Grundlagen der Graphdatenverarbeitung

Oberseminar „Datenbanksysteme - Aktuelle Trends“

Sommersemester 2019 bei Prof. Dr.-Ing. Kudraß

Hannah Beck (2. Semester Medieninformatik Master, Matr.-Nr: 72643)

Inhaltsverzeichnis

1. Einführung

In den letzten 10 Jahren wurde die Erfassung von Daten über Verbindungen und Beziehungen immer wichtiger. Bisher kommen häufig SQL-Datenbanken zum Einsatz. [1] Doch die relationale Datenhaltung kommt durch die zunehmend wachsenden, unstrukturierten und stark vernetzten Datenmengen oft an ihre Grenzen. Aus diesem Grund kommen vermehrt auch NoSQL-Ansätze zur Anwendung. Dazu zählen auch Graphdatenbanken, bei denen Daten nicht in Form von Tabellen, sondern als Graphen gespeichert werden. [2]

Für verschiedene Anwendungsfälle hat sich bereits ein graphenorientiertes Datenmodell etabliert. Einige dieser Anwendungsfälle sind [1]:

- Analyse sozialer Netzwerke
- Produktempfehlungen
- Analyse von Geschäftsprozessen
- Optimierung von Logistikprozessen
- Routenplanung
- Wissensrepräsentation

Die Verwaltung von Graphdaten ist allerdings auch verknüpft mit neuen Herausforderungen für Datenmanagementsysteme. Es sind spezifische graphenorientierte Konzepte in Bezug auf die Modellierung, Speicherung, Repräsentation und Verarbeitung der Daten notwendig.

2. Eigenschaften von Graphen

Bei einem Graphen handelt es sich um eine Sammlung von Knoten und Kanten. Die Knoten repräsentieren reale Objekte/Konzepte. Die Kanten verbinden die Knoten miteinander und zeigen, in welcher Beziehung die Objekte/Konzepte in der realen Welt zueinanderstehen. [3] Durch die Orientierung an der Realität lassen sich durch Graphen reale Szenarien einfach visualisieren und für den Menschen verständlich darstellen. [4]

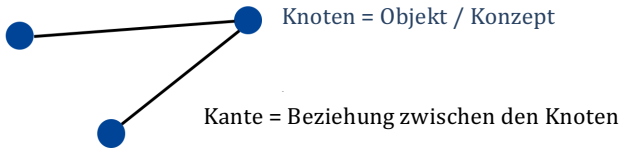


Abb. 1 Darstellung eines Graphen

A. Richtung eines Graphen

Bei einem gerichteten Graphen weist die Kante (Beziehung) zwischen den Knoten eine bestimmte Richtung auf. Bei einem gerichteten Graphen hat eine Kante somit immer einen Ursprungsknoten und einen Zielknoten. Bei einem ungerichteten Graphen weist die Kante zwischen 2 Knoten keine festgelegte Richtung auf. [5]

Abb. 2 links: gerichteter Graph, rechts: ungerichteter Graph

B. Zusammenhang eines Graphen

Ein Graph ist zusammenhängend, falls es zu jedem Knoten im Graphen einen Pfad zu jedem anderen Knoten des Graphen gibt. Falls es isolierte Knoten(-gruppen) gibt, spricht man von einem unzusammenhängenden Graphen. [6]

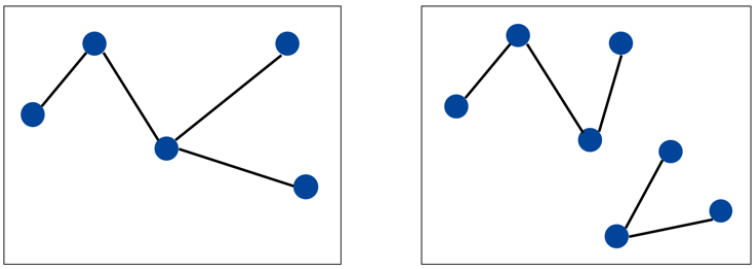


Abb. 3 links: zusammenhängender Graph, rechts: unzusammenhängender Graph

C. Grad eines Knotens

Der Grad eines Knotens wird bestimmt durch die Anzahl der von einem Knoten ausgehenden Kanten. [6]

D. Gewicht eines Graphen

Ein Graph ist gewichtet, wenn seine Kanten gewichtende Eigenschaften besitzen. Dabei kann es sich z.B. um Zeitangaben handeln. Das Gesamtgewicht eines Graphen ist die Summe sämtlicher Gewichte der Kanten. Das Gewicht eines Graphen wird beispielsweise zur Berechnung des kürzesten Weges zwischen 2 Knoten benötigt. [6]

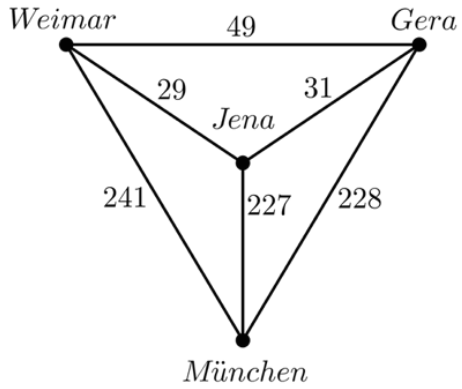


Abb. 4 Gewichteter Graph (mit Zeitangaben auf den Kanten)

3. Graphdatenmodelle

Es existieren verschiedene Graphdatenmodelle, die zwar alle auf den grundlegenden Eigenschaften von Graphen basieren, sich jedoch hinsichtlich bestimmter Aspekte unterscheiden.

A. Property Graph Model

Das Property Graph Model ist das gängigste Modell für Graphdaten. Nach dem Modell besteht ein Graph aus Knoten und gerichteten Kanten. Die Knoten und Kanten haben Namen und können Eigenschaften haben, welche als Attribut-Wert-Paare wiedergegeben werden. [7]

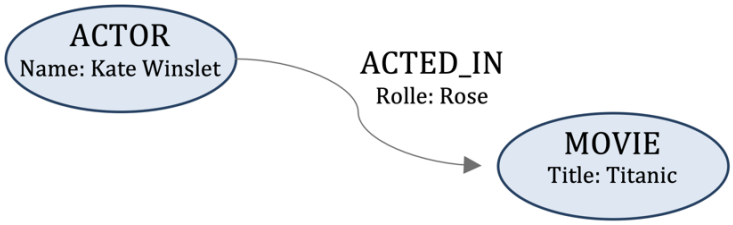


Abb. 5 Property Graph

B. Hypergraph Modell

Bei dem Hypergraph Modell können die Kanten eines Graphen beliebig viele Knoten miteinander verbinden. Anders als bei dem Property Graph Modell, hat eine Kante demnach nicht immer nur exakt einen festgelegten Start- und Endknoten. Das Modell eignet sich, um Beziehungen zwischen mehr als nur 2 Objekten auszudrücken. [4]

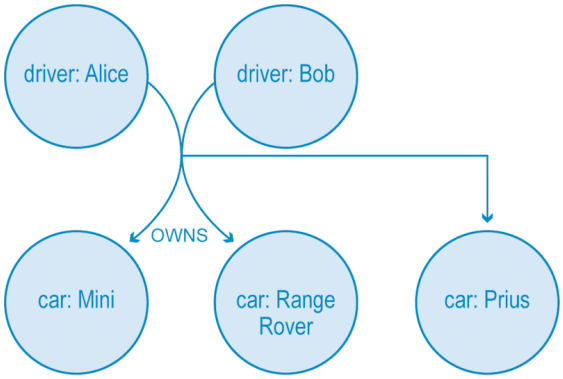


Abb. 6 Hypergraph

C. RDF-Modell

RDF steht für Resource Description Framework. Dabei handelt es sich um ein vom w3C standardisiertes Datenmodell. Es bildet das Fundament für den Bereich des Semantic Webs und soll Informationen zwischen unterschiedlichen Anwendungen austauschbar und von Maschinen lesbar machen. [1]

Ein RDF-Ausdruck ist ein Tripel bestehend aus Subjekt, Prädikat und Objekt. Das Subjekt ist die Ressource, die beschrieben wird. Das Prädikat ist die Eigenschaft über die Ressource, die beschrieben werden soll. Das Objekt ist der konkrete Wert der Eigenschaft.

