Ejercicio 3 – Grafos

Se cuenta con un grafo que representa el circuito eléctrico de un subsistema importante de un avión, donde los vértices representan componentes y las aristas, el cableado que une estos componentes. Sabiendo que no encontraremos más de una arista entre componentes y que los tipos Componente y Cableado ya se dan implementados.

```
public record Componente(Integer id, String nombre, Double coste) {…}
public record Cableado(Integer id, Double cms) {…}
```

SE PIDE:

Apartado A: Los componentes cuentan con un atributo *coste* que representa el coste de sustituir el componente en caso de avería. Obtener un subgrafo que contenga únicamente aquellos componentes con un coste mayor a 20.000€ y las aristas que los unen. El método recibirá como entrada el grafo original y debe retornar el subgrafo calculado. Este subgrafo no debe ser un grafo nuevo.

```
public static Graph<Componente, Cable> apartadoA(SimpleWeightedGraph<Componente, Cable> g) {
    Predicate<Componente> pv = c -> c.coste() > 20000.0;
    Predicate<Cable> pa = c -> true;
    Graph<Componente, Cable> sg = SubGraphView.of(g, pv, pa);
    return sg;
}
```

Apartado B: Cuando un componente falla, el fallo se propaga a los componentes adyacentes, produciendo en el sistema una salida incorrecta. Calcular el menor conjunto de componentes donde hay que colocar sensores para detectar cualquier salida errónea. El método recibirá como entrada el grafo original y debe retornar el conjunto de componentes calculado.

```
public static Set<Componente> apartadoB(SimpleWeightedGraph<Componente, Cable> g) {
    GreedyVCImpl<Componente, Cable> algA = new GreedyVCImpl<>(g);
    Set<Componente> componentesSalidaCorrecta = algA.getVertexCover();
    return componentesSalidaCorrecta;
}
```

Apartado C: Dados dos componentes A y B, y sabiendo que las aristas llevan asociado un peso correspondiente a la longitud del cable, encuentre el camino más corto entre los componentes A y B. El método recibirá como entradas el grafo original y dos objetos de tipo Componente, representando los componentes A y B. Finalmente, debe retornar el camino calculado.

```
public static GraphPath<Componente, Cable> apartadoC(SimpleWeightedGraph<Componente, Cable> g, Componente a,
        Componente b) {
    DijkstraShortestPath<Componente, Cable> alg = new DijkstraShortestPath<>(g);
    GraphPath<Componente, Cable> path = alg.getPath(a, b);
    return path;
}
```

Apartado D: Se sabe que cuando uno de los componentes falla, el error se manifiesta en él o en los que están conectados a él de manera directa o indirecta a través de otros componentes. Dado un componente C, encontrar el subconjunto de componentes en los que podría manifestarse el error. El método debe recibir como entrada el grafo original, y un objeto de tipo Componente, representando el componente C. El resultado debe ser

el conjunto de componentes calculado.

```
public static Set<Componente> apartadoD(SimpleWeightedGraph<Componente, Cable> g, Componente c) {
    ConnectivityInspector<Componente, Cable> alg = new ConnectivityInspector<>(g);
    List<Set<Componente>> compoConex = alg.connectedSets();
    System.out.println("Componentes conexas: " + compoConex);
    Set<Componente> conj = compoConex.stream().filter(comp -> comp.contains(c)).findFirst().get();
    return conj;

}
```

Tiempo estimado: 45 minutos **Puntuación: ADDA 25%, EDA 33,3%**