Algorithmik SS 2003 Prof. Dr. Heiner Klocke

Praktikum 3:
- Entwurf und Implementation einer Adapterklasse zur Anbindung von Graph-Algorithmen an XML-Bäume
- Analyse konkreter Graph-Algorithmen

Späteste Abnahme: A1, B1, C1: 05.06. A2, B2, C2: 12.06.

Name: ..... Matr-Nr: .....

Datum: …… Unterschrift des Dozenten (wenn bestanden): ……..……

Aufgabe 1 (Graph-Algorithmen und XML-Strukturen)

Ein gerichteter, gewichteter **Graph** G(N, E) wird durch eine Menge von Knoten N (Nodes) und Kanten (Edges) beschrieben. Den Kanten können Gewichte (Kosten) zugeordnet werden. Der Graph ist dann durch eine Menge von Tupeln der Form

(*fromNode*, *toNode*, *cost*)

beschrieben. Bei vielen Graphalgorithmen, die man als Klassen im Web findet, werden die Tupel für die Eingabe in einem Textfile erwartet. In dieser **Aufgabe** sollen die Knoten, Kanten und Kantengewichte jedoch aus einer Datenbank gelesen und dann als austauschbare Datenstruktur in Form von XML-Bäumen generiert bzw. ausgegeben werden. Damit die XML-Bäume von Graphalgorithmen als Eingabe benutzt werden können, sollen Sie eine **Adapterklasse** schreiben, welche die XML-Strukturen in die von den Algorithmen erwartete Eingabeform transformiert.

Schreiben Sie eine Adapterklasse für die Klasse *Graph.java*. Die Klasse *Graph* implementiert vier Graph-Algorithmen, u. a. den Algorithmus von Dijkstra zur Berechnung der Kürzesten Wege. Das Verständnis des Algorithmus ist für die Lösung dieser Aufgabe nicht relevant. Er wird später in der Vorlesung erklärt.

Laden Sie sich von meiner Homepage die Files **code.zip** und **docs.zip**. Sie finden darin die Klassen *Graph* und die benötigten Hilfsklassen. Mit NetBeans klappt die Kompilierung ohne Probleme. Die Ausführung kann sowohl unter NetBeans – dazu muss das Working-Direktory entsprechend gesetzt werden – oder auch im DOS-Fenster erfolgen (java Graph <textfile.txt>, z.B. java Graph graph1.txt). Nach dem Start werden der Start- und der Zielknoten abgefragt.

Die Adapterklasse soll jede *Kante* des XML-Baumes zur internen Repräsentation der Kanten in der Klasse *Graph* hinzufügen. Ebenso muss der Start- und Endknoten sowie der gewählte Algorithmus – z. B. d für Dijkstra – als Parameter übergeben werden.

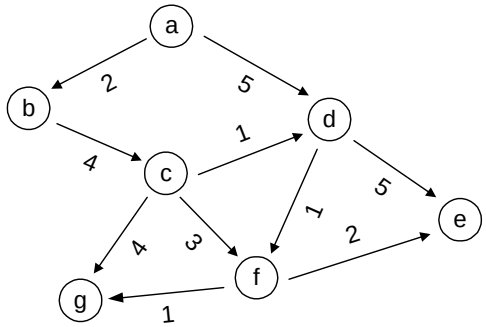
Beispiel: Knoten und Kanten des Graphen als XML-Baum:

```
<graph>
    <edge>
        <from>a</from>
        <to>b</to>
        <cost>2</cost>
    </ edge >

    < edge >
        <from>b</from>
        <to>c</to>
        <cost>4</cost>
    </ edge >

    < edge >
        <from>c</from>
        <to>g</to>
        <cost>4</cost>
    </ edge >

    …
</graph>
```



Ausgeben soll die Adapterklasse

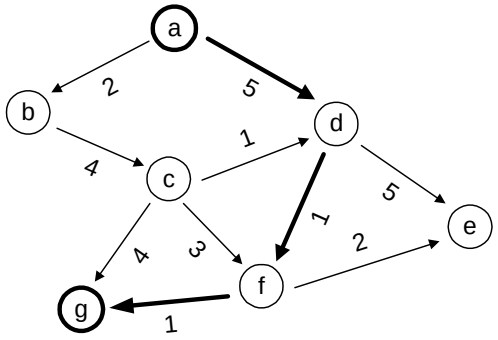
- den Start-und Zielknoten,
- den Wert des kürzesten Weges und
- die Kanten, welche den kürzesten Weg bilden

als XML-Baum. Damit ist eine beliebige Darstellung des Ergebnisses durch XSL-Regeln möglich.

Ergebnis: Dijkstras shortestpath als XML-Baum:

```
<dijkstra>
    <shortestpath>
        <from>a</from>
        <to>g</to>
        <cost>7</cost>
    </shortestpath>
    <edges>
        <edge>
            <from>a</from>
            <to>d</to>
            <cost>5</cost>
        </edge>
        <edge>
            <from>d</from>
            <to>f</to>
            <cost>1/cost>
        </edge>
        <edge>
            <from>f</from>
            <to>g</to>
            <cost>1/cost>
        </edge>

    </edges>
</dijkstra>
```



Aufgabe 2 (Analyse)

Analysieren Sie den Algorithmus von Dijkstra mit Hilfe des Ratio-Tests und des Powertests. Erzeugen Sie dazu Konten und Kanten durch einen Zufallsgenerator. Verändern Sie bei der Analyse die Anzahl der Knoten und der Kanten. Welche Ergebnisse liefern die Tests bei kantenreichen (viele Kanten) und kantenarmen (= bzgl. der Knotenzahl wenig Kanten) Graphen? Stellen Sie die Ergebnisse grafisch dar.