PRACTICA INDIVIDUAL 3

MEMORIA 1: Jose Luis Navarro (josnavlop4)

UNIVERSIDAD Đ SEVILLA

EJERCICIO 1

```
public class Ejercicio1 {

//  APARTADO A

    public static void apartadoA (SimpleDirectedGraph<Persona,Relacion> g, String file) {
        Graph<Persona,Relacion> vista = SubGraphView.of(g,
                x->padres(g,x).size()>1?
                        padres(g,x).stream().allMatch(y->y.anio().equals(padres(g,x).get
(0).anio())
                                && y.ciudad().equals(padres(g,x).get(0).ciudad())):false,
                        null);

        GraphColors.toDot(g,"resultados/ejercicio1/" + file + "apartadoA" + ".gv",
                x->x.nombre() , x->" ",
                v->GraphColors.colorIf(Color.blue, Color.black, vista.containsVertex(v)),
                e->GraphColors.style(Style.solid));

        System.out.println(file + "apartadoA" + ".gv generado en " +
"resultados/ejercicio1");
    }

    private static List<Persona> padres (SimpleDirectedGraph<Persona,Relacion> g,Persona p)
{
        return Graphs.predecessorListOf(g, p);
    }


//  APARTADO B

    public static Set<Persona> apartadoB (SimpleDirectedGraph<Persona,Relacion> g,Persona
p) {

        Graph<Persona,Relacion> vista = SubGraphView.of(g,x->ancestros(g,x,p),null);
        return vista.vertexSet();

    }

    public static Boolean ancestros (SimpleDirectedGraph<Persona,Relacion> g,Persona
p,Persona v) {
        Boolean ac = false;
        if(p.equals(v)){
            ac = true;
        } else if(Graphs.vertexHasPredecessors(g, v)) {
            List<Persona> ancestros = Graphs.predecessorListOf(g, v);
            for(int i=0;i<ancestros.size();i++) {
                ac = ancestros(g,p,ancestros.get(i));
                if(ac==true)break;
            }
        }
        return ac;
    }

//  APARTADO C

    public static void apartadoC (SimpleDirectedGraph<Persona,Relacion> g,Persona
p1,Persona p2) {
```

```
        son s = son.OTROS;

        Set<Persona> abuelosP1 = abuelos(g,p1);
        Set<Persona> abuelosP2 = abuelos(g,p1);

        if(sonHermanos(g,p1,p2)) {
            s = son.HERMANOS;
        } else if(abuelosP1.equals(abuelosP2)) {
            s = son.PRIMOS;
        }

        System.out.println(p1.nombre() + " y " + p2.nombre() + " son " + s.name());

    }

    public static Boolean sonHermanos (SimpleDirectedGraph<Persona,Relacion> g,Persona
p1,Persona p2) {

        List<Persona> padresP1 = padres(g,p1);
        List<Persona> padresP2 = padres(g,p2);

        Boolean ac = false;
        for (int i = 0; i<padresP1.size(); i++) {
            if(padresP2.contains(padresP1.get(i))) {
                ac = true;
            }
        }
        return ac;
    }

    public static Set<Persona> abuelos (SimpleDirectedGraph<Persona,Relacion> g,Persona p){

        Set <Persona> abu = new HashSet<>();
        List<Persona> padres = padres(g,p);
        for (Persona per :padres) {
            List<Persona> a = Graphs.predecessorListOf(g, per);
            abu.addAll(a);
        }
        return abu;
    }
//  APARTADO D

    public static void apartadoD (SimpleDirectedGraph<Persona,Relacion> g, String file) {

        Graph<Persona,Relacion> vista = SubGraphView.of(g,x->disPadres(g,x),null);

        GraphColors.toDot(g,"resultados/ejercicio1/" + file + "apartadoD" + ".gv",
                x->x.nombre() , x->" ",
                v->GraphColors.colorIf(Color.blue, Color.black, vista.vertexSet().contains
(v)),
                e->GraphColors.style(Style.solid));

        System.out.println(file + "apartadoD" + ".gv generado en " +
"resultados/ejercicio1");

    }

    public static Boolean disPadres (SimpleDirectedGraph<Persona,Relacion> g,Persona p) {

        Boolean ac = false;
        List<Persona> suc = Graphs.successorListOf(g, p);
```

```
        if (suc.size()>=2) {
            List<Persona> pad1 = Graphs.predecessorListOf(g, suc.get(0));
            for (Persona per:suc) {
                if(!Graphs.predecessorListOf(g, per).equals(pad1)) {
                    ac = true;
                }
            }
        }
        return ac;
    }

//  APARTADO E

    public static void ApartadoE (Graph<Persona,Relacion> g,String file) {

        GreedyVCImpl<Persona,Relacion> vCover = new GreedyVCImpl<>(g);
        Set<Persona> selected = vCover.getVertexCover();

        GraphColors.toDot(g,"resultados/ejercicio1/" + file + "apartadoE" + ".gv",
                x->x.nombre() , x->" ",
                v->GraphColors.colorIf(Color.blue, Color.black, selected.contains(v)),
                e->GraphColors.style(Style.solid));

        System.out.println(file + "apartadoE" + ".gv generado en " +
"resultados/ejercicio1");

    }


}
```

