XML-Text im Graphen

Contents

1 Inhalt

{:.no_toc}

- Will be replaced with the ToC, excluding the “Contents” header {:toc}

2 Textmodelle im Graphen

2.1 Text als Graph

Die Diskussion über Modellierungsansätze von Text als Graph hält aktuell an.[1]

2.1.1 XML-Dateien als Ketten von Zeichen

Da die technische Grundlage von XML Textdateien sind, handelt es sich bei XML um eine eindimensionale Kette von Tokens[2].

2.1.2 Modellierungsüberlegungen

Prinzipiell können XML-Dateien ohne größere Probleme in einen Graphen importiert werden, da sie einen geerdeten, gerichteten azyklischen Graphen, der vielfache Elternbeziehungen verhindert, und damit ein Ordered Hierarchy of Content Objects (OHCO) darstellen. Es gibt vor allem im Bereich des

[1] Vgl. zuletzt @DekkerHaentjensItmorejust2017.

[2] @HuitfeldtMarkupTechnologyTextual2014, S. 161 sieht digitale Dokumente prinzipiell als lineare Sequenz von Zeichen

Mixed-Content verschiedene Ansätze, XML-Strukturen im Graphen abzubilden[3]. Überlegungen zur Auslagerung von Annotationen aus XML in eine Graphdatenbank brachte schon Desmond Schmidt in die Diskussion ein:

Embedded annotations can also be removed from TEI texts. The elements `<note>`*,* `<interp>`*, and* `<interpGrp>` *describe content that, like metadata, is about the text, not the text itself. These are really annotations, and should ideally be represented via the established standards and practices of external annotation (Hunter and Gerber 2012). Annotations are stored in triple stores or graph databases like Neo4J,20 which record the identifiers of each component of the annotation and its data*[4].

2.1.2.1 Granularität des Modells – Was ist ein Token ?

Für den Bereich der historisch-kritischen und philologischen Editionen ist es in der Regel ausreichend, beim Import von XML-kodierten Texten in den Graphen jeweils ein Wort in einen Knoten zu importