Aufgabenblock 3

TECHNISCHE UNIVERSITÄT DARMSTADT

Softwarepraktikum, WS 17/18
Abgabetermin: 30.11.2017 23:59 Uhr
FlowGraph, ResidualGraph

Hinweis

Wir messen der Einhaltung der Grundregeln der wissenschaftlichen Ethik größten Wert bei. Mit der Abgabe einer Lösung bestätigen Sie, dass Sie/Ihre Gruppe der alleinige Autor/die alleinigen Autoren des gesamten Materials sind. Falls Ihnen die Verwendung von Fremdmaterial gestattet war, so müssen Sie dessen Quellen deutlich zitiert haben. Weiterführende Informationen finden Sie unter http://www.es.tu-darmstadt.de/lehre/plagiatshinweise.html.
Weiterhin dürfen **keine Klassen unterhalb** von org.sopra.internal.* referenziert werden, dies führt zu einer Bewertung mit **0 Punkten** für die betreffende Aufgabe. Jede Abgabe muss alle notwendigen Dateien zum Kompilieren enthalten. Das in der jeweiligen Teilaufgabe geforderte Schema der Bezeichnung der Dateien ist einzuhalten.
Zusätzlich soll in jeder Klasse das Interface org.sopra.api.exercises.ExerciseSubmission implementiert werden und die in der Dokumentation beschriebene Funktionalität besitzen.

Lerninhalte

- Kennenlernen der Strukturen eines `FlowGraph` und seiner Kanten durch Erstellen von generischen Klassen, die einen `FlowGraph` und seine Kanten repräsentieren.
- Umgang mit JUnit durch Festlegen der Testkriterien und Testen des `FlowGraph` üben.
- Kennenlernen der Strukturen eines `ResidualGraph` und seiner Kanten durch Erstellen von generischen Klassen, die einen `ResidualGraph` und seine Kanten repräsentieren.

Aufgabe 3.1 - FlowEdge (5 Punkte)

In dieser Aufgabe sollen Sie die generische Klasse `FlowEdgeImpl` erstellen, die Kanten eines `FlowGraph` repräsentiert. Speichern Sie die Klasse im Paket `solutions.exercise3` ab. Die Klasse soll das Interface `org.sopra.api.exercises.exercise3.FlowEdge` implementieren und die in den Javadocs beschriebene Funktionalität besitzen.
Sämtliche Attribute der Klasse sollen von außen **nicht** sichtbar sein. Um generische Knotentypen zu unterstützen, verwenden Sie den Typparameter `V`.
Die Klasse soll außerdem über einen öffentlichen Konstruktor verfügen, der als Parameter Start- und Endknoten sowie die Kapazität in der genannten Reihenfolge übergeben bekommt. Der Fluss einer Kante soll mit `0` initialisiert werden und wird mit der Methode `setFlow(int i)` aktualisiert.

Weiterführende Informationen: Generics

Aufgabe 3.2 - FlowGraph (7 Punkte)

In dieser Aufgabe sollen Sie die generische Klasse `FlowGraphImpl` erstellen, die den aus der Vorlesung bekannten `FlowGraph` repräsentiert. Speichern Sie die Klasse im Paket `solutions.exercise3` ab. Die Klasse soll das Interface `org.sopra.api.exercises.exercise3.FlowGraph` implementieren und die in den Javadocs beschriebene Funktionalität besitzen.
Um generische Knotentypen zu unterstützen, verwenden Sie den Typparameter `V`. Sämtliche Attribute der Klasse dürfen von außen **nicht** sichtbar sein.
Der Graph soll als Map-Datenstruktur repräsentiert werden, die folgende Eigenschaften hat:

- Zu einem Key der äußeren Map sollen in einer weiteren Map alle ausgehenden Kanten abgelegt werden. Der Endknoten bildet bei der geschachtelten Map den Key und die Kante des Typs `FlowEdge<V>` den zugehörigen Wert eines Elements.
- Die Map soll folgenden Typparameter aufweisen: `Map<V, Map<V, FlowEdge<V>>>`.
- Die Map soll einmalig im Konstruktor als `java.util.HashMap` initialisiert werden.
- Die Startknoten der Kanten des Graphen sollen als Keys der äußeren Map verwendet werden.
- Kanten des Typs `FlowEdgeImpl` (in Aufgabe 3.1 implementiert) sollen in der Methode `addEdge` beim Hinzufügen einer Kante erzeugt werden.
- Beachten Sie darüber hinaus die Angaben in den Javadocs.

