Algorithmen und Datenstrukturen
Übungszettel 9

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>

24. Mai, zur Besprechung am 31. Mai

Aufgabe 1) Graphen In der Vorlesung wurden verschiedene Möglichkeiten vorgestellt, um Graphen auf einem Rechner zu repräsentieren.

1. Implementieren Sie nachfolgendes Interface als
   (a) Adjazenzmatrix
   (b) Adjazenzliste.

```
1  import java.util.Set;
2
3  public interface Graph {
4
5      /**
6       * Ein einfaches Interface fuer Graphen G = (V,E)
7       */
8
9      /**
10      * @return die Anzahl der Knoten im Graphen
11      */
12     public int size();
13
14     /**
15      * @param  node ein Knoten
16      * @return wahr falls node in V
17      */
18     public boolean containsNode(int node);
19
20     /**
21      * @param  source ein Knoten
22      * @param  target ein Knoten
23      * @return wahr falls (source,target) in E
24      */
25     public boolean containsEdge(int source, int target);
26
27     /**
```

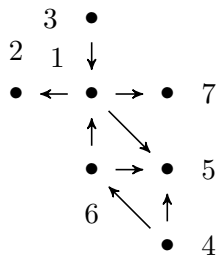
```
28          * @return einen frischen Knoten
29          */
30         public int freshNode();
31
32         /**
33          * @param  source ein Knoten
34          * @param  target ein Knoten
35          * @return fuegt die Kante (source,target) in den Graphen ein
36          */
37         public void addEdge(int source, int target) throws
               NodeNotFoundException;
38
39         /**
40          * loescht die Kante (source,target) aus dem Graphen falls diese
                existiert
41          * @param  source ein Knoten
42          * @param  target ein Knoten
43          */
44         public void removeEdge(int source, int target);
45
46         /**
47          * loescht den Knoten node aus dem Graphen falls dieser existiert
48          * @param  node ein Knoten
49          */
50         public void removeNode(int node);
51
52         /**
53          * @param  node ein Knoten
54          * @return die Menge der Vorgaenger des gegebenen Knotens falls
55          *         dieser vorhanden, ansonsten ein leeres Set
56          */
57         public Set<Integer> successors(int node);
58
59         /**
60          * @param  node ein Knoten
61          * @return die Menge der Nachfolger des gegebenen Knotens falls
62          *         dieser vorhanden, ansonsten ein leeres Set
63          */
64         public Set<Integer> predecessors(int node);
65
66         /**
67          * @param  node ein Knoten
68          * @return Eingangsgrad des gegebenen Knoten, ansonsten 0
69          */
70         public int inDegree(int node);
71
72         /**
73          * @param  node ein Knoten
74          * @return Ausgangsgrad des gegebenen Knoten, ansonsten 0
75          */
76         public int outDegree(int node);
77 }
```

2. Bestimmen Sie für beide Implementierungen die Komplexitäten der jeweiligen Methoden.

Aufgabe 2) Topologisches Sortieren

1. Zeigen Sie, dass ein Graph (V, E) *genau dann* azyklisch ist falls eine topologische Sortierung für (V, E) existiert.

2. In der Vorlesung wurde eine effiziente Methode zum topologisches Sortieren eingeführt. Wenden Sie diesen Algorithmus auf den folgenden Graphen an:



3. Implementieren Sie die Methode `topsort(Graph graph)` wobei hier `Graph` dem Interface aus Aufgabe 1 entspricht.

```java
import java.util.Stack;
import java.util.Iterator;

public class TopologicalSort {

    /**
     * Berechnet eine Topologische Sortierung
     *
     * @param graph
     *             Der betrachtete Graph (V,E).
     *             Zur Erleichterung kann V={1..graph.size()}
     *             angenommen werden.
     *
     * @return null falls der gegebene Graph zyklisch ist,
     *          ansonsten ein Array a der Groesse graph.size() fuer
     *          das gilt: a[v] == ord(v).
     */

    public int[] topsort(Graph graph) {
    throw new RuntimeException("zu implementieren!");
    }
}
```

4. Bestimmen Sie die Komplexität Ihres Algorithmus.