TEST EJERCICIO 1

```
public class TestEjercicio1 {

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

    public static void testEjercicio1(String file) {

        SimpleDirectedGraph<Persona,Relacion> dg = GraphsReader
                .newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleDirectedGraph);

        System.out.println("\nArchivo " + file + ".txt \n" + "Datos de entrada: " + dg);

//      Para mostrar el grafo original

        GraphColors.toDot(dg,"resultados/ejercicio1/" + file + "_Datos" + ".gv",
                x->x.nombre(), x->"" , //Atributos del vertice y de la arista
                v->GraphColors.color(Color.black), //Propiedades del vertice
                e->GraphColors.color(Color.black)); //Propiedades de la arista


//      APARTADO A

        Ejercicio1.apartadoA (dg, file);


//      APARTADO B

        if (file == "PI3E1A_DatosEntrada") {
             Persona p = Persona.of(13, "Maria", 2008, "Sevilla");

             GraphColors.toDot(dg,"resultados/ejercicio1/" + file + "apartadoB" + ".gv",
                        x->x.nombre() , x->" ",
                        v->colorea(p,v, Ejercicio1.apartadoB(dg, p)),
                        e->GraphColors.style(Style.solid));
                System.out.println(file + "apartadoB" + ".gv generado en " +
"resultados/ejercicio1");

        } else {

             Persona p = Persona.of(13, "Raquel", 1993, "Sevilla");

             GraphColors.toDot(dg,"resultados/ejercicio1/" + file + "apartadoB" + ".gv",
                        x->x.nombre() , x->" ",
                        v->colorea(p,v, Ejercicio1.apartadoB(dg, p)),
                        e->GraphColors.style(Style.solid));
                System.out.println(file + "apartadoB" + ".gv generado en " +
"resultados/ejercicio1");
        }

//      APARTADO C

        Persona p1 = null;
        Persona p2 = null;
```

```
        if (file == "PI3E1A_DatosEntrada") {
            p1 = Persona.of(16, "Rafael", 2020, "Malaga");
            p2 = Persona.of(14, "Sara", 2015, "Jaen");
        } else {
            p1 = Persona.of(14, "Julia", 1996, "Jaen");
            p2 = Persona.of(6, "Angela", 1997, "Sevilla");
        }
        Ejercicio1.apartadoC (dg,p1,p2);

//      APARTADO D

        Ejercicio1.apartadoD(dg, file);

//      APARTADO E

        Graph<Persona,Relacion> g = GraphsReader
                .newGraph("ficheros/" + file + ".txt",
                        Persona::ofFormat,
                        Relacion::ofFormat,
                        Graphs2::simpleGraph);

        Ejercicio1.ApartadoE(g, file);


    }

    private static Map<String,Attribute> colorea (Persona p, Persona v, Set<Persona> per){
        Map<String,Attribute> res = GraphColors.color(Color.black);
        if(p.equals(v)) {
            res = GraphColors.color(Color.red);
        }else if(per.contains(v)) {
            res = GraphColors.color(Color.blue);
        }
        return res;
    }



}
```

