Graphen: Dijkstra-Algorithmus

**1. Anwendungsmöglichkeiten**

**1.1 Netzwerke**

Open Shortest Path First (OSPF):

um den besten Pfad zwischen den Quell- und dem Zielrouter zu finden

Dateiserver festlegen:

Um einen Dateiserver zu bestimmen, minimiert Dijkstra die Anzahl der Hops zwischen den Netzwerken

**1.2 Navigation**

Reiseauskunft:

Bestimmung der frühsten Ankunftszeit für das Ziel bei gegebenem Startpunkt und Startzeit

Auswahl der bestmöglichen Verbindung

Navigationssyteme

Bestimmung des kürzesten Weg zwischen dem aktuellen Standort und dem Ziel zu finden

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) 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

Gewichtete Graphen

Kanten haben ein beliebiges Gewicht z.B. Kosten, Entfernung, Menge enthalten

Kante wird als Triplet (u, v, w) dargestellt und können in gerichteten und 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 Algorithmus**

Beginnt an dem von Ihnen gewählten Knoten und analysiert den Graphen, um den kürzesten Weg zwischen diesem Knoten und allen anderen Knoten im Graphen zu finden

Verfolgt die aktuell bekannte kürzeste Entfernung von jedem Knoten zum Quellknoten und aktualisiert die Entfernung, wenn ein kürzerer Pfad gefunden wurde

Wurde ein kürzester Pfad zwischen dem Quellknoten und einem anderen Knoten gefunden, wird dieser Knote als besucht markiert und dem Pfad hinzugefügt

Der Vorgang wird fortgesetzt, bis alle Knoten zum Pfad hinzugefügt wurden.

**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**

* 1. **A\* Search Algorithmus**

Ineffizienz von Dijkstra

Dijkstra‘s Algorithm verarbeitet jede vielversprechendste Edge 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

Manhatten Distance

Berechnung des absoluten Abstandes von einem Knoten zum Zielknoten

Euclidean Distance

Berechnung des Pfads entlang einer Linie

Genauer als die Manhatten Distanz, aber langsamer, da ein größeres Gebiet abgesucht werden muss