Jedes Tripel repräsentiert somit eine logische Aussage bezüglich einer Beziehung zwischen dem Subjekt und dem Objekt. Mehrere dieser RDF-Statements bilden einen zusammenhängenden RDF-Graphen, der als semantisches Netzwerk betrachtet werden kann (Siehe Abb. 7). [8]

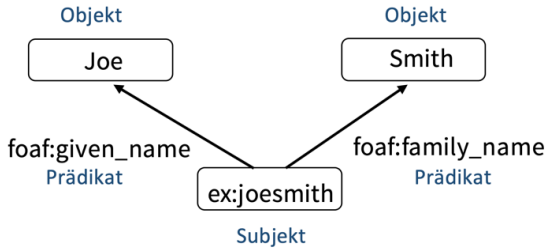


Abb. 7 RDF-Graph

4. Speicherung von Graphen

A. Adjazenzmatrix

Bei einer Adjazenzmatrix wird ein Graph in Form einer Matrix gespeichert. Dazu werden alle Knoten sowohl senkrecht als auch waagrecht entlang der Matrix eingetragen. Die Einträge in der Matrix geben an, welche Knoten mit anderen Knoten über wie viele Kanten verbunden sind. [9]

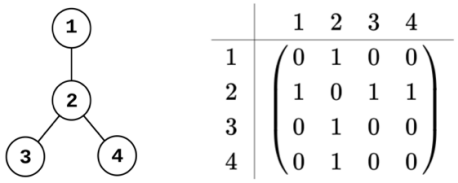


Abb. 8 links: Graph, rechts: Darstellung des Graphen als Adjazenzmatrix

Die Speicherung eines Graphen als Adjazenzmatrix ist allerdings insbesondere bei großen Graphen ineffizient, da häufig viele Nullen abgespeichert werden müssen und es zum Herausfinden aller Nachbarknoten eines Knotens notwendig ist, ganze Zeilen/Spalten auszulesen. [9]

B. Adjazenzliste

Adjazenzlisten stellen eine Alternative zu Adjazenzmatrizen dar. In einer Adjazenzliste werden bei einem Knoten alle von ihm ausgehenden Kanten gespeichert. Somit müssen nicht mehr ganze Zeilen/Spalten abgefragt werden, um alle Nachbarknoten eines Knotens zu identifizieren. [9]

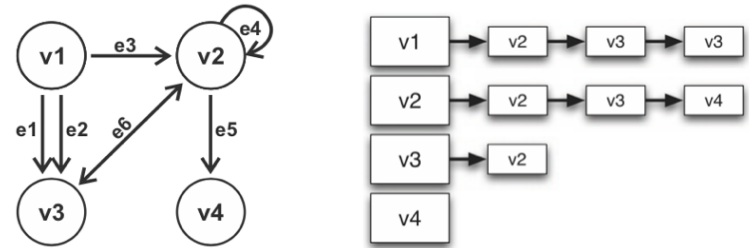


Abb. 9 links: Graph, rechts: Darstellung des Graphen als Adjazenzliste

C. Compressed Sparse Rows

Eine weitere Alternative stellt die Speicherung in Form von Compressed Sparse Rows dar. Dabei handelt es sich um ein Verfahren, das häufig dafür genutzt wird, schwach besetzte Matrizen zu speichern. Bei dem Verfahren werden nur Einträge gespeichert, die nicht 0 sind und somit keine unnötigen Einträge. Die Speicherung erfolgt statt in Form einer Matrix in komprimierter Form in 3 Arrays.

Beispiel:

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

val	10	-2	3	9	3	7	8	7	3 ··· 9	13	4	2	-1
col_ind	1	5	1	2	6	2	3	4	1 ··· 5	6	2	5	6

row_ptr	1	3	6	9	13	17	20

Abb. 10 oben: Matrix A, unten: Repräsentation der Matrix A in 3 Arrays

In dem Array val werden alle Matrix-Einträge gespeichert, die nicht 0 sind. In dem Array col_ind wird für jeden Eintrag im Array val der Spalten-Index gespeichert. Die Werte im Array row_ptr geben an, zu welcher Matrix-Zeile die Einträge des Arrays val gehören.