EJERCICIO 2

```
public class Ejercicio2 {

//  APARTADO A
    public static void ApartadoA (Graph<Ciudad,Transporte> g, String file) {

        ConnectivityInspector<Ciudad, Transporte> conIns= new ConnectivityInspector<>(g);
        List<Set<Ciudad>> conComps = conIns.connectedSets();

        GraphColors.toDot(g,
                        "resultados/ejercicio2/" + file + " ApartadoA.gv",
                        v -> v.ciudad(),
                        e -> "",
                        v -> GraphColors.color(asignaColor(v,conComps,conIns)),
                        e -> GraphColors.color(asignaColor(g.getEdgeSource(e), conComps,
conIns)));


        System.out.println(file + "_ApartadoA.gv generado en resultados/ejercicio2");

    }

    private static Color asignaColor(Ciudad v, List<Set<Ciudad>> ls,
ConnectivityInspector<Ciudad, Transporte> alg) {
        Color[] vc = Color.values(); //Una lista de colores
        Set<Ciudad> s = alg.connectedSetOf(v); //La componente conexa a la que pertenece el
vértice
        return vc[ls.indexOf(s)]; //Qué componente conexa es dentro de la lista
    }

//  APARTADO B

    public static void ApartadoB (Graph<Ciudad,Transporte> g, String file) {

        ConnectivityInspector<Ciudad, Transporte> conIns= new ConnectivityInspector<>(g);
        List<Set<Ciudad>> conComps = conIns.connectedSets();

        Set<Ciudad> c1 = conComps.get(0);
        Set<Ciudad> c2 = conComps.get(1);

        Graph<Ciudad,Transporte> g1 = grafoConexo(g,c1);
        Graph<Ciudad,Transporte> g2 = grafoConexo(g,c2);

        List<Ciudad> vertices1 = g1.vertexSet().stream().toList();
        List<Ciudad> vertices2 = g2.vertexSet().stream().toList();

        List<Transporte> aristas1 = g1.edgeSet().stream().toList();
        List<Transporte> aristas2 = g2.edgeSet().stream().toList();

        Double suma1 = vertices1.stream().mapToDouble(x->x.puntuacion()).sum();
        Double suma2 = vertices2.stream().mapToDouble(x->x.puntuacion()).sum();

        if(suma1>suma2) {

            GraphColors.toDot(g,
                    "resultados/ejercicio2/" + file + "_ApartadoB.gv",
                    v->v.ciudad(),
                    v->v.precio().toString(),
                    v -> GraphColors.colorIf(Color.blue,vertices1.contains(v)),
```

```
                    e -> GraphColors.colorIf(Color.blue,aristas1.contains(e)));

            System.out.println(file + "_ApartadoB.gv generado en resultados/ejercicio2");

        } else {

            GraphColors.toDot(g,
                    "resultados/ejercicio2/" + file + "_ApartadoB.gv",
                    v->v.ciudad(),
                    v->v.precio().toString(),
                    v -> GraphColors.colorIf(Color.blue,vertices2.contains(v)),
                    e -> GraphColors.colorIf(Color.blue,aristas2.contains(e)));

            System.out.println(file + " ApartadoB.gv generado en resultados/ejercicio2");

        }


    }

//  APARTADO C

    public static void ApartadoC (Graph<Ciudad,Transporte> g, String file) {

        ConnectivityInspector<Ciudad, Transporte> conIns= new ConnectivityInspector<>(g);
        List<Set<Ciudad>> conComps = conIns.connectedSets();

        Set<Ciudad> c1 = conComps.get(0);
        Set<Ciudad> c2 = conComps.get(1);

        Graph<Ciudad,Transporte> g1 = grafoConexo(g,c1);
        Graph<Ciudad,Transporte> g2 = grafoConexo(g,c2);

        List<Ciudad> lc1 = new HeldKarpTSP<Ciudad,Transporte>().getTour(g1).getVertexList
();
        List<Transporte> lt1= new HeldKarpTSP<Ciudad,Transporte>().getTour(g1).getEdgeList
();

        List<Ciudad> lc2 = new HeldKarpTSP<Ciudad,Transporte>().getTour(g2).getVertexList
();
        List<Transporte> lt2= new HeldKarpTSP<Ciudad,Transporte>().getTour(g2).getEdgeList
();

        lc1.addAll(lc2);
        lt1.addAll(lt2);

        GraphColors.toDot(g,
                "resultados/ejercicio2/" + file + "_ApartadoC.gv",
                v->v.ciudad(),
                v->v.precio().toString(),
                v -> GraphColors.colorIf(Color.blue,lc1.contains(v)),
                e -> GraphColors.colorIf(Color.blue,lt1.contains(e)));

        System.out.println(file + "_ApartadoC.gv generado en resultados/ejercicio2");

    }

    public static Graph<Ciudad,Transporte> grafoConexo (Graph<Ciudad,Transporte> g,
Set<Ciudad> g1){
        return SubGraphView.of(g,x->g1.contains(x),null);
    }
```

```java
//  APARTADO D

    public static void ApartadoD (Graph<Ciudad,Transporte> g, String file) {

        ConnectivityInspector<Ciudad, Transporte> conIns= new ConnectivityInspector<>(g);
        List<Set<Ciudad>> conComps = conIns.connectedSets();

        Set<Ciudad> c1 = conComps.get(0);
        Set<Ciudad> c2 = conComps.get(1);

        Graph<Ciudad,Transporte> g1 = grafoConexo(g,c1);
        Graph<Ciudad,Transporte> g2 = grafoConexo(g,c2);

        List<Ciudad> vertices1 = g1.vertexSet().stream().toList();
        List<Ciudad> vertices2 = g2.vertexSet().stream().toList();

        Set<GraphPath<Ciudad,Transporte>> sg1 = floyd(g1,vertices1);
        Set<GraphPath<Ciudad,Transporte>> sg2 = floyd(g2,vertices2);

        GraphPath<Ciudad,Transporte> q1 = minimo(sg1);
        GraphPath<Ciudad,Transporte> q2 = minimo(sg2);

        List<Ciudad> lc1 = q1.getVertexList();
        List<Transporte> lt1 = q1.getEdgeList();

        List<Ciudad> lc2 = q2.getVertexList();
        List<Transporte> lt2 = q2.getEdgeList();

        lc1.addAll(lc2);
        lt1.addAll(lt2);

        GraphColors.toDot(g,
                "resultados/ejercicio2/" + file + "_ApartadoD.gv",
                c -> c.ciudad().toString(),
                e -> e.duracion().toString(),
                v -> GraphColors.colorIf(Color.blue, lc1.contains(v)),
                e -> GraphColors.colorIf(Color.blue, lt1.contains(e)));

        System.out.println(file + "_ApartadoD.gv generado en resultados/ejercicio2");

    }

    public static GraphPath<Ciudad,Transporte> minimo (Set<GraphPath<Ciudad,Transporte>>
sg){
        return sg.stream().filter(x->x.getLength()>=2).min(Comparator.comparing(x-
>x.getWeight())).get();
    }

    public static Set<GraphPath<Ciudad,Transporte>> floyd (Graph<Ciudad,Transporte> g,
List<Ciudad> vertices){

        FloydWarshallShortestPaths<Ciudad, Transporte> a  = new
FloydWarshallShortestPaths<>(g);
        Set<GraphPath<Ciudad,Transporte>> s = new HashSet<>();

        for(int i = 0;i<vertices.size();i++) {
            for(int j = i+1;j<vertices.size();j++) {
                s.add(a.getPath(vertices.get(i), vertices.get(j)));
            }
        }

        return s;
```

