Algorithmen und Datenstrukturen
Musterlösung 10

Martin Avanzini <martin.avanzini@uibk.ac.at>
Thomas Bauereiß <thomas.bauereiss@uibk.ac.at>
Herbert Jordan <herbert@dps.uibk.ac.at>
René Thiemann <rene.thiemann@uibk.ac.at>

31. Mai, zur Besprechung am 7. Juni

Aufgabe 1) Klassifizierung von Kanten

1. Die Klasse `DepthFirstClassifier` implementiert die Klassifizierung von Kanten mit Hilfe der Tiefensuche:

```
1  import java.util.Set;
2  import java.util.List;
3  import java.util.LinkedList;
4  import java.util.HashMap;
5
6  public class DepthFirstClassifier implements EdgeClassifier {
7
8      enum Color { WHITE, GRAY, BLACK };
9
10     HashMap<Integer, Color> color;
11     HashMap<Integer, Integer> d;
12     HashMap<Integer, Integer> f;
13     int time;
14
15     public List<EdgeClassification> classifyEdges(Graph g) {
16         // Initialisierung
17         color = new HashMap<Integer, Color>();
18         d = new HashMap<Integer, Integer>();
19         f = new HashMap<Integer, Integer>();
20         List<EdgeClassification> edges = new LinkedList<
               EdgeClassification>();
21
22         // Farbe aller Knoten auf weiss setzen
23         // Das Graph-Interface bietet leider keine bequeme
24         // Moeglichkeit, alle Knoten aufzuzaehlen, daher werden
25         // beginnend bei 0 aufsteigend Indizes geprueft, bis die
26         // richtige Anzahl an Knoten gefunden wurde.
27         int numNodes = 0;
28         for (int i = 0; numNodes < g.size(); i++) {
29             if (g.containsNode(i)) {
30                 color.put(i, Color.WHITE);
```

```
31                  numNodes++;
32              }
33          }
34
35          time = 0;
36
37          // Tiefensuche fuer alle (noch) weissen Knoten starten
38          for (Integer v : color.keySet()) {
39              if (color.get(v) == Color.WHITE) {
40                  // Klassifizierung mit Startknoten v
41                  // durchfuehren und das Ergebnis der Liste
42                  // klassifizierter Kanten hinzufuegen
43                  edges.addAll(classifyEdges(g, v));
44              }
45          }
46
47          return edges;
48      }
49
50      public List<EdgeClassification> classifyEdges(Graph g, int node) {
51          List<EdgeClassification> edges = new LinkedList<
                EdgeClassification>();
52          // Knoten als grau markieren und Anfangszeit speichern
53          color.put(node, Color.GRAY);
54          d.put(node, ++time);
55
56          // Nachfolger untersuchen
57          for (Integer u : g.successors(node)) {
58              // Fallunterscheidung nach Farben, um Kanten zu
59              // klassifizieren (siehe Folie 331)
60              switch (color.get(u)) {
61                  case WHITE:
62                      // Kante zu weissem Knoten ist Baumkante
63                      edges.add(new EdgeClassification(node, u,
                            EdgeClassification.Type.TREE));
64                      // falls Knoten noch weiss ist,
65                      // rekursiv besuchen
66                      edges.addAll(classifyEdges(g, u));
67                      break;
68
69                  case GRAY:
70                      // Kante zu grauem Knoten ist
71                      // Rueckwaerstkante
72                      edges.add(new EdgeClassification(node, u,
                            EdgeClassification.Type.BACK));
73                      break;
74
75                  case BLACK:
76                      // Kante zu schwarzem Knoten ist
77                      // entweder Vorwaerts- oder Querkante,
78                      // je nach Anfangszeiten
79                      if (d.get(node) < d.get(u)) {
80                          edges.add(new EdgeClassification(node, u,
                                EdgeClassification.Type.FORWARD));
81                      } else {
82                          edges.add(new EdgeClassification(node, u,
                                EdgeClassification.Type.CROSS));
83                      }
84                      break;
85              }
86          }
87
```

```
88            // Knoten als schwarz markieren und Endzeit speichern
89            color.put(node, Color.BLACK);
90            f.put(node, ++time);
91
92            return edges;
93        }
94    }
```

2. Für die Kanten des Graphen von Folie 329 ergibt sich folgende Klassifizierung:

```
(0, 1): TREE
(1, 4): TREE
(4, 3): TREE
(3, 1): BACK
(0, 3): FORWARD
(2, 4): CROSS
(2, 5): TREE
(5, 5): BACK
```

   Der Graph, der in den Java-Sourcen zu diesem Übungsblatt enthalten war, entspricht nicht ganz diesem Graphen, da aufgrund eines Tippfehlers die Kante (e, f) statt der Kante (e, d) enthalten war. Für diesen Graphen ergibt sich folgende Klassifizierung:

```
(0, 1): TREE
(1, 4): TREE
(4, 5): TREE
(5, 5): BACK
(0, 3): TREE
(3, 1): CROSS
(2, 4): CROSS
(2, 5): CROSS
```

Aufgabe 2) Starke Zusammenhangskomponenten (verschoben auf Blatt 11)