Team: Gruppe 1, Team 1; Alex Mantel, Daniel Hofmeister

Aufgabenaufteilung:

1. Alex Mantel
   Implementierung der GUI und des BFS sowie Architektur des Projektes.

2. Daniel Hofmeister,
   Implementierung des File load, Graph Generation, File save.

Quellenangaben: Von Kommilitone Simon Kosch haben wir den RegEx übernommen.

Bearbeitungszeitraum:	Alex Mantel	70 Stunden
	Daniel Hofmeister	70 Stunden

Aktueller Stand: Alles fertig implementiert.

Dokumentation:

Algorithmus:

Breadth First:

Der Algorithmus macht eine vollstaendige Abdeckung der schwachen Komponente. Wir erstellen zunächst eine Map<String, Integer> welche die Knoten und deren Bewertung darstellt. In dieser werden landen alle Knoten der schwachen Komponente. Aus dieser Abbildung lassen lässt sich der kürzeste Weg ermitteln, falls im Keyset die target enthalten ist. Im anderen Fall gibt es keinen Weg zwischen Source ↔ Target.

Datenstrukturen:

Es wurden keine eigenen Datenstrukturen angelegt.

Implementierung:

Wir haben uns an der MVC (Model-View-Control) Architektur orientiert um unser Projekt erstellen.
Die Hauptaspekte der grundlegenden Implentierung, das Laden und Speichern der Dateien, sowie das erstellen der Graphen aus den geladenen Dateien, wurden in zwei seperaten Dateien behandelt, dem FileHandler und GraphHandler.
Der FileHandler gibt durch das Laden der Datei eine ArrayList<String> zurück, welche die Informationen des Graphen enthält. Die einzelnen Tupel werden dabei im Format: „Knoten1+ (- - / - >)* + Knoten2* + (:)* + Gewicht“ ausgegeben.
Der GraphHandler nimmt nun dieses Format und parst die wichtigen Informationen heraus um die Graphen zu stellen. Der fertige Graph wird dann über den Maincontroller an die View gegeben um visualisiert zu werden.
Genau andersrum funktioniert das Prinzip beim speichern, wo der GraphHandler die relevanten Informationen aus dem Graph liest und der Filehandler diesen dann in einer Datei speichert.

Tests:

Hier haben wir rudimentäre Tests durchgeführt um das korrekte Laden und Speichern von den verschiedenen Graphen sicher zu stellen. Weiterhin haben auch das Erstellen der Graphen und die Breitensuche auf ihre korrekte Funktionalität überprüft.

Beantwortung der Fragen:

Was passiert, wenn Knotennamen mehrfach auftreten?

Mehrfache Knotennamen werden ignoriert. Jeder Knoten muss einen einzigartigen (unique) Namen besitzen um ihn als Quell- oder Zielknoten von Kanten bestimmen zu können. Gäbe es zwei Knoten mit dem Namen "Node1" und einen Knoten "Node2" und man hätte eine Kante "Node1 -> Node2" so würde nicht klar sein, von welchem der zwei "Node1"-Knoten diese Kante nun ausgeht. Auch eine Kante "Node1—Node1" zum Beispiel könnte in diesem Fall ein Loop in einem der beiden sein, oder eine Verbindung zwischen den beiden.

Wie unterscheidet sich der BFS für gerichtete und ungerichtete Graphen?

Der Algorithmus wird auf seiden beider gleich aufgerufen. Man erstellt eine neue Instanz der Algorithmusimplementierung und uebergibt diesen im Konstruktor den Graphen.
Die konzeptuellen Unterschiede wurden in Hilfsmethoden behandelt. Beispielsweise wenn man alle adjazenten, erreichbaren Knoten müssen wir zwischen gerichteten und ungerichteten unterscheiden. Bei einem ungerichteten Graphen erhalten wir alle adjazenten Knoten als „erreichbar“ wobei bei dem gerichteten Graphen wir nur von den ausgehenden Kanten, deren Ziele als „erreichbar“ einstufen können.
Wir haben uns an den Implementationen anderer orientiert und eine Map aufgestellt, welche von String auf Integer abbildet. Die Strings stellen Knoten und die dazugehörigen Integer deren Bewertung dar.
Bei der Ermittelung des kürzesten Weges muss zusätzlich bei den adjazenten Knoten unterschieden werden zwischen „erreichbar von“ und „nicht erreichbar von“. Damit ist gemeint, dass man beim Rückweg vom Ziel zum Startpunkt zwischen bei den Kanten geht. Bei einem ungerichteten ist dies wieder egal, da man immer vom Ziel ↔ Start kommt. Bei einem gerichteten müssen wir von den eingehenden Kanten den Startknoten nehmen.

Wie können Sie testen, dass Ihre Implementation auch für sehr große Graphen richtig ist?

Es ist prinzipiell nicht möglich zu beweisen oder sicher zu stellen, dass eine Implentation für beliebig Große Graphen korrekt ist, da man früher oder später an Limits des Speichers oder des Prozessors stößt. Hat man bereits einen Algorithmus implementiert der sicher funktioniert, so kann man durch Vergleich der beiden Algorithmen seine eigene Implementation testen. Hierfür erstellt man zufällige Graphen, lässt beide Algorithmen über Sie laufen und vergleicht anschließend die Ergebnisse.