Memoria Práctica 3 juaorecar

EJERCICIO 1

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.function.Predicate;
import org.jgrapht.Graph;
import org.jgrapht.alg.shortestpath.DijkstraShortestPath;
import org.jgrapht.alg.vertexcover.GreedyVCImpl;
import org.jgrapht.graph.SimpleDirectedGraph;
import org.jgrapht.traverse.DepthFirstIterator;
import datos.Familia;
import datos.Persona;
import datos.Relacion;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Color;
import us.lsi.colors.GraphColors.Style;
import us.lsi.graphs.Graphs2;
import us.lsi.graphs.views.SubGraphView;

public class Ejercicio1 {

	public static void apartadoA(String file, Graph<Persona, Familia> g,
			Predicate<Persona> pv, String nombreVista) {

		Graph<Persona, Familia> vista = SubGraphView.of(g, pv, null);
		Set<Persona> set = vista.vertexSet();
		List<Persona> personas = set.stream().toList();
		List<String> ls = new ArrayList<>();
		for(int i=0; i<personas.size(); i++) {
			ls.add(personas.get(i).nombre());
		}
		System.out.println("Personas cuyos padres aparecen en el grafo y cumplen los
requisitos: " + ls);
		String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
		GraphColors.toDot(vista, fileRes,
				v->v.nombre(),
				e->"",
				v->GraphColors.colorIf(Color.red, vista.containsVertex(v)),
				e->GraphColors.colorIf(Color.black, vista.containsEdge(e)));
		System.out.println("Se ha generado" + fileRes);

	}

	public static void apartadoB(String file, SimpleDirectedGraph<Persona, Familia> g,
String p, String nombreVista) {
		Graph<Persona, Familia> grafo = Graphs2.inversedDirectedGraph(g);
```

```
		Persona persona = g.vertexSet().stream().filter(v->v.nombre().equals(p)).findFirst().get(); //Encontrar a dicha persona en el grafo
		List<Persona> ancestros = new ArrayList<>();
		DepthFirstIterator<Persona, Familia> dfi = new DepthFirstIterator<>(grafo, persona);
		dfi.forEachRemaining(v->ancestros.add(v));
		Map<Persona, Color> m = new HashMap<>();
		for(Persona person: g.vertexSet()) {
			if(person.nombre().equals(p)) {
				m.put(person, Color.red);
			}else if(ancestros.contains(person)) {
				m.put(person, Color.blue);
			}else {
				m.put(person, Color.black);
			}
		}
		List<String> ls = new ArrayList<>();
		for(int i=0; i<ancestros.size(); i++) {
			ls.add(ancestros.get(i).nombre());
		}
		System.out.println("Ancestros de " + p + ": "+ ls);
		String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
		GraphColors.toDot(g, fileRes,
				v->v.nombre(),
				e->"",
				v->GraphColors.color(m.get(v)),
				e->GraphColors.color(Color.black));
		System.out.println("Se ha generado" + fileRes);
	}


	public static void apartadoC(String file, Graph<Persona, Familia> g, String p1, String p2) {
		Persona persona1 = g.vertexSet().stream().filter(v->v.nombre().equals(p1)).findFirst().get();
		Persona persona2 = g.vertexSet().stream().filter(v->v.nombre().equals(p2)).findFirst().get();
		DijkstraShortestPath<Persona, Familia> circuito = new DijkstraShortestPath<>(g);
		Integer longitud = circuito.getPath(persona1, persona2).getLength();
		if(longitud==2) {
			System.out.println(persona1.nombre() + " y " + persona2.nombre() + " son " + Relacion.HERMANOS);
		}else if(longitud == 4) {
			System.out.println(persona1.nombre() + " y " + persona2.nombre() + " son " + Relacion.PRIMOS);
		}else {
			System.out.println(persona1.nombre() + " y " + persona2.nombre() + " son " + Relacion.OTROS);
		}
	}
```

```
	public static void apartadoD(String file, Graph<Persona, Familia> g,
			Predicate<Persona> pv, String nombreVista) {
		Graph<Persona, Familia> vista = SubGraphView.of(g, pv, null);
		Set<Persona> set = vista.vertexSet();
		List<Persona> personas = set.stream().toList();
		List<String> ls = new ArrayList<>();
		for(int i=0; i<personas.size(); i++) {
			ls.add(personas.get(i).nombre());
		}
		System.out.println("Personas que tienen hijos/as con distintas personas: " +
ls);
		String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
		GraphColors.toDot(vista, fileRes,
				v->v.nombre(),
				e->"",
				v->GraphColors.colorIf(Color.red, vista.containsVertex(v)),
				e->GraphColors.colorIf(Color.black, vista.containsEdge(e)));
		System.out.println("Se ha generado" + fileRes);
	}


