Team: <3_4>, <Patrick Steinhauer, Jan Dennis Bartels>

Aufgabenaufteilung:

Die Aufgabe wurde weitestgehend von beiden Team Mitgliedern bearbeitet, bis auf ein paar Kleinigkeiten jeweils.

Quellenangaben:

https://github.com/jgrapht/jgrapht/blob/master/jgrapht-core/src/main/java/org/jgrapht/graph/DefaultWeightedEdge.java

http://www.vogella.com/tutorials/JavaRegularExpressions/article.html

http://openbook.galileocomputing.de/javainsel/javainsel_15_004.html#dodtpac233475-9da3-4cec-9eab-d68a36830773

http://jgraph.github.io/mxgraph/docs/js-api/files/layout/mxGraphLayout-js.html

Code Übernahme von dem Studenten Simon Kosch um den Graph in einer Kreisform anzugeben.

Bearbeitungszeitraum: Bearbeitungszeitraum war von letzte Woche Donnerstag, bis heute.

Bearbeitungszeit liegt hier bei ca. 50 Stunden

Aktueller Stand:

Der aktuelle Stand ist, dass der Graph eingelesen und visualisiert werden kann. Angefangen wurde mit dem BFS, nur die Tests wurden noch nicht bearbeitet.

Beantwortung der Fragen:

1. Wenn Knotennamen mehrfach auftreten, taucht dieser nur einmal im Graphen auf. Hierbei werden dann nur die Kanten hinzugefügt, die von dem Knoten ausgehen oder auf ihn zeigen.
2. Bei dem BFS ist der Unterschied, dass bei gerichteten Graphen nur eine Richtung gibt, die abgesucht werden muss. Bei den ungerichteten Graphen ist dies etwas problematischer, da es keine Richtung gibt müsste man in beide Richtungen schauen, oder Knoten die vorher waren nicht mehr betrachten.
3. Dies können wir testen, indem wir einen sehr Großen Graphen zum testen verwenden und dort unseren Algorithmus verwenden. Möglicherweise kann man die Graphen in kleinere Graphen zerteilen und dann so versuchen herauszufinden ob der weg der kürzeste ist.

Dokumentation:

Graphen aus Textdatei auslesen:

Der Graph wird mithilfe des bufferedreaders ausgelesen. Hierbei wird dem bufferedreader in einem Filereader eine Datei gegeben (ganzer DateiPfad). Danach speichern wir mithilfe des readline Befehls, die Zeilen, die ausgelesen werden sollen und speichern diese in eine Variable herein.

Graph von Text Datei in Graphen umwandeln:

Hier verwenden wir den matcher , um mithilfe der Regulären Ausdrücke zu prüfen, ob dieser mit etwas aus einem bestimmten String Matched.

Dann werden zwei Graphen erstellt entweder ein ungerichteter oder ein gerichteter, je nachdem, was für Zeichen übergeben werden.

Danach wird überprüft, ob der matcher weiterhin was findet. Falls dies der Fall ist werden die Knoten dem Graphen hinzugefügt.

Visualisierung von Graphen:

Für die Visualisierung:

Wir haben bei der Visualisierung den JGraphModelAdapter benutzt. Weiterhin benutzen wir den PseudoGraphen (wurde uns von Commolitonen empfohlen), denn wir haben vorher den Listenable Graphen benutzt. Um die Knoten in einer Kreisform anzugeben hat uns ein Kommmolitone uns seinen Code zur Verfügung gestellt, den wir hierfür verwenden.

Das Positionieren der Graphen kommt von dem Beispiel auf der JGraphT Seite.

BFS Algorithmus:

Unseren BFS Algorithmus haben wir implementiert, indem wir eine queue und eine Map ersteinmal verwenden.

Hierbei wird am Anfang das Startelement in die queue getan. Danach fragen wir innerhalb einer while Schleife, solange die Queue nicht leer ist führe den weiteren Code aus.

Hier tun wir als erstes das oberste Element der queue in eine Variable getan.

Hier wird als erstes geprüft, ob der Knoten schon das gesuchte Element ist.

Wenn dies der Fall ist wird die Anzahl direkt zurückgegeben.

Für die Zielknoten wird dann geprüft, ob dieser noch nicht in unserer Map gespeichert ist.

Wenn es so ist, fügen wir den Knoten der Map hinzu.

Ebenfalls wird der Targetknoten auch in die queue getan. Dies wird alles wiederholt, bis der krzeste Weg gefunden wird.

Die map dient hierbei für die Initialisierung der Knoten mit den einzelnen Indexen.