TEST EJERCICIO 2

```
public class TestEjercicio2 {

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

    }

    private static void testEjercicio2(String file) {

        Graph<Ciudad, Transporte> g = GraphsReader
                .newGraph("ficheros/" + file + ".txt",
                        Ciudad::ofFormat,
                        Transporte::ofFormat,
                        Graphs2::simpleWeightedGraph);

        System.out.println("\nArchivo " + file + ".txt \n" + "Datos de entrada: " + g);

//      Para mostrar el grafo original
        GraphColors.toDot(g,"resultados/ejercicio2/" + file + "_Datos.gv",
                x->x.ciudad(), x->x.duracion().toString()+" min", //Atributos del vertice y
de la arista
                v->GraphColors.color(Color.black), //Propiedades del vertice
                e->GraphColors.color(Color.black)); //Propiedades de la arista

        Ejercicio2.ApartadoA(g, file);

        Ejercicio2.ApartadoB(g, file);

        Ejercicio2.ApartadoC(g, file);

        Ejercicio2.ApartadoD(grafoApartD(file), file);

    }

    private static Graph<Ciudad,Transporte> grafoApartD (String file){
        return GraphsReader
                .newGraph("ficheros/" + file + ".txt",
                        Ciudad::ofFormat,
                        Transporte::ofFormat,
                        Graphs2::simpleWeightedGraph,
                        Transporte::duracion);
    }

}
```

EJERCICIO 3

```
public class Ejercicio3 {

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

        GreedyColoring<String,DefaultEdge> gColoring = new GreedyColoring<>(g);
        Coloring<String> coloring = gColoring.getColoring();

        List<Set<String>> composicion = coloring.getColorClasses();
        System.out.println(String.format("Nº de franjas necesarias: ",composicion.size()));

        for(int i=0;i<composicion.size();i++) {
            System.out.println(String.format("Franja nº %d: %s", i, composicion.get(i)));

        }

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

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

        System.out.println(file + "_ApartadoAyB.gv generado en resultados/ejercicio3");

    }


}
```

TEST EJERCICIO 3

```
public class TestEjercicio3 {

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

    }

    private static void testEjercicio3(String file) {

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

        Files2.streamFromFile("ficheros/"+file+".txt").forEach(linea->{
            String[] v = linea.split(":")[1].split(",");
            for (int i=0;i<v.length;i++) {
                String v0 = v[i].trim();
                g.addVertex(v0);
                for (int j=0;j<v.length;j++) {
                    String v1 = v[j].trim();
                    g.addVertex(v1);
                    if(i!=j) {
                        g.addEdge(v0,v1);
                    }
                }
            }
        });

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

        Ejercicio3.todosLosApartados(g, file);
    }

}
```

DATOS

CIUDAD

```
public record Ciudad(String ciudad, Double puntuacion) {

    public static Ciudad ofFormat(String[] v) {
        String ciudad=v[0];
        Double puntuacion = Double.valueOf(v[1].split("p")[0]);
        return new Ciudad(ciudad,puntuacion);
    }

}
```

TRANSPORTE

```
public record Transporte(Double precio, Double duracion) {

    public static Transporte ofFormat(String[] v) {
        Double precio = Double.valueOf(v[2].split("euros")[0]);
        Double duracion = Double.valueOf(v[3].split("min")[0]);
        return new Transporte(precio,duracion);
    }
}
```

PERSONA

```
public record Persona(Integer id, String nombre, Integer anio, String ciudad) {
    public static Persona ofFormat(String[] v) {
        Integer id = Integer.valueOf(v[0]);
        String nombre = v[1];
        Integer anio = Integer.valueOf(v[2]);
        String ciudad = v[3];
        return Persona.of(id,nombre,anio,ciudad);
    }

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

}
```

RELACION

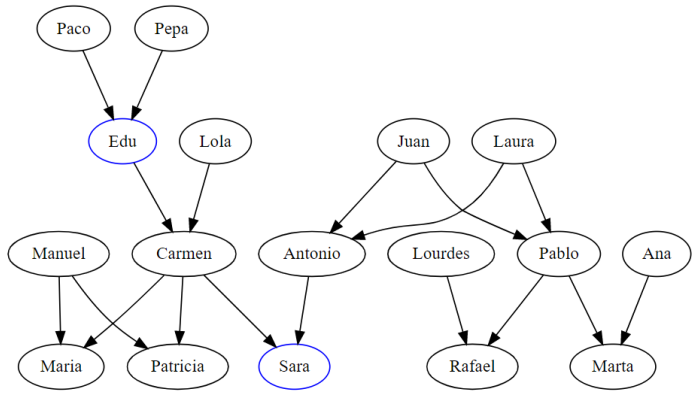
```
public record Relacion(Integer id,Integer r1, Integer r2) {

    private static int num = 0;

    public static Relacion ofFormat(String[] v) {
        Integer re1 = Integer.valueOf(v[0]);
        Integer re2 = Integer.valueOf(v[1]);
        Integer id = num;
        num++;
        return new Relacion(id,re1,re2);
    }
}
```

