Übung 7: Graphen

Abgabetermin: 05.05.2014

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	☐	

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

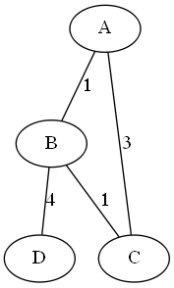
Gegeben ist ein Graph der aus Knoten (vertex) und Kanten (edge) besteht. Kanten können ungerichtet oder gerichtet sein, und können ein Kantengewicht haben. Implementieren Sie in der Klasse *Graphs* folgende Algorithmen: Depth-First-Search, Breadth-First-Search, Minimal Spanning Tree und Shortest Path. Folgendes Beispiel zeigt, wie die Vorgabeklassen und Ihre Graphs-Methoden verwendet werden sollen und wie die Ausgabe aussehen soll (Grafik).

```
Graph g = new Graph();
Vertex a = new Vertex('A', true);  g.addVertex(a);
Vertex b = new Vertex('B', false); g.addVertex(b);
Vertex c = new Vertex('C', false); g.addVertex(c);
Vertex d = new Vertex('D', false); g.addVertex(d);
g.addEdge(new Edge(a, b, 1, Edge.Kind.Undirected));
g.addEdge(new Edge(b, c, 1, Edge.Kind.Undirected));
g.addEdge(new Edge(b, d, 4, Edge.Kind.Undirected));
g.addEdge(new Edge(c, a, 3, Edge.Kind.Undirected));
Out.open("g.dot");
Out.println(DotMaker.makeDotForGraph(g));
Out.close();
Graphs.depthFirstSearch(g);
Graphs.breadthFirstSearch(g);
Graph mst = Graphs.makeMinimalSpanningTree(g, a);
Out.open("mst.dot");
Out.println(DotMaker.makeDotForGraph(mst));
Out.close();
Graph sp = Graphs.makeShortestPath(g, a, d);
Out.open("sp.dot");
Out.println(DotMaker.makeDotForGraph(sp));
Out.close();
```

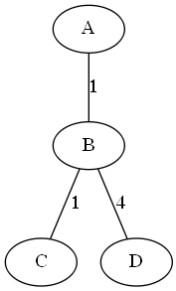
DFS
A BC
B DC
D C
C

BFS
A BC
B CD
C D
D

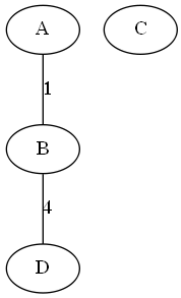
g.dot



mst.dot



sp.dot



Verwenden Sie die Vorgabeklassen aus dem Paket *at.jku.ssw* (alles *public*):

```
class Graph {
  final List vertices;
  List edges;
  Graph() {...}
  void addVertex(Vertex v) {...}
  void addEdge(Edge e) {...}
  Vertex getRoot() {...}
  Edge[] getEdges(Vertex v) {  ... }
  Edge getEdge(Vertex start, Vertex end) {...}
}
class VertexArrayPriorityQueue extends ... {
  VertexArrayPriorityQueue(Comparator c) {...}
  // offer(), contains(), poll() etc.
  void upHeap(Vertex v) {...}
  void print() {...}
}
```

```
class Vertex {
  final Object value;
  final boolean isRoot;
  boolean marked;
  Vertex dad;
  int minWeight = 0;
  int distance = 0;
  Vertex(Object val, boolean isRoot) {...}
}
class Edge {
  enum Kind { Directed, Undirected }
  final Vertex start, end;
  final int weight;
  final Kind kind;
  Edge(Vertex start, Vertex end,
       int weight, Kind kind) {...}
}
```

Implementieren Sie die Klasse *Graphs* mit folgender Schnittstelle:

```
class Graphs {
  public static void depthFirstSearch(Graph graph) {...}
  public static void breadthFirstSearch(Graph graph) {...}
  public static Graph makeMinimalSpanningTree(Graph graph, Vertex start) {...}
  public static Graph makeShortestPath(Graph graph, Vertex from, Vertex to) {...}
}
```

Implementierungshinweise:

a) Die Methoden *depthFirstSearch* und *breadthFirstSearch* durchlaufen den Graphen und geben jeden besuchten Knoten und dazu den aktuellen Stack- bzw. Queueinhalt aus (siehe Abbildung auf Seite 1). Verwenden Sie die Klassen *ArrayStack* bzw. *ArrayQueue* aus der Vorgabedatei.

b) Die Methode *makeMinimalSpanningTree* erzeugt einen neuen Graphen welcher nur mehr die Kanten des Minimal Spanning Tree enthält. Verwenden Sie dafür die vorgegebene Prioritätswarteschlange *VertexArrayPriorityQueue*, der Sie eine Instanz von *MinWeightVertexComparator* übergeben, um die Knoten nach *minWeight* zu gewichten.

c) Die Methode *makeShortestPath* erzeugt einen neuen Graphen welcher nur mehr die Kanten des kürzesten Pfades enthält. Verwenden Sie dafür die vorgegebene Prioritätswarteschlange *VertexArrayPriorityQueue*, die Sie mit einer Instanz von *DistanceVertexComparator* erzeugen, um die Knoten im Heap nach *distance* zu gewichten.

d) Die Methode *DotMaker.makeDotForGraph(Graph graph)* erzeugt GraphViz-Darstellungen Ihrer Graphen. DotMaker zeichnet Kanten mit *kind == Edge.Kind.Directed* als gerichtete Kanten und beschriftet Kanten mit *weight != 0* mit dem Kantengewicht.

e) Implementieren Sie sich eine private Hilfsmethode welche einen Graphen kopiert und arbeiten Sie in den Algorithmen *makeMinimalSpanningTree* und *makeShortestPath* mit der Kopie des ursprünglichen Graphens.

f) Testen Sie Ihre Implementierung mit der Vorgabedatei *GraphTest.java* und vergleichen Sie Ihre Ergebnisse mit *GraphTest.Output.txt*.

Abzugeben ist: Java-Programm und Testergebnisse