Praktikumsaufgaben

Graphentheoretische Konzepte und Algorithmen

WiSe 20

7. Oktober 2020

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

Erfolgreiche Bearbeitung der Aufgaben

Das Ergebnis der Bearbeitung einer Aufgabe wird von jeder Gruppe im Praktikum vorgestellt.
Das heißt, die Aufgabe muss *vor* dem Praktikumstermin fertig bearbeitet sein und die Lösungsdokumentation muss **spätestens um 8:00 des Praktikumstags in MS-Teams** hochgeladen sein.
Über die Vorstellung hinaus wird für jede Aufgabe das Folgende erwartet:

1. die Implementierung der gestellten Aufgabe, also
   - eine korrekte und möglichst effiziente Implementierung in Java, die der vorgegebenen 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. die Lösungsdokumentation (schriftliche Erläuterung Ihrer Lösung) *etwa 4-7 Seiten* mit
   - den folgenden Daten im Kopf Ihrer Lösungsdokumentation:
     - **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
   - der kurzer Beschreibung der Algorithmen und Datenstrukturen,
   - den wesentlichen Entwurfsentscheidungen Ihrer Implementierung,
   - der Umsetzung möglicher Aspekte der Implementierung,
   - der umfassenden Dokumentation der Testfälle, wobei die Abdeckung der Testfälle diskutiert werden soll und
   - der 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 bis um 8:00 des Praktikumstags in MS-Teams hochgeladen haben,
2. Sie im Praktikum Ihre Implementierung vorgestellt haben und diese von mir (mindestens) als ausreichend anerkannt wurde.

Qualität der Implementierung

Die Qualität der Implementierung spielt eine wesentliche Rolle, es genügt nicht, einen Algorithmus irgendwie zu implementieren. Es wird eine professionelle Lösung erwartet.
Beim Programmieren sollen als Ergänzung (wenn nicht ohnehin schon Bestandteil der PR-Vorlesung) zum in PR1 und PR2 Gelernten die Tipps von Joshua Bloch in 'Effective Java Third Edition Best practices for' befolgt werden, insbesondere:

Item 9: Prefer 'try-with-resources' to 'try-finally'

Item 11: Always override 'hashCode' when you override 'equals'

Item 26: Don't use raw types

Kapitel 7 komplett; Lambdas und Streams benutzen; übersichtlicher

Item 49: Check Parameters for validity
Wichtig auch für die Doku: Sinnvolle Preconditions überlegen und testen

Item 54: Return empty collections or arrays, not nulls
Wenn kein Ergebnis gefunden wird, ist die Liste eben leer

Item 54: Return optionals judiciously

Item 57: Minimize the scope of local variables

Item 58: Prefer for-each loops to traditional for loops

Item 59: Know and use the libraries
Insbesondere RegExp

Item 60: Avoid 'float' and 'double' if exact answers are required
z.B. Kantengewichte

Item 61: Prefer primitive types to boxed primitives (Wrapperklassen)
Wir bauen Algorithmen; da ist Schnelligkeit gefragt

1 Visualisierung, Speicherung und Traversierung von Graphen

Die Graphen, mit denen Sie arbeiten, sollen in diesem Format (siehe auch VL-Folien) gespeichert und gelesen werden:

gerichtet

```
<name node1>[ -> <name node2>[(<edge name>)][: <edgeweight>]];
```

ungerichtet

```
<name node1>[ -- <name node2> [(<edge name>)][: <edgeweight>]];
```

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)
- das Implementieren eines Algorithmus zur Traversierung eines Graphen (Breadth-First Search (BFS)),
- dabei soll als Ergebnis der kürzeste Weg und die Anzahl der benötigten Kanten angegeben werden.
- einfache JUnit-Tests, die Ihre Methoden überprüfen und
- JUnit-Tests, die das Lesen, das speichern sowie den BFS-Algorithmus überprüfen unter Benutzung der gegeben *.gka*-Dateien
- die Beantwortung der folgenden Fragen:
  1. Was passiert, wenn Knotennamen mehrfach auftreten?
  2. Wie unterscheidet sich der BFS für gerichtete und ungerichtete Graphen?
  3. Wie können Sie testen, dass Ihre Implementierung auch für sehr große Graphen funktioniert?