	public static void apartadoE(String file, Graph<Persona, Familia> g, String nombreVista)
{
		GreedyVCImpl<Persona, Familia> vCover= new GreedyVCImpl<>(g);
		Set<Persona> personas = vCover.getVertexCover();
		String fileRes = "resultados/ejercicio1/" + file + nombreVista + ".gv";
		GraphColors.toDot(g, fileRes,
				v->v.nombre(),
				e->"",
				v->GraphColors.colorIf(Color.red, personas.contains(v)),
				e->GraphColors.style(Style.solid));
		System.out.println("Se ha generado" + fileRes);
	}
}
```

EJERCICIO 2

```
import java.util.HashSet;
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 org.jgrapht.graph.SimpleWeightedGraph;

import datos.Ciudad;
import datos.Trayecto;
import us.lsi.colors.GraphColors;
```

```
import us.lsi.colors.GraphColors.Color;
import us.lsi.colors.GraphColors.Style;
import us.lsi.common.Trio;
import us.lsi.graphs.views.SubGraphView;

public class Ejercicio2 {

	public static List<Set<Ciudad>> componentesConexas(Graph<Ciudad, Trayecto> gf){
		var cc = new ConnectivityInspector<>(gf);
		List<Set<Ciudad>> componentes = cc.connectedSets();
		return componentes;
	}

	public static Boolean esConexo(Graph<Ciudad, Trayecto> gf) {
		var ec = new ConnectivityInspector<>(gf);
		Boolean conexo = ec.isConnected();
		return conexo;
	}

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

	public static void apartadoA(Graph<Ciudad, Trayecto> gf, String file, String
nombreVista) {
		var esConexo = esConexo(gf);
		var cc = new ConnectivityInspector<>(gf);
		if(esConexo) {
			System.out.println("Solo hay un grupo de ciudades");
		}else {
			System.out.println("Numero de grupos de ciudades: " +
componentesConexas(gf).size());
			for(int i=0; i<componentesConexas(gf).size(); i++) {
				System.out.println("\n Grupo numero " + (i+1) + ":" +
componentesConexas(gf).get(i));
			}
		}
		String fileRes = "resultados/ejercicio2/" + file + nombreVista + ".gv";
		GraphColors.toDot(gf, fileRes,c->"", v->"",
				v -> GraphColors.color(asignaColor(v,
componentesConexas(gf), cc)),
				e -> GraphColors.color(asignaColor(gf.getEdgeSource(e),
componentesConexas(gf),cc)));

	}


	public static Set<Ciudad> setApartadoB(Graph<Ciudad, Trayecto> gf) {
		List<Set<Ciudad>> lsComponentes = componentesConexas(gf);
```

```
		Integer maximo = 0;
		Set<Ciudad> res = new HashSet<>();
		for(int i=0; i<lsComponentes.size(); i++) {
			Integer suma = lsComponentes.get(i).stream()
					.mapToInt(Ciudad::puntuacion).sum();
			if(suma>maximo) {
				maximo = suma;
				res = lsComponentes.get(i);
			}
		}

		return res;
	}


	public static void apartadoB(Graph<Ciudad, Trayecto> gf, String file, String
nombreVista) {
		Set<Ciudad> set = setApartadoB(gf);
		Graph<Ciudad, Trayecto> g = SubGraphView.of(gf, set);
		String fileRes = "resultados/ejercicio2/" + file + nombreVista + ".gv";
		GraphColors.toDot(g, fileRes,
			v->v.nombre(),
			e->"",
			v->GraphColors.colorIf(Color.blue, g.containsVertex(v)),
			e->GraphColors.colorIf(Color.blue, g.containsEdge(e)));
		System.out.println("Se ha generado" + fileRes);
	}



	public static void apartadoC(SimpleWeightedGraph<Ciudad, Trayecto> gf, String file,
String nombreVista) {

			ConnectivityInspector<Ciudad, Trayecto> cc = new
ConnectivityInspector<>(gf);
			HeldKarpTSP<Ciudad, Trayecto> camino = new HeldKarpTSP<>();
			Trio<List<Ciudad>, List<Trayecto>, Double> trio = Trio.of(null, null,
Double.MAX_VALUE);

			for(Set<Ciudad> grupo:cc.connectedSets()) {
				Graph<Ciudad, Trayecto> vista = SubGraphView.of(gf, grupo);
				List<Trayecto> lsTrayectos =
camino.getTour(vista).getEdgeList();
				List<Ciudad> lsCiudades =
camino.getTour(vista).getVertexList();
				Double totalPrecio = lsTrayectos.stream().mapToDouble(x-
>x.precio()).sum();
				if(totalPrecio<trio.third()) {
					trio = Trio.of(lsCiudades, lsTrayectos, totalPrecio);
				}
			}
```

