```java
/*******************************************************************************
 * Copyright 2015 Maximilian Stark | Dakror <mail@dakror.de>
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 ******************************************************************************/

package de.dakror.wseminar.graph.algorithm;

import static de.dakror.wseminar.util.Benchmark.Type.*;

import java.util.ArrayList;
import java.util.Collections;

import de.dakror.wseminar.Const.State;
import de.dakror.wseminar.graph.Edge;
import de.dakror.wseminar.graph.Graph;
import de.dakror.wseminar.graph.Path;
import de.dakror.wseminar.graph.Vertex;
import de.dakror.wseminar.graph.VertexData.InfPath;
import de.dakror.wseminar.graph.WeightedEdge;
import de.dakror.wseminar.graph.algorithm.base.PathFinder;
import de.dakror.wseminar.util.Benchmark.Type;
import de.dakror.wseminar.util.Visualizer;

/**
 * @author Maximilian Stark | Dakror
 */
public class Dijkstra<V> extends PathFinder<V> {
  ArrayList<Vertex<V>> list;

  public Dijkstra(Graph<Vertex<V>> graph, boolean animate) {
    super(graph, animate);

    list = new ArrayList<>();
    metaClasses = new Class<?>[] { InfPath.class };
  }

  @SuppressWarnings("unchecked")
  @Override
  public Path<Vertex<V>> findPath(Vertex<V> from, Vertex<V> to) {
    Visualizer.resetAll(graph, true, false);
    BM.time();

    for (Vertex<V> v : graph.getVertices()) {
      v.add(new InfPath<>());
      if (v.equals(from)) v.get(InfPath.class).d = 0;
      list.add(v);
      BM.add(v);
      BM.add(OPEN_LIST_SIZE);
      Visualizer.setVertexState(v, State.OPENLIST, true);
    };

    boolean found = false;

    Vertex<V> v = null;

    while (!list.isEmpty()) {
      Collections.sort(list, (a, b) -> {
        int c = Float.compare(a.get(InfPath.class).d, b.get(InfPath.class).d);
        if (c == 0) return a.data().toString().compareTo(b.data().toString());
        return c;
      });
      BM.add(SORTS);
      v = list.remove(0);
      BM.sub(OPEN_LIST_SIZE);
      BM.add(CLOSED_LIST_SIZE);
      Visualizer.setVertexState(v, State.CLOSEDLIST, true);

      if (v.equals(to)) {
        found = true;
        break;
      }

      takeStep(null, v, to);
    }

    if (!found) return null;

    Path<Vertex<V>> p = new Path<Vertex<V>>();
    p.setUserData("Dijkstra" + (animate ? " anim" : "") + " " + from.data() + "->" +
    to.data());

    for (Edge<Vertex<V>> e : graph.getEdges()) {
      Visualizer.setEdgePath(e, false, false, false);
    }

    while (v != null) {
      p.add(0, v);

      if (v.get(InfPath.class).parent != null) {
        Visualizer.setEdgePath(graph.getEdge(v, v.get(InfPath.class).parent), true, true);
      }
      Visualizer.setVertexState(v, State.BACKTRACK, false);
      v = v.get(InfPath.class).parent;

      BM.add(PATH_CREATION);
    }
    p.calculateCost(graph);

    p.setBenchmark(BM);


    BM.time();
    cleanup();
    Visualizer.resetAll(graph, true, false);

    return p;
  }

  @SuppressWarnings("unchecked")
  @Override
  protected boolean takeStep(Vertex<V> parent, Vertex<V> node, Vertex<V> to) {
    for (Edge<Vertex<V>> e : graph.getEdges(node)) {
      Vertex<V> oe = e.getOtherEnd(node);
      BM.add(oe);
      if (list.contains(oe)) {
        float alt = node.get(InfPath.class).d + (e instanceof WeightedEdge ?
        ((WeightedEdge<Vertex<V>>) e).getWeight() : 1);
        if (alt < oe.get(InfPath.class).d) {
          oe.get(InfPath.class).d = alt;
          oe.get(InfPath.class).parent = node;
          Visualizer.setEdgePath(e, true, true);
          BM.add(Type.OVERRIDES);
        }
      }
    }

    return false;
  }
}
```