```java
package ejercicios;

import java.util.Comparator;

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)),
```

```
 74                      e -> GraphColors.colorIf(Color.blue,aristas1.contains(e)));
 75
 76              System.out.println(file + "_ApartadoB.gv generado en resultados/ejercicio2");
 77
 78          } else {
 79
 80              GraphColors.toDot(g,
 81                      "resultados/ejercicio2/" + file + "_ApartadoB.gv",
 82                      v->v.ciudad(),
 83                      v->v.precio().toString(),
 84                      v -> GraphColors.colorIf(Color.blue,vertices2.contains(v)),
 85                      e -> GraphColors.colorIf(Color.blue,aristas2.contains(e)));
 86
 87              System.out.println(file + "_ApartadoB.gv generado en resultados/ejercicio2");
 88
 89          }
 90
 91
 92      }
 93
 94  //  APARTADO C
 95
 96      public static void ApartadoC (Graph<Ciudad,Transporte> g, String file) {
 97
 98          ConnectivityInspector<Ciudad, Transporte> conIns= new ConnectivityInspector<>(g);
 99          List<Set<Ciudad>> conComps = conIns.connectedSets();
100
101          Set<Ciudad> c1 = conComps.get(0);
102          Set<Ciudad> c2 = conComps.get(1);
103
104          Graph<Ciudad,Transporte> g1 = grafoConexo(g,c1);
105          Graph<Ciudad,Transporte> g2 = grafoConexo(g,c2);
106
107          List<Ciudad> lc1 = new HeldKarpTSP<Ciudad,Transporte>().getTour(g1).getVertexList
     ();
108          List<Transporte> lt1= new HeldKarpTSP<Ciudad,Transporte>().getTour(g1).getEdgeList
     ();
109
110          List<Ciudad> lc2 = new HeldKarpTSP<Ciudad,Transporte>().getTour(g2).getVertexList
     ();
111          List<Transporte> lt2= new HeldKarpTSP<Ciudad,Transporte>().getTour(g2).getEdgeList
     ();
112
113          lc1.addAll(lc2);
114          lt1.addAll(lt2);
115
116          GraphColors.toDot(g,
117                  "resultados/ejercicio2/" + file + "_ApartadoC.gv",
118                  v->v.ciudad(),
119                  v->v.precio().toString(),
120                  v -> GraphColors.colorIf(Color.blue,lc1.contains(v)),
121                  e -> GraphColors.colorIf(Color.blue,lt1.contains(e)));
122
123          System.out.println(file + "_ApartadoC.gv generado en resultados/ejercicio2");
124
125      }
126
127      public static Graph<Ciudad,Transporte> grafoConexo (Graph<Ciudad,Transporte> g,
     Set<Ciudad> g1){
128          return SubGraphView.of(g,x->g1.contains(x),null);
129      }
130
```

```
131 //  APARTADO D
132
133     public static void ApartadoD (Graph<Ciudad,Transporte> g, String file) {
134
135         ConnectivityInspector<Ciudad, Transporte> conIns= new ConnectivityInspector<>(g);
136         List<Set<Ciudad>> conComps = conIns.connectedSets();
137
138         Set<Ciudad> c1 = conComps.get(0);
139         Set<Ciudad> c2 = conComps.get(1);
140
141         Graph<Ciudad,Transporte> g1 = grafoConexo(g,c1);
142         Graph<Ciudad,Transporte> g2 = grafoConexo(g,c2);
143
144         List<Ciudad> vertices1 = g1.vertexSet().stream().toList();
145         List<Ciudad> vertices2 = g2.vertexSet().stream().toList();
146
147         Set<GraphPath<Ciudad,Transporte>> sg1 = floyd(g1,vertices1);
148         Set<GraphPath<Ciudad,Transporte>> sg2 = floyd(g2,vertices2);
149
150         GraphPath<Ciudad,Transporte> q1 = minimo(sg1);
151         GraphPath<Ciudad,Transporte> q2 = minimo(sg2);
152
153         List<Ciudad> lc1 = q1.getVertexList();
154         List<Transporte> lt1 = q1.getEdgeList();
155
156         List<Ciudad> lc2 = q2.getVertexList();
157         List<Transporte> lt2 = q2.getEdgeList();
158
159         lc1.addAll(lc2);
160         lt1.addAll(lt2);
161
162         GraphColors.toDot(g,
163                 "resultados/ejercicio2/" + file + "_ApartadoD.gv",
164                 c -> c.ciudad().toString(),
165                 e -> e.duracion().toString(),
166                 v -> GraphColors.colorIf(Color.blue, lc1.contains(v)),
167                 e -> GraphColors.colorIf(Color.blue, lt1.contains(e)));
168
169         System.out.println(file + "_ApartadoD.gv generado en resultados/ejercicio2");
170
171     }
172
173     public static GraphPath<Ciudad,Transporte> minimo (Set<GraphPath<Ciudad,Transporte>>
    sg){
174         return sg.stream().filter(x->x.getLength()>=2).min(Comparator.comparing(x-
    >x.getWeight())).get();
175     }
176
177     public static Set<GraphPath<Ciudad,Transporte>> floyd (Graph<Ciudad,Transporte> g,
    List<Ciudad> vertices){
178
179         FloydWarshallShortestPaths<Ciudad, Transporte> a  = new
    FloydWarshallShortestPaths<>(g);
180         Set<GraphPath<Ciudad,Transporte>> s = new HashSet<>();
181
182         for(int i = 0;i<vertices.size();i++) {
183             for(int j = i+1;j<vertices.size();j++) {
184                 s.add(a.getPath(vertices.get(i), vertices.get(j)));
185             }
186         }
187
188         return s;
```

```
189
190     }
191 }
192
```