```
                        List<Ciudad> mejoresCiudades = trio.first();
                        List<Trayecto> mejoresTrayectos = trio.second();

                        String fileRes = "resultados/ejercicio2/" + file + nombreVista + ".gv";
                        GraphColors.toDot(gf, fileRes,
                                        v->v.nombre(),
                                        e->"",
                                        v->GraphColors.colorIf(Color.blue,
mejoresCiudades.contains(v)),
                                        e->GraphColors.colorIf(Color.blue,
mejoresTrayectos.contains(e)));
                        System.out.println("Se ha generado" + fileRes);
        }


        public static void apartadoD(Graph<Ciudad, Trayecto> gf, String file, String
nombreVista) {
                List<Set<Ciudad>> lsComponentes = componentesConexas(gf);
                gf.edgeSet().forEach(e-> gf.setEdgeWeight(e, e.duracion()));
                for(int i=0; i<lsComponentes.size(); i++) {
                        Graph<Ciudad, Trayecto> componente =
SubGraphView.of(gf,lsComponentes.get(i));
                        List<Ciudad> ls = componente.vertexSet().stream().toList();
                        Double minimo = Double.POSITIVE_INFINITY;
                        GraphPath<Ciudad, Trayecto> res = null;
                        for(int j=0; j<ls.size(); j++) {
                                for(int k=j+1; k<ls.size(); k++) {
                                        Ciudad vertice1 = ls.get(j);
                                        Ciudad vertice2 = ls.get(k);
                                        DijkstraShortestPath<Ciudad, Trayecto> dj = new
DijkstraShortestPath<>(componente);
                                        GraphPath<Ciudad, Trayecto> camino =
dj.getPath(vertice1, vertice2);
                                        if(camino.getLength()>= 2) {
                                                if(camino.getWeight()<minimo) {
                                                        minimo = camino.getWeight();
                                                        res = camino;
                                                }
                                }
                        }
                        }
                        System.out.println("Para el grupo: " + componente.vertexSet() + ", las
ciudades no conectadas directamente entre las que se puede viajar en menor tiempo son:\r\n"
                                        + "Origen: " + res.getStartVertex() + " y Destino: " +
res.getEndVertex());

                        List<Ciudad> lsVertices = res.getVertexList();
                        List<Trayecto> lsAristas = res.getEdgeList();
                        String fileRes = "resultados/ejercicio2/" + file + nombreVista + ".gv";
                        GraphColors.toDot(gf, fileRes,
                                        v->v.nombre(),
                                        e->e.precio().toString() + " euros",
```

```
                                        v->GraphColors.styleIf(Style.bold, lsVertices.contains(v)),
                                                e->GraphColors.styleIf(Style.bold,
lsAristas.contains(e)));
                                System.out.println("Se ha generado" + fileRes);
                        }


                }

}
```

EJERCICIO 3

```
import java.util.Map;
import org.jgrapht.Graph;
import org.jgrapht.alg.color.GreedyColoring;
import org.jgrapht.alg.interfaces.VertexColoringAlgorithm.Coloring;
import org.jgrapht.graph.DefaultEdge;
import us.lsi.colors.GraphColors;
import us.lsi.colors.GraphColors.Style;

public class Ejercicio3 {

        public static void apartadoA(Graph<String, DefaultEdge> gf, String file) {
                var c = new GreedyColoring<>(gf);
                Coloring<String> res = c.getColoring();
                System.out.println("Franjas horarias necesarias: " + res.getNumberColors());

                System.out.println("Composicion de las actividades");
                var actividades = res.getColorClasses();
                for(int i=0; i<actividades.size(); i++) {
                        System.out.println("Franja horaria numero "+(i+1)+": " +
actividades.get(i));
                }
        }
        public static void apartadoB(Graph<String, DefaultEdge> gf, String file, String
nombreVista){

                var c = new GreedyColoring<>(gf);
                Coloring<String> res = c.getColoring();
                Map<String, Integer> map = res.getColors();
                String fileRes = "resultados/ejercicio3/" + file + nombreVista + ".gv";
                GraphColors.toDot(gf, fileRes,
                                v->v.toString(),
                                e->"",
                                v -> GraphColors.color(map.get(v)),
                                e -> GraphColors.style(Style.solid));