Compressed Sparse Rows ermöglichen eine effizientere Speicherung, da keine unnötigen Daten und ganze Matrizen abgespeichert werden müssen. Allerdings erschwert die indirekte Adressierung auch den Zugriff auf die Daten und das Hinzufügen neuer Daten. [10]

D. Triple Table

Triple Tables werden zur Speicherung von RDF-Graphen in RDF-Datenbanken (Triple Stores) verwendet. Bei einer Triple Table handelt es sich um eine 3 spaltige Tabelle, bei der jeweils eine Spalte für das Subjekt, Prädikat und Objekt eines Tripels steht. Jede Zeile einer Triple Table stellt somit eine Subjekt-Prädikat-Objekt-Relation dar. [11]

Bei Triple Tables kommen oftmals Wörterbuchverfahren zu Einsatz, um den Speicherumfang zu verringern. Bei dem Wörterbuchverfahren werden häufig wiederkehrende Zeichenketten in einer Wörterbuchtabelle ausgelagert, in der ihnen ein Ersatzsymbol (z.B. die Eintragsnummern) zugeordnet wird. Ein Nachteil von Triple Tables ist, dass die Abfrage komplexe JOIN-Abfragen erfordert. Um die Performance zu optimieren, kann bei Triple Tables Indexierung eingesetzt werden. Üblich ist dabei die vollständige 6-fach Indexierung, bei der alle möglichen Ausprägungen eines Tripels indexiert werden. Die Wörterbuchcodierung ermöglicht dabei kompaktere Indizes. [1]

5. Anforderungen an Graphdatenbanksysteme

Um Graphdaten effizient, sicher und performant verwalten und bereitstellen zu können, müssen Graphdatenbanken bestimmte Anforderungen erfüllen.

A. Leistungsstarkes, flexibles Graphdatenmodell

Graphendatenbanken sollten in der Lage sein, nicht nur homogene Graphen zu verarbeiten, sondern auch Graphen mit Knoten und Kanten unterschiedlicher Art und mit unterschiedlichen Attributen. Dies sollte ohne festes Schema möglich sein. Das Graphdatenmodell sollte außerdem in der Lage sein, nicht nur einzelne Graphen, sondern auch Graphensammlungen darzustellen und zu verarbeiten. Zudem sollte das Graphdatenmodell verschiedene leistungsfähige Graphenoperatoren zur Verfügung stellen, die für die Graphenverarbeitung und –analyse verwendet werden können.

B. Leistungsstarke Abfrage- und Analysefunktionen

Es sollte für den Nutzer möglich sein, Graphdaten mithilfe einer deklarativen Abfragesprache zu analysieren. Das System sollte auch die Verarbeitung von komplexen Analysen unterstützen, die meist eine iterierte Verarbeitung von dem gesamten Graphen oder großen Bereichen davon benötigen. Dazu zählen beispielsweise Graph-Mining-Prozesse.

C. Hohe Leistung und Skalierbarkeit

Die Graphenverarbeitung und –analyse sollte schnell erfolgen und skalierbar sein, sodass es auch möglich ist, sehr umfangreiche Graphen mit einer Vielzahl unterschiedlicher Knoten und Kanten zu erstellen und zu verarbeiten. Dies erfordert in der Regel eine verteilte Graphenverarbeitung und In-Memory-Verarbeitung.

D. Persistente Graphenspeicherung und Transaktionssicherheit

Trotz der notwendigen In-Memory-Speicherung von Graphen, ist eine persistente (dauerhafte) Speicherung der Graphdaten und Analyseergebnisse notwendig. Zudem sollte Transaktionssicherheit gewährleistet werden. Wünschenswert ist dabei die OLTP-Funktionalität mit ACID-Transaktionen.

