# Graphen: Dijkstra-Algorithmus

## 1. Anwendungsmöglichkeiten

### 1.1 Netzwerke

**Open Shortest Path First (OSPF)**

* Ermittlung des besten Pfades zwischen Quell- und Zielrouter

**Festlegung eines Dateiservers**

* Minimierung der Anzahl der Hops (Übergänge zwischen Subnetzen)

### 1.2 Navigation

**Reiseauskunft**

* Bestimmung der frühsten Ankunftszeit für das Ziel bei gegebenem Startpunkt und Startzeit
* Auswahl der bestmöglichen Verbindung von A nach B

**Navigationssyteme**

* Bestimmung des kürzesten Weges zwischen dem aktuellen Standort und dem Ziel
* z. B. Google Maps, Apple Karten, Drohnen, Roboter (Digital Warehouse), Autos, …

## 2. Dijkstra’s Shortest Path Algorithm

### 2.1 Graphen

**Ungerichtete Graphen**

* Kanten haben keine Orientierung oder Richtung
* Kante (u, v) zwischen den Knoten u und v ist identisch mit der Kante (v, u)

**Gerichtete Graphen**

* Kanten haben eine Orientierung oder Richtung
* Kante (u, v) ist die Kante von Knoten u zu Knoten v, nicht umgekehrt

**Gewichtete Graphen**

* Kanten haben ein beliebiges Gewicht z.B. Kosten, Entfernung, Menge enthalten
* Kanten werden als Triplet (u, v, w) dargestellt und können sowohl in gerichteten als auch in ungerichteten Graphen vorkommen

### 2.2 Voraussetzung

Der Graph darf keine negative Kanten haben, da Dijkstra im Gegensatz zu Floyd-Warshall Algorithmus keinen negativen Zyklus berücksichtigt.

### 2.3 Ablauf des Algorithmus

* Beginnt an Startknoten und analysiert den Graphen, um kürzeste Pfade zu allen anderen Knoten im Graphen zu finden, nimmt initial unendliche Kosten zu allen anderen Knoten an
* Analysiert Kosten zu allen Knoten, zu denen eine Kante vom aktuellen Knoten existiert, aktualisiert Kosten zu Knoten, wenn kostengünstigerer Pfad gefunden wird
* Nach Analyse aller benachbarten Knoten Auswahl des unbesuchten Knoten mit den aktuell geringsten Kosten und Wiederholung des letzten Schrittes
* Beendigung wenn alle Knoten besucht wurden oder die Kosten zu kostengünstigstem unbesuchten Knoten unendlich sind

### 2.4 Komplexität

|  |  |  |
| --- | --- | --- |
| Implementation | Time Complexity | Space Complexity |
| Lazy mit Array |  |  |
| Lazy mit Priority Queue |  |  |
| Eager mit Indexed PQ |  |  |
| Eager mit Binary Heap |  |  |
| Eager mit D-ary Heap |  |  |
| Eager mit Fibonacci Heap |  |  |

## 3. Erweiterung von Dijkstra’s Algorithm

### 3.1 A\* Search Algorithmus

**Probleme von Dijkstra**

* Dijkstra‘s Algorithm verarbeitet vielversprechendste Edges der Reihe nach,
* das für ggf. dazu, dass Dijkstra in die falsche Richtung geht, da die Kanten ein niedrigeres Gewicht aufweisen

**Heuristik des A\***

* Bei jedem Schritt wird vorausgeplant, damit bessere Entscheidung getroffen werden
* Speichert die geschätzen Gesamtkosten des Pfades durch alle Vertices und der Kosten bis zum Erreichen des Ziels von einer Edge

**Manhattan-Distanz**

* Berechnung des absoluten Abstandes eines Knotens zum Zielknoten

**Euklidische Distanz**

* Berechnung des Pfads entlang einer Linie
* Genauer als die Manhattan Distanz, aber langsamer, da ein größeres Gebiet abgesucht werden muss