Dijkstras Algorithmus[1]

Kürzeste Wege im öffentlichen Nahverkehr

„Wie komme ich von dem Ort, an dem ich gerade stehe, am schnellsten mit der Straßenbahn zur Stadtmitte?"

Die Frage nach der kürzesten öffentlichen Nahverkehrsverbindung ist selbst wenn jeder sie sich oder einem anderen schon einmal gestellt haben dürfte nicht immer ganz einfach zu beantworten. Oftmals ist das Erreichen der Zielhaltestelle über verschiedene Linien möglich. Außerdem können ganz verschiedene Faktoren optimiert werden: Soll die Fahrt möglichst günstig sein, soll das Umsteigen vermieden werden oder steht eine besonders kurze Fahrzeit im Mittelpunkt der Betrachtung.

Das Kürzeste-Wege-Problem

Allgemeine Definition

Diese verallgemeinerte Problemstellung nennt sich **Single Source Shortest Path Problem** (oder kurz SSSP, das zweite P entfällt bei dieser Schreibweise).

Abbildung 2.1-1

Dazu wird aus der konkreten Vorgabe ein Graph gebildet, der sich aus **Knoten** und **Kanten** zusammensetzt (siehe Abb. 2.1-1). Als Gewicht sind ausschließlich positive Werte zulässig. Kanten können der Einfachheit halber vorerst in beiden Richtungen durchlaufen werden, es ist allerdings auch denkbar, mit gerichteten Kanten zu arbeiten. Die Knoten können beliebig auch zirkulär mit einander verbunden sein.

Das zu lösende Problem besteht nun darin, den Weg mit dem geringsten Gewicht zu finden, der zwei im Voraus festgelegte Knoten miteinander verbindet. Das Gewicht eines Weges ergibt sich dabei aus der Summe der Gewichte der durchlaufenen Kanten.

Anwendbarkeit

Mittels dieser Abstraktion lassen sich erstaunlich viele Szenarien modellieren. Hier einige Beispiele aus dem Umfeld des öffentlichen Nahverkehrs:

- **Standardanordnung**

  Der Normalfall, mit einer festgelegten Anzahl von Haltestellen und bidirektionalen Verbindungen, stellt die einfachste Anforderung dar. Dabei ist das Gewicht eines Weges von einer Haltestelle zu einer anderen unabhängig von der Richtung, in die man sich bewegt. Das heißt pro Kante ist nur ein Gewicht festlegbar, welches für beide Durchlaufrichtungen gültig ist.

[1] Aus: verschiedene Quellen, vor allem: http://www.sciface.com/education/data/web/Dijk03r.html

Verbindungen für nur eine Fahrtrichtung

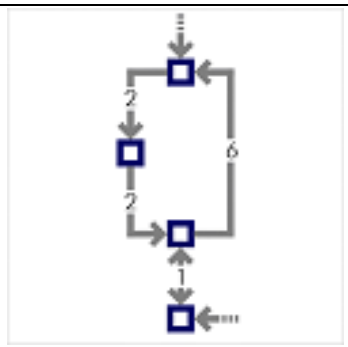


Abbildung 2.2-1

Allerdings ist es auch denkbar, dass eine Kante nur in einer Richtung durchlaufen werden kann. Führt eine Straßenbahnlinie z. B. durch eine Einbahnstraße, so kann es vorkommen, dass Bahnen in Gegenrichtung einen Umweg durch eine andere Straße fahren müssen. Folglich werden manche Haltestellen nur in einer Richtung angefahren oder es nehmen Fahrten zwischen zwei Haltestellen abhängig von der Fahrtrichtung unterschiedlich viel Zeit in Anspruch.

Zur Umsetzung dieses Szenarios müssen die Kanten gerichtet sein, d.h. nur eine Durchlaufrichtung zulassen. Eine ungerichtete Kante kann dann durch zwei entgegengesetzt gerichtete Kanten dargestellt werden. In Abbildung 2.2-1 ist ein entsprechender Aufbau abgebildet.

Wabenbasierte Tarifsysteme

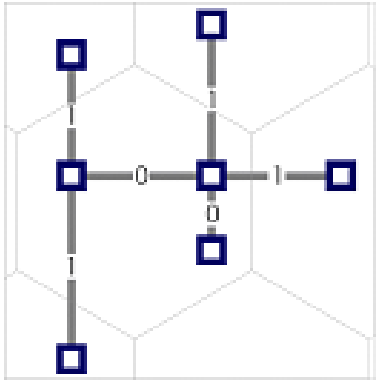


Abbildung 2.2-2

Viele Verkehrsbetriebe berechnen ihre Fahrpreise durch Einteilung des Stadtgebiets in Waben. Es fallen dann jeweils Kosten beim Erreichen einer neuen Wabe an. Im Rahmen unseres Modells ist es ein Leichtes diese Anordnung umzusetzen. Alle Kanten, die Knoten innerhalb der selben Wabe verbinden, werden mit einem Gewicht gleich Null definiert. Nur wabengrenzen-überschreitenden Kanten wird ein Gewicht zugeordnet. Abbildung 2.2-2 stellt diesen Sachverhalt beispielhaft dar.

