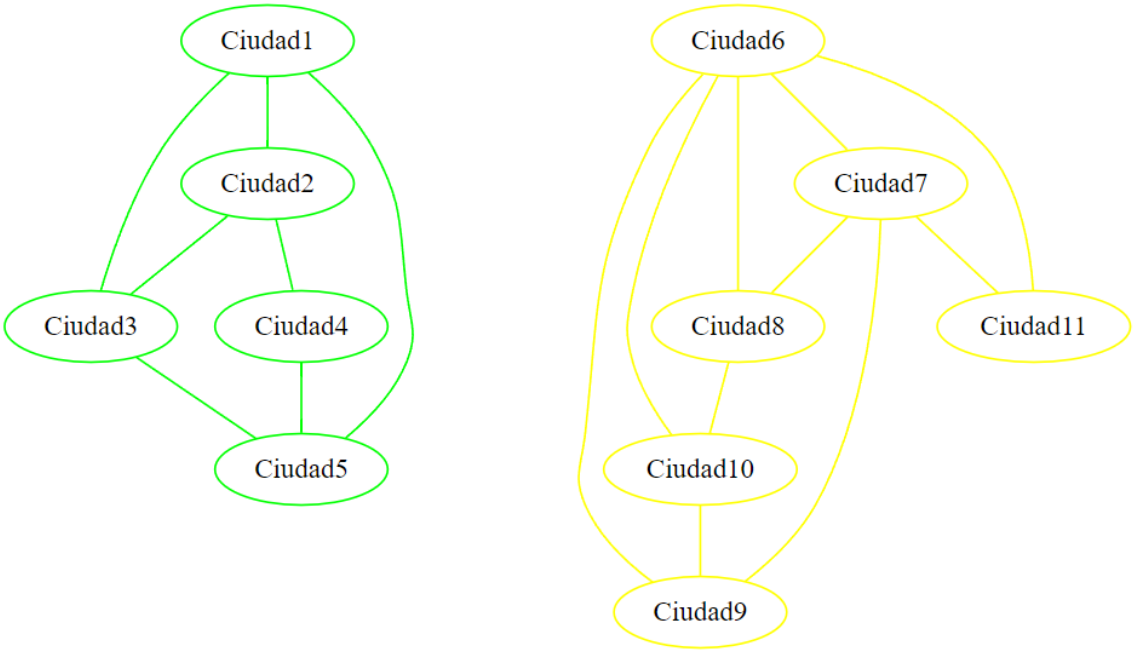
```java
//------------APARTADO A-------------------------------
    public static void TestApartadoA(String file) {
        Graph<Ciudad, Trayecto> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
        Ciudad::ofFormat,
        Trayecto::ofFormat,
        Graphs2::simpleGraph);


        List<Set<Ciudad>>ls = Ejercicio2.apartadoA(gf);

        System.out.println("====================APARTADO A ================================================");
        System.out.println("Hay " + ls.size() + " grupos de ciudades ");
        for(int i = 0; i<ls.size();i++) {
            System.out.println("Grupo numero "+(i+1)+":"+ls.get(i));
            }
        System.out.println("================================================================================");
        }
//------------APARTADO B-------------------------------
    public static void TestApartadoB(String file) {
        Graph<Ciudad, Trayecto> gf = GraphsReader.newGraph("ficheros/" + file + ".txt",
        Ciudad::ofFormat,
        Trayecto::ofFormat,
        Graphs2::simpleGraph);


        Set<Ciudad>ls = Ejercicio2.apartadoB(gf);

        System.out.println("====================APARTADO B ================================================");
        System.out.println("Grupo de ciudades que maximiza la suma de puntuaciones: "+ ls);
        System.out.println("================================================================================");
        }
//------------APARTADO C-------------------------------
    public static void TestApartadoC(String file) {
