Übung 6: Graphen

Abgabetermin: 10.05.2011

Name: _____ **Matrikelnummer:** _____

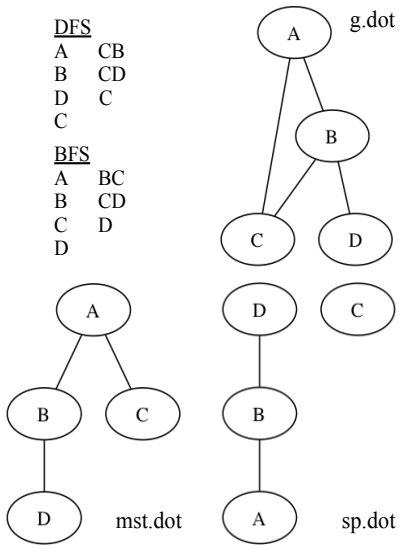
Gruppe: ☐ **G1** Di 10:15 ☐ **G2** Di 11:00 ☐ **G3** Di 12:45

Aufgabe	Punkte	gelöst	abzugeben schriftlich	abzugeben elektronisch	Korr.	Punkte
Aufgabe 1	24	☐	Java-Programm, Testfälle und Ergebnisse	Java-Programm	☐	

Aufgabe 1: DFS, BFS, Minimal Spanning Tree, ShortestPath (24 Punkte)

Implementieren Sie einen Graphen der Knoten (vertex) und Kanten (edge) zwischen diesen Knoten speichern kann. Knoten speichern einen Buchstaben. Kanten können ungerichtet oder gerichtet und ungewichtet oder gewichtet sein. Implementieren Sie für den Graphen folgende Algorithmen: Depth-First-Search, Breadth-First-Search, Minimal Spanning Tree und Shortest Path. Folgendes Beispiel zeigt wie der Graph verwendet wird und was ausgegeben wird:

```
Graph g = new Graph();
Vertex a = new Vertex('A', true);  g.addVertex(a);
       b = new Vertex('B', false); g.addVertex(b);
       c = new Vertex('C', false); g.addVertex(c);
       d = new Vertex('D', false); g.addVertex(d);
g.addEdge(new Edge(a, b, 0, Edge.Kind.Undirected));
g.addEdge(new Edge(b, c, 0, Edge.Kind.Undirected));
g.addEdge(new Edge(b, d, 0, Edge.Kind.Undirected));
g.addEdge(new Edge(c, a, 0, Edge.Kind.Undirected));
g.makeDot("g.dot");
g.depthFirstSearch();
g.breadthFirstSearch();
g.makeMinimalSpanningTree(a);
g.makeDot("mst.dot");
g.makeShortestPath(a, d);
g.makeDot("sp.dot");
```



Implementieren Sie die Klassen *Graph*, *Vertex* und *Edge* mit folgenden Schnittstellen (alles *public*):

```
class Graph {
  ArrayList vertices, edges;
  Graph() { ... }
  void addVertex(Vertex v) { ... }
  void addEdge(Edge e) { ... }
  void makeDot(String filename) { ... }
  void depthFirstSearch() { ... }
  void breadthFirstSearch() { ... }
  void makeMinimalSpanningTree(
    Vertex start) { ... }
  void makeShortestPath(
    Vertex from, Vertex to) { ... }
}
```

```
class Vertex implements Comparable {
  final char value;
  final boolean isRoot;
  Vertex(char value, boolean isRoot) { ... }
  int compareTo(Object o) { ... }
  boolean equals(Object o) { ... }
  int hashCode() { ... }
}
class Edge {
  enum Kind { Directed, Undirected }
  final Vertex start;
  final Vertex end;
  final int weight;
  final Kind kind;
  Edge(Vertex start, Vertex end,
    int weight, Kind kind) { ... }
  boolean equals(Object o) { ... }
  int hashCode() { ... }
}
```

Implementierungshinweise:

a) Die Methode *addVertex* soll nur einen Wurzelknoten erlauben. Die Methode *addEdge* soll nur Kanten mit bekannten Knoten erlauben und Duplikate verbieten. Lösen Sie Exceptions aus, wenn diese Regeln verletzt werden.

b) Die Methode *makeDot(String filename)* erstellt eine Datei zur Anzeige in GraphViz (http://graphviz.org). Verwenden Sie dazu *DotMakerGraph.makeDot(Graph g)* aus der Vorgabe. Achtung, die Vorgabe setzt die Schnittstelle von *Graph*, *Vertex* und *Edge* wie oben spezifziert voraus.

c) Die Methoden *depthFirstSearch* und *breadthFirstSearch* durchlaufen den Graphen und geben jeden besuchten Knoten und dazu den aktuellen von Stack- bzw. Queueinhalt aus (siehe Abbildung auf Seite 1). Verwenden Sie die Klassen *java.util.Stack* bzw. *java.util.LinkedList* aus dem JDK.

d) Die Methoden *makeMinimalSpanningTree* und *makeShortestPath* sollen direkt die Kanten des Graphen, für den diese Methoden aufgerufen werden, verändern.

e) Wenn Sie für *makeMinimalSpanningTree* und *makeShortestPath* zusätzliche Felder benötigen, zB minWeight oder heapPos, ergänzen Sie diese in der Klasse *Vertex*.

f) Für *makeMinimalSpanningTree* und *makeShortestPath* benötigen Sie einen Heap. Die Klasse *Vertex* implementiert *Comparable*, d.h. Sie können Ihren Heap aus Übung 5 verwenden. Passen Sie Ihre Heap-Implementierung an, wo notwendig. Implementieren Sie die Methode *compareTo* in *Vertex* so, dass der Heap wie gewünscht sortiert.

g) Die *DotMakerGraph*-Vorgabedatei zeichnet Kanten mit *kind == Edge.Kind.Directed* als gerichtete Kanten und beschriftet Kanten mit *weight != 0* mit dem Kantengewicht.

Abzugeben ist: Java-Programm, Testfälle und Ergebnisse