Programmierkurs Python II – SS 2012

Übung 3

---

1 Wikipedia-Graph mit Gewichten (2 Punkte)

Die Datei `wikipedia.txt` enthält eine kleine Sammlung von Wikipedia-Links. Die Links sind als Listen von String-Paaren gespeichert, wobei jeder String ein Seitentitel (im Wikipedia-Titel-Format) ist. Die Strings sind mit Tabulatoren getrennt; der erste markiert die Quelle, der zweite das Ziel eines Links.

Implementiere eine Methode `parse(dateiname)`, die einen Dateinamen als Parameter nimmt und einen gerichteten Graphen zurückgibt, dessen Knoten mit den Wikipedia-Seitentiteln markiert sind und dessen Kanten Wikipedia-Links symbolisieren. Du kannst hierfür die Graph-Infrastruktur aus Übung 2 benutzen.

Die Kanten des Graphen sollen gewichtet werden. Dafür musst Du die Datenstruktur erweitern. Implementiere eine Methode `weight(node,node)`, die das Gewicht einer Kante zwischen zwei Knoten zurück gibt. Wenn zwischen den beiden Knoten keine Kante existiert, soll die Methode 0 zurück geben.

Im Wikipedia-Graph soll jede Kante symbolisch ein Gewicht bekommen, das anzeigt, wie „wahrscheinlich " es ist, dass der Link benutzt wird. Benutze für eine Kante `u,v` das Gewicht `100*indeg(v)/outdeg(u)`, also Ein- und Ausgangsgrad der beiden Knoten (auf `int` gerundet). Implementiere hierfür eine Methode `computeWeights()`, die in `__init__` aufgerufen wird (ggf. optional), aber auch getrennt aufgerufen werden kann.

2 Dijkstra (3 Punkte)

1. Der Dijkstra-Algorithmus funktioniert nicht mit negativen Kantengewichten. Erkläre wieso. **(1 Punkt)**

2. Implementiere den Dijkstra-Algorithmus in einer Methode `dijkstra(node)` in der Graphklasse, die den Dijkstra-Algorithmus mit `node` als Startknoten ausführt. Du kannst ihn auf dem Wikipedia-Graphen testen, z.B. mit „Apple" als Startknoten. Rückgabe des Algorithmus soll ein Dictionary sein: Schlüssel des Dictionaries sind alle Knoten des Graphen, Werte sind Paare aus 1. der

Distanz zum Startknoten und 2. dem Vorgängerknoten bei der Berechnung dieser Distanz. **(2 Punkte)**

3 Topologische Sortierung (3 Punkte)

Implementiere topologische Sortierung für Graphen in einer Methode `tsort()` In der Graphklasse. Parse die Datei `frikadellen.txt` zu einem Testgraphen (gleiches Format wie `wikipedia.txt`).

Füge der Graph-Klasse eine Methode `tsort` hinzu, die die Knoten des Graphen topologisch sortiert in einer Liste zurück gibt. Du kannst die Ergebnisse Deiner Methode mit dem Output des UNIX-Tools `tsort` vergleichen, das Listen wie die in `frikadellen.txt` als Graph betrachtet und topologisch sortiert:

```
dhcp104-206: Michaela$ tsort < frikadellen.txt
```

(Beachte hierbei, dass es mehrere mögliche topologische Sortierungen gibt.)

4 Max-Flow Min-Cut (Bonusaufgabe, 6 Punkte)

Die Bonusaufgabe beschäftigt sich mit dem Ford-Fulkerson-Algorithmus aus der Vorlesung. Zum Testen haven wir ein kleines Netzwerk in der Datei `zuege.txt` zur Verfuegung gestellt (= das Netzwerk von Folie 26). Eine High-Level Beschreibung des Algorithmus findet sich z.B. auch hier:
`http://www.cse.yorku.ca/~aaw/Wang/MaxFlowMinCutAlg.html`

Es gibt Teilpunkte für folgende Tasks:

1. Erweitere die Rerpräsentation für gewichtete Graphen in Netzwerke, indem Du eine Klasse `Network` von der Graph-Klasse ableitest. Es soll dafür die Methoden `capacity(node,node)` und `flow(node,node)` geben, um die Gesamt-Kapazität und den aktuellen Fluss von Kanten abzufragen. Äquivalent dazu soll es die Methoden `addFlow(node,node)` und `setCapacity(node,node)` geben, um den aktuellen Fluss zu verändern oder die Gesamtkapazität zu setzen. *(1 Punkt)*

2. Implementiere in der Netzwerkklasse die Methode `maxFlow(source,sink)`, die

   - Den Betrag des max flow zurück gibt *(2 Punkte)*

- Zusätzlich auch die finalen Flows aller Kanten speichert *(1 Punkt)*

3. Implementiere eine Methode `minCut(source,sink)`, die unter Benutzung der `maxFlow`-Methode den Graphen entlang des minimalen Schnittes teilt und als Ergebnis die Teilnetzwerke zurück gibt, die bei diesem Schnitt entstehen. *(2 Punkte)*

Abgabe bis Freitag, 18.05.2012, 08:30 Uhr