Team: 4_2, Marcel Schlegel, Florian Bauer

Aufgabenaufteilung:

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

ii. 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: Aufgabe 1 ist Fertig. Bei Aufgabe 2 Fehlen noch 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: Es werden zunächst mögliche Marker aller Knoten entfernt und lediglich der Startknoten bekommt einen Marker mit der Distanz 0 und sich selber als Vorgänger gesetzt. Anschließend wird der Startknoten als erster Knoten in die Queue aufgenommen. Damit ist der Iterator fertig initialisiert und der Nutzer kann beginnen ihn zu benutzen. Dazu wird zunächst die Methode .hasNext aufgerufen. Diese Methode prüft anhand der Queue, ob noch Knoten verfügbar sind und gibt dies zurück. Danach kann mit .next zum nächsten Knoten gesprungen werden. .next holt sich dazu den nächsten Knoten aus der Queue, setzt seinen Marker auf visited, sucht alle noch zu bearbeitenden Nachbarn, und gibt den Koten an den Aufrufer aus. Nachbarknoten werden gefunden in dem über alle direkt verbundenen Knoten iteriert wird und alle Knoten, die noch keinen Marker haben, diesen erhalten und in die Queue eingefügt 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:
Zunächst wird eine Distanzmatrix bzw. eine Distanzmap gebaut. Hierfür werden für alle Knoten des Graphen genau deren Nachbarknoten mit der jeweiligen Kantengewichtung gespeichert. Hierbei wird ein Knotenpaar als Key und die Kantengewichtung als Value in einer Map gespeichert.
Als nächstes werden für alle Knoten des Graphen deren Quell- und Zielknoten bestimmt. Existiert für einen Knoten im Graphen deren Quell- und Zielknoten in der Distanzmatrix (direkter Weg zwischen 2 Knoten) wird diese Kantengewichtung gespeichert bzw. wenn nicht wird Integer.MAX_VALUE gespeichert (in A).
Danach wird der Weg vom Quellknoten zum aktuellen Knoten mit dem Weg vom aktuellen Knoten zu dessen Zielknoten addiert und gespeichert (in B). Der kürzere Weg (entweder A oder B) wird anschließend in eine Transitmatrix bzw. Transitmap gespeichert.
Jetzt kann durch Berechnung der Transitiven hülle mithilfe der Distanz- und Transitmap der kürzeste Weg von einem Start- zu einem Endknoten gefunden werden.

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

BFS
testIterator: Es wird programmatisch ein Graph erstellt und anschließend mittels BFS Iterator mit den Methoden .hasNext und .next iteriert. Dabei wird bei jedem Schritt die Rückgabewerte der Methoden geprüft. Es wird anschließend geprüft ob .hasNext zum Ende False zurück gibt. Zum Schluss wird geprüft ob alle Knoten die erwarteten Vorgänger in Ihren Markern gesetzt bekommen haben.