Die drei Beispiele machen klar, dass die vorgenommene Verallgemeinerung dem ursprünglichen Problem also anscheinend in seiner ganzen Vielseitigkeit gerecht wird. Daher können wir uns nun einem Lösungsansatz zuwenden.

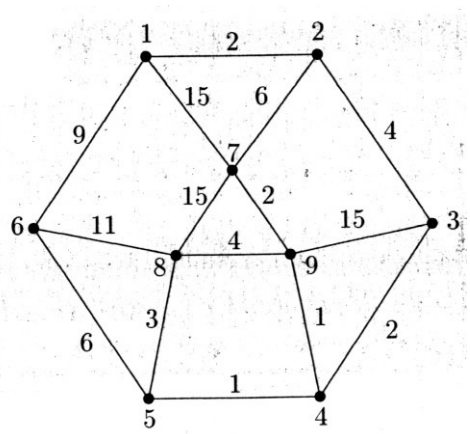
Dijkstras Algorithmus

Allgemein

Einen heutzutage gängigen Weg zur Berechnung kürzester Strecken in Graphen stellt ein 1959 vom Mathematiker Edsger Wybe Dijkstra vorgeschlagener Algorithmus dar, der unter dem Namen „Dijkstras Algorithmus" bekannt geworden ist. Es wird vom Startknoten ausgehend der gesamte Graph erforscht und der kürzeste Weg zu dem Zielknoten in einer vollständigen Suche (Breitensuche) ermittelt. Das Ergebnis ist garantiert der kürzeste Weg und wird immer gefunden.

Er basiert auf einem Algorithmus von L.R. Ford Jr. aus dem Jahre 1956, dessen Grundideen er übernimmt:

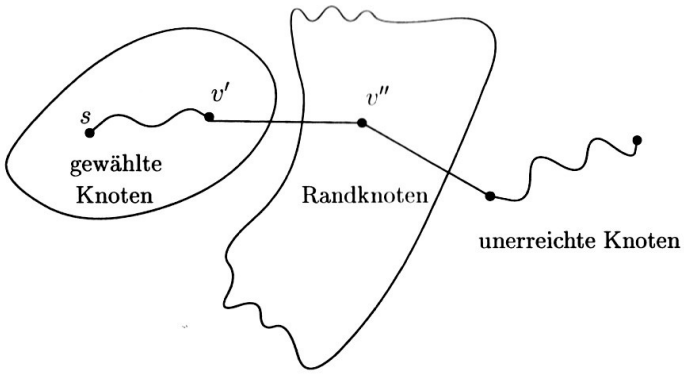
- Es ist aus Sicht des Algorithmus nicht aufwendiger die kürzesten Wege von einem Knoten des Graphen zu ***allen*** anderen zu bestimmen, als ***einen*** kürzesten Weg von einem Start- zu einem Zielknoten zu finden. Die Angabe des Endknotens kann eventuell genutzt werden, um die Berechnung aller kürzesten Wege abzubrechen, nachdem der Endknoten erreicht wurde. Ansonsten endet der Vorgang, wenn alle Knoten entdeckt sind.



- Im Verlauf der Ausführung ermittelt der Algorithmus für jeden Knoten des ihm übergebenen Graphs zwei Angaben: Nämlich das Gewicht des kürzesten Weges vom Startpunkt aus und jeweils einen Verweis auf den vorhergehenden Knoten über den dieser kürzeste Weg verläuft.

Beschreibung und Beispiel

Im abgebildeten Graphen soll der kürzeste Weg von Knoten 1 zu Knoten 8 gefunden werden.

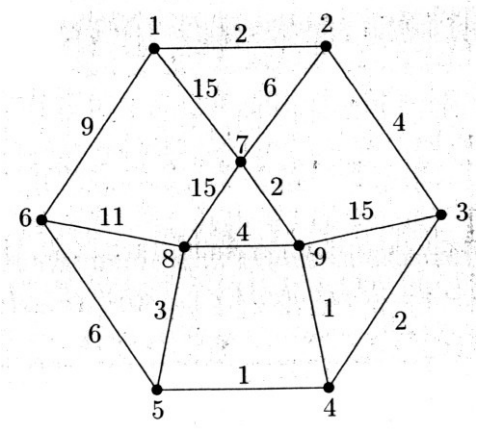


Die grundlegende Idee ist, einen Weg, von dem man weiß, dass er bis zu einem Punkt der kürzeste ist, so zu verlängern, dass der neu entstehende Weg wieder der kürzeste ist.