Weiterführende Informationen: Generics, Collection-API, HashMap

Aufgabe 3.3 - Testen von FlowGraph (14 Punkte)

In dieser Aufgabe sollen Sie eine Implementierung der Klasse `FlowGraph<V>` auf ihre Funktionsfähigkeit testen. Erstellen Sie dazu die Klasse `FlowGraphTest`, die von `AbstractFlowGraphTest` erbt und das Interface `ExerciseSubmission` implementiert. Speichern Sie die Klasse im Paket `solutions.exercise3` ab.
Die Superklasse stellt die Knoten des in Abbildung 1 dargestellten Graphen im vererbten Attribut `nodes` sowie die Kanten im vererbten Attribut `edges` bereit. Die Knoten des Graphen entsprechen String-Objekten, die jeweils den Namen der Knoten entsprechen. Die zu testende Implementierung ist über das Attribut `sut` (*System Under Test*) des Typs `FlowGraph<String>` bereitgestellt und wird vor jedem Testfallaufruf neu initialisiert. Der Graph in `sut` enthält vor jedem Testdurchlauf weder Knoten noch Kanten.
Verwenden Sie die zur Verfügung gestellten Knoten und Kanten als Eingabedaten zur Implementierung folgender Testfälle:

a) Vervollständigen Sie den Testfall `test_addNode`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.addNode` korrekt ist. Prüfen Sie außerdem das Verhalten bei der Übergabe fehlerhafter Parameter.

b) Vervollständigen Sie den Testfall `test_addEdge`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.addEdge` korrekt ist. Prüfen Sie außerdem das Verhalten bei der Übergabe fehlerhafter Parameter.

c) Vervollständigen Sie den Testfall `test_edgesFrom`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.edgesFrom` korrekt ist. Verwenden Sie die zur Verfügung gestellten Testdaten, um den Graphen zu füllen und vergleichen Sie anschließend das Soll-Verhalten mit der tatsächlichen Ausgabe nach Aufruf von `sut.edgesFrom`. Prüfen Sie außerdem das Verhalten bei der Übergabe fehlerhafter Parameter.

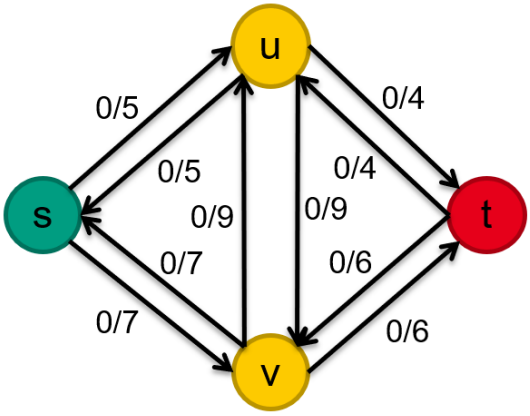

Abbildung 1: Durch die Klasse `AbstractFlowGraphTest` in den Attributen `nodes` und `edges` bereitgestellte Testdaten.

d) Vervollständigen Sie den Testfall `test_getEdges`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.getEdges` korrekt ist. Verwenden Sie die zur Verfügung gestellten Testdaten, um den Graphen zu füllen und vergleichen Sie anschließend das Soll-Verhalten mit der tatsächlichen Ausgabe nach Aufruf von `sut.getEdges`.

e) Vervollständigen Sie den Testfall `test_getEdge`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.getEdge` korrekt ist. Verwenden Sie die zur Verfügung gestellten Testdaten, um den Graphen zu füllen und vergleichen Sie anschließend das Soll-Verhalten mit der tatsächlichen Ausgabe nach Aufruf von `sut.getEdge`.

f) Vervollständigen Sie den Testfall `test_getNodes`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.getNodes` korrekt ist. Verwenden Sie die zur Verfügung gestellten Testdaten, um den Graphen zu füllen und vergleichen Sie anschließend das Soll-Verhalten mit der tatsächlichen Ausgabe nach Aufruf von `sut.getNodes`.

g) Vervollständigen Sie den Testfall `test_containsNode`. Dieser Testfall soll prüfen, ob das Verhalten der Methode `sut.containsNode` korrekt ist. Verwenden Sie die zur Verfügung gestellten Testdaten, um den Graphen zu füllen und vergleichen Sie anschließend das Soll-Verhalten mit der tatsächlichen Ausgabe nach Aufruf von `sut.containsNode`. Prüfen Sie außerdem das Verhalten bei der Übergabe fehlerhafter Parameter.