        SimpleWeightedGraph<Ciudad, Trayecto> graph = GraphsReader.newGraph("ficheros/" + file + ".txt",
                Ciudad::ofFormat,
                Trayecto::ofFormat,
                Graphs2::simpleWeightedGraph,
                Trayecto::euros);


        List<List<String>> ls = Ejercicio2.apartadoC(graph);

        Double Precio = Double.parseDouble(ls.get(1).get(0));

        System.out.println("====================APARTADO C ================================================");
        System.out.println("Grupo de ciudades a visitar que dan lugar al camino cerrado de menor precio: "+ ls.get(0)+"⟶ "+Precio + " euros");
        System.out.println("================================================================================");
        }



//------------APARTADO D-------------------------------
    public static void TestApartadoD(String file) {
        SimpleWeightedGraph<Ciudad, Trayecto> graph = GraphsReader.newGraph("ficheros/" + file + ".txt",
                Ciudad::ofFormat,
                Trayecto::ofFormat,
                Graphs2::simpleWeightedGraph,
                Trayecto::minutos);

        ;

        ConnectivityInspector alg = new ConnectivityInspector<>(graph);
        List<Set<Ciudad>> ls = alg.connectedSets();

        Set<Ciudad>componente1 = ls.get(0);
        Set<Ciudad>componente2 = ls.get(1);

        Double tiempo1 = Ejercicio2.apartadoD(graph,"Apartado D(componente1)", "Ciudad4","Ciudad3");
        Double tiempo2 = Ejercicio2.apartadoD(graph,"Apartado D(componente2)", "Ciudad8","Ciudad9");