Dabei liegt diesem Gedanken das **Optimalitätsprinzip** zu Grunde: Für jeden kürzesten Weg $p=(v_1, v_2, ..., v_k)$ von v_0 nach v_k gilt, dass jeder Teilweg $p'=(v_i, ..., v_j)$ daraus für sich gesehen auch ein kürzester Weg ist. Warum?

Nun teilt man die Knoten des Graphen in drei Klassen ein: die *schon gewählten* Knoten, die *Randknoten* und die *unerreichten* Knoten. Zu jedem gewählten Knoten ist der kürzeste Weg zum Startpunkt schon bekannt; zu jedem Randknoten kennt man einen Weg vom Startpunkt aus und für jeden unerreichten Knoten kennt man noch keinen Weg. Wir merken uns für jeden Knoten die bisher berechnete, vorläufige Entfernung zum Anfangsknoten und den Vorgänger auf dem bisher berechneten (kürzesten) Weg.

Folgendermaßen läuft dann der Algorithmus ab:

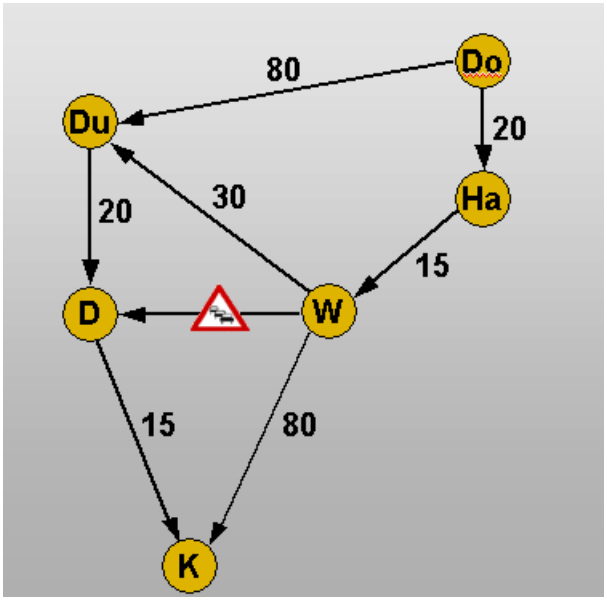


Man sieht, wie sich vorläufige Distanzen von Randknoten ändern können, etwa am Beispiel des Knotens 8. Wenn die von Knoten 1 ausgesandte äquidistante Welle mit aktueller Distanz 8 den Knoten 7 erreicht hat, wird Knoten 8 als mit 7 adjazenter Knoten zu einem Randknoten; seine vorläufige Distanz zu Knoten 1, die er über den Vorgängerknoten 7 realisiert, beträgt 23. Nach der Wahl des Knotens 6 verringert sich diese Distanz auf 20, nach Wahl von Knoten 9 auf 13 und nach Wahl von Knoten 5 schließlich auf 12. Anhand der Liste der gewählten Knoten und der dazugehörigen Vorgängerinformation lässt sich ein kürzester Weg von Knoten 1 zu jedem anderen Knoten rekonstruieren. Für Knoten 8 beispielsweise findet man den Vorgänger 5, für 5 den Vorgänger 4, für 4 den Knoten 3, für 3 den Knoten 2 und für 2 schließlich den Knoten 1 als Vorgänger.

Knoten ≙ (Nr., Entfernung, Vorgänger)

gewählt	Randknoten
(1,0,1)	(2,2,1), (6,9,1), (7,15,1)
(2,2,1)	(6,9,1), (7,8,2), (3,6,2)
(3,6,2)	(6,9,1), (7,8,2), (4,8,3), (9,21,3)
(7,8,2)	(6,9,1), (4,8,3), (9,10,7), (8,23,7)
(4,8,3)	(6,9,1), (8,23,7), (9,9,4), (5,9,4)
(6,9,1)	(9,9,4), (5,9,4), (8,20,6)
(9,9,4)	(5,9,4), (8,13,9)
(5,9,4)	(8,12,5)
(8,12,5)	∅

Aufgabe: Führe das Verfahren von Dijkstra für folgende Streckenkarte durch:[2]



[2] Weitere Quelle: http://www14.in.tum.de/skripten/ead_ws9899_html/node81.html