HAW HAMBURG

Dijsktra-Algorithmus

Torben Kunow, Helena Lajevardi

December 6, 2017

Abstract

Der Dijsktra-Algorithmus löst das Problem der kürzesten Pfade. Er berechnet den kürzesten Pfad von einem Startknoten zu allen anderen Knoten im Graphen. Die Komplexität des Algorithmus ist dabei abhängig von der Implementation des Graphen. Zwei mögliche Implementationen sollen im Folgenden anhand von Zeitmessung und eines Operationszählers verglichen werden.

1 EINLEITUNG

Die folgende Dokumentation beschreibt die Implementationen des Graphen mit Adjazenzlisten sowie Adjazenzmatrix und vergleicht deren Laufzeiten bei Graphen der Größe 10^1 bis 10^4 und dem Grad 10 anhand von Zeitmessung sowie eines Operationszählers. Dabei wird darauf Wert gelegt das Zähler, der Dijsktra-Algorithmus sowie die Graphenimplementation voneinander abgekoppelt sind, und die jeweilige Graphenimplementation unabhängig von Zähler und Dijsktra-Algorithmus ohne Performance-Verlust benutzt werden kann.

2 Implementation des Graphen

Im Folgenden sollen die beiden benutzten Graphenimplementationen an denen der Dijsktra-Algorithmus durchgeführt wird näher beschrieben werden.

2.1 Adjazenzmatrix

Die Adjazenzmatrix speichert welche Knoten durch eine Kante verbunden sind, sowie das Gewicht dieser Kante. Da das Gewicht der Kante von jedem Knoten zu jedem Knoten in der Matrix dargestellt ist beträgt die Größe der Matrix n^2 wobei n die Anzahl der Knoten des Graphes ist. Wenn man wissen will ob der i-te Knoten mit dem j-ten Knoten verbunden ist, kann das Gewicht der Kante an der Stelle (i, j), also an der i-ten Zeile und j-ten Spalte abgelesen werden. Zwei nicht miteinander verbundene Knoten werden durch eine -1 dargestellt. Bei einem ungerichteten Graphen ist die Adjazenzmatrix an der Diagonalen gespiegelt, d.h. sie müsste eigentlich nur zur Hälfte abgespeichert werden. Eine Adjazenzmatrix lohnt sich eher für Graphen mit einem hohem Grad, da ansonsten sehr viel Speicher angelegt wird, nur um die Information zu speichern dass zwei Knoten nicht verbunden sind. Andererseits lässt sich an der Matrix das Gewicht einer Kante sehr schnell (mit **O(1)** Aufwand) ablesen.

2.2 Adjazenzlisten

Für jeden Knoten im Graphen wird eine Liste von allen seinen Nachbarn gehalten. Hierbei ist der Speicheraufwand im Regelfall deutlich geringer als bei der Adjazenzmatrix. Im schlimmsten (und seltenen) Falle wäre jeder Knoten mit jedem Knoten benachbart was wie bei der Adjazenzmatrix zu einem Speicherplatzbedarf von $O(n^2)$ führen würde. Bei dieser Implementation ist es jedoch aufwändiger herauszufinden ob zwei Knoten benachbart sind, da man zunächst die Liste der Knoten nach dem ersten Knoten und dann dessen Adjazenzliste nach dem zweiten Knoten durchsuchen muss. Die Gewichte der Kanten werden in einer Hash-Map gespeichert werden.

3 Entkoppelung des Zählers und des Dijsktra-Algorithmus' von der Graphenimplementation

Im Rahmen der Untersuchungen des Algorithmus' wurde ein Operationszähler eingebaut, der jedoch die eigentliche Implementation des Graphens oder des Algorithmus' nicht verändern sollte. Hierfür wurde ein Graph-Wrapper eingeführt, der jede Operation des Graphen-interfaces anbietet. Der Wrapper bekommt eine der beiden Implementationen übergeben wird nun eine Operation an ihm aufgerufen inkrementiert er einen Zähler und führt diese Operation an der übergebenen Implementation durch. Der Graphen-Wrapper sowie die beiden Graphenimplementation implementieren also alle das Interface Graph. Der Dijkstra-Algorithmus findet in einer eigenen Klasse Dijsktra statt. Alle Informationen die der Algorithmus braucht(z.B. Ob ein Knoten bereits besucht wurde) verwaltet er intern in einer inneren Klasse Node. Die Klasse Dijkstra modifiziert die jeweilige, ihr übergebene Graphen-Implementation, also nicht.