        System.out.println("====================APARTADO C ================================================");
        System.out.println("Para el grupo "+componente1 +" las ciudades no conectadas directamente entre las que\r\n"
                + "se puede viajar en menor tiempo son:\r\n"
                + "Origen: Ciudad8 y Destino: Ciudad9 ⟶ Tiempo: "+tiempo1 +" minutos");
        System.out.println("                                                                                ");
        System.out.println("Para el grupo "+componente2 +" las ciudades no conectadas directamente entre las que\r\n"
                + "se puede viajar en menor tiempo son:\r\n"
                + "Origen: Ciudad8 y Destino: Ciudad9 ⟶ Tiempo: "+tiempo2 +" minutos");

        System.out.println("================================================================================");
            }

}
```

```
================================================================================
                            EJERCICIO 2
================================================================================

====================APARTADO A ================================================
Hay 2 grupos de ciudades
Grupo numero 1:[Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]
Grupo numero 2:[Ciudad8, Ciudad11, Ciudad10, Ciudad6, Ciudad7, Ciudad9]
================================================================================
====================APARTADO B ================================================
Grupo de ciudades que maximiza la suma de puntuaciones: [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]
================================================================================
====================APARTADO C ================================================
Grupo de ciudades a visitar que dan lugar al camino cerrado de menor precio: [Ciudad8, Ciudad10, Ciudad9, Ciudad7, Ciudad11, Ciudad6, Ciudad8]⟶ 60.0 euros
================================================================================
====================APARTADO C ================================================
Para el grupo [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1] las ciudades no conectadas directamente entre las que
se puede viajar en menor tiempo son:
Origen: Ciudad8 y Destino: Ciudad9 ⟶ Tiempo: 40.0 minutos

Para el grupo [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
================================================================================
```

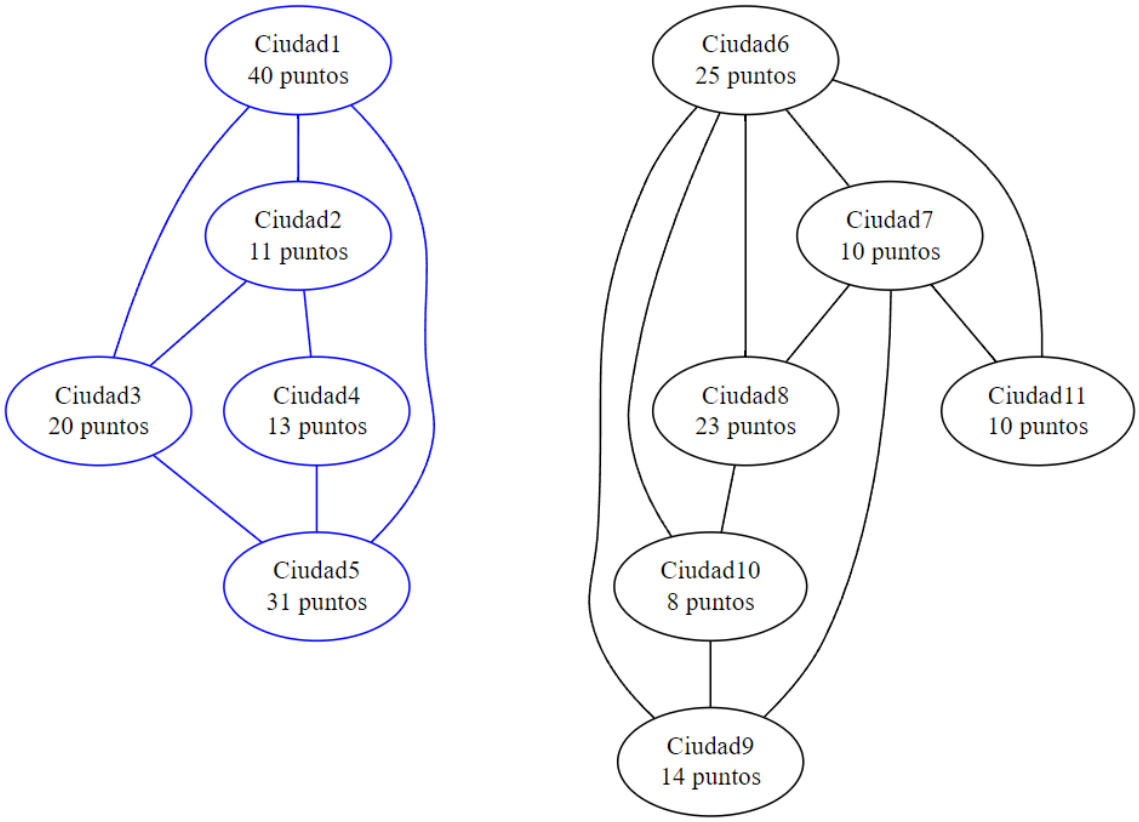
ENTRADA



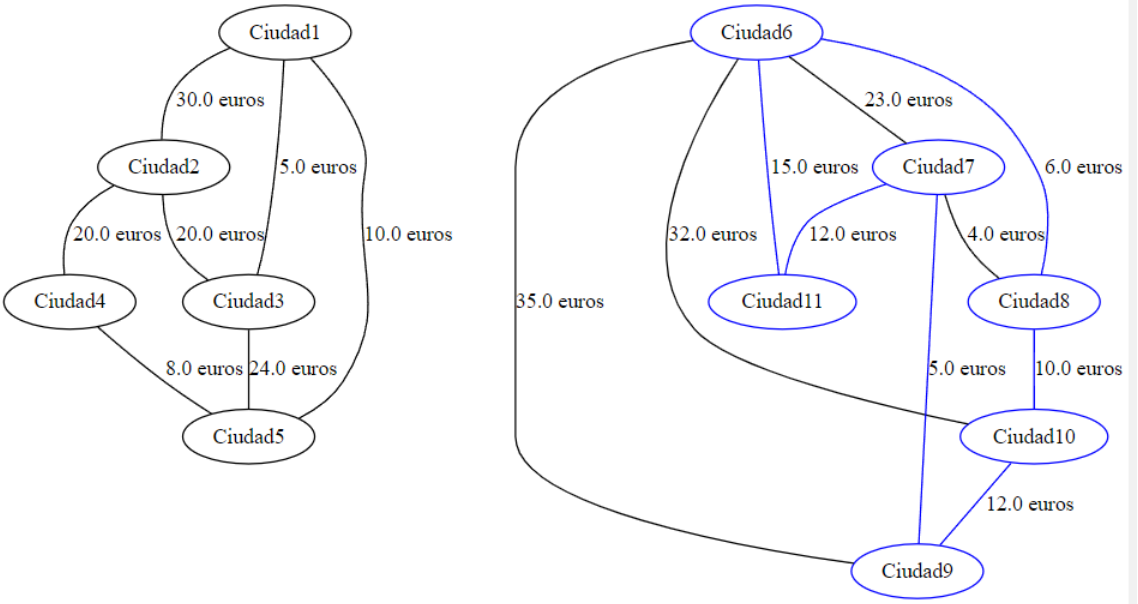
```
==================APARTADO A ============================================
Hay 2 grupos de ciudades
Grupo numero 1:[Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]
Grupo numero 2:[Ciudad8, Ciudad11, Ciudad10, Ciudad6, Ciudad7, Ciudad9]
================================================================================
```



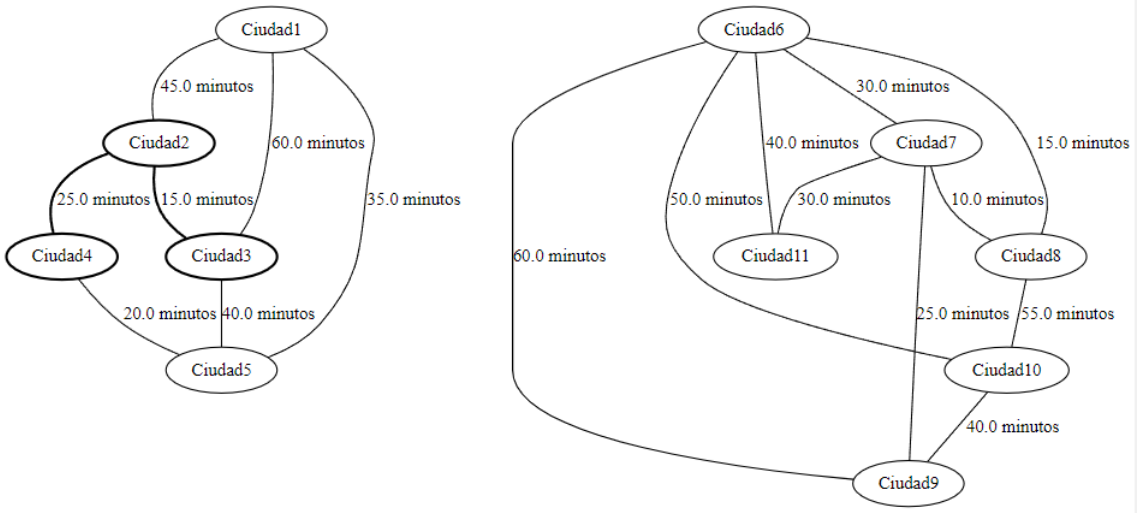
```
==================APARTADO B ================================================
Grupo de ciudades que maximiza la suma de puntuaciones: [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]
