technische universität dortmund

Fakultät für Informatik
Lehrstühle 2, 7, 11, 13

Prof. Dr. C. Sohler, D. Köppl
R. Kalkreuth, D. Siedhoff, I. Kamehkhosh, M. Ludewig
`http://tiny.cc/dap2praktikum`

Sommersemester 2016

DAP2 Praktikum – Blatt 14

Ausgabe: 6. Juli — Abgabe: 19. Juli–22. Juli

Wichtig: Der Quellcode ist natürlich (wie immer) mit sinnvollen Kommentaren zu versehen.

Langaufgabe 14.1 (4 Punkte)

Lernziel: Eigene Graph-Klasse

Implementieren sie eine Klasse `Graph`, die ungerichtete Graphen modelliert. Implementieren sie dafür zuerst die Hilfsklassen `Edge` und `Node`.

`Edge`

- Die Klasse `Edge` modelliert eine Kante zwischen zwei Knoten.
- Ihr Konstruktor soll mit zwei `Node`-Objekten aufgerufen werden.
- Diese sollen in privaten Variablen `src` und `dst` abgelegt und über passende getter-Methoden zugänglich gemacht werden.

`Node`

- Die Klasse `Node` modelliert Knoten und enthält eine Adjazenzliste `ArrayList<Edge>`.
- Ihr Konstruktor soll mit einem Integer-Wert `id` aufgerufen werden.
- Ihre Attribute sollen als `private` deklariert und über getter-Methoden zugänglich gemacht werden.
- Sie soll die folgenden Methoden bieten:
  - `addEdge(Node dst)` die eine Kante zwischen diesem Knoten und dem Zielknoten `dst` zieht.
  - `equals(Object other)` die für einen übergebenen Node `true` zurück gibt, wenn dieser die gleiche `id` hat wie dieser.

`Graph`

- Die Klasse `Graph` soll bei Aufruf einen leeren Graphen erzeugen. Verwenden sie eine `ArrayList<Node>` um die Knoten des Graphen zu verwalten.
- Sie soll die folgende Methoden bieten:
  - `contains(int id)` die prüft, ob ein `Node` mit der `id` im Graphen vorhanden ist.
  - `addNode(int id)` die einen neuen `Node` in den Graphen einfügt, wenn die `id` frei ist.
  - `getNode(int id)` die den `Node` mit der übergebenen `id` zurück gibt, sonst `null`.
  - `addEdge(int src, int dst)` die eine Kante zwischen den Knoten mit den übergebenen `id`s hinzufügt, wenn diese vorhanden sind.
  - `static Graph fromFile(String filepath)` die einen Graphen aus einer Textdatei ausliest und zurück gibt.

Testen sie jede der Klassen zuerst einzeln mit je einer eigenen `main`-Methode. **Hinweis:** Eine `toString()`-Methode, die die Attribute des Objekts als String zurück gibt, kann dabei sehr hilfreich sein.

Langaufgabe 14.2 **(4 Punkte)**

Lernziel: Breitensuche

Verwenden sie ihre Klasse `Graph`, um den aus der Vorlesung bekannten Algorithmus zur Breitensuche auf ungerichteten Graphen zu implementieren. Gehen Sie dabei wie folgt vor:

- Schreiben Sie eine Methode `bfs`, die einen Graphen und eine Knoten-ID übergeben bekommt und von diesem Knoten ausgehend eine Breitensuche durchführt. Es ist also nach kürzesten Wegen von `s` zu allen anderen Knoten gesucht.
- Benutzen Sie dabei für die Färbung der Knoten `ArrayList`s und zur Verwaltung der noch zu durchsuchenden Knoten eine `Queue` (z.B. eine `LinkedList`) oder ähnliche Klassen aus der Java-API und nutzen Sie die Methoden `contains`, `add` bzw. `offer` und `remove` bzw. `poll`/`peek`, um die im Algorithmus beschriebenen Operationen durchführen zu können.
- Implementieren Sie die `main`-Methode, die als Parameter den Pfad zu einer `.graph`-Datei und eine nicht-negative Ganzzahl erhält. Die `.graph`-Datei soll eingelesen und daraus mittels `Graph.fromFile` ein Graph erzeugt werden, auf dem dann eine Breitensuche mit der übergebenen Ganzzahl als Knoten-ID durchgeführt werden soll. Achten Sie insbesondere auf spezielle Rückgaben der verwendeten Methoden. Geben Sie zum Schluss den Abstand aller Knoten zum Startknoten aus, die von diesem erreicht werden können.
- **Optional**: Geben Sie zu jedem erreichbaren Knoten aus, welcher Pfad der kürzeste ist. Schreiben Sie dazu eine Hilfsmethode, die auf der `ArrayList` und dem jeweiligen Zielknoten als Parameter aufbaut. Benutzen Sie dabei, dass jeder Knoten nur einmal als Zielknoten einer Kante vorkommt und überlegen Sie, warum das so ist.