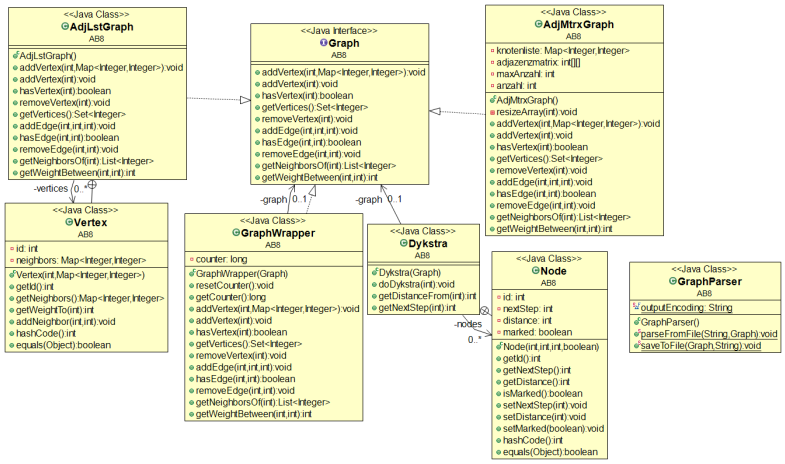

Figure 3.1: Klassendiagramm

4 Vergleich der Implementationen

Für den Vergleich der beiden Implementationen wurde auf zufällige Graphen der jeweiligen Implementation mit 10^1 bis 10^4 Knoten und einem Grad d von 10 der Dijsktra-Algorithmus angewendet. Dabei wurden die Zeit gemessen(Fig. 4.1) sowie Rechenoperationen gezählt(Fig. 4.2).

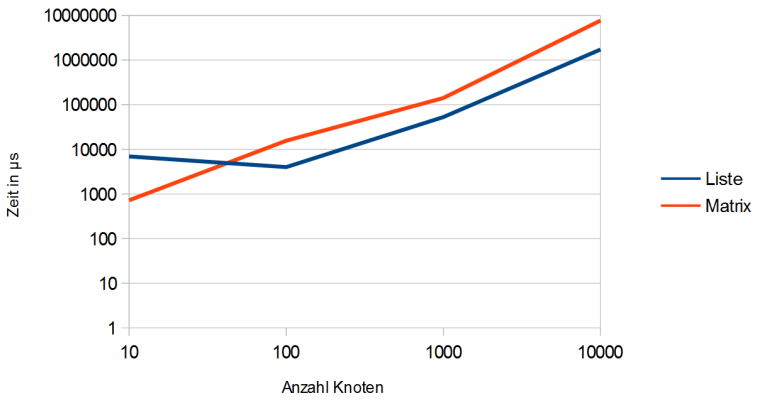

Figure 4.1: Vergleich der Zeitkomplexität

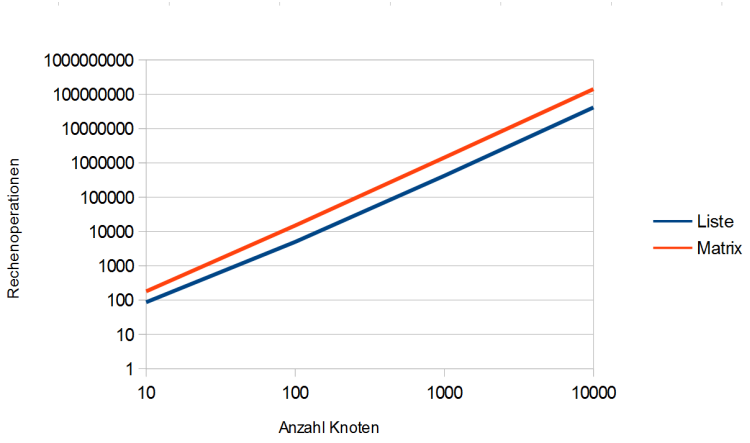

Figure 4.2: Vergleich der Anzahl der Rechenoperationen

Bei dem Vergleich der beiden Implementationen schneidet die Implementation mit Adjazenzliste deutlich besser ab. Für eine Listengröße von 10 ist die Implementation mit Matrix jedoch schneller, in diesem Fall ist ist die Anzahl der Knoten n gleich dem Grad d. Für die Iteration über alle Nachbarn hat man bei der Matrix-Implementation einen Aufwand von $O(n)$ und bei Listen-Implementation einen Aufwand von $O(d)$, wenn Grad und Anzahl Knoten gleich sind fällt dieser Nachteil der Matrix-Implementaion weg. Die Matrix ist außerdem schneller darin auf das Gewicht zwischen zwei Knoten zuzugreifen was ihr in diesem Fall einen Vorteil gegenüber der Listenimplementation verschafft, Die Matrix-Implementation ist also generell eher geeignet wenn der Grad nah an der Anzahl der Knoten liegt.

5 Fazit

Für den Dijsktra-Algorithmus ist also die Listenimplementation schneller für Graphen bei denen die Anzahl der Knoten deutlich höher ist als der Grad. Die Implementation mit Adjazenzmatrix lohnt sich für Graphen bei denen der Grad ähnlich groß wie die Anzahl der Knoten ist. Eindeutig zu bestimmen welche Implenmentation die bessere ist, ist also nicht möglich und vom Anwendungsfall, sowie den Operationen die man am Graphen ausführen will abhängig.