**Team**: <Teamnummer?>, Marcel Schlegel, Florian Bauer

**Aufgabenaufteilung**:

1. Aufgaben: GUI, Floyd-Warshall
   Dateien: siehe JavaDoc, bzw. https://github.com/flbaue/gka-wise14

2. Aufgaben: Speichern & Laden, Dijkstra, BFS Iterator
   Dateien: siehe JavaDoc, bzw. https://github.com/flbaue/gka-wise14

**Quellenangaben**: Es wurde kein Quellcode übernommen.

**Bearbeitungszeitraum**: Erster GitHub Commit Montag 20.10.14, aktuell letzter GitHub Commit Montag, 27.10.14 (https://github.com/flbaue/gka-wise14/commits/master)

**Aktueller Stand**: Es fehlen noch einige Unit-Tests.

Algorithmus:

Allgemein: Wir haben Konten als Vertex Klasse Implementiert, wobei Vertex Instanzen durch den gegebenen Namen eindeutig sind. Jeder Vertex kann ein Marker Objekt enthalten. Marker Objekte dienen dazu während der Verarbeitung durch Algorithmen, Informationen wie Distanz, Vorgänger, Fertig(Visited), etc. zu speichern. Zu beginn eines Algorithmus werden daher alle Marker der im Graph enthaltenen Knoten entfernt. Am Ende kann man so von jedem Knoten die entsprechenden Informationen abfragen.

BFS: Wir haben den BFS Algorithmus in zwei Varianten implementiert.
Variante 1 nutzt ein Java Iterable/Iterator Interface um einen Iterator bereit zu stellen, welcher mit .hasNext und .next über den Graphen iteriert. Dabei wird mit einem gegebenen Startpunkt begonnen, dann alle direkten Nachbarn dieses Knoten und dann der Reihe nach Alle jeweiligen Nachbarn der zuvor gesammelten Nachbarn. Der jeweils betrachtete Knoten wird von der .next Methode zurückgegeben und kann so weiter betrachtet werden.

Dijkstra: Es werden zunächst alle Knoten vorbereitet, das heißt sie bekommen einen Marker mit „null“ als Vorgänger sowie Integer.MAX_VALUE als Distanz. Der Startknoten erhält sich selbst als Vorgänger sowie die Distanz 0. Anschließend wird in einer Schleife stets der Knoten mit der Geringsten Distanz ermittelt und die Distanzwerte aller seiner nicht fertigen Nachbarn überprüft, und ggf. aktualisiert. Dabei wird auch der Vorgänger entsprechend gesetzt. Kann kein Unbefugter Die Schleife beginnt nun von Vorne. Kann kein nicht fertiger Knoten mehr ermittelt werden, ist der Algorithmus fertig. Detaillierte Ergebnisse, Distanz und Vorgänger, können dann direkt über die Knoten abgefragt werden.

Floyd-Warshal:…

Datenstrukturen & Implementierung

Graph: Wir verwenden für Graphen die Klassen DirectedWeightedPseudograph sowie WeightedPseudograph des JGraphT Frameworks in Kombination mit DefaultWeightedEdge als Kante und der selbst implementierten Vertex Klasse für Knoten.
Der Grund warum wir uns für den Pseudograph entschieden haben ist, da dieser die größte

mögliche Freiheit bietet. So sind bei anderen Graph-Klassen Dinge wie doppelte Kanten oder Loops nicht erlaubt.

GraphLoader: Hier wird nur eine einfache Java Hash Map vorgehalten. Sie dient der Speicherung bereits erzeugter Vertex Instanzen und deren einfacher Wiederverwendung unter Zuhilfenahme des String Namens.
Zwischen den Methoden werden Listen von FileEntry Objekten weitergereicht. FileEntry stellt eine Zeile der eingelesenen Datei dar. Die Zeile wird beim erzeugen von FileEntry analysiert und validiert. Dabei werden alle relevanten Informationen extrahiert und sind über die Getter-Methoden des FileEntry Objekts abrufbar. Schlägt die validieren einer Zeile fehl, wird diese Zeile ignoriert.

GraphSaver: Hier werden alle Kanten des Graphen als EdgeSet genommen und jede Kannte in ein FileEntry Objekt überführt. Anschließend werden noch alle Knoten ohne Kanten ermittelt und diese ebenfalls in FileEntry Objekte überführt.
Anschließend werden die String Repräsentationen der FileEntry Objekte zeilenweise in die gegebene Datei geschrieben.

Tests

GraphIO
testReadGraphFromFile: Es wird ein einfacher Testgraph in Form einer Textdatei gelesen und geprüft ob die Menge der der Erzeugten Knoten und Kanten mit den Erwartungen übereinstimmt.
testSaveGraphAsFile: Es wird ein programmatisch erzeugter Graph in eine Datei gespeichert und anschließend geprüft ob diese Datei erwartete Zeilen enthält.
testGraphIOWithGraphGka6: Es wird der gegebene beispiel Graph graph6.gka eingelesen und die Anzahl der erzeugten kanten und Konnten mit den Erwartungen verglichen
testGraphIOWithGraphGka2: Es wird der gegebene beispiel Graph graph2.gka eingelesen und die Anzahl der erzeugten kanten und Konnten mit den Erwartungen verglichen