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

TEST EJERCICIO 1

```
import java.util.HashSet;
import java.util.Set;
import java.util.function.Predicate;
import org.jgrapht.Graph;
import org.jgrapht.graph.SimpleDirectedGraph;
import datos.Familia;
import datos.Persona;
import ejercicios.Ejercicio1;
import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class TestEjercicio1 {

	public static void main(String[] args) {
		testApartadoA("1A");
		testApartadoA("1B");
		testApartadoB("1A", "Maria");
		testApartadoB("1B", "Raquel");
		testApartadoC("1A", "Rafael", "Sara");
		testApartadoC("1A", "Maria", "Patricia");
		testApartadoC("1A", "Carmen", "Rafael");
		testApartadoD("1A");
		testApartadoD("1B");
		testApartadoE("1A");
		testApartadoE("1B");
	}


	public static SimpleDirectedGraph<Persona, Familia> grafoDirigido(String file) {
		SimpleDirectedGraph<Persona, Familia> g =
GraphsReader.newGraph("ficheros/PI3E" + file + "_DatosEntrada.txt",
					Persona::ofFormat,
					Familia::ofFormat,
					Graphs2::simpleDirectedGraph);
		return g;
	}

	public static Graph<Persona, Familia> grafoSimple(String file) {
		Graph<Persona, Familia> g = GraphsReader.newGraph("ficheros/PI3E" + file +
"_DatosEntrada.txt",
					Persona::ofFormat,
					Familia::ofFormat,
					Graphs2::simpleGraph);
		return g;
	}

	public static void testApartadoA(String file) {
		SimpleDirectedGraph<Persona, Familia> g = grafoDirigido(file);
		Predicate<Persona> pv = v -> mismoAnyoYCiudad(g, g.incomingEdgesOf(v));
```

```
		Ejercicio1.apartadoA(file, g, pv, "Apartado A");
	}

	public static Boolean mismoAnyoYCiudad(Graph<Persona, Familia> g, Set<Familia>
familia) {
		Set<String> ciudad = new HashSet<>();
		Set<Integer> anyo = new HashSet<>();
		if(familia.size()==2) { //si tiene padres
			for(Familia f: familia) {
				ciudad.add(g.getEdgeSource(f).ciudad_nacimiento()); //Ciudad
de nacimiento de los padres
				anyo.add(g.getEdgeSource(f).anyo_nacimiento()); //Años de
nacimiento de los padres
			}
			if(anyo.size() == 1 && ciudad.size() == 1) { //Al ser un set y no poder
haber elementos repetidos, comprobamos si hay un solo elemento en cada set
				return true;
			}else {
				return false;
			}
		}else {
			return false;
		}
	}

	public static void testApartadoB(String file, String nombre) {
		SimpleDirectedGraph<Persona, Familia> g= grafoDirigido(file);
		Ejercicio1.apartadoB(file, g, nombre, "Apartado B");
	}

	public static void testApartadoC(String file, String p1, String p2) {
		Graph<Persona,Familia> g = grafoSimple(file);
		Ejercicio1.apartadoC(file, g, p1, p2);
	}


	public static void testApartadoD(String file) {
		SimpleDirectedGraph<Persona, Familia> g = grafoDirigido(file);
		Predicate<Persona> pv = v->HijoConPadresDiferentes(g, v);
		Ejercicio1.apartadoD(file, g, pv, "Apartado D");
	}

	public static Boolean HijoConPadresDiferentes(SimpleDirectedGraph<Persona,
Familia> gf, Persona p) {
		Set<Persona> conj = new HashSet<>();
		if(gf.outDegreeOf(p)>0) {
			for(Familia target: gf.outgoingEdgesOf(p)) {
				Persona hijo = gf.getEdgeTarget(target);
				for(Familia padre:gf.incomingEdgesOf(hijo)) {
					conj.add(gf.getEdgeSource(padre));
				}
			}
```

```
		}
		return conj.size()>2;
	}

	public static void testApartadoE(String file) {
		Graph<Persona, Familia> g = grafoSimple(file);
		Ejercicio1.apartadoE(file, g, "Apartado E");
	}

}
```

