# Praktikum 2

Abgabe von : Moritz Spindelhirn

## Quellenangaben:

Es wurden keine Fremdquellen verwendet.

Für das Verständiniss des Floy-Warshall Algorythmus wurden folgende Quellen genutzt: https://www.youtube.com/watch?v=9QV6QpyhN0o ( User: Romesh Malinga Perera )

## Bearbeitungszeitraum:

- 5 Stunden ( Refactoring )
- 4 Stunden ( Dijkstra )
- 5 Stunden ( FloydWarshall )
- 1 Stunde ( Dokumentation )
- => Zum Praktika 15 Stunden
- 1 Stunde Nachbearbeitung Dijkstra
- 2 Stunden Nachbearbeitung Floyd-Warshall
- 3 Stunden Nachbearbeitung Graph Generierung
- 3 Stunden Nachbearbeitung Erstellung Tests
- => Nachbearbeitung 9 Stunden
- => Total 24 Stunden

## Dijkstra

Nach der Refaktorierung ist es das die GraphFactory verschiedene Knotenklassen verwenden kann. Für Dijkstra wurde die Knotenklasse `DijkstraVertex` erstellt, welche die Zusatzinformation Distanz, Vorgänger und Status der Beendung beinhaltet.

### Ablauf

In der Initalisierung wird jeder Knoten mit der Siatanz `Integer.MAX_VALUE` gefüllt. Der Startknoten bekommt die Distanz `0`.
Der Startknoten wird in eine Bearbeitungsliste eingefügt. Danach wird so lange der erste Knoten aus der Bearbeitungsknoten genommen und als aktueller Knoten betrachtet. Neue Nachbarknoten werden sortiert in die Bearbeitungsliste eingefügt.
Wenn der Zielknoten fertig als final markiert wurde beendet sich der Algorithmus und liefert den Weg zum Ziel.

Die Länge des Pfades kann man aus dem letzten Vertex - dem Ziel - auslesen.

# Floyd Warshall

In der Initalisierungsphase wird die Distanzmatrix aufgebaut.
Als erster Schritt legen wir dazu eine Übersetzungsmap an. Damit übersetzen wir die Knotennamen in Nummern. Dies erleichtert die Erstellung und Bearbeitung der Distanzmatrix.
Nach der Initalisierung iterieren wir über die Distanzmatrix und aktualiseren die Werte der Distanzen.
Am Ende steht in der Matrix somit für jede mögliche Knotenverbindung die kürzeste Distanz.

Neben des Ausgabe des Pfades steht hier die länge des Angefraten Weges über die Methode `getLastDistance()`

# Graph Generierung

Die abstrake, generische Klasse `GraphGenerator<V, E>` kann zur Generierung großer zufälliger Graphen genutzt werden.
Um einen Graph generieren zu können müssen die abstraken Methoden `createGraph`, `createVertex`, `createEdge` und `createEdgeWeight` überschrieben werden. Somit ist eine möglichst große Flexibilität möglich.
Im Test `DijsktaFloydWarshallTest` findet sich eine Implementierung für gewichtete, gerichtete Graphen mit einem zufälligen Kantengewicht.
Bei jedem Aufruf von `generate` wird ein euer Graph erzeugt.

# Tests

Neben Tests für Dijkstra und Flyod-Warshall gibt es einen Test ( `DijsktaFloydWarshallTest` ) der beide Algorithemen gegeneinader laufen lässt und kontrolliert ob beide die gleiche Angabe für den kürzesten Weg haben. Dieser Test gibt zusätzlich auch einen Vergleich der Laufzeiten an.

Beispielausgabe:

```
Dijkstra: 22,000000 | Floyd-Warshall: 22,000000
Dijkstra: 42,000000 | Floyd-Warshall: 42,000000
Different ways!
Floyd-Warshall: [v1, v51, v42, v243, v90]
Dijkstra: [v1, v286, v267, v194, v90]
Dijkstra: 32,000000 | Floyd-Warshall: 32,000000
Dijkstra: 22,000000 | Floyd-Warshall: 22,000000
Dijkstra: 32,000000 | Floyd-Warshall: 32,000000
Dijkstra: 32,000000 | Floyd-Warshall: 32,000000
Dijkstra: 20,000000 | Floyd-Warshall: 20,000000

Dijkstra time for 50 graphs: 231ms ( AVG: 4ms )
Floyd-Warshall time for 50 graphs: 231ms ( AVG: 4ms )
```

Bei dem Test hat der Graph 300 Knoten und 2000 Kanten mit einem Katengewicht von 10 oder 12. Da das Kantengewicht also immer recht nah aneinander liegt finden die beiden Algorithmen regeläßig unterschieldiche Wege gleicher Länge.