```java
/**-------------------------------------------
 * -- Estructuras de Datos.  2018/19
 * -- 2º Curso del Grado en Ingeniería [Informática | del Software | de Computadores].
 * -- Escuela Técnica Superior de Ingeniería en Informática. UMA
 * --
 * -- Examen 4 de febrero de 2019
 * --
 * -- ALUMNO/NAME:
 * -- GRADO/STUDIES:
 * -- NÚM. MÁQUINA/MACHINE NUMBER:
 * --
 * -------------------------------------------
 */

package dataStructures.graph;

import java.util.Iterator;
import java.util.Objects;

import dataStructures.dictionary.Dictionary;
import dataStructures.dictionary.HashDictionary;

import dataStructures.set.Set;
import dataStructures.set.HashSet;
import dataStructures.tuple.Tuple2;

public class DictionaryWeightedGraph<V, W extends Comparable<? super W>> implements WeightedGraph<V, W> {

    static class WE<V1, W1 extends Comparable<? super W1>> implements WeightedEdge<V1, W1> {

        V1 src, dst;
        W1 wght;

        WE(V1 s, V1 d, W1 w) {
            src = s;
            dst = d;
```

```
37              wght = w;
38          }
39
40          public V1 source() {
41              return src;
42          }
43
44          public V1 destination() {
45              return dst;
46          }
47
48          public W1 weight() {
49              return wght;
50          }
51
52          public String toString() {
53              return "WE(" + src + "," + dst + "," +
    wght + ")";
54          }
55
56          public int hashCode() {
57              return wght.hashCode();
58          }
59
60          public boolean equals(Object obj) {
61
62              boolean ok = false;
63              if(obj instanceof  WE){
64                  WE aux = (WE) obj;
65                  ok = wght.equals(aux.wght) &&
66                          ((src.equals(aux.src) && dst.
    equals(aux.dst)) ||
67                                  (src.equals(aux.dst
    ) && dst.equals(aux.src)));
68              }
69
70              return ok;
71          }
72
73          public int compareTo(WeightedEdge<V1, W1> o
    ) {
```

```
 74              return wght.compareTo(o.weight());
 75          }
 76      }
 77
 78      /**
 79       * Each vertex is associated to a dictionary
    containing associations
 80       * from each successor to its weight
 81       */
 82      protected Dictionary<V, Dictionary<V, W>> graph;
 83
 84      public DictionaryWeightedGraph() {
 85          graph = new HashDictionary<>();
 86      }
 87
 88
 89      public void addVertex(V v) {
 90
 91          if(!graph.isDefinedAt(v)) {
 92              graph.insert(v, new HashDictionary<V, W
    >());
 93          }
 94
 95      }
 96
 97      public void addEdge(V src, V dst, W w) {
 98
 99          if(!graph.isDefinedAt(src) || !graph.
    isDefinedAt(dst)){
100              throw new GraphException("Vértice/s no
    encontrado");
101          }
102
103          Dictionary<V, W> ed = graph.valueOf(src);
104          ed.insert(dst, w);
105          graph.insert(src, ed);
106
107
108          ed = graph.valueOf(dst);
109          ed.insert(src, w);
110          graph.insert(dst, ed);
```

```
111     }
112
113     public Set<Tuple2<V, W>> successors(V v) {
114
115         if(!graph.isDefinedAt(v)){
116             throw new GraphException("Vértice no
    encontrao");
117         }
118
119         Dictionary<V, W> aux = graph.valueOf(v);
120         Set<Tuple2<V, W>> zuzezore = new HashSet
    <>();
121
122         for (V suc: aux.keys()) {
123             W er_pezo = aux.valueOf(suc);
124             zuzezore.insert(new Tuple2<>(suc,
    er_pezo));
125         }
126
127         return zuzezore;
128     }
129
130
131     public Set<WeightedEdge<V, W>> edges() {
132
133         Set<WeightedEdge<V, W>> ed = new HashSet
    <>();
134
135         Set<V> vertices = vertices();
136
137         for (V v: vertices) {
138             Set<Tuple2<V, W>> suc = successors(v);
139             for (Tuple2<V, W> aux : suc) {
140                 ed.insert(new WE<>(v, aux._1(), aux.
    _2()));
141             }
142         }
143
144         return ed;
145     }
146
```

```
147
148
149
150
151
152     /** DON'T EDIT ANYTHING BELOW THIS COMMENT **/
153
154
155     public Set<V> vertices() {
156         Set<V> vs = new HashSet<>();
157         for (V v : graph.keys())
158             vs.insert(v);
159         return vs;
160     }
161
162
163     public boolean isEmpty() {
164         return graph.isEmpty();
165     }
166
167     public int numVertices() {
168         return graph.size();
169     }
170
171
172     public int numEdges() {
173         int num = 0;
174         for (Dictionary<V, W> d : graph.values())
175             num += d.size();
176         return num/2;
177     }
178
179
180     public String toString() {
181         String className = getClass().getSimpleName
    ();
182         String s = className + "(vertices=(";
183
184         Iterator<V> it1 = vertices().iterator();
185         while (it1.hasNext())
186             s += it1.next() + (it1.hasNext() ? ", "
```

```
186  : "");
187          s += ")";
188
189          s += ", edges=(";
190          Iterator<WeightedEdge<V, W>> it2 = edges().
    iterator();
191          while (it2.hasNext())
192              s += it2.next() + (it2.hasNext() ? ", "
     : "");
193          s += "))";
194
195          return s;
196      }
197 }
198
```