TEST EJERCICIO 2

```
import org.jgrapht.Graph;
import org.jgrapht.graph.SimpleWeightedGraph;
import datos.Ciudad;
import datos.Trayecto;
import ejercicios.Ejercicio2;
import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class TestEjercicio2 {

	public static void main(String[] args) {
		testApartadoA("2");
		testApartadoB("2");
		testApartadoC("2");
		testApartadoD("2");
	}
	public static Graph<Ciudad, Trayecto> grafoSimple(String file) {
		Graph<Ciudad, Trayecto> g = GraphsReader.newGraph("ficheros/PI3E" + file +
"_DatosEntrada.txt",
				Ciudad::ofFormat,
				Trayecto::ofFormat,
				Graphs2::simpleGraph);
		return g;
	}
	public static SimpleWeightedGraph<Ciudad, Trayecto> grafoSimplePeso(String file) {
		SimpleWeightedGraph<Ciudad, Trayecto> g =
GraphsReader.newGraph("ficheros/PI3E" + file + "_DatosEntrada.txt",
				Ciudad::ofFormat,
				Trayecto::ofFormat,
				Graphs2::simpleWeightedGraph);
		return g;
	}
	public static void testApartadoA(String file) {
		Graph<Ciudad,Trayecto> g = grafoSimple(file);
		Ejercicio2.apartadoA(g, file, "Apartado A");
	}

	public static void testApartadoB(String file) {
		Graph<Ciudad, Trayecto> g = grafoSimple(file);
```

```
		Ejercicio2.apartadoB(g, file, "Apartado B");
	}

	public static void testApartadoC(String file) {
		SimpleWeightedGraph<Ciudad, Trayecto> g = grafoSimplePeso(file);
		Ejercicio2.apartadoC(g, file, "Apartado C");
	}

	public static void testApartadoD(String file) {
		SimpleWeightedGraph<Ciudad, Trayecto> g = grafoSimplePeso(file);
		Ejercicio2.apartadoD(g, file, "Apartado D");
	}
}
```

TEST EJERCICIO 3

```
import org.jgrapht.Graph;
import org.jgrapht.graph.DefaultEdge;
import ejercicios.Ejercicio3;
import us.lsi.common.Files2;
import us.lsi.graphs.Graphs2;

public class TestEjercicio3 {

	public static void main(String[] args) {
		testApartadoA("3A");
		testsApartadoB("3A");
		testsApartadoB("3B");
	}

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

		Files2.streamFromFile("ficheros/PI3E" + file +
"_DatosEntrada.txt").forEach(linea -> {
			String[] lineaSinPuntos = linea.split(":");
			String[] lineaSinComas = lineaSinPuntos[1].replace(" ", "").split(",");
			for(int i=0; i<lineaSinComas.length; i++) {
				for(int j = i +1; j<lineaSinComas.length; j++) {
					g.addVertex(lineaSinComas[i]);
					g.addVertex(lineaSinComas[j]);
					g.addEdge(lineaSinComas[i], lineaSinComas[j]);
				}
			}

		});
		return g;
	}

	public static void testApartadoA(String file) {
		Graph<String, DefaultEdge> g = grafo(file);
```

```
            Ejercicio3.apartadoA(g, file);
        }

        public static void testsApartadoB(String file) {

            Graph<String, DefaultEdge> g = grafo(file);
            Ejercicio3.apartadoB(g, file, "Apartado B");
        }

}
```

RESULTADOS

Ejercicio 1

Fichero A

a)

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

```
strict digraph G {
  1 [ color="red" label="Edu" ];
  2 [ color="red" label="Sara" ];
}
```

b)

Ancestros de Maria: [Maria, Carmen, Lola, Edu, Pepa, Paco, Manuel]

```
strict digraph G {
  1 [ color="blue" label="Paco" ];
  2 [ color="blue" label="Pepa" ];
  3 [ color="blue" label="Edu" ];
  4 [ color="blue" label="Lola" ];
  5 [ color="black" label="Juan" ];
  6 [ color="black" label="Laura" ];
  7 [ color="blue" label="Manuel" ];
  8 [ color="blue" label="Carmen" ];
  9 [ color="black" label="Antonio" ];
  10 [ color="black" label="Pablo" ];
  11 [ color="black" label="Ana" ];
  12 [ color="black" label="Patricia" ];
  13 [ color="red" label="Maria" ];
  14 [ color="black" label="Sara" ];
  15 [ color="black" label="Marta" ];
  16 [ color="black" label="Rafael" ];
  17 [ color="black" label="Lourdes" ];
  1 -> 3 [ color="black" ];
  2 -> 3 [ color="black" ];
  3 -> 8 [ color="black" ];
  4 -> 8 [ color="black" ];
  5 -> 9 [ color="black" ];
  5 -> 10 [ color="black" ];
  6 -> 9 [ color="black" ];
  6 -> 10 [ color="black" ];
  7 -> 13 [ color="black" ];
  8 -> 13 [ color="black" ];
  7 -> 12 [ color="black" ];
  8 -> 12 [ color="black" ];
  8 -> 14 [ color="black" ];
  9 -> 14 [ color="black" ];
  10 -> 15 [ color="black" ];
  11 -> 15 [ color="black" ];
  10 -> 16 [ color="black" ];
  17 -> 16 [ color="black" ];
}
```

d)