============================================================================
```



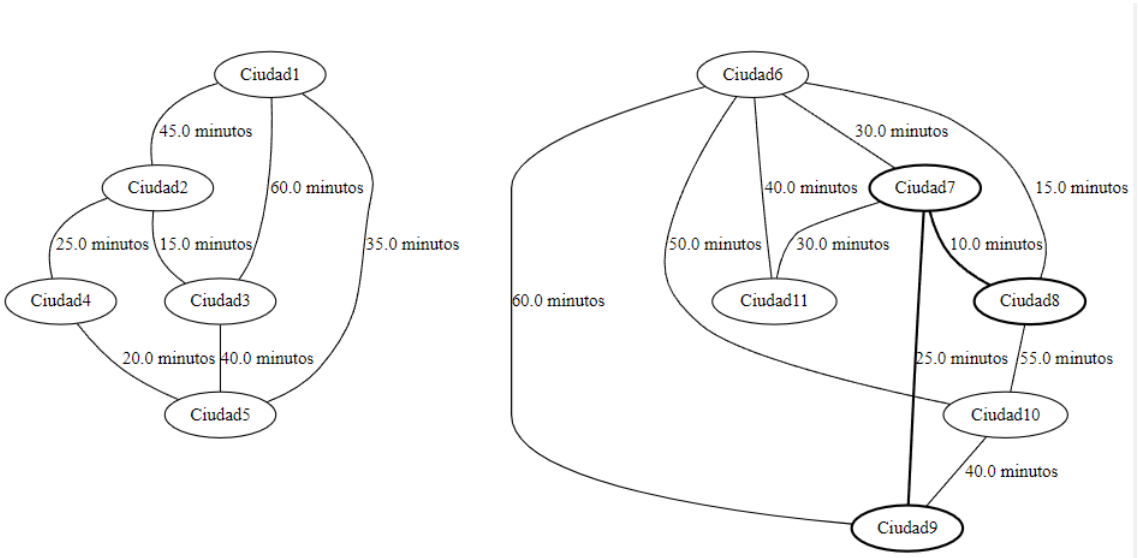
```
==================APARTADO C ================================================
Grupo de ciudades a visitar que dan lugar al camino cerrado de menor precio: [Ciudad8, Ciudad10, Ciudad9, Ciudad7, Ciudad11, Ciudad6, Ciudad8]⟶ 60.0 euros
================================================================================
```




```
==================APARTADO D ================================================
Para el grupo [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1] las ciudades no conectadas directamente entre las que
se puede viajar en menor tiempo son:
Origen: Ciudad8 y Destino: Ciudad9 ⟶ Tiempo: 40.0 minutos

Para el grupo [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
================================================================================
```



```
==================APARTADO D ================================================
Para el grupo [Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1] las ciudades no conectadas directamente entre las que
se puede viajar en menor tiempo son:
Origen: Ciudad8 y Destino: Ciudad9 ⟶ Tiempo: 40.0 minutos

Para el grupo [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
================================================================================
```



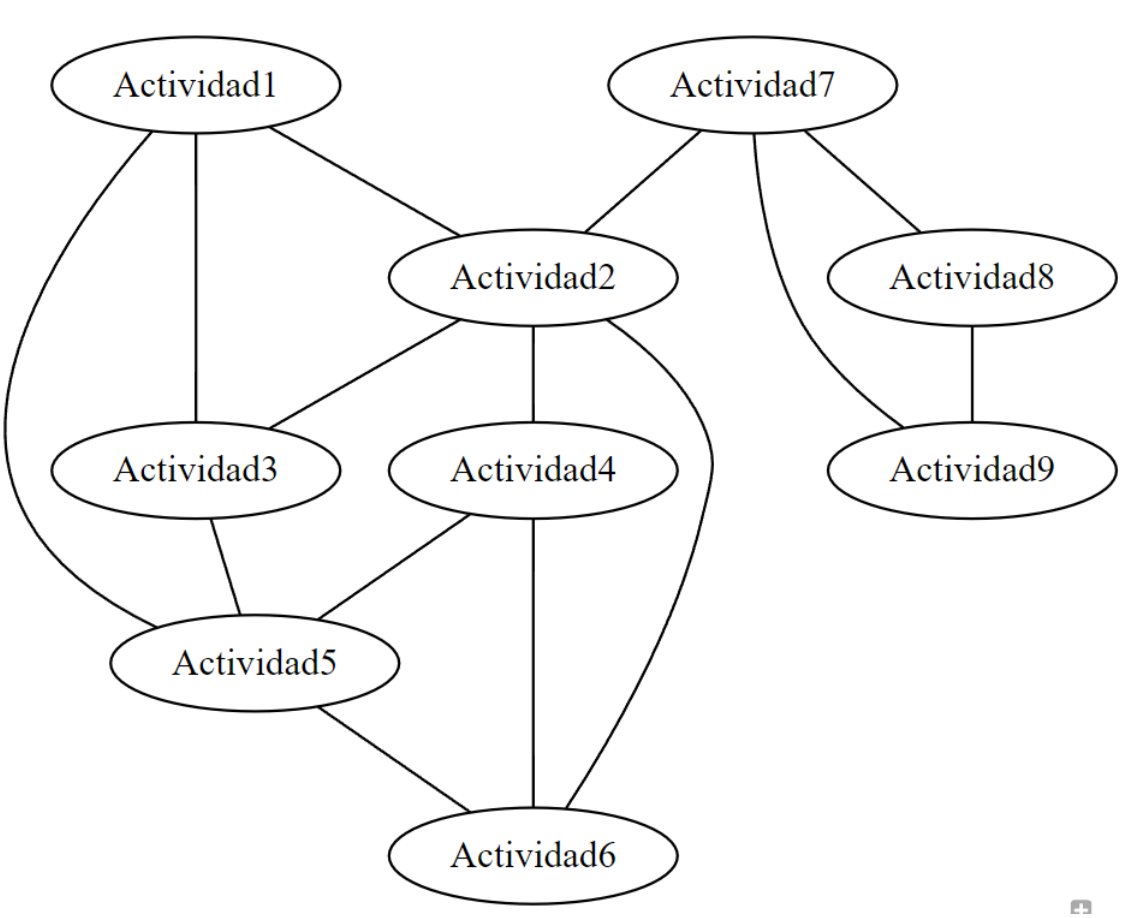
EJERCICIO 3

```java
public class Ejercicio3 {
    public static void todosLosApartados(Graph<String, DefaultEdge> gf, String file) {
        GreedyColoring<String, DefaultEdge> alg = new GreedyColoring<>(gf);