Weiterführende Informationen: JUnit

Aufgabe 3.4 - ResidualEdge (5 Punkte)

In dieser Aufgabe sollen Sie die generische Klasse `ResidualEdgeImpl` erstellen, die Kanten eines `ResidualGraph` repräsentiert. Speichern Sie die Klasse im Paket `solutions.exercise3` ab.
Die Klasse soll das Interface `org.sopra.api.exercises.exercise3.ResidualEdge` implementieren und die in den Javadocs beschriebene Funktionalität besitzen.
Verwenden Sie den Typparameter `V`, um generische Knotentypen zu unterstützen. Sämtliche Attribute der Klasse dürfen von außen **nicht** sichtbar sein.

Implementieren Sie einen Konstruktor, der folgende Parameter in der genannten Reihenfolge übergeben bekommt und jeweils einem Objektattribut zuweist:

- Den Startknoten `startNode` des generischen Typs `V`.

- Den Endknoten `endNode` des generischen Typs `V`.
- Die Kantenkapazität `capacity` des primitiven Datentyps `int`.

Weiterführende Informationen: Generics

Aufgabe 3.5 - ResidualGraph (9 Punkte)

In dieser Aufgabe sollen Sie die generische Klasse `ResidualGraphImpl` erstellen, die den in der Frontalveranstaltung vorgestellten `ResidualGraph` repräsentiert. Speichern Sie die Klasse im Paket `solutions.exercise3` ab.
Die Klasse soll das Interface `org.sopra.api.exercises.exercise3.ResidualGraph` implementieren und die in den Javadocs beschriebene Funktionalität besitzen.
Verwenden Sie den Typparameter `V`, um generische Knotentypen zu unterstützen. Sämtliche Attribute der Klasse dürfen von außen **nicht** sichtbar sein.

Der Graph soll als Map-Datenstruktur repräsentiert werden, die folgende Eigenschaften hat:

- Die Map soll folgenden Typparameter aufweisen: `Map<V, List<ResidualEdge<V>>>`.
- Die Keys der Map sollen die Startknoten der im Graph enthaltenen Kanten sein.
- Zu jedem Startknoten soll in der Map eine Liste der ausgehenden Kanten des Typs `ResidualEdge<V>` als zugehöriger Wert abgelegt werden.
- Die Map soll einmalig im Konstruktor als `java.util.HashMap` initialisiert werden.

Implementieren Sie außerdem einen Konstruktor, der einen Parameter `flowGraph` des Typs `org.sopra.api.exercises.exercise3.FlowGraph<V>` übergeben bekommt. Im Konstruktor soll aus dem übergebenen FlowGraph der ResidualGraph folgendermaßen erzeugt werden:

- Alle in `flowGraph` enthaltenen Knoten sollen ebenfalls im ResidualGraph auftreten.
- Da eine ungerichtete Kante mit Kapazität $c(e_{ij})$ im FlowGraph als zwei gerichtete gegenläufige Kanten, die jeweils Kapazität $c(e^r_{ij}) = c(e^r_{ji}) = c(e_{ij})$ besitzen, repräsentiert werden, ist Folgendes zu beachten:

  Zu einer Kante e_{ij} mit Kapazität $c(e_{ij})$ und dem Fluss $f(e_{ij})$ im FlowGraph existiert eine gegenläufige Kante e_{ji} mit Kapazität $c(e_{ji})$ und Fluss $f(e_{ji})$. Die Kapazität der Kante e^r_{ij} im Residual-Graph soll $c(e_{ij}) - f(e_{ij})$ sein. Die Kapazität der Kante e^r_{ji} soll entsprechend $c(e_{ji}) - f(e_{ji})$ sein.

  Beachten Sie außerdem, dass für zwei gegenläufige Kanten aus dem FlowGraph im ResidualGraph nur insgesamt zwei Kanten erzeugt werden dürfen.
- **Hinweis:** Für die Flussgraphen gilt immer, dass beide gegenläufigen Kanten die gleiche Kapazität besitzen und nur eine von ihnen einen Fluss größer Null. Unter diesen Voraussetzungen können alle Informationen des Flussgraphen in zwei Werte der gegenläufigen Kanten im Residualgraphen übertragen werden.

Weiterführende Informationen: Generics, Collection-API, ResidualGraph mit ungerichteten Kanten

Kritik, Verbesserungsvorschläge und Bug-Report

Sollten Sie Kritik oder Verbesserungsvorschläge haben bzw. Bugs finden, dann nutzen Sie dafür bitte den Bug-Report Button im Moodle-Kurs.