Vorstellung und Abgabe der Lösungen	der Teams in	am
	GKAP/02 und GKAP/05	9.11.
	GKAP/03 und GKAP/06	16.11.
	GKAP/01 und GKAP/04	23.11.

*

2 Optimale Wege

In dieser Teilaufgabe wird dem Dijkstra-Algorithmus aus der VL ein (etwas) effizienterer gegenübergestellt und die beiden sollen experimentell untersucht werden. Dabei sollen größere Graphen (größer hinsichtlich Kanten- und Knotenanzahl) zum Vergleich genutzt werden.

Vergleich Dijkstra mit Floyd-Warshall

Für gerichtete Graphen mit beliebigen Kantenbewertungen liefert der Algorithmus von Floyd und Warshall kürzeste Wege zwischen allen Knotenpaaren, falls der Graph keine Kreise mit negativer Kantenbewertungssumme besitzt. Andernfalls findet er einen solchen Kreis.

Der Algorithmus arbeitet mit zwei $|V| \times |V|$-Matrizen, der Distanzmatrix $D = (d_{ij})$ und der Transitmatrix $T = (t_{ij})$. Am Ende des Algorithmus gibt d_{ij} die Länge des kürzesten Weges von v_i nach v_j an, und t_{ij} benennt den Knoten mit der höchsten Nummer auf diesem Weg (hierdurch läßt sich der Weg rekonstruieren).

Algorithmus

1. Setze $d_{ij} := \begin{cases} l_{ij} & \text{für } v_i v_j \in E \text{ und } i \neq j \\ 0 & \text{für } i = j \\ \infty & \text{sonst} \end{cases}$ und $t_{ij} := 0$
2. Für $j = 1, \ldots, |V|$:
   - Für $i = 1, \ldots, |V|;\ i \neq j$:
     - Für $k = 1, \ldots, |V|;\ k \neq j$:
       - Setze $d_{ik} := \min\{d_{ik}, d_{ij} + d_{jk}\}$.
       - Falls d_{ik} verändert wurde, setze $t_{ik} := j$.
     - Falls $d_{ii} < 0$ ist, bricht den Algorithmus vorzeitig ab.
       (Es wurde ein Kreis negativer Länge gefunden.)

Die Aufgabe umfasst:

1. die Implementierung des Dijkstra- und des Floyd-Warshall Algorithmus'
2. einfache JUnit-Tests, die die Methoden überprüfen und
3. JUnit-Tests, die die Algorithmen überprüfen:
   - Testen Sie für *graph*3 in graph3.gka dabei Floyd-Warshall gegen Dijkstra und geben Sie den kürzesten Weg, sowie die Anzahl der Zugriffe auf den Graphen an
   - eine allgemeine Konstruktion eines gerichteten Graphen für eine vorgegebene Anzahl von Knoten und Kanten mit beliebigen, aber unterschiedlichen, nicht-negativen Kantenbewertungen
   - Implementierung eines gerichteten Graphen *BIG* mit 100 Knoten und etwa 3500 Kanten. Beschreiben Sie die Konstruktion von *BIG* und weisen Sie für zwei nicht-triviale Knotenpaare die kürzesten Wege nach.
   - Lassen Sie bitte beide Algorithmen auf dem Graphen *BIG*, die kürzesten Wege berechnen und vergleichen diese.
4. die Beantwortung der folgenden Fragen:
   a) Bekommen Sie für einen Graphen immer den gleichen kürzesten Weg? Warum?
   b) Was passiert, wenn der Eingabegraph negative Kantengewichte hat?
   c) Wie allgemein ist Ihre Konstruktion von *BIG*, kann jeder beliebige, gerichtete Graph erzeugt werden?
   d) Wie testen Sie für *BIG*, ob Ihre Implementierung den kürzesten Weg gefunden hat?
   e) Wie müssten Sie Ihre Lösung erweitern, um die Menge der kürzesten Wege zu bekommen?
   f) Wie müssten Sie Ihre Lösung erweitern, damit die Suche nicht-deterministisch ist?

Vorstellung und Abgabe der Lösungen	GKAP/02 und GKAP/05	30.11.
	GKAP/03 und GKAP/06	7.12.
	GKAP/01 und GKAP/04	14.12.