        Coloring<String> coloring = alg.getColoring();
        System.out.println("Numero de franjas horarias necesarias: " + coloring.getNumberColors());
        System.out.println("Actividades para impartirse en paralelo por franja horaria: ");
        List<Set<String>> composicion = coloring.getColorClasses();
        for (int i = 0; i < composicion.size(); i++) {
            System.out.println("Franja numero " + (i + 1) + ": " + composicion.get(i));
        }

        Map<String, Integer> map = coloring.getColors();

        GraphColors.toDot(gf,
                "resultados/ejercicios/ejercicio3/" + file + ".gv",
                v -> v.toString(),
                e -> "",
                v -> GraphColors.color(map.get(v)),
                e -> GraphColors.style(Style.arrowhead));


    }
}
```

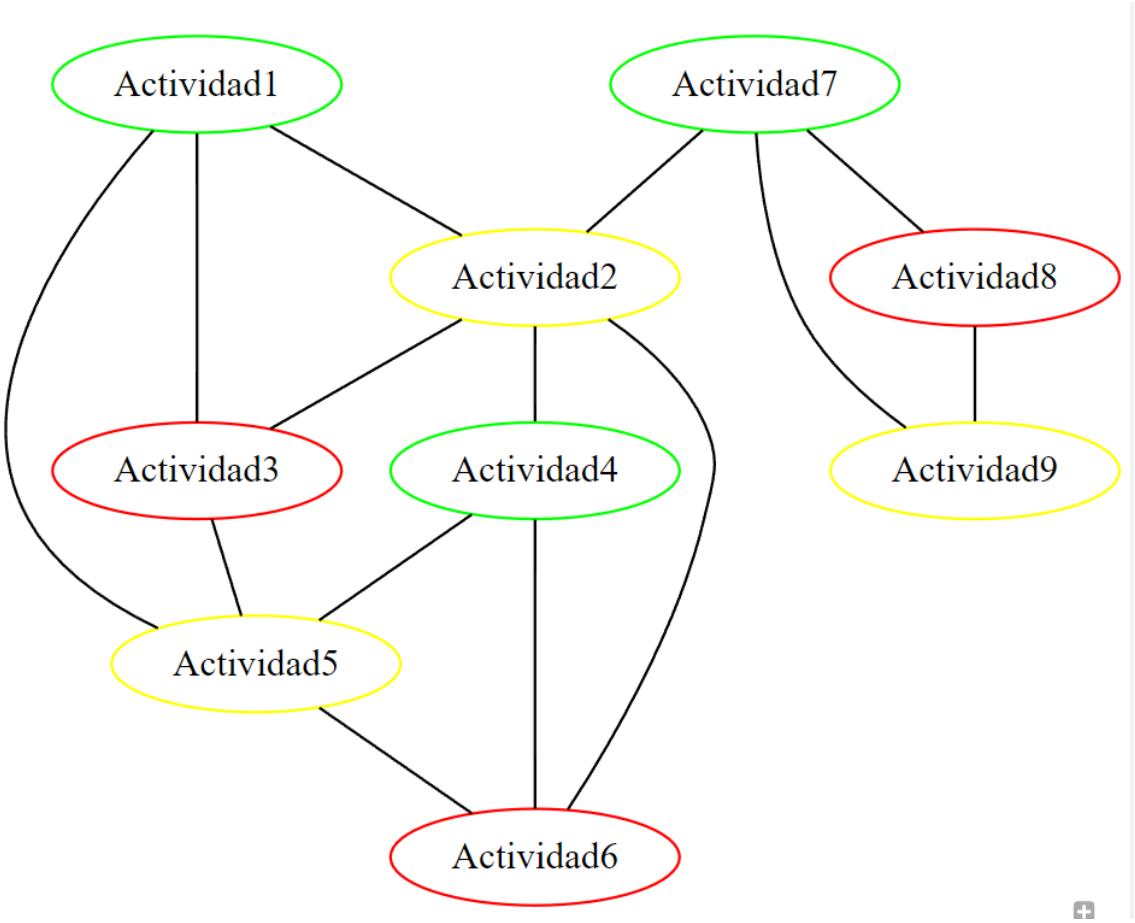
EJERCICIO 3 TEST

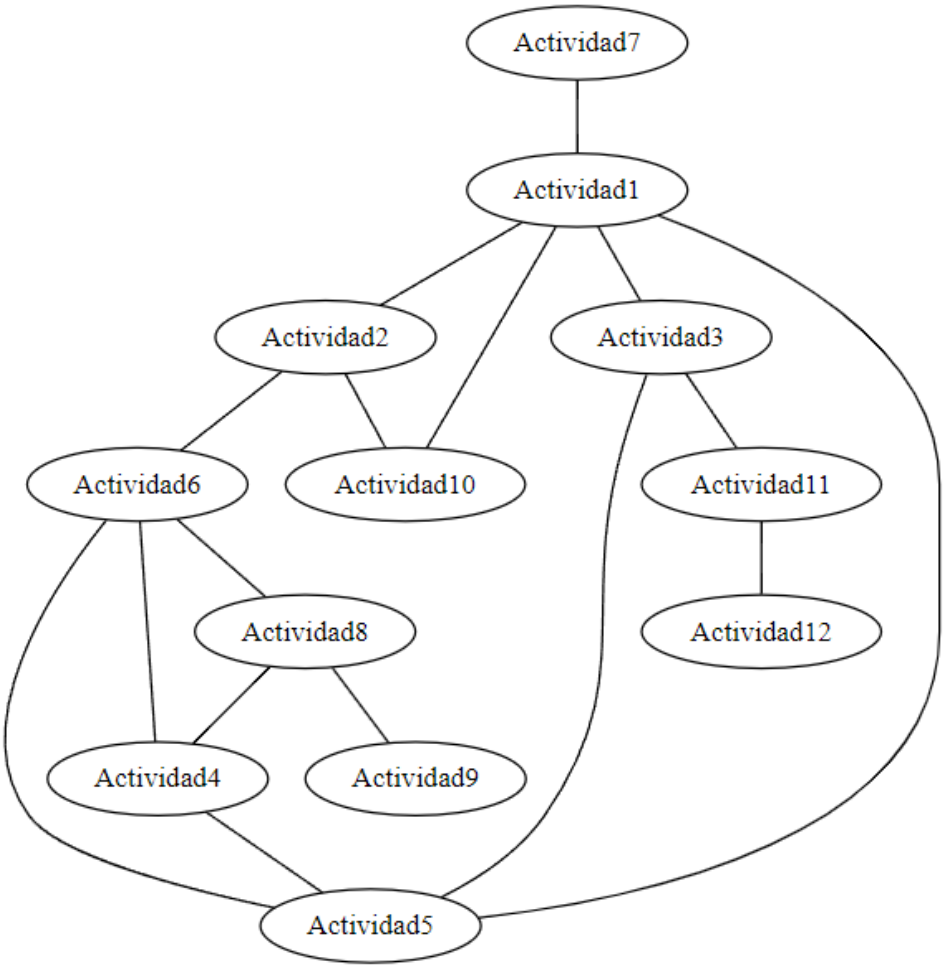
```java
public class TestEjercicio3 {
    public static void main(String[] args) {

        System.out.println("================================================================================");
        System.out.println("                    ENTRADA A ");
        System.out.println("================================================================================");
        TestEjercicio3A("PI3E3A_DatosEntrada");
        System.out.println("================================================================================");
        System.out.println("                    ENTRADA B ");