SON

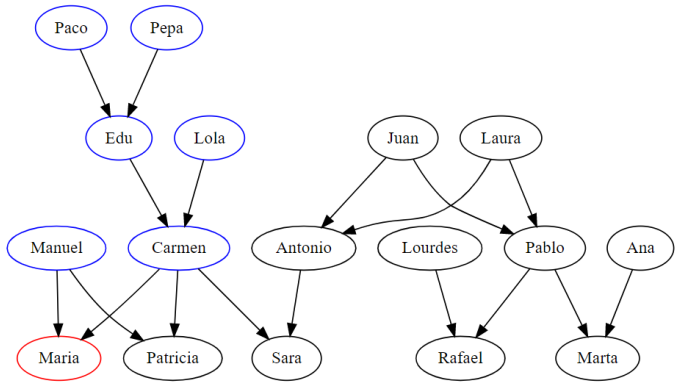
```
public enum son {
    HERMANOS,PRIMOS,OTROS
}
```

RESULTADOS EJERCICIO 1
PI3E1A

- **APARTADO A**



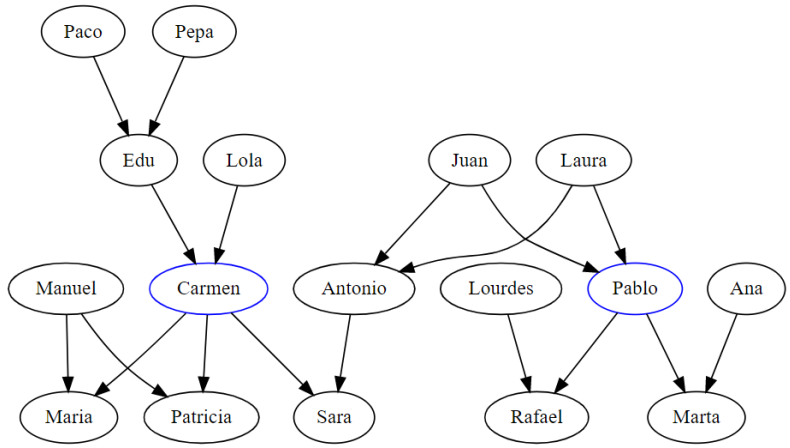
- **APARTADO B**



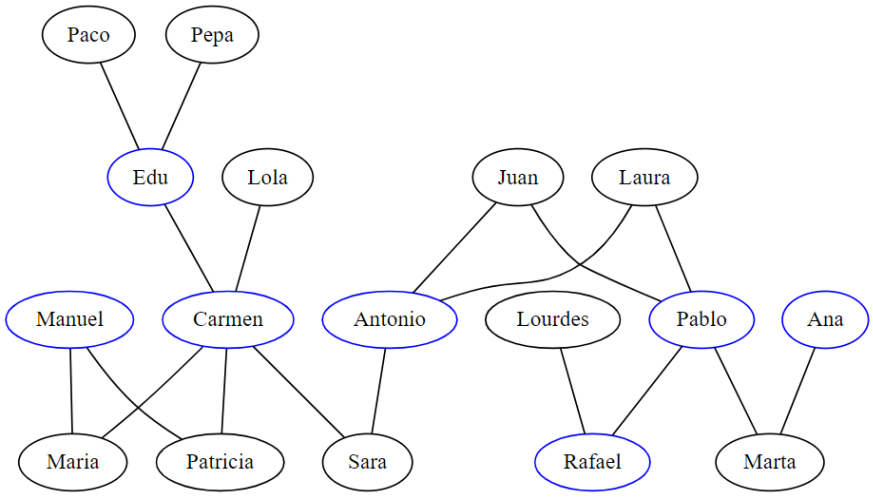
- **APARTADO C**

```
Rafael y Sara son PRIMOS
```

