Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

13.07.2009

14. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Montag, 20.07.2009, 10:00 Uhr

Graphen

Aufgabe 1 (Gleichheit von Graphen)

Eine häufiges Problem im Umgang mit Graphen ist die Frage, wann zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ “gleich” sind. Allgemein spricht man in der Graphentheorie von einem *Graphisomorphismus*, wenn es eine bijektive Abbildung

$$f : V_1 \rightarrow V_2$$

gibt mit der Eigenschaft, dass $\{v_i, v_j\} \in E_1 \Rightarrow \{f(v_i), f(v_j)\} \in E_2$ (d.h. wenn es eine Kante in G_1 zwischen v_i und v_j gibt, so gibt es auch eine Kante zwischen den Abbildungen $f(v_i)$ und $f(v_j)$ im Graph G_2).

Wir betrachten nun den Sonderfall, dass die Bijektion f genau die Identitatsabbildung ist, d.h. $f(v_i) = v_i \forall v_i \in V_1$. Wir nehmen also an, dass zwei Graphen genau dann gleich sind, wenn sie

- die gleichen Knoten enthalten
- jeweils die gleichen Knoten mit einer Kante verbunden sind (d.h. die beiden Graphen enthalten auch genau dieselben Kanten).

Skizzieren Sie einen möglichst effizienten Algorithmus in Pseudocode, der feststellt, ob zwei Graphen nach dieser Definition gleich sind. In der Vorlesung wurden mehrere Repräsentationen für Graphen behandelt (Knotenlisten, Adjazenzmatrizen, . . .) – welche erscheint Ihnen am geeignetsten für diese Aufgabe? Schreiben Sie Ihren Algorithmus in Pseudocode auf und geben Sie explizit an, auf welcher Graph-Repräsentation er arbeitet. Geben Sie zudem die Best- und Worst-Case-Laufzeit Ihres Algorithmus abhängig von der Anzahl Knoten $|V|$ und der Anzahl Kanten $|E|$ an. Gehen Sie bei der Bearbeitung der Aufgabe durchgehend von *gerichteten* Graphen aus.

(7 Punkte)

Aufgabe 2 (Implementierung eines Graphalgorithmus)

Auf der Vorlesungsseite befindet sich die abstrakt eKlasse **`AbstractGraph`**, welche das ebenfalls dort verfügbare Interface **`Graph`** zur Modellierung von *gerichteten Graphen* implementiert.

Intern wird ein Graph durch eine Adjazenz-Hashtabelle (`HashMap<V,List<V>> adjacency`) dargestellt: Die zu einem Knoten u adjazenten Knoten werden als Liste an der zum Schlüssel u gehörigen Position in der Hashtabelle abgelegt.

a) Leiten Sie die Klasse `AdjacencyGraph` von der Klasse `AbstractGraph` ab und implementieren Sie die abstrakte Methode `addEdge(V src, V dst)`, welche eine Kante vom Knoten `src` zum Knoten `dst` zum Graphen hinzufügt.

Falls die Knoten `src` oder `dst` im Graph noch nicht vorhanden sind, soll in der Hashtabelle zunächst an entsprechender Stelle eine leere Knotenliste erzeugt werden.

(6 Punkte)

b) Implementieren Sie ferner den Algorithmus `DFS` zur Tiefensuche in der Methode `dfs` der Klasse `AbstractGraph`.

Verwenden Sie dabei folgende Datenstrukturen, mit den Bezeichnern entsprechend zur Vorlesung auf Folie 9-35, 9-36:

```
HashMap<V,Color> farbe  HashMap<V,Integer> d
HashMap<V,Integer> f    HashMap<V,V> pi
```

(12 Punkte)

c) Implementieren Sie schließlich die Methode `hasCycles`, welche `true` zurückgibt, falls der gespeicherte Graph einen Kreis enthält und ansonsten `false`.

Hinweis: Verwenden Sie die Tiefensuche zum Finden von Kreisen.

(4 Punkte)

Aufgabe 3 (Einfache Netzwerkanalyse)

Graphen werden in vielen Bereichen als formales Modell für wissenschaftliche Untersuchungen verwendet. Beispiele hierfür sind

- Planung und Modellierung von Stromnetzen
- Soziale Netzwerke
- Zitationsgraphen (Graph von Zitierungen) von wissenschaftlichen Arbeiten
- Modellierung der Verlinkungsstruktur im WWW
- ...

Obwohl somit bei den verschiedenen Anwendungen unterschiedliche Objekte und Beziehungen dargestellt werden, weisen obige Beispiele gemeinsame Charakteristika auf.

Eine Gemeinsamkeit zeigt sich, wenn man die Verteilung der *Ausgangsgrade* in einem Graph betrachtet. Der Ausgangsgrad eines Knotens u in einem Gerichteten Graph $G = (V, E)$ bezeichnet dabei die Anzahl von ausgehenden Kanten von u, d.h. $|\{(u, v) \in E \mid v \in V\}|$. Es zeigt sich, dass es jeweils sehr viele Knoten mit niedrigem und sehr wenige mit hohem Ausgangsgrad gibt

und dass die Knotenzahl pro Ausgangsgrad einer Exponentialfunktion folgend mit steigendem Ausgangsgrad fällt.

Schreiben Sie eine Methode **printDegreeDistribution**, welche zu jedem im Graph vorhandenen Ausgangsgrad die Anzahl der entsprechenden Knoten ausgibt. Die Ausgabe soll dabei nach Ausgangsgrad aufsteigend Sortiert erfolgen.

Verwenden Sie zur Implementierung der Methode eine HashMap, welche Ausgangsgrade auf Zähler abbildet. Durchlaufen Sie alle Knoten im Graph und erhöhen Sie zu jedem Knoten den Zähler entsprechend zum Ausgangsgrad des Knotens um eins.

Auf der Vorlesungsseite befindet sich die Klasse **GraphTest**, welche nacheinander verschiedene Graphen aus der Datei **testgraphs.txt** dargestellt als Knotenliste einliest und die Ausgangsgrad-Verteilung ausgibt. Die Ausgabe für die ersten drei Graphen soll dabei wie folgt aussehen:

```
1) Degree distribution for graph with 6 vertices and 11 edges:
   [0] 1   [1] 1   [2] 2   [3] 2
2) Degree distribution for graph with 6 vertices and 8 edges:
   [0] 1   [1] 3   [2] 1   [3] 1
3) Degree distribution for graph with 100 vertices and 148 edges:
   [0] 1   [1] 60   [2] 29   [3] 10
```

Welche der eingelesenen Graphen besitzen die oben beschriebene Eigenschaft? (8 Punkte)

Viel Spaß und viel Erfolg!