        System.out.println("================================================================================");
        TestEjercicio3B("PI3E3B_DatosEntrada");
    }
    public static void TestEjercicio3A(String file) {
        Graph<String, DefaultEdge> g = Graphs2.simpleGraph(String::new, DefaultEdge::new, false);

        List<String> lineas=Files2.streamFromFile("ficheros/" + file + ".txt").toList();
        Integer numlinea=0;
        for(String linea:lineas) {
            numlinea++;
             String []linea2 = linea.replace("Alumno"+numlinea+": ", "").trim().split(",");

            for(int i = 0; i<linea2.length;i++) {
                if(linea2.length>1) {
                if(i+1 < linea2.length) {
                g.addVertex(linea2[i].trim());
                g.addVertex(linea2[i+1].trim());
                g.addEdge(linea2[i].trim(), linea2[i+1].trim());}
                else {
                    g.addVertex(linea2[i].trim());
                    g.addVertex(linea2[0].trim());
                    g.addEdge(linea2[i].trim(), linea2[0].trim());
                }}
            }
        }

        GraphColors.toDot(g, "resultados/ejercicios/ejercicio3/" + file + "_inicialA.gv");


    ejercicios.Ejercicio3.todosLosApartados(g, "Ejercicio3 A");
    }
```

```java
public static void TestEjercicio3B(String file) {
    Graph<String, DefaultEdge> g = Graphs2.simpleGraph(String::new, DefaultEdge::new, false);

    List<String> lineas=Files2.streamFromFile("ficheros/" + file + ".txt").toList();
    Integer numlinea=0;
    for(String linea:lineas) {
        numlinea++;