E. Einfache Benutzbarkeit und Graphenvisualisierung

Umfangreiche, stark vernetzte Graphdaten können oftmals äußerst komplex und schwer durchsuchbar sein. Darum sollte die Analyse der Daten so einfach wie möglich gemacht werden, beispielsweise durch leistungsfähige Operatoren und Analysefunktionen. [12]

6. Query Interfaces

Um Graphen effizient verarbeiten zu können, müssen Query Interfaces eine größtmögliche Parallelisierung der Verarbeitung ermöglichen. Dafür gibt es verschiedene Ansätze, die von Graphenverarbeitungssystemen unterstützt werden können.

A. Knotenzentrierte Verarbeitung

Nach dem knotenzentrierten Modell wird ein Algorithmus aus der Sicht eines einzelnen Knotens geschrieben („Think like a vertex"). Für jeden Knoten sind die einzigen Informationen, die er hat, seine eigenen Eigenschaften und seine Nachbarliste. Dementsprechend kann ein Knoten bei einem knotenzentrierten Algorithmus ausschließlich seine eigenen Informationen manipulieren, Nachrichten von benachbarten Knoten empfangen und Nachrichten an sie senden. [13]

Bei einem knotenzentrierten Algorithmus werden folgende Phasen durchlaufen [1]:

- **Gather:** Der aktive Knoten empfängt Nachrichten von seinen Nachbarknoten mit Informationen zu deren Daten. Die übermittelten Daten werden an die Apply-Funktion übergeben.
- **Apply:** Die Apply-Funktion wird auf dem aktiven Knoten ausgeführt. Dabei handelt es sich um eine benutzerdefinierte Funktion, welche die Daten des Knotens neu berechnet.
- **Scatter:** Der aktive Knoten sendet die in der Apply-Phase berechneten Daten an seine Nachbarknoten.

Der Algorithmus wird beendet, sobald keine Nachrichten mehr zwischen den Knoten versendet werden, da dies auf einen Stopp hinweist. [13]

Die knotenzentrierte Verarbeitung ist am weitesten verbreitet. Allerdings hat das Verfahren auch Nachteile. Da das Verfahren darauf beruht, dass in verschiedenen Iterationen Nachrichten zwischen Knoten ausgetauscht werden, kann der Vorgang viel Zeit in Anspruch nehmen. Da nicht alle Knoten gleich viele Kanten haben, kann es außerdem zu einem Arbeitsungleichgewicht kommen. Knoten mit vielen Kanten erfordern mehr Aktionen als Knoten mit wenigen Kanten. Insbesondere, wenn unterschiedliche Computerressourcen für die verteilte, parallele Verarbeitung genutzt werden, kann dies dazu führen, dass einzelne Computerressourcen nicht maximal genutzt werden. [14]

B. Kantenzentrierte Verarbeitung

Das kantenzentrierte Modell ist ähnlich wie das knotenzentrierte Modell, allerdings wird nicht mehr auf den Knoten, sondern auf den Kanten eines Graphen gerechnet. Während der Phasen Gather Apply und Scatter werden auf den Kanten eines Graphen anhand der Daten des Ursprungsknotens die Daten des Zielknotens berechnet und übermittelt. [1]

C. Graphenzentrierte Verarbeitung

Bei der graphenzentrierten Verarbeitung wird ein Graph partitioniert (in kleinere Graphenabschnitte unterteilt). Anschließend wird eine benutzerdefinierte Funktion auf die gesamten Graphenpartitionen angewendet. Demnach beschränken sich graphenzentrierte Algorithmen nicht mehr auf die Knoten oder Kanten eines Graphens, sondern beziehen alle Unterstrukturen mit ein. Graphenzentrierte Algorithmen erfordern eine komplexere

Programmierung. [1] Andererseits kann eine höhere Flexibilität und bessere Performance erreicht werden, da Informationen nicht mehr wie bei knoten- und kantenzentrierten Algorithmen schrittweise durch den gesamten Graphen transportiert werden müssen. [15]

7. Systeme zur Graphdatenverarbeitung

Es existieren nicht nur zahlreiche Graphdatenbanksystemen, sondern auch verschiedene Frameworks zur Graphenverarbeitung, welche darauf ausgelegt sind, komplexe und umfangreiche Graphdaten zu verarbeiten. Die Frameworks zeichnen sich vor allem durch den Einsatz von verteilten und parallelen Verarbeitungsstrategien aus. Zwei bekannte Graphverarbeitungssysteme sind Pregel und Apache Giraph.

