Vergleich zweier Implementation zum kürzeste-Wege-Algorithmus von Dijkstra

Einzureichen am Donnerstag, Dezember 8, 2017

Emira Zorgarti, Menan Ulfat

1

***Abstract*–In dieser Analyse wurde das Verhalten von zwei verschiedenen Implementationen auf ihren Aufwand (unterschiedliche Belastungen)überprüft.Die Implementationen unterscheiden sich in der Implementation des Graphen(Adjazenzenliste, Adjazenzenmatrix)).**

Einleitung

Beim kürzeste-Weg-Algorithmus(Dijkstra) handelt es sich um einen Algorithmus, der einem den kürzesten Pfad von einem gegebenen Startknoten aus in einem Graphen zurückgibt. Die folgende Dokumentation beschreibt zwei Implementationsvarianten des Graphen im kürzester-Weg-Algorithmus.Zunächst werden die verschiedenen Implementationsansätze detailliert beschrieben. In Abschnitt 3 wird das Laufzeitverhalten der beiden Implementationsvarianten überprüft.

2

2.1 Adjazenzliste

Bei der ersten Implemenatation wird der Graph mit Hilfe von Adjazenzenlisten implementiert. Hier werden die Daten in Knoten gespeichert, welche jeweils eine Map enthalten, in der die Nachbarknoten und die dazugehörigen Gewichte gespeichert werden. Fr jeden Knoten im Graph gibt es eine Liste in der die jeweiligen Nachbarknoten enthalten sind. Der Speicheraufwand ist in der Regel um einiges geringer als der bei einer Adjazentematrix. Allerdings entsteht bei einem extrem hohen Grad, für jeden einzelnen Knoten, ein ähnlicher Speicheraufwand $O(N^2)$ wie bei der Adjazenzmatrix. Um zu prüfen, ob zwei Knoten benachbart sind haben wir einen Aufwand von O(d), dieser ist um einiges Aufwändiger als der Aufwand der Adjazenzmatrix bei derselben Operation O(1). Die Gewichte der Kanten könnte man in einer Map speichern.

2.2 Adjazenzmatrix

Bei der zweiten Implementation wird der Graph über eine Adjazenzenmatrix imlplementiert. Diese hält ein zweifaches Integer-Array der Größe n*n (n=Anzahl der Elemente), in dem die Gewichte gespeichert werden. Anhand dieser Tabelle, kann nun für jeden beliebige Element überprüft werden, ob eine Verbingung zu einem anderen besteht und mit welchem Gewicht diese verbunden sind. Dadurch wird die Methode getNeighborsOf(int knotenId) länger im Vergleich zur Adjazenzliste. Da nun alle Elemente überprüft werden müssen und verglichen wird, ob das Gewicht z.B. -1 ist(keine Nachbarn). Diese Vergleiche haben einen Aufwand von O(1), da die Gewichte in der Matrix direkt ablesen werden können. Bei einem ungerichteten Graphen ist die Adjazenzmatrix an der Diagonalen gespiegelt, d.h. sie müsste eigentlich nur zur Hälfte abgespeichert werden. Eine Adjazenzmatrix im vergleich zu einer Adjazenzliste lohnt sich eher für Graphen mit einem hohem Grad, da ansonsten sehr viel Speicher angelegt wird, um die Information zu speichern dass zwei Elemente nicht verbunden sind.

2.3 Dijkstra Algorithmus

Der Dijkstra Algorithmus berechnet den kürzesten bzw. günstigsten Weg von jedem Punkt in einem Graphen zu einem bestimmten Zielpunkt zu gelangen. Dafür werden zunächst alle Nachbarn des Zielpunktes betrachtet. Es wird für jeden Punkt die Kosten zum Zielpunkt, den nächsten Punkt und ein Flag(true=kürzester Weg) gespeichert. Nun werden diese Punkte verglichen und für derjenige mit den gerigsten Kosten das Flag auf true gesetzt. Als nächsten Schritt wird für die aktuellen Punkt und die Nachbarpunkte des zuletzt bearbeiteten Punktes jeweils verglichen, ob die Kosten geringer werden, wenn über diesen Punkt gegangen wird. Wenn ja, wird der nächste Punkt und die Kosten angepasst. Diese Schritte werden jetzt solange wiederholt, bis alle Flags auf true gesetzt werden.