Personas que tienen hijos/as con distintas personas: [Pablo, Carmen]

```
strict digraph G {
  1 [ color="red" label="Pablo" ];
  2 [ color="red" label="Carmen" ];
}
```

e)

```
strict graph G {
  1 [ label="Paco" ];
  2 [ label="Pepa" ];
  3 [ color="red" label="Edu" ];
  4 [ label="Lola" ];
  5 [ label="Juan" ];
  6 [ label="Laura" ];
  7 [ color="red" label="Manuel" ];
  8 [ color="red" label="Carmen" ];
  9 [ color="red" label="Antonio" ];
  10 [ color="red" label="Pablo" ];
  11 [ color="red" label="Ana" ];
  12 [ label="Patricia" ];
  13 [ label="Maria" ];
  14 [ label="Sara" ];
  15 [ label="Marta" ];
  16 [ color="red" label="Rafael" ];
  17 [ label="Lourdes" ];
  1 -- 3 [ style="solid" ];
  2 -- 3 [ style="solid" ];
  3 -- 8 [ style="solid" ];
  4 -- 8 [ style="solid" ];
  5 -- 9 [ style="solid" ];
  5 -- 10 [ style="solid" ];
  6 -- 9 [ style="solid" ];
  6 -- 10 [ style="solid" ];
  7 -- 13 [ style="solid" ];
  8 -- 13 [ style="solid" ];
  7 -- 12 [ style="solid" ];
  8 -- 12 [ style="solid" ];
  8 -- 14 [ style="solid" ];
  9 -- 14 [ style="solid" ];
  10 -- 15 [ style="solid" ];
  11 -- 15 [ style="solid" ];
  10 -- 16 [ style="solid" ];
  17 -- 16 [ style="solid" ];
}
```

Fichero B

a)

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

```
strict digraph G {
  1 [ color="red" label="Angela" ];
  2 [ color="red" label="Julia" ];
  3 [ color="red" label="Raquel" ];
  4 [ color="red" label="Josefina" ];
  4 -> 3 [ color="black" ];
  4 -> 2 [ color="black" ];
}
```

c)

```
Rafael y Sara son PRIMOS
Maria y Patricia son HERMANOS
Carmen y Rafael son OTROS
```

b)

Ancestros de Raquel: [Raquel, Ramon, Manuela, Francisco, Josefina, Encarna, Pedro, Daniel, Irene]
Se ha generado resultados/ejercicio1/1BApartado B.gv

```
strict digraph G {
  1 [ color="blue" label="Francisco" ];
  2 [ color="blue" label="Manuela" ];
  3 [ color="black" label="Laura" ];
  4 [ color="blue" label="Ramon" ];
  5 [ color="black" label="Marcos" ];
  6 [ color="black" label="Angela" ];
  7 [ color="blue" label="Irene" ];
  8 [ color="blue" label="Daniel" ];
  9 [ color="blue" label="Pedro" ];
  10 [ color="blue" label="Encarna" ];
  11 [ color="blue" label="Josefina" ];
  12 [ color="black" label="Javier" ];
  13 [ color="red" label="Raquel" ];
  14 [ color="black" label="Julia" ];
  15 [ color="black" label="Alvaro" ];
  1 -> 3 [ color="black" ];
  2 -> 3 [ color="black" ];
  3 -> 6 [ color="black" ];
  5 -> 6 [ color="black" ];
  1 -> 4 [ color="black" ];
  2 -> 4 [ color="black" ];
  7 -> 9 [ color="black" ];
  8 -> 9 [ color="black" ];
  9 -> 11 [ color="black" ];
  10 -> 11 [ color="black" ];
  11 -> 13 [ color="black" ];
  11 -> 14 [ color="black" ];
  4 -> 13 [ color="black" ];
  4 -> 14 [ color="black" ];
  11 -> 15 [ color="black" ];
  12 -> 15 [ color="black" ];
}
```

c)

```
Julia y Angela son PRIMOS
Alvaro y Raquel son HERMANOS
Laura y Raquel son OTROS
```

d)

Personas que tienen hijos/as con distintas personas: [Josefina]
Se ha generado resultados/ejercicio1/1BApartado D.gv

```
strict digraph G {
  1 [ color="red" label="Josefina" ];
}
```

