Dokumentation

SKIZZE ZUR AUFGABE 3 AUS DER VORLESUNGSREIHE GKA

TEAM: ANTON, & MESUT

Aufgabenaufteilung

Grundsätzlich wurden alle Entwicklungsvorgänge und Aufgaben zusammen erarbeitet. Trotz dessen haben wir uns entschieden, die Aufgaben aufzuteilen, sodass wir schneller vorankommen.

- Mesut: Run.java, GraphReader.java, GraphSaver.java & GraphBuilder.java
- Anton: BreadFirstSearch.java und alle zugehörigen Tests

Quellenangaben

GraphStream Dokumentation: https://github.com/graphstream

Verschiedene Lösungswege von stackoverflow.com

Lehrveranstaltung

Bearbeitungszeitraum

Datum	**Dauer in Stunden**
Anton	20
Mesut	20
Gesamt	**40**

Aktueller Stand

Unsere Software funktioniert vollständig und die Tests durchlaufen alle eine positive Instanz.

Funktionen unserer Software

Folgende Funktionen haben wir aus dem Aufgabenblatt herausgelesen, sodass wir Sie ebenfalls auch implementiert haben:

- Es können .gka-Dateien eingelesen werden
  - Es können ebenfalls .gka-Dateien gespeichert werden
- Es kann mithilfe des Ford Fulkerson und Edmond Algorithmen den den maximalen Flow von einem Vertex zu einem anderen angegebenen Vertex finden.
- Vollständige Tests
- Eine vollständige Javadoc.

Algorithmus

Ford Fulkerson:

Der Algorithmus wurde mit Hilfe der Folien von Frau Padberg und Herrn Klauck umgesetzt.
Wir starten bei unserem Startknoten und arbeiten und bis zu der Senke durch.
Bei der Implementierung haben wir darauf geachtet, dass wir zwischen Ford und Edmond unterscheiden können und dadurch den maximalen Fluss mit dem gewünschten Algorithmus berechnen können.
Beschreibung des Algorithmus:
Der Algorithmus startet bei unserem Anfangsknoten(Start) und hat die Senke(Endknoten) als Ziel.
Jede Kante zwischen den vorhandenen Knoten hat zwei Attribute: die Kapazität und den Fluss.
Die Kapazität, wie der Name schon sagt gibt an wieviel „Einheiten“ durch eine Kante transportiert werden können. Der Flow(Fluss) ist der tatsächliche Transport von „Einheiten“.
Das Ziel ist es einen optimalen Fluss zu erhalten von Start zur Senke und dabei soll der Fluss maximal wie möglich sein.
Zu beachten ist, dass man nicht mehr transportieren kann, als die Kapazität erlaubt. Sollte eine Kapazität ausgelastet sein, so muss man sich einen anderen Weg suchen um den Fluss zu optimieren.
Der Algorithmus terminiert sobald der maximale Fluss erreicht wurde. Dabei gilt zu beachten, dass die Ausgangsflussstärke (am Start) gleich der Eingangsflussstärke (an der Senke) ist.

Edmond und Karp:

Der Edmond und Karp Algorithmus arbeitet im Prinzip genau wie der Ford Fulkerson mit dem Unterschied, dass wir für den kürzesten vergrößerten Weg eine Queue benutzen statt einen beliebigen vergrößerten Weg.

JUnit-Testfälle

Unsere Testfälle beinhalten positive sowie negative Testfälle. Folgende Situationen wurden getestet:

- Einlesen der graph04.gka Datei und Test.
- Alle unerfüllten Bedingungen die für ein Netzwerk gelten.
- Einen selbsterstellten Graphen mit 9 Kanten.
- Einen Graphen aus dem Internet mit 5 Knoten und 4 Kanten.
- Ein Testfall mit 50 Knoten und 800 Kanten.
- Ein Testfall mit (100*800) Knoten und ca. 2Mio. Kanten. (OutOfMemory Exception)

Fragebogen

1. **Welcher Algorithmus/welche Implementierung ist schneller? Wie schnell für die Netzwerke Ihrer Praktikumsgruppe?**
   Der Edmond und Karp Algorithmus ist schneller. Für die Praktikumsgruppen haben wir bis jetzt keine Messergebnisse.
2. **Was haben Sie unternommen, um eine bessere Laufzeit zu erreichen?**
   Wir speichern alle Knoten des Graphen in eine Liste ab, damit wir nicht immer über den Graphen iterieren müssen und benutzen keine zusätzlichen Datenstrukturen, sondern Attribute.
3. **Lasst sich die Laufzeit Ihrer Implementierung durch andere Datenstrukturen verbessern?**
   Nicht durch andere Datenstrukturen, aber durch das Residualnetzwerk können wir(wahrscheinlich) effizientere Laufzweit erreichen.
4. **Was passiert, wenn Sie nicht-ganzzahlige Kantengewichte wählen?**
   Dieser Fall wird von unserem Algorithmus abgefangen und als ungültiger Graph bewertet (Exception Meldung. -> ungültiger Graph).
5. **Was passiert bei negativen Kantengewichten?**
   Dieser Fall wird von unserem Algorithmus abgefangen und als ungültiger Graph bewertet (Exception Meldung. -> ungültiger Graph).