Pregel ist eine API, die von Google entwickelt wurde, um Algorithmen für die die Verarbeitung von umfangreichen Graphdaten zu schreiben. Pregel folgt dem „Think-like-a-vertex"-Ansatz und somit dem knotenzentrierten Modell. Pregel orientiert sich an dem Bulk Synchronous Parallel Modell von Leslie Valiant und führt demnach Graphenberechnungen in verschiedenen Supersteps parallel auf allen Knoten des Graphens aus. [11]

Apache Giraph sollte als Open-Source-Pendant zu dem von Google entwickelten Pregel dienen. Bei Giraph handelt es sich um ein Graphverarbeitungssystem, das vor allem auf eine hohe Skaliebarkeit ausgerichtet ist. Es wird beispielsweise von Facebook genutzt, um Soziogramme (die Vernetzung von unterschiedlichen Nutzern und deren Beziehung) zu analysieren. [16] Apache Giraph wurde auf Grundlage von Apache Hadoop entwickelt, ein in Java geschriebenes Frameworks für verteilt arbeitende und leicht skalierbare Software zur Verarbeitung von großen Datenmengen. [17]

8. Zusammenfassung und Ausblick

Die Graphdatenverarbeitung gewinnt zunehmend an Bedeutung in Bezug auf die Repräsentation und Analyse von vernetzten Daten in zahlreichen Anwendungsgebieten. Zu den Anwendungsgebieten zählen alle Bereiche, in denen eine Vielzahl vernetzter Daten anfallen (z.B. soziale Netzwerke, Routenplanung etc.). Ein wesentlicher Vorteil der graphenorientierten Datenhaltung ist, dass stark vernetzte Daten flexibel abgespeichert und ohne komplexe JOIN-Analysen abgefragt werden können. Die performante Verarbeitung von Graphdaten erfordert allerdings eine effiziente Speicherung, die stark von der klassischen relationalen Datenhaltung abweicht. Durch Graphalgorithmen wie beispielsweise dem Shortest-Path-Algorithmus können bestimmte Grapheneigenschaften identifiziert werden, um Aussagen bezüglich der Daten treffen zu können. Dabei kann ein knoten-, kanten- oder graphenzentrierter Ansatz verfolgt werden. Um komplexe Analysen an Graphdaten durchführen zu können, sind parallele Verarbeitungsprozesse notwendig, die von den gängigen Graphdatenverarbeitungssystemen unterstützt werden.

Graphdatenbanken werden bereits von vielen Unternehmen eingesetzt. Angesichts der zunehmenden Relevanz und dem steigenden Wert bezüglich der Auswertung von Beziehungen zwischen unterschiedlichen Daten, ist ein weiteres Wachstum in Bezug auf die Popularität von Graphdatenbanken in den nächsten Jahren zu erwarten. [18]

9. Erläuterung des Demo-Beispiels

Als Demo-Beispiel wurde gezeigt, wie die Berechnung der kürzesten Tram-Strecke zwischen der Tram-Station „Hauptbahnhof“ und der Tram-Station „HTWK“ mit Hilfe der Graphdatenbank Neo4J und dem Shortest-Path-Algorithmus berechnet werden kann.

