Theoretische Aufgaben

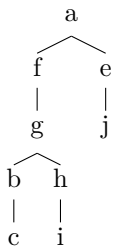
Aufgabe 1

Listing 1: Aufgabe 1

```
1  squareGraph(G, adjList)
2  list l
3  for i=0 to G.size                          //nodes
4      for j=0 to adjList[i]                  //edges
5          add adjList[i].adjList[j] to l
6      add l to adjList[i]
7      l.removeAll
8
9  squareGraph(G, adjMatrix)
10 for i=0 to adjMatrix.size                  //height
11     for j=0 to adjMatrix[0].size           //width
12         if (adjMatrix[i][j]==true)
13             for k=0 to adjMatrix[]         //line of chosen node
14                 if (adjMatrix[j][k] == true)
15                     adjMatrix[i][k] = true
```

Bei den Listen müssen nur alle Kantenlisten der Knoten durchgegangen werden und danach verknüpft werden. Rechnet man mit linearem verknüpfen, liegt die Laufzeit in $\Theta(V \cdot E)$.
Die Matrix-Variante ist viel ineffizienter, da für jeden Knoten die ganze Linie möglicher Verbindungen durchgegangen werden muss, um dann bei einem Treffer die Linie des betreffenden verknüpften Knotens durchzugehen, um herauszufinden, mit welcher dieser verknüpft ist. Die dreifach verschachtelte Schleife führt deshalb zu $\Theta(V^3)$.

Aufgabe 2



$d =$ `a:0, e:1, f:1, j:2, g:2, b:3, h:3, c:4, i:4 | d:` ∞
$\pi =$ `a:null, e:a, f:a, j:e, g:f, b:g, h:g, c:b, i:h | d:null`

Aufgabe 3

d[u] muss der kürzeste Abstand vom Startknoten sein[1]. d[u] ändert sich nicht, da durch Änderung der Reihenfolge in der Adjazenzliste die Distanz nicht ändert.

Aufgabe 4

[1] Theorem 22.5 im Buch

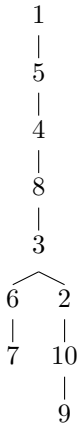
Listing 2: Aufgabe 4

```
16 bfs (G, boolean[] adj, int[] s)
17         int maxDist = inf
18         Node[] nodes = new Node[]
19
20         //create loose nodes from adjacency matrix,
21         //with arguments 'color', 'distance', 'index', 'parent'
22         for (int i=0; i<G.size; i++)
23                 nodes[i] = new Node("white", maxDist, i, null)
24
25         s.color = "grey"
26         s.depth = 0
27         s.parent = null
28
29         //initialize Queue
30         Queue<Node> Q = new LinkedList<Node>()
31
32         //trigger wave
33         Q.add(s)
34
35         while (Q.size() != 0)
36                 Node u = Q.remove();
37                 for (int i=0; i<G.size; i++)
38
39                         //check matrix if edge exists.
40                         //here's the main difference to normal BFS
41                         if (adj[u.index][i] == true)
42                                 Node v = nodes[i]
43                                 if (v.color == "white")
44                                         v.color = "grey";
45                                         v.depth = u.d + 1;
46                                         v.parent = u;
47                                         Q.add(v);
48                 u.c = "black";
```

Aufgabe 5

Beginnend beim Knoten 'a' als Beispiel beginnend, kann der Knoten 'j' entweder über 'e' oder aber über 'f' gefunden werden, je nach Priorisierung in der Adjazenzliste. Je nachdem, welcher Knoten zuerst auftaucht, wird zuerst in die Prioritätswarteschlange `Q` eingefügt und bearbeitet. Der Unterschied im Breitensuchbaum bestünde darin, dass 'j' entweder ein Kind von 'e' oder von 'f' sein könnte.

Aufgabe 6



$d, f =$ `1:1/20, 5:2/19, 4:3/18, 8:4/17, 3:5/16, 2:6/11, 10:7/10, 9:8/9, 6:12/15, 7:13/14`

$\pi =$ `1:null, 5:1, 4:5, 8:4, 3:8, 2:3, 10:2, 9:10, 6:3, 7:6`

Aufgabe 7