Dokumentation ALG UE4 -advanced 3

# 1. Installations- und Bedienungsanleitung

Im entpackten Archiv finden Sie eine klassische Visual Studio 2015 Solution Verzeichnisstruktur. Eine ausführbare .exe Datei ist unter Fahrplanauskunft\Debug mit Testdaten zu finden. Doppelklick auf die exe-Datei startet das Programm.

Wenn Sie das Programm über die Commandline öffnen, dann bitte so:   
Fahrplanauskunft.exe [Dateiname]  
Der Dateiname der Datei die den Graphen enthält wird nicht unbedingt verlangt, dieser kann auch später eingelesen werden.  
Jeder Menupunkt darf nur einmal ausgeführt werden, wenn Sie zum Beispiel den Dateinamen schon über die Commandline mitgeteilt haben, darf 1 nicht mehr gewählt werden, wenn Sie schon einmal gesucht haben und nocheinmal suchen wollen muss das Programm neu gestartet werden.

# 2. Beschreibung des Algorithmus

## 2.1 Dijkstra

Die Grundidee des Algorithmus ist es, immer derjenigen Kante zu folgen, die den kürzesten Streckenabschnitt vom Startknoten aus verspricht. Andere Kanten werden erst dann verfolgt, wenn alle kürzeren Streckenabschnitte beachtet wurden. Dieses Vorgehen gewährleistet, dass bei Erreichen eines Knotens kein kürzerer Pfad zu ihm existieren kann. Eine einmal berechnete Distanz zwischen dem Startknoten und einem besuchten Knoten wird nicht mehr geändert. Distanzen zu noch nicht abgearbeiteten Knoten können sich hingegen im Laufe des Algorithmus durchaus verändern, nämlich verringern. Dieses Vorgehen wird fortgesetzt, bis die Distanz des Zielknotens berechnet wurde (single-pair shortest path) oder die Distanzen aller Knoten zum Startknoten bekannt sind (single-source shortest path).

Der Algorithmus lässt sich durch die folgenden Schritte beschreiben. Es werden sowohl die kürzesten Wegstrecken als auch deren Knotenfolgen berechnet.

* Weise allen Knoten die beiden Eigenschaften „Distanz“ und „Vorgänger“ zu. Initialisiere die Distanz im Startknoten mit 0 und in allen anderen Knoten mit ∞ (in unserem Fall ist hier kein Verweis = keine Kante im Knoten gespeichert).
* Solange es noch unbesuchte Knoten gibt, wähle darunter denjenigen mit minimaler Distanz aus und speichere, dass dieser Knoten schon besucht wurde.
* Berechne für alle noch unbesuchten Nachbarknoten die Summe des jeweiligen Kantengewichtes und der Distanz im aktuellen Knoten. Ist dieser Wert für einen Knoten kleiner als die dort gespeicherte Distanz, aktualisiere sie und setze den aktuellen Knoten als Vorgänger. Dieser Schritt wird auch als Update oder Relaxieren bezeichnet.

In dieser Form berechnet der Algorithmus ausgehend von einem Startknoten die kürzesten Wege zu allen anderen Knoten. Da wir aber nur an einem Ziel interessiert sind bricht man in Schritt (2) schon ab, wenn der gesuchte Knoten der aktive ist.

Dijkstra war unsere einzige Möglichkeit, da wir für andere Algorithmen mehr Zusatzinformationen (wie zum Beispiel Koordinaten für den A\*) benötigen. Die Linienumstiege werden zwar beachtet verbessern aber nicht die Performance des Algorithmus an sich. Verbesserungsmöglichkeiten wären vorhanden, wenn wir zu den Stationen noch Koordinaten einlesen könnten.

## 2.2 Datenstrukturen

* Stationen sind in einer Hashmap gespeichert um möglichst schnell auf sie zugreifen zu können (insbesondere beim Einlesen und zurücksetzen der besuchten Punkte wichtig)
* Connections sind in einer Linked-List gespeichert (Da nicht mehr wie 10 oder 20 pro Station zu erwarten sind und diese Art der Implementation wesentlich einfacher ist und weniger Speicherplatz verbraucht)
* Die Vorgänger der Stationen werden in den Stationen selbst gespeichert und aktualisiert, wenn ein minimaler Weg zu der jeweiligen Station gefunden wurde.
* Die Heapelemente des Dijkstra sind in einer geordneten Linked-List gespeichert, da diese Implementation einfach und Speichersparend ist und das „poppen“ des ersten Elements mit O(1) geht, einfügen ist O(N).

# 3. Testprotokoll

## 3.1 Testdaten

Waren die zur Verfügung gestellten Files: ubahn.txt und ubahnstraßenbahn.txt, die die Wiener Netze beinhalten. Es wurden aufgrund von Zeitmangel keine eigenen Testdateien erstellt.

## 3.2 Testvorgang

* Programmstart
* Testdaten einlesen (entweder als Argument (cmd) oder händisch über Menupunkt)
* 2 Stationen eingeben
* Gefundenen Weg und Zeit mit händisch ausgerechnetem Weg vergleichen

## 3.3 Testdurchführung

* Zugegeben wir haben nicht ausreichend getestet, wir hatten einen Meister in Prokrastination in der 2-er Gruppe dem einer der Mitglieder praktisch die ganze Arbeit am 05.06.2017 um 22:00 bis 23:55 (also nedmal 2 Stunden vor der Abgabe) abnehmen musste…
* 10 Tests mit verschiedenen Stationen durchgeführt, alle lieferten das erwartete Ergebnis zurück.