Universität Karlsruhe
Fakultät für Informatik
ITI Prof. Dr. D. Wagner

Praktikum
WS 2008
09. Mai 2008

Übungsblatt 2

Abgabe: **30. Mai 2008**

Aufgabe 1: Über den Tellerand schauen

Schauen Sie sich die Ursprungsdateien der Datensätze noch einmal genauer an.

Einzelne Zeile der `Email`-Ursprungsdatei:
`[timestamp];[sender];[sender-institute];[recipient];[recipient-institute];[id]`

Material in der (vollständigen) `Patent`-Ursprungsdatei:

- `id` und `timestamp`
- Alle beteiligten Erfinder inklusive Ländern und Adressen
- Der Anmelder inklusive Land und Adresse
- internationaler Klassifikationsschlüssel IPC. Dieser ist hierarchisch eingeteilt in Sektionen (Buchstabe), diese in Klassen (Ziffern) und Unterklassen (Buchstabe) und diese wiederum in ca. 70.000 Haupt- und Untergruppen (Ziffern):
  Bsp.: `A21B 1/04` ⇒ `Bakers' ovens heated by fire before baking only`

Ein Graph entspricht der Modellierung eines bestimmten Aspekts der Daten. Betrachten Sie *beide* Datensätze und schlagen Sie jeweils mehrere Modelle vor und dokumentieren Sie diese.

Wie sinnvoll sind Ihre Modelle? Was bedeutet in Ihren Modellen:

(a) zwei Knoten sind adjazent

(b) ein Knoten hat einen hohen Grad

(c) das Gewicht einer Kante

(d) die kürzeste Entfernung zwischen zwei Knoten beträgt 4 Hops

Was kann man anhand der gefundenen sinnvollen Graphmodelle über die beiden Datensätze sagen?

BITTE WENDEN!

Aufgabe 2: Kürzeste Wege

Viele Maße benutzen kürzeste Wege Algorithmen.

(a) Einer der bekanntesten Algorithmen ist der Dijkstra Algorithmus. Implementieren Sie diesen wie folgt:

```
public static void dijkstra(Graph graph,
                            Node s,
                            boolean directed,
                            DataProvider cost,
                            NodeMap dist,
                            NodeMap pred)
```

Schauen Sie für genaue Spezifikationen dieser Parameter in die yFiles-Dokumentation: http://www.yworks.com/products/yfiles/doc/api/y/algo/ShortestPaths.html

(b) Überprüfen Sie die Korrektheit Ihrer Dijkstra Implementierung mit der folgenden Klasse: *y.algo.ShortestPaths*

(c) Erstellen Sie eine Klasse, die den *Durchmesser* eines Graphen bestimmt.

```
int diameter(Graph graph, DataProvider edgeCost)
```

Berechnen Sie nun für Ihren erstellten Graphen von Übungsblatt 1 den Durchmesser.

(d) Machen Sie sich klar was der kürzeste Pfad zwischen zwei Knoten und der Durchmesser eines Graphen bedeuten. Interpretieren Sie diese Maße nun für Ihr Graphmodell.

Aufgabe 3: Heavy Load

Es stehen nun mehr Daten zur Verfügung (Emaildaten komplett, Patentdaten teilweise). Generieren Sie daraus einen Graphen und wiederholen Sie die Aufgabe 4 von Übungsblatt 1.

Versuchen Sie Ihre Algorithmen aus Aufgabe 2 auf diese Graphen anzuwenden.

BITTE WENDEN!

Aufgabe 4: Zentralität

Zentralitätsmaße werden in der Netzwerkanalyse genutzt um die Intuition zu quantifizieren, dass manche Knoten in einem Netzwerk eine zentralere Rolle spielen als andere. Diese Maße weisen i.d.R. jedem Knoten (oder jeder Kante) einen Skalar zu. Implementieren sie die folgenden wichtigen Zentralitätsmaße, nehmen Sie dazu an, dass V die Knotenmenge bezeichnet, und $d(v, w)$ für $v, w \in V$ die Länge des kürzesten Weges von v nach w ist.

(a) Eccentricity-Centrality:

$$c_E(v) = \frac{1}{e(v)} = \frac{1}{\max\{d(v, w) : w \in V\}}$$

(b) Closeness:

$$c_C(v) = \frac{1}{\sum_{w \in V} d(v, w)}$$

(c) Shortest-Path Node-Betweenness:

$$c_B(v) = \sum_{s \neq v \in V} \sum_{t \neq v \in V} \delta_{st}(v)$$

Dabei bezeichnet

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}} = \frac{\text{Anzahl } s\text{-}t\text{-SPs welche durch Knoten } v \text{ verlaufen}}{\text{Anzahl } s\text{-}t\text{-SPs}}$$

Dabei steht s-t-SP für einen kürzesten Weg zwischen den Knoten s und t. Beachten Sie für diese Definition, dass es mehrere kürzeste – und somit gleich lange – s-t-Wege geben kann. Somit können auch mehrere von diesen durch den Knoten v verlaufen.

Implementieren Sie diese Zentralitätsmaße nach folgender API:

```
void abcCentrality(Graph graph, NodeMap centrality, DataProvider edgeCosts)
```

Interpretieren Sie die Bedeutungen der einzelnen Maße in ihrem Netzwerkmodell. Beachten Sie hierbei die Bedeutung und den Umgang von *Zusammenhangskomponenten*, welche Sie z.B. mit Hilfe von Übungsblatt 1 einfach finden können.

BITTE WENDEN!

Aufgabe 5: Modul Programmierung (optional!)

Man kann den Editor yEd um Module erweitern, welche dann in die graphische Oberfläche eingebunden werden. Diese können dann z.B. Algorithmen direkt auf dem angezeigten Graphen durchführen und das Ergebnis wiederum visualisieren.

Die Klasse y.module.YModule ist die abstrakte Basisklasse für yEd Module. Um ein eigenes Modul zu schreiben, müssen Sie Folgendes tun:

(a) Eine Klasse erzeugen, welche von y.module.YModule ableitet.

(b) Einen Konstruktor in dieser Klasse definieren, welcher keine Argumente entgegennimmt.

(c) Die Methode mainrun() überschreiben.

Die Klasse sieht dann beispielsweise so aus:

```
package <username>;
import java.io.*;
import y.base.*;
import y.module.YModule;
import y.view.Graph2D;

public class HelloWorldModule extends YModule {

  public HelloWorldModule() {
    super("","","");
  }

  protected void mainrun() {
    Graph2D G=getGraph2D();
    System.out.println("Hello World");
  }
}
```

Jedes Modul besitzt eine Referenz auf den Graphen, der gerade im YEd dargstellt wird. Auf diesen Graphen kann über getGraph2D() zugegriffen werden.

Module Deklarieren
Damit YEd auf neue Module zugreifen kann, müssen diese deklariert werden. Module, die unter Tools (Werkzeuge) in YEd ansprechbar sein sollen, werden in der Datei tools.pkg im Verzeichnis ∼/.yed deklariert (dies wird nach dem ersten Aufruf von YEd automatisch erzeugt). Diese Datei sieht beispielsweise folgendermaßen aus:

```
<PACKAGE name="Tools">
  <PACKAGE name="myName">
  <MODULE name="Dijkstra" class="myPackage.myDijkstraClass"></MODULE>
  <MODULE name="Diameter of Graph" class="myPackage.myDiameter"></MODULE>
  <MODULE name="Eccentricity" class="myPackage.myEccentricity"></MODULE>
  </PACKAGE>
</PACKAGE>
```