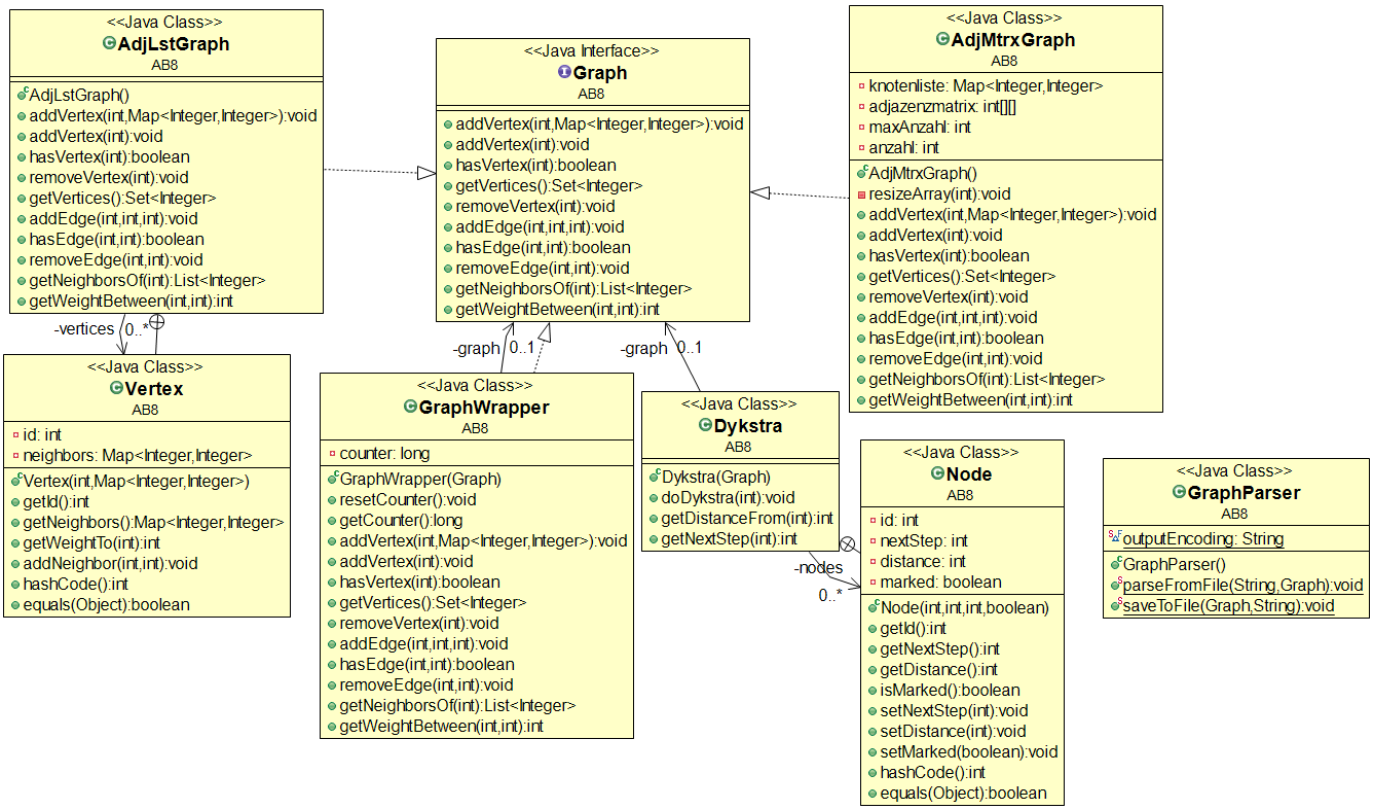

Figure 1: UML-Diagramm

In der Abbildung 1 ist das dazugehörige UML-Diagramm zuerkennen.

3

Für den Vergleich der beiden Implementationen wurde auf zufälligen 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. 2) sowie Rechenoperationen gezählt(Fig 3).