Erstellung des Datensatzes:

/*Erstellung der Knoten (=Tram-Stationen) des Graphen*/

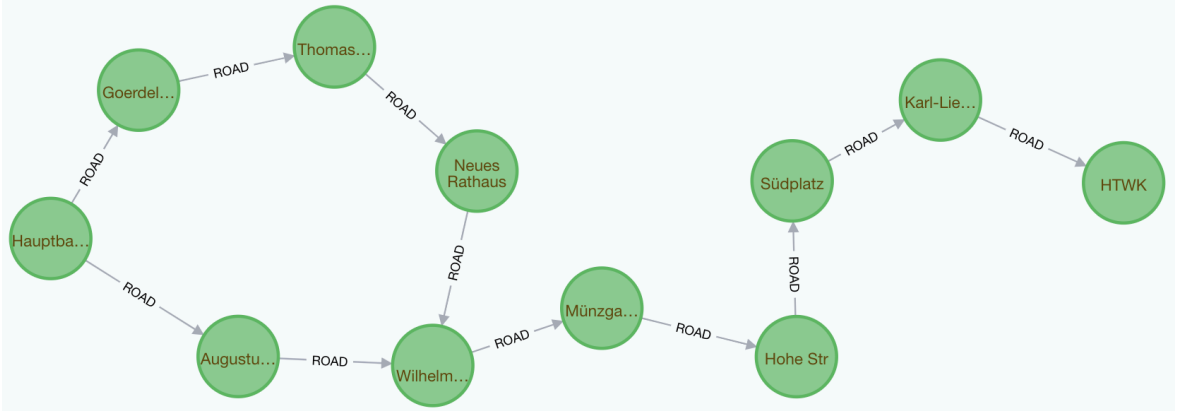
```
MERGE (a:Station {name:"Hauptbahnhof"})
MERGE (b:Station {name:"Augustusplatz"})
MERGE (c:Station {name:" Wilhelm-Leuschner-Platz"})
MERGE (d:Station {name:"Münzgasse "})
MERGE (e:Station {name:"Hohe Str"})
MERGE (f:Station {name:"Südplatz"})
MERGE (g:Station {name:"Karl-Liebknecht"})
MERGE (h:Station {name:"HTWK"})
MERGE (i:Station {name:"Goerdelerring"})
MERGE (j:Station {name:"Thomaskirche"})
MERGE (k:Station {name:"Neues Rathaus"})
```

/*Erstellung der Kanten des Graphen (=Verbindungen zwischen den Stationen) mit Angabe der Eigenschaft cost (=Fahrzeit zwischen den Stationen)*/

```
MERGE (a)-[:ROAD {cost:3}]->(b)
MERGE (a)-[:ROAD {cost:3}]->(i)
MERGE (b)-[:ROAD {cost:2}]->(c)
MERGE (c)-[:ROAD {cost:2}]->(d)
MERGE (d)-[:ROAD {cost:1}]->(e)
MERGE (e)-[:ROAD {cost:1}]->(f)
MERGE (f)-[:ROAD {cost:2}]->(g)
MERGE (g)-[:ROAD {cost:1}]->(h)
MERGE (i)-[:ROAD {cost:2}]->(j)
MERGE (j)-[:ROAD {cost:1}]->(k)
MERGE (k)-[:ROAD {cost:2}]->(c);
```

Graph:

Enthält die Verbindungen zwischen den Tram-Stationen. Es sind 2 mögliche Routen zwischen dem Hauptbahnhof und der HTWK erkennbar. Die benötigte Zeit zwischen den Stationen ist auf den Kanten als Eigenschaft gespeichert (in Neo4J nicht visuell in dem Graphen erkennbar).



Route 1 (Linie 9): Hauptbahnhof > Goerdelerring > Thomaskirche > Neues Rathaus > Wilhelm-Leuschner-Platz > Münzgasse > Hohe Str. > Südplatz > Karl-Liebknecht/Kurt-Eisner-Str. > HTWK
Route 2 (Linie 11): Hauptbahnhof > Augustusplatz > Wilhelm-Leuschner-Platz > Münzgasse > Hohe Str. > Südplatz > Karl-Liebknecht/Kurt-Eisner-Str. > HTWK

Anwendung des Shortest-Path-Algorithmus:

```
MATCH (start:Station{name:'Hauptbahnhof'}), (end:Station{name:'HTWK'})
CALL algo.shortestPath.stream(start, end, 'cost')
YIELD nodeId, cost AS time
RETURN algo.asNode(nodeId).name AS Station, time
```

Ergebnis:
Die kürzeste Route zwischen dem Hauptbahnhof und der HTWK wird angezeigt mit Angabe der benötigten Fahrzeit (12 Minuten):

