HOCHSCHULE FURTWANGEN UNIVERSITY HFU

Prof. Dr. W. Rülling
Fakultät Informatik

11.1.20

Algorithmen und Datenstrukturen
Übung 10 (AIN2)

In dieser Übung soll der Dijkstra-Algorithmus zur Berechnung eines Kürzeste-Wege-Baums in einem Graphen implementiert werden.

Implementieren Sie dazu eine Klasse `Graph` zur Beschreibung eines gerichteten Graphen mit Kantengewichten, so dass das vorgegebene Hauptprogramm funktioniert.
In dem Hauptprogramm wird ein Graph konstruiert, dessen Knoten durch Namen wie `"Hamburg"` oder `"München"` beschrieben werden. Intern kann ein solcher Knoten beispielsweise als Objekt einer (inneren) Klasse `Node` dargestellt werden, das neben dem Namen auch Zusatzinformationen, wie etwa eine Entfernungsangabe (`d`-Wert) enthält. Eine weitere für die Aufgabe nützliche Zusatzinformation ist die Liste der vom Knoten ausgehenden Kanten.

Mit der Methode
`void insertEdges(String a, String b, int w)`
sollen die beiden gerichteten Kanten *(a,b)* und *(b,a)* mit dem Kantengewicht *w* in den Graphen eingetragen werden. Die Knotenmenge des Graphen ergibt sich aus der Menge der verwendeten Strings. Es wäre deshalb hilfreich in der Klasse `Graph` ein Wörterbuch anzulegen, dass die verwendeten Städtenamen auf `Node`-Objekte abbildet.

Schließlich soll mit der Methode
`public void Dijkstra(String s, Map<String,Integer> result)`
der **Dijkstra-Algorithmus** am Startknoten `s` beginnend ausführt werden. Als Ergebnis soll man in der Abbildung `result` für jeden Knotennamen die Entfernung des Knotens von `s` erhalten.

```
import java.util.*;
public class GraphTest
{   public static void main(String[] args)
      { Graph G= new Graph();
        G.insertEdges("Hamburg",   "Münster",   281);
        G.insertEdges("Hamburg",   "Hannover",  151);
        G.insertEdges("Hamburg",   "Berlin",    280);
        G.insertEdges("Münster",   "Aachen",    560);
        G.insertEdges("Aachen",    "Koblenz",   154);
        G.insertEdges("Koblenz",   "Frankfurt", 125);
        G.insertEdges("Hannover",  "Frankfurt", 351);
        G.insertEdges("Frankfurt", "Würzburg",  119);
        G.insertEdges("Würzburg",  "Stuttgart", 151);
        G.insertEdges("Stuttgart", "Basel",     273);
        G.insertEdges("Stuttgart", "München",   221);
        G.insertEdges("Koblenz",   "Basel",     403);
        G.insertEdges("Aachen",    "Basel",     539);
        G.insertEdges("Hannover",  "Erfurt",    217);
        G.insertEdges("Erfurt",    "Würzburg",  195);
        G.insertEdges("Berlin",    "Leipzig",   192);
        G.insertEdges("Leipzig",   "Erfurt",    146);
        G.insertEdges("Leipzig",   "München",   431);
        G.insertEdges("Basel",     "München",   394);

        Map<String,Integer> result= new HashMap<String,Integer>();
        G.dijkstra("Hamburg",result);

        for (String v:G.getNodes())
           System.out.println(v + ": " + result.get(v));
      }
}
```

Hinweise: Ein wichtiger Aspekt des Dijkstra-Algorithmus besteht darin, dass man in jedem Schleifendurchlauf einen Knoten mit minimalem d-Wert ermittelt. Dazu sollten Sie die Knoten des Graphen nach den d-Werten sortiert in einer sortierten Menge M oder einer Priority-Queue ablegen.

Immer wenn sich der d-Wert eines Knoten ändert, muss der alte Eintrag aus M entfernt werden und der aktualisierte Knoten wieder eingefügt werden. Auf diese Weise bleibt die Sortierung ständig aktuell.

Beachten Sie auch, dass die Sortierung der Einträge in M eindeutig sein muss. D.h. für zwei verschiedene Knoten a und `b` darf die Vergleichsmethode `compareTo` nie den Wert 0 (Gleichheit) liefern.