         String []linea2 = linea.replace("Alumno"+numlinea+": ", "").trim().split(",");

        for(int i = 0; i<linea2.length;i++) {
            if(linea2.length>1) {
            if(i+1 < linea2.length) {
            g.addVertex(linea2[i].trim());
            g.addVertex(linea2[i+1].trim());
            g.addEdge(linea2[i].trim(), linea2[i+1].trim());}
            else {
                g.addVertex(linea2[i].trim());
                g.addVertex(linea2[0].trim());
                g.addEdge(linea2[i].trim(), linea2[0].trim());
            }}
        }
    }

    GraphColors.toDot(g, "resultados/ejercicios/ejercicio3/" + file + "_inicialB.gv");


ejercicios.Ejercicio3.todosLosApartados(g, "Ejercicio3 B");
}
```

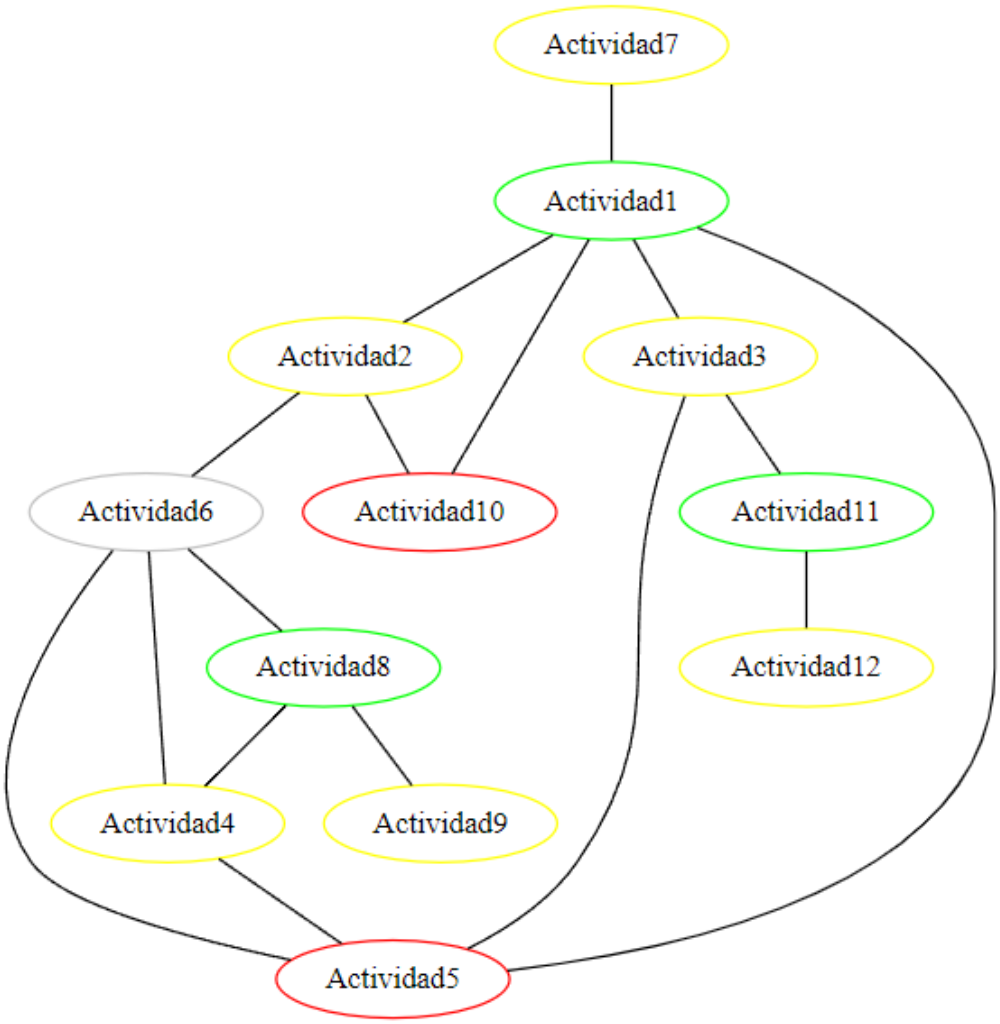


```
================================================================================
                          ENTRADA A
================================================================================
Numero de franjas horarias necesarias: 3
Actividades para impartirse en paralelo por franja horaria: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






```
================================================================================
                        ENTRADA B
================================================================================
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, Actividad7]
Franja numero 3: [Actividad10, Actividad5]
Franja numero 4: [Actividad6]
```


Actividad7
Actividad1
Actividad2
Actividad3
Actividad6
Actividad10
Actividad11
Actividad8
Actividad12
Actividad4
Actividad9
Actividad5