Datenstrukturen und Algorithmen: Hausübung 7

Felix Schrader, 3053850
Jens Duffert, 2843110
Eduard Sauter, 3053470

29. November 2015

Aufgabe 1

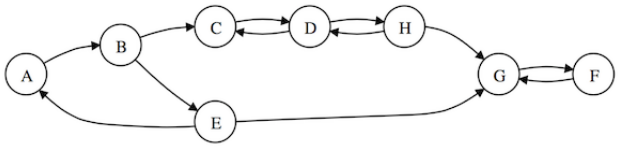
```
1  class AMDFGraph implements ADTUGraph{
2      // Die Knoten werden wie im Tiefendurchlauf geordnet.
3      var adjacencyMatrix, depthFirstList;
4      function create() {
5          // Die Adjazenzmatrix wird als verkettete Liste von
6          // verketteten Listen implementiert.
7          adjacencyMatrix = new LinkedList;
8          adjacencyMatrix.create();
9          // Auch die Knoten werden in einer verketteten Liste
10         // gespeichert.
11         depthFirstList = new LinkedList;
12         depthFirstList.create();
13     }
14     function nodeValue(u) {
15         return depthFirstList.retrieve(u);
16     }
17     function areAdjacent?(u, v) {
18         if(adjacencyMatrix.retrieve(u).retrieve(v) == 0 &&
19                 adjacencyMatrix.retrieve(v).retrieve(u) == 0) {
20             // Wenn die Eintraege in der Adjazenzmatrix mit
21             // Indizes u und v null sind, gibt es keine Kante
22             // zwischen u und v.
23             return false;
24         }
25         return true;
26     }
27     function edgeWeight(u, v) {
28         return adjacencyMatrix.retrieve(u).retrieve(v);
29     }
30     function sort() {
31         // Ordnet die Knoten in der Reihenfolge eines
32         // Tiefendurchlaufs.
33     }
34     function iterationStart() {
35         sort();
36         return depthFirstList.retrieve(0);
37     }
38     function iterationNext(u) {
39         if(u = depthFirstList.length() - 1) {
40             return NIL;
41         }
```

```
42          return u + 1;
43      }
44      function adjacentStart(u) {
45          sort();
46          for(i = 0; i < depthFirstList.length(); i++) {
47              if(adjacencyMatrix.retrieve(i).retrieve(u) != 0) {
48                  return i;
49              }
50          }
51          return NIL;
52      }
53      function adjacentNext(u, v) {
54          for(i = v + 1; i < depthFirstList.length(); i++) {
55              // Sucht beginnend bei v+1 nach einer Kante
56              // (Eintrag ungleich null in der Adjazenzmatrix).
57              if(adjacencyMatrix.retrieve(u).retrieve(i) != 0) {
58                  return i;
59              }
60          }
61          return NIL;
62      }
63      function insert(x) {
64          elements = depthFirstList.length();
65          // Das Element wird ans Ende von depthFirstList angefuegt.
66          depthFirstList.insert(elements, x);
67          // Die Adjazenzmatrix wird mit verketteten Listen
68          // gefuellt.
69          newColumn = new LinkedList;
70          newColumn.create();
71          adjacencyMatrix.insert(newColumn);
72          for(i = 0; i < elements; i++) {
73              // Die letzte Spalte der Matrix wird mit Nullen
74              // gefuellt (da es zu x keine Kanten gibt).
75              adjacencyMatrix.retrieve(i).insert(elements, 0);
76              // Es wird noch eine Zeile fuer x in die Matrix
77              // eingefuegt.
78              adjacencyMatrix.retrieve(elements + 1).insert(i, 0));
79          }
80      }
81      function remove(u) {
82          for(i = 0; i < depthFirstList.length(); i++) {
83              // Die zu u gehoerige Zeile in der Adjazenzmatrix wird
84              // geloescht.
85              adjacencyMatrix.retrieve(u).delete(i);
86          }
87          for(i = 0; i < depthFirstList.length() - 1; i++) {
88              // Die zugehoerige Spalte wird geloescht.
89              adjacencyMatrix.retrieve(i).delete(u);
90          }
91          // Der Knoten wird geloescht.
92          depthFirstList.delete(u);
93      }
94      function changeWeight(u, v, w) {
95          // Das Gewicht der Kante (u, v) wird in der
96          // Adjazenzmatrix geaendert.
97          adjacencyMatrix.retrieve(u).retrieve(v) = w;
98      }
99  }
```

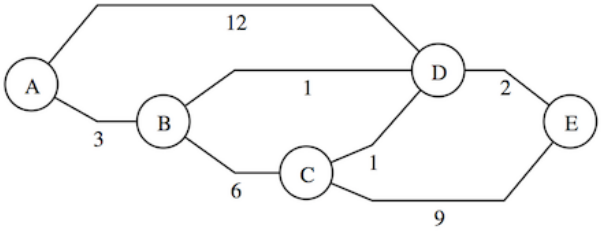
Aufgabe 3

a) Die *graphlib* von cpettit. Hier eine Zusammenfassung des Abschnittes “Graph Concepts” aus der Dokumentation:

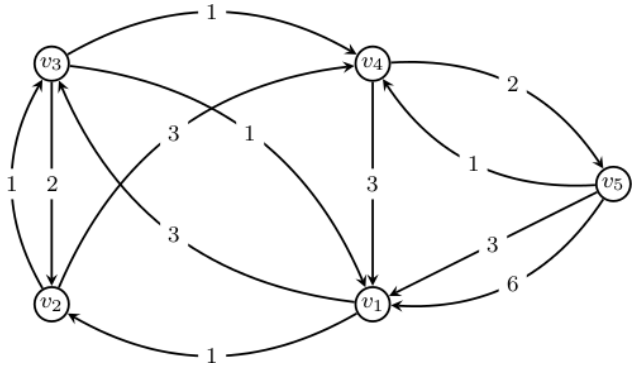
Directed Dies ist der Standard-Typ. Die Kanten sind gerichtet.



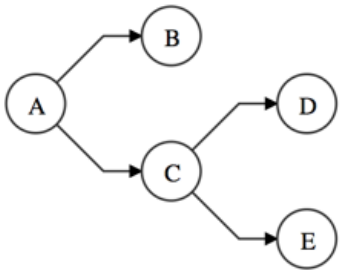
Undirected Diese sind analog zu denen in der Vorlesung.



Multigraph Hier können mehrere Kanten zwei gleiche Knoten verbinden. Nicht zu verwechseln mit einem “Hypergraphen”, bei dem eine Kante mehrere Knoten verbinden kann.



Compound Diese sind wie Bäume mit Wurzeln. Es lässt sich eine Kind-Elternteil Hirarchie auf den Knoten beschreiben.



b) *Der Tarjan Algorithmus*
Das Ziel dieses Algorithmus ist es, die Starken zusammenhangskomponenten eines gerichteten Graphen zu bestimmen.