e)

```
strict graph G {
  1 [ label="Francisco" ];
  2 [ label="Manuela" ];
  3 [ color="red" label="Laura" ];
  4 [ color="red" label="Ramon" ];
  5 [ color="red" label="Marcos" ];
  6 [ label="Angela" ];
  7 [ label="Irene" ];
```

```
  8 [ label="Daniel" ];
  9 [ color="red" label="Pedro" ];
  10 [ label="Encarna" ];
  11 [ color="red" label="Josefina" ];
  12 [ color="red" label="Javier" ];
  13 [ label="Raquel" ];
  14 [ label="Julia" ];
  15 [ label="Alvaro" ];
  1 -- 3 [ style="solid" ];
  2 -- 3 [ style="solid" ];
  3 -- 6 [ style="solid" ];
  5 -- 6 [ style="solid" ];
  1 -- 4 [ style="solid" ];
  2 -- 4 [ style="solid" ];
  7 -- 9 [ style="solid" ];
  8 -- 9 [ style="solid" ];
  9 -- 11 [ style="solid" ];
  10 -- 11 [ style="solid" ];
  11 -- 13 [ style="solid" ];
  11 -- 14 [ style="solid" ];
  4 -- 13 [ style="solid" ];
  4 -- 14 [ style="solid" ];
  11 -- 15 [ style="solid" ];
  12 -- 15 [ style="solid" ];
}
```

Ejercicio 2

a)

```
Numero de grupos de ciudades: 2

 Grupo numero 1:[Ciudad5, Ciudad2, Ciudad4, Ciudad3, Ciudad1]

 Grupo numero 2:[Ciudad8, Ciudad11, Ciudad10, Ciudad6, Ciudad7, Ciudad9]
```

```
strict graph G {
  1 [ color="green" ];
  2 [ color="green" ];
  3 [ color="green" ];
  4 [ color="green" ];
  5 [ color="green" ];
  6 [ color="yellow" ];
  7 [ color="yellow" ];
  8 [ color="yellow" ];
  9 [ color="yellow" ];
  10 [ color="yellow" ];
  11 [ color="yellow" ];
  1 -- 2 [ color="green" ];
  2 -- 3 [ color="green" ];
  3 -- 5 [ color="green" ];
  2 -- 4 [ color="green" ];
  4 -- 5 [ color="green" ];
  1 -- 3 [ color="green" ];
  5 -- 1 [ color="green" ];
  6 -- 8 [ color="yellow" ];
  9 -- 6 [ color="yellow" ];
  8 -- 10 [ color="yellow" ];
  10 -- 9 [ color="yellow" ];
  6 -- 10 [ color="yellow" ];
  6 -- 7 [ color="yellow" ];
  7 -- 9 [ color="yellow" ];
  7 -- 11 [ color="yellow" ];
  11 -- 6 [ color="yellow" ];
  7 -- 8 [ color="yellow" ];
}
```

b)

```
strict graph G {
  1 [ color="blue" label="Ciudad5" ];
  2 [ color="blue" label="Ciudad2" ];
  3 [ color="blue" label="Ciudad4" ];
  4 [ color="blue" label="Ciudad3" ];
  5 [ color="blue" label="Ciudad1" ];
  5 -- 2 [ color="blue" ];
  4 -- 1 [ color="blue" ];
  2 -- 4 [ color="blue" ];
  3 -- 1 [ color="blue" ];
```

```
  1 -- 5 [ color="blue" ];
  2 -- 3 [ color="blue" ];
  5 -- 4 [ color="blue" ];
}
```

c)

```
strict graph G {
  1 [ color="blue" label="Ciudad1" ];
  2 [ color="blue" label="Ciudad2" ];
  3 [ color="blue" label="Ciudad3" ];
  4 [ color="blue" label="Ciudad4" ];
  5 [ color="blue" label="Ciudad5" ];
  6 [ label="Ciudad6" ];
  7 [ label="Ciudad7" ];
  8 [ label="Ciudad8" ];
  9 [ label="Ciudad9" ];
  10 [ label="Ciudad10" ];
  11 [ label="Ciudad11" ];
  1 -- 2 [ ];
  2 -- 3 [ color="blue" ];
  3 -- 5 [ ];
  2 -- 4 [ color="blue" ];
  4 -- 5 [ color="blue" ];
  1 -- 3 [ color="blue" ];
  5 -- 1 [ color="blue" ];
  6 -- 8 [ ];
  9 -- 6 [ ];
  8 -- 10 [ ];
  10 -- 9 [ ];
  6 -- 10 [ ];
  6 -- 7 [ ];
  7 -- 9 [ ];
  7 -- 11 [ ];
  11 -- 6 [ ];
  7 -- 8 [ ];
}
```

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: Ciudad5 y Destino: Ciudad3
Se ha generadoresultados/ejercicio2/2Apartado D.gv
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
Se ha generadoresultados/ejercicio2/2Apartado D.gv
```

