

```

/*****
 * 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.HashMap;
import java.util.List;
import java.util.stream.Collectors;

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.PathCommons;
import de.dakror.wseminar.graph.WeightedEdge;
import de.dakror.wseminar.graph.algorithm.base.PathFinder;
import de.dakror.wseminar.util.Visualizer;

/**
 * @author Maximilian Stark | Dakror
 */
public class DFS<V> extends PathFinder<V> {
    HashMap<Vertex<V>, PathCommons<V>> meta;

    public DFS(Graph<Vertex<V>> graph, boolean animate) {
        super(graph, animate);
        meta = new HashMap<>();
    }

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

        if (!takeStep(null, from, to)) return null;

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

```

```

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

    BM.add(PATH_CREATION);
}

p.add(0, from);
BM.add(PATH_CREATION);
p.calculateCost(graph);

p.setBenchmark(BM);

BM.time();
cleanup();
Visualizer.resetAll(graph, false, false);
return p;
}

@Override
protected boolean takeStep(Vertex<V> parent, Vertex<V> node, Vertex<V> to) {
    PathCommons<V> pc = new PathCommons<>();
    pc.parent = parent;
    meta.put(node, pc);
    Visualizer.setVertexState(node, State.OPENLIST, false);
    BM.add(OPEN_LIST_SIZE);

    if (node.equals(to)) return true;

    List<Edge<Vertex<V>>> edges = graph.getEdgesFrom(node).stream().filter(e -> {
        Vertex<V> v = e.getOtherEnd(node);

        BM.add(v);

        boolean free = meta.get(v) == null;
        Visualizer.setEdgeActive(e, free, false);
        return free;
    }).sorted((a, b) -> Float.compare(a instanceof WeightedEdge ? ((WeightedEdge<Vertex<V>>)
a).getWeight() : 0,
                                     b instanceof WeightedEdge ? ((WeightedEdge<Vertex<V>>)
b).getWeight() : 0)).collect(Collectors.toList());

    BM.add(SORTS);

    // is target reachable?
    for (Edge<Vertex<V>> e : edges) {
        Vertex<V> oe = e.getOtherEnd(node);
        BM.add(oe);
        if (oe.equals(to)) {
            BM.sub(OPEN_LIST_SIZE);
            BM.add(CLOSED_LIST_SIZE);
            Visualizer.setVertexState(node, State.CLOSEDLIST);
            Visualizer.setEdgePath(e, true);
            return takeStep(node, oe, to);
        }
    }
}

```

```
// take next step
for (Edge<Vertex<V>> e : edges) {
    Vertex<V> oe = e.getOtherEnd(node);
    BM.add(oe);
    Visualizer.tick();

    BM.sub(OPEN_LIST_SIZE);
    BM.add(CLOSED_LIST_SIZE);
    Visualizer.setVertexState(node, State.CLOSEDLIST);
    Visualizer.setEdgePath(e, true);

    for (Edge<Vertex<V>> e1 : edges) {
        if (e1 == e) continue;
        Visualizer.setEdgeActive(e1, false, false);
    }

    if (takeStep(node, oe, to)) return true;

    for (Edge<Vertex<V>> e1 : edges) {
        if (e1 == e) continue;
        Visualizer.setEdgeActive(e1, true, false);
    }

    Visualizer.setEdgePath(e, false);
}

BM.add(BACK_TRACKS);
Visualizer.setVertexState(node, Visualizer.isEnabled() ? State.BACKTRACK : null, false);

return false;
}
}
```