Praktikumsaufgaben

Graphentheoretische Konzepte und Algorithmen

SoSe 19

30. Januar 2019

J. Padberg **Aufgaben**

Allgemeines zu allen Aufgaben

Die Aufgabenstellung ist aus folgenden Gründen nicht ganz genau spezifiziert:

- Sie sollen einen Entwurfsspielraum haben. Das heißt, Sie können relativ frei entscheiden, wie Sie die Aufgabe lösen, allerdings sollten Sie Ihre Entscheidungen bewusst fällen und begründen können. Insofern gibt es auch keine Musterlösung.
- Durch die unterschiedlichen Lösungen können Sie von Ihren Kommilitonen noch lernen und das *Structured Walk-Through* bleibt spannend.

Darüber hinaus dürfen Sie gerne unterschiedliche Quellen nutzen, aber nur wenn Sie diese auch angeben. Sie sollen auch nicht unbedingt alles selber programmieren, nutzen Sie den vorgegeben Datentyp oder gerne auch andere Libaries.
Für die Bearbeitung der Praktikumsaufgaben erhalten Sie auf der Homepage der LV

- Beispielgraphen für das Praktikum
- Links zu verschiedenen Libaries

Das Ergebnis der Bearbeitung einer Aufgabe wird von jeder Gruppe im Praktikum vorgestellt. Das heißt, die Aufgaben muss *vor Praktikumstermin fertig bearbeitet* sein. Über die Vorstellung hinaus wird für jede Aufgabe erwartet,

1. Implementierung der gestellten Aufgabe, also
   - eine korrekte und möglichst effiziente Implementierung in Java, die der vorgegeben Beschreibung entspricht,
   - die Kommentierung der zentralen Eigenschaften/Ereignisse etc. im Code und
   - hinreichende Testfälle in JUnit und ihre Kommentierung.

   Hinweis: Wenn Sie in der Implementierung englische Fachbegriffe nutzen wollen, können Sie die im Index von Diestel nachschlagen.

2. Schriftliche Erläuterung Ihrer Lösung (Lösungsdokumentation) *etwa 4-7 Seiten*

   - Bitte geben Sie folgende Daten im Kopf Ihrer Lösungsdokumentation an:
     - **Team:** Teamnummer sowie die Namen der Teammitglieder
     - **Aufgabenaufteilung:**
       a) Aufgaben, für die Teammitglied 1 verantwortlich ist;
       Dateien, die komplett/zum Teil von Teammitglied 1 implementiert/bearbeitet wurden
       b) Aufgaben, für die Teammitglied 2 verantwortlich ist;
       Dateien, die komplett/zum Teil von Teammitglied 2 implementiert/bearbeitet wurden
     - **Quellenangaben:** Angabe von wesentlichen Quellen, z.B. Web-Seiten/Bücher, von denen Quellcode/Algorithmen übernommen wurden , Namentliche Nennung von Studierenden der HAW, von denen Quellcode übernommen wurde
     - **Bearbeitungszeitraum:** Datum und Dauer der Bearbeitung an der Aufgabe von allen Teammitgliedern und Angabe der gemeinsamen Bearbeitungszeiten
   - kurze Beschreibung der Algorithmen und
   - Datenstrukturen
   - wesentliche Entwurfsentscheidungen ihrer Implementierung
   - Umsetzung der Aspekte der Implementierung
   - umfassende Dokumentation der Testfälle, wobei die Abdeckung der Testfälle diskutiert werden soll
   - Beantwortung der in der Aufgabe gestellten Fragen

Es gibt drei Praktikumsaufgaben, weitere Details werden in der jeweiligen Aufgabenstellung angegeben.

Eine Praktikumsaufgabe gilt als erledigt, wenn

1. Sie die Lösungsdokumentation am Praktikumsanfang abgegeben haben,
2. Sie im Praktikum Ihre Implementierung vorgestellt haben und diese von mir (mindestens) als ausreichend anerkannt wurde.

Aufgabe 1:
Visualisierung, Speicherung und Traversierung von Graphen

Die Graphen, mit denen Sie arbeiten, sollen in diesem Format gespeichert und gelesen werden: Dabei ist folgendes Format (in etwa die EBNF) zu verwenden:

```
["#directed;"]
node1,[":"attr1],[","node2,[" :"attr2],[" ("edge")"],[" :: "weight]];
```

`node1`, `node2` und `edge` sind Zeichenketten
`attr1`, `attr2` und `weight` sind Zahlen

Die Aufgabe umfasst:

- die Einarbeitung in die gewählte library
- das Einlesen/ Speichern von ungerichteten sowie gerichteten Graphen und die Visualisierung der Graphen,
  (Hinweis: reguläre Aussdrücke in Java nutzen)
- die Implementierung des Breadth-First Search (BFS) Algorithmus zur Traversierung eines Graphen,
  gefordert werden dabei
  - als Ergebnis der kürzeste Weg und die Anzahl der benötigten Kanten und
  - einfache JUnit-Tests, die Ihre Methoden überprüfen und
  - JUnit-Tests, die das Lesen, das Speichern sowie den BFS-Algorithmus überprüfen unter Benutzung der gegeben *.graph*-Dateien.

Wenn Sie sich wegen dieser Aufgabe per email an mich wenden, bitte geben Sie im Betreff die Teamnummer und die Aufgabennummer an.

Aufgabe 2: Berechnung des Spannbaums

In der Vorlesung wurde der Algorithmus von Kruskal zur Berechnung des Spannbaums vorgestellt.
Der Algorithmus von Prim dient ebenfalls der Berechnung eines minimalen Gerüstes in einem zusammenhängenden, ungerichteten, kantengewichteten Graphen.
Der Algorithmus beginnt mit einem trivialen Graphen T, der aus einem beliebigen Knoten des gegebenen Graphen besteht. In jedem Schritt wird nun eine Kante mit minimalem Gewicht gesucht, die einen weiteren Knoten mit T verbindet. Diese Kante und der entsprechende Knoten werden zu T hinzugefügt. Das Ganze wird solange wiederholt, bis alle Knoten in T vorhanden sind; dann ist T ein minimales Gerüst.

Algorithmus

Wähle einen beliebigen Knoten als Startgraph T.
Solange T noch nicht alle Knoten enthält:

- Wähle eine Kante e minimalen Gewichts aus, die einen noch nicht in T enthaltenen Knoten v mit T verbindet.
- Füge e und v dem Graphen T hinzu.

Für eine effiziente Implementierung des Algorithmus von Prim muss man möglichst einfach eine Kante finden, die man dem entstehenden Baum T hinzufügen kann. Dafür **soll** eine einfache Priority-Queue **oder** ein Fibonacci-Heap benutzt werden.
In der Priority Queue sind alle Knoten gespeichert, die noch nicht zu T gehören. Alle Knoten haben einen Wert, der dem der leichtesten Kante entspricht, durch die der Knoten mit T verbunden werden kann. Existiert keine solche Kante, wird dem Knoten der Wert ω zugewiesen. Die Warteschlange liefert nun immer einen Knoten mit dem kleinsten Wert zurück. Die Effizienz des Algorithmus hängt infolgedessen von der Implementierung der Warteschlange ab.
Bei Verwendung eines Fibonacci-Heaps ergibt sich eine optimale Laufzeit.

Es sollen Algorithmen für minimale Spannbäume so implementiert werden, dass der minimale Spannbaum visualisiert und die Kantengewichtssumme des minimalen Spannbaums berechnet werden kann.