```
strict graph G {
  1 [ style="solid" label="Ciudad1" ];
  2 [ style="solid" label="Ciudad2" ];
  3 [ style="solid" label="Ciudad3" ];
  4 [ style="solid" label="Ciudad4" ];
  5 [ style="solid" label="Ciudad5" ];
  6 [ style="solid" label="Ciudad6" ];
  7 [ style="bold" label="Ciudad7" ];
  8 [ style="bold" label="Ciudad8" ];
  9 [ style="bold" label="Ciudad9" ];
  10 [ style="solid" label="Ciudad10" ];
  11 [ style="solid" label="Ciudad11" ];
  1 -- 2 [ style="solid" label="45 euros" ];
  2 -- 3 [ style="solid" label="15 euros" ];
  3 -- 5 [ style="solid" label="40 euros" ];
  2 -- 4 [ style="solid" label="25 euros" ];
  4 -- 5 [ style="solid" label="20 euros" ];
  1 -- 3 [ style="solid" label="60 euros" ];
  5 -- 1 [ style="solid" label="35 euros" ];
  6 -- 8 [ style="solid" label="15 euros" ];
  9 -- 6 [ style="solid" label="60 euros" ];
  8 -- 10 [ style="solid" label="55 euros" ];
  10 -- 9 [ style="solid" label="40 euros" ];
  6 -- 10 [ style="solid" label="50 euros" ];
  6 -- 7 [ style="solid" label="30 euros" ];
  7 -- 9 [ style="bold" label="25 euros" ];
  7 -- 11 [ style="solid" label="30 euros" ];
  11 -- 6 [ style="solid" label="40 euros" ];
  7 -- 8 [ style="bold" label="10 euros" ];
}
```

Ejercicio 3

Fichero A

a)

```
Franjas horarias necesarias: 3
Composicion de las actividades
Franja horaria numero 1: [Actividad1, Actividad4, Actividad7]
Franja horaria numero 2: [Actividad2, Actividad9, Actividad5]
Franja horaria numero 3: [Actividad3, Actividad6, Actividad8]
```

b)

```
strict graph G {
  1 [ color="green" label="Actividad1" ];
  2 [ color="yellow" label="Actividad2" ];
  3 [ color="red" label="Actividad3" ];
  4 [ color="yellow" label="Actividad5" ];
  5 [ color="green" label="Actividad4" ];
  6 [ color="red" label="Actividad6" ];
  7 [ color="green" label="Actividad7" ];
  8 [ color="yellow" label="Actividad9" ];
  9 [ color="red" label="Actividad8" ];
  1 -- 2 [ style="solid" ];
  1 -- 3 [ style="solid" ];
  2 -- 3 [ style="solid" ];
  1 -- 4 [ style="solid" ];
  3 -- 4 [ style="solid" ];
  2 -- 5 [ style="solid" ];
  2 -- 6 [ style="solid" ];
  5 -- 6 [ style="solid" ];
  5 -- 4 [ style="solid" ];
  4 -- 6 [ style="solid" ];
  7 -- 8 [ style="solid" ];
  9 -- 8 [ style="solid" ];
  7 -- 9 [ style="solid" ];
  7 -- 2 [ style="solid" ];
}
```

Fichero B

a)

```
Franjas horarias necesarias: 4
Composicion de las actividades
Franja horaria numero 1: [Actividad1, Actividad11, Actividad8]
Franja horaria numero 2: [Actividad2, Actividad4, Actividad3, Actividad9, Actividad12, Actividad7]
Franja horaria numero 3: [Actividad10, Actividad5]
Franja horaria numero 4: [Actividad6]
```

b)

```
strict graph G {
  1 [ color="green" label="Actividad1" ];
  2 [ color="yellow" label="Actividad2" ];
  3 [ color="red" label="Actividad10" ];
  4 [ color="yellow" label="Actividad3" ];
  5 [ color="red" label="Actividad5" ];
  6 [ color="green" label="Actividad8" ];
  7 [ color="yellow" label="Actividad4" ];
  8 [ color="gray" label="Actividad6" ];
  9 [ color="yellow" label="Actividad9" ];
  10 [ color="yellow" label="Actividad7" ];
  11 [ color="green" label="Actividad11" ];
  12 [ color="yellow" label="Actividad12" ];
  1 -- 2 [ style="solid" ];
  1 -- 3 [ style="solid" ];
  2 -- 3 [ style="solid" ];
  1 -- 4 [ style="solid" ];
  1 -- 5 [ style="solid" ];
  4 -- 5 [ style="solid" ];
  6 -- 7 [ style="solid" ];
  6 -- 8 [ style="solid" ];
  7 -- 8 [ style="solid" ];
  7 -- 5 [ style="solid" ];
  5 -- 8 [ style="solid" ];
  6 -- 9 [ style="solid" ];
```

```
  10 -- 1 [ style="solid" ];
  4 -- 11 [ style="solid" ];
  2 -- 8 [ style="solid" ];
  11 -- 12 [ style="solid" ];
}
```