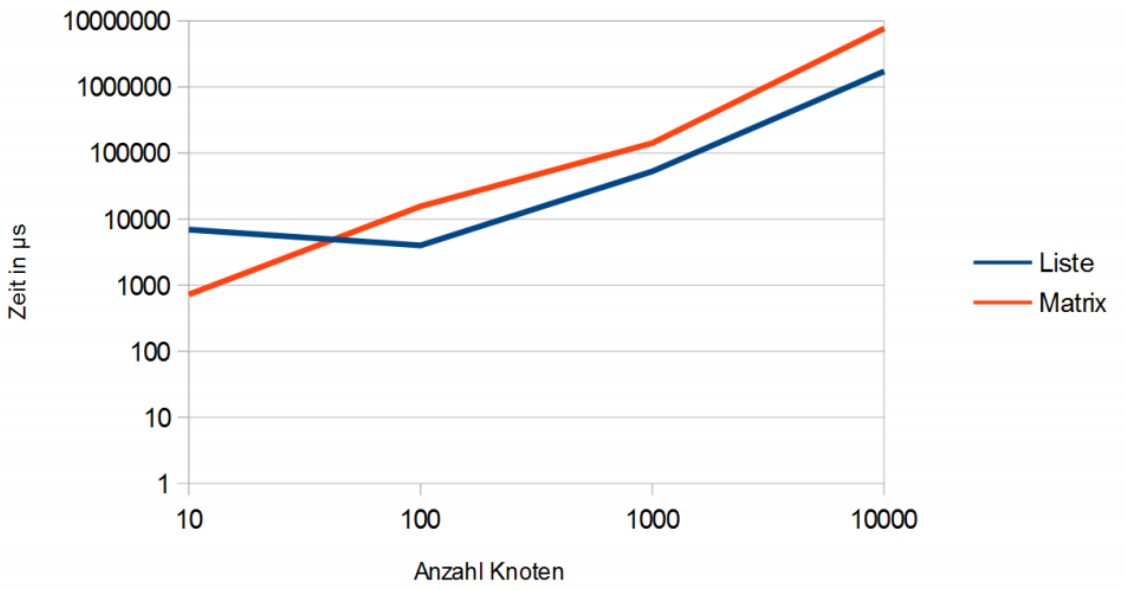

Figure 2: Graphische Darstellung der Zeitkomplexität beider Implementationen

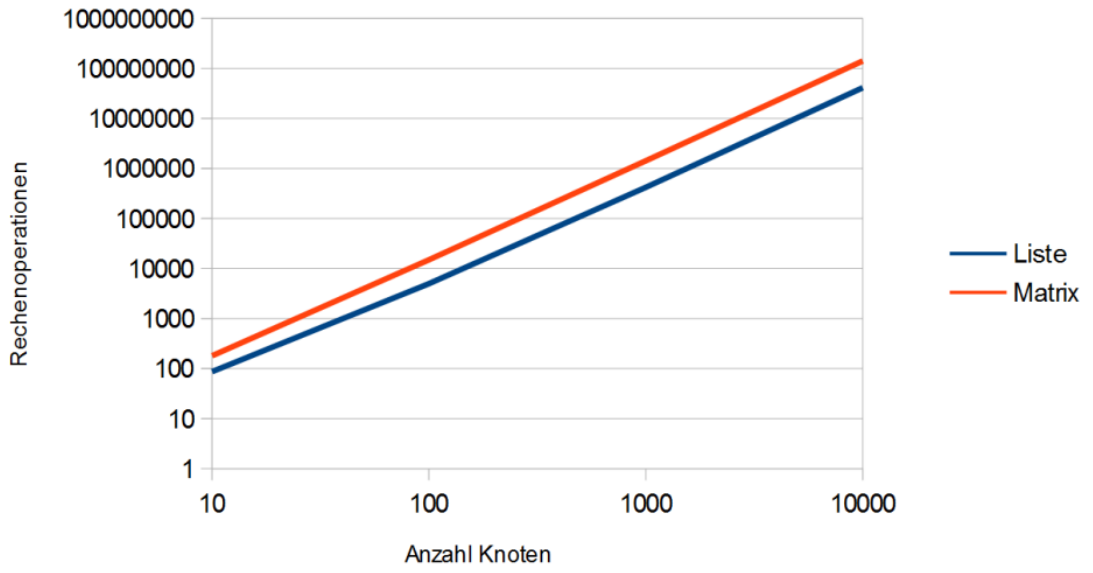

Figure 3: Graphische Darstellung des vergleichs der Rechenoperationen beider Implementationen

3 continued on next page...

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

4

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 Implementation die bessere ist, ist also nicht möglich und vom Anwendungsfall, sowie den Operationen die man am Graphen ausführen will abhängig.

References

[S.Pareigis, 2017] "'Algorithmen und Datenstrukturen fur Technische Informatiker"', Stephan Preigis Hamburg, 10.10.2017