- **APARTADO D**

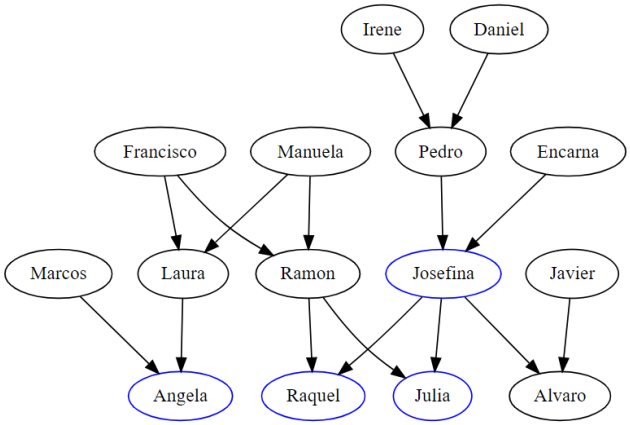


- **APARTADO E**

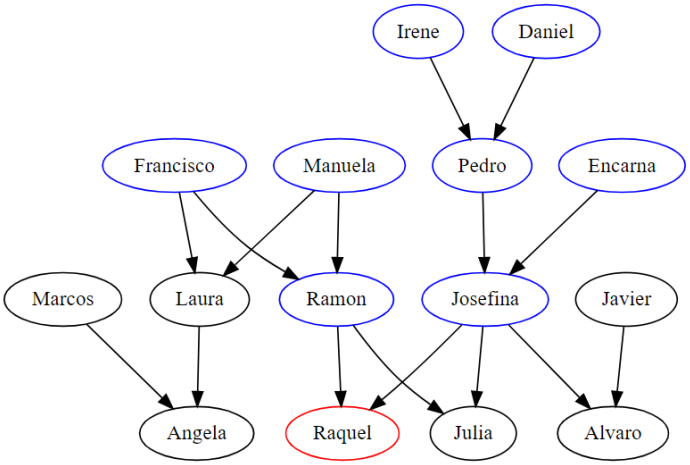

Paco
Pepa
Edu
Lola
Juan
Laura
Manuel
Carmen
Antonio
Lourdes
Pablo
Ana
Maria
Patricia
Sara
Rafael
Marta

PI3E1B

- **APARTADO A**



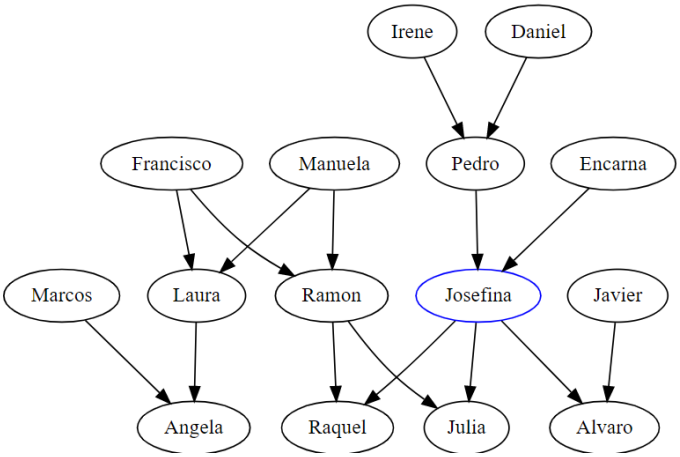
- **APARTADO B**



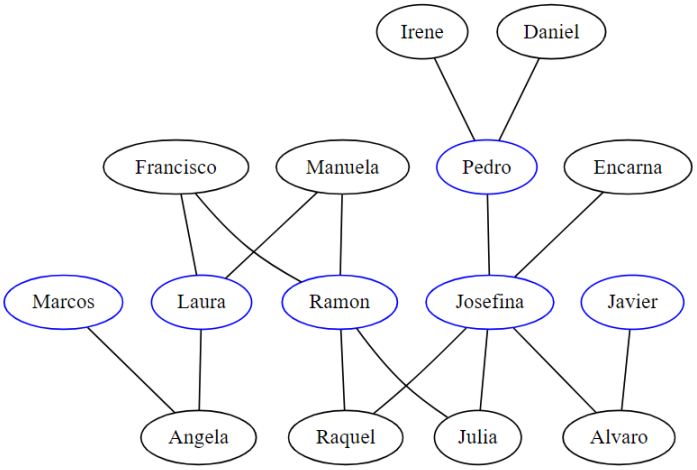
- **APARTADO C**

```
Julia y Angela son PRIMOS
```