Station	time
"Hauptbahnhof"	0.0
"Augustusplatz"	3.0
" Wilhelm-Leuschner-Platz"	5.0
"Münzgasse "	7.0
"Hohe Str"	8.0
"Südplatz"	9.0
"Karl-Liebknecht"	11.0
"HTWK"	12.0

10. Quellen

[1] Marcus Paradies, Hannes Voigt: Big Graph Data Analytics on Single Machines – An Overview, in: Datenbank-Spektrum Bd. 17, Heft 2, Juli 2017 S.97-105

[2] Andreas Meier, Michael Kaufmann: SQL- & NoSQL-Datenbanken, 8., überarb. u. erw. Aufl., Berlin, Heidelberg : Springer (2016) S. 3

[3] Ian Robinson, Jim Webber, Emil Eifrem: Graph Databases, O'Reilly (2015): S.1

[4] Ian Robinson, Jim Webber, Emil Eifrem: Graph Databases, O'Reilly (2015): S.207

[5] Peter Eades, Karsten Klein: Graph Visualization, Erschienen in: Graph Data Management : Fundamental Issues and Recent Developments / Fletcher, George et al. (Hrsg.). Springer (2018): S. 60

[6] Andreas Meier, Michael Kaufmann: SQL- & NoSQL-Datenbanken, 8., überarb. u. erw. Aufl., Berlin, Heidelberg : Springer (2016): S. 61-63

[7] Andreas Meier, Michael Kaufmann: SQL- & NoSQL-Datenbanken, 8., überarb. u. erw. Aufl., Berlin, Heidelberg : Springer (2016): S. 14

[8] Renzo Angles, Claudio Gutierrez: An Introduction to Graph Data Management, Erschienen in: Graph Data Management : Fundamental Issues and Recent Developments / Fletcher, George et al. (Hrsg.). Springer (2018): S. 12

[9] Stefan Edlich, Achim Friedland, Jens Hampe, Benjamin Brauer: NoSQL: Einstieg in die Welt nichtrelationaler Datenbanken, 2., akutalisierte und erweiterte Auflage, München : Carl Hanser Verlag (2011): S. 217-218

[10] Richard Barrett, Michael Berry, Tony F. Chan, James Demmel, June M. Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, Henk Van der Vorst: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods: <http://www.netlib.org/templates/templates.pdf> S. 57-58

[11] Renzo Angles, Claudio Gutierrez: An Introduction to Graph Data Management, Erschienen in: Graph Data Management : Fundamental Issues and Recent Developments / Fletcher, George et al. (Hrsg.). Springer (2018): S. 24-26

[12] Martin Junghanns, Andre Petermann, Martin Neumann, Erhard Rahm: Management and Analysis of Big Graph Data: Current Systems and Open Challenges, Handbook of Big Data Technologies (2017), S. 457-459

[13] Siyuan's Blog (2017): Vertex-centric graph processing: the what and why <https://shawnliu.me/post/vertex-centric-graph-processing-the-what-and-why/> (letzter Zugriff: 20.06.2019)

[14] Cornell University: <http://www.cs.cornell.edu/courses/cs6453/2017sp/slides/graph_processing.pdf > (letzter Zugriff: 22.06.2019)

[15] Yuanyuan Tian, Andrey Balmin, Severin Andreas Corsten, Shirish Tatikonda, John McPherson: From "Think Like a Vertex" to "Think Like a Graph" (2013) <https://researcher.watson.ibm.com/researcher/files/us-ytian/giraph++.pdf> (letzter Zugriff: 21.06.2019)

[16] The Apache Software Foundation: <https://giraph.apache.org> (letzter Zugriff: 21.06.2019)

[17] Wikimedia Foundation Inc. (2008); Apache Hadoop <https://de.wikipedia.org/wiki/Apache_Hadoop> (letzter Zugriff: 22.06.2019)

[18] ap Verlag GmbH (2017) <http://ap-verlag.de/ausblick-2017-die-zukunft-der-graphtechnologie/30364/> (letzter Zugriff: 25.06.2019)

11. Abbildungsverzeichnis