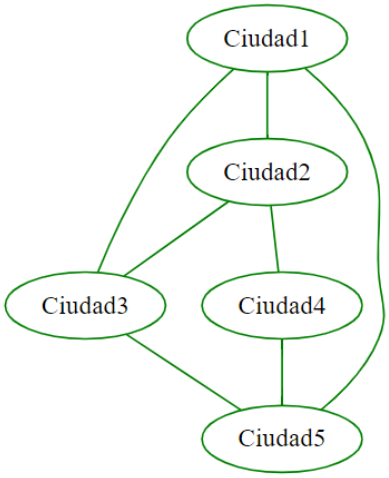
- **APARTADO D**

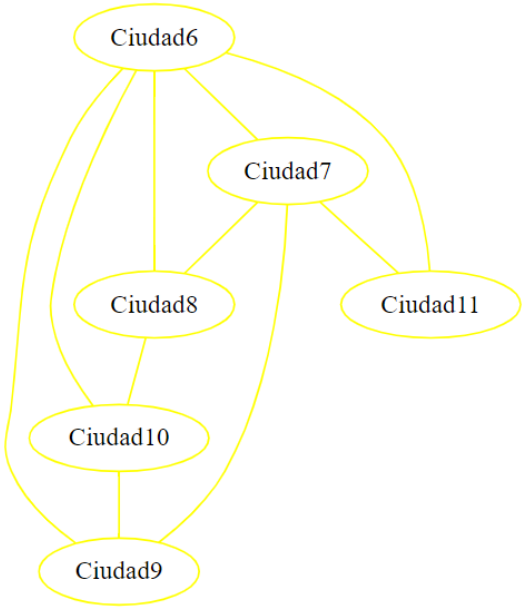


- **APARTADO E**

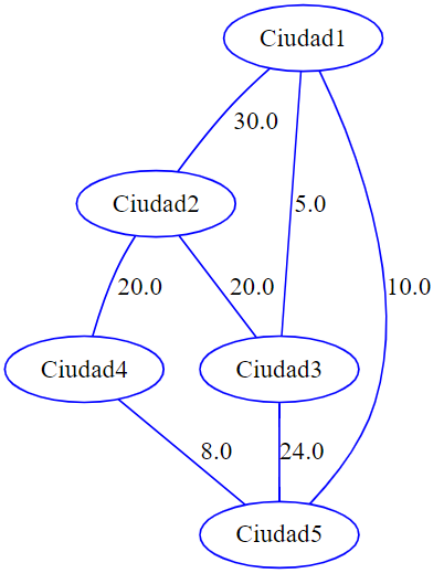

Irene
Daniel
Francisco
Manuela
Pedro
Encarna
Marcos
Laura
Ramon
Josefina
Javier
Angela
Raquel
Julia
Alvaro

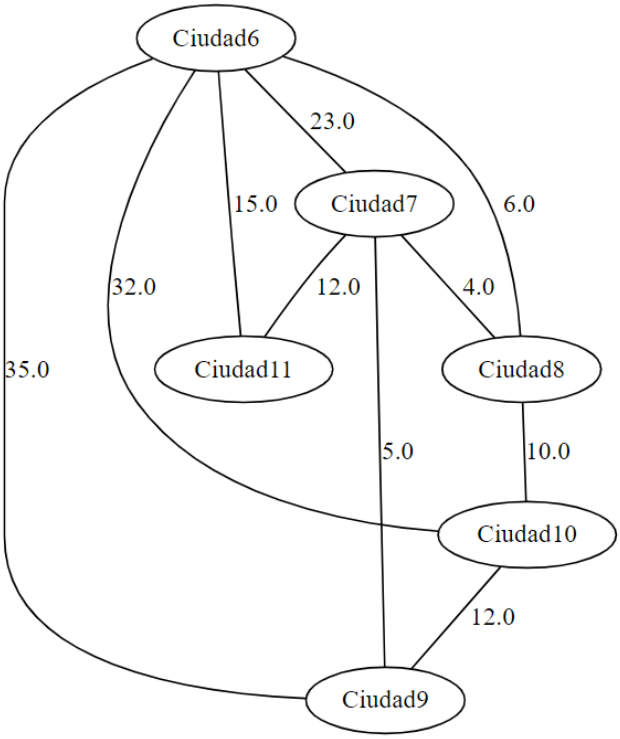
RESULTADOS EJERCICIO 2
APARTADO A



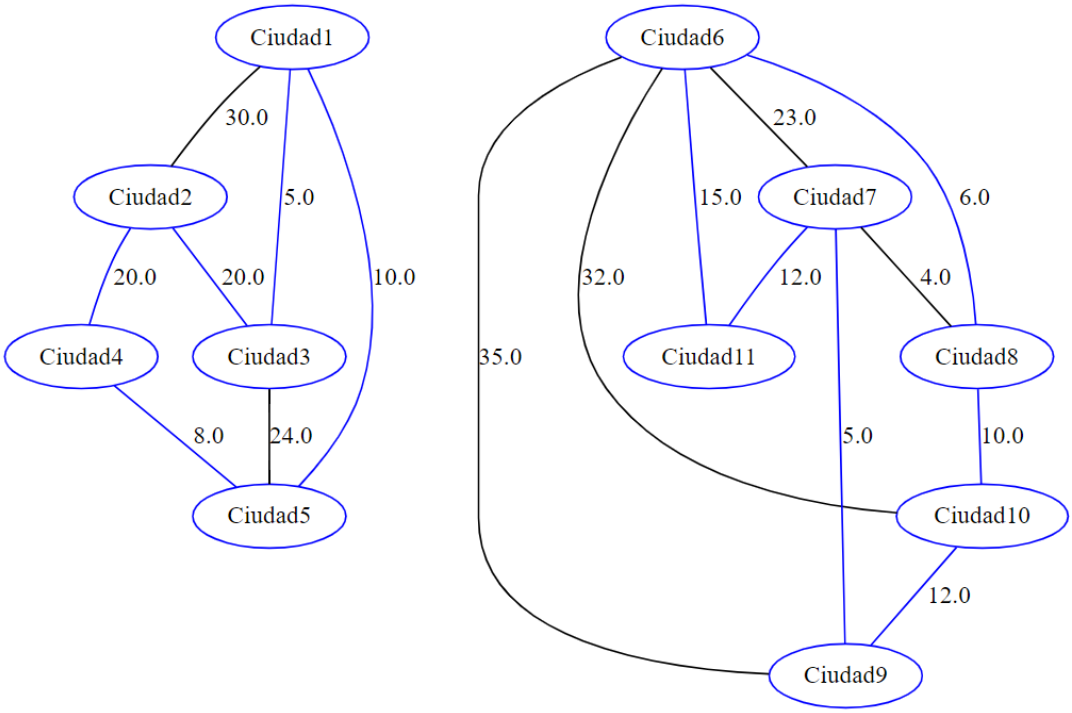


APARTADO B

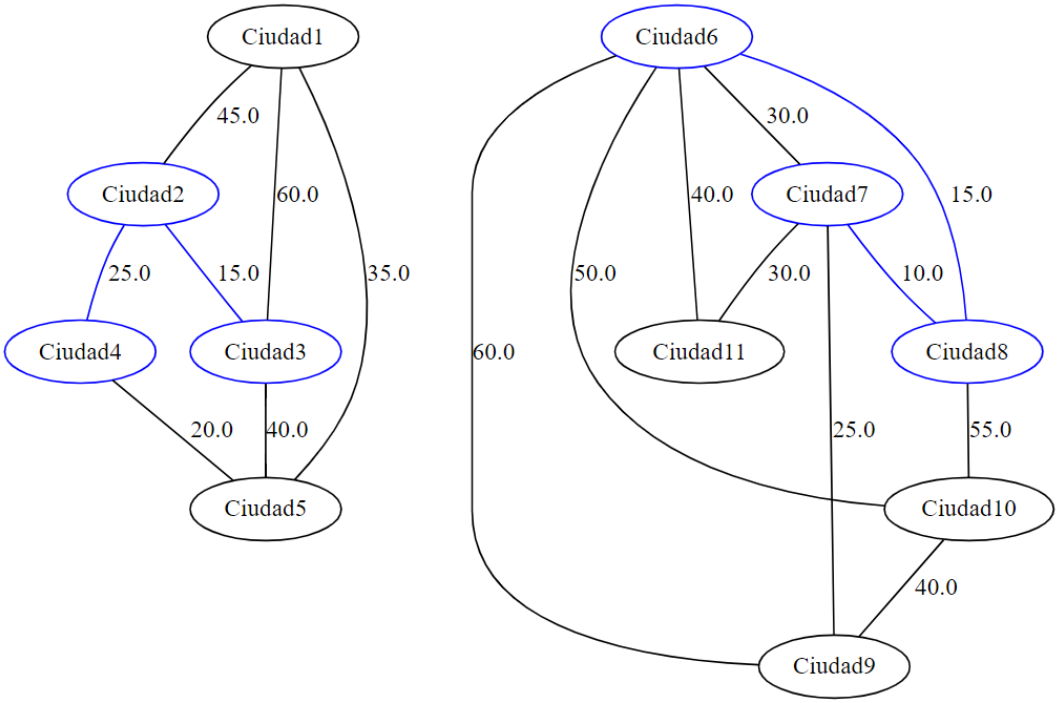




APARTADO C



APARTADO D

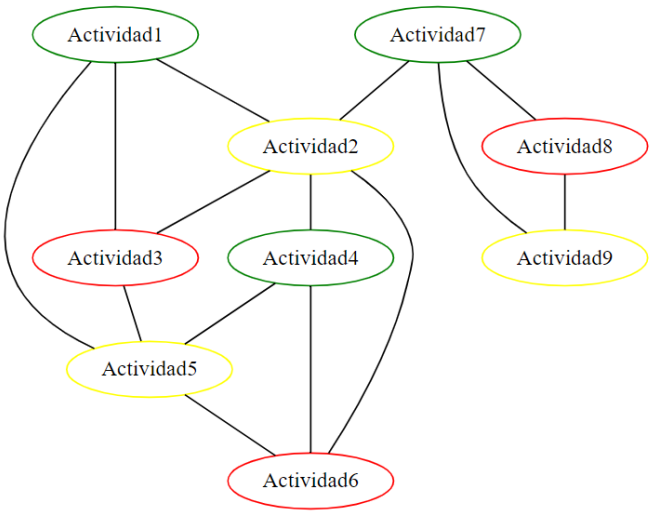


RESULTADOS EJERCICIO 3

PI3E3A

- **APARTADO A Y B**

N◆ de franjas necesarias:
Franja n◆ 0: [Actividad1, Actividad4, Actividad7]
Franja n◆ 1: [Actividad2, Actividad9, Actividad5]
Franja n◆ 2: [Actividad3, Actividad6, Actividad8]



PI3E3B

- **APARTADO A Y B**

N◆ de franjas necesarias:
Franja n◆ 0: [Actividad1, Actividad11, Actividad8]
Franja n◆ 1: [Actividad2, Actividad4, Actividad3, Actividad9, Actividad12, Actividad7]
Franja n◆ 2: [Actividad10, Actividad5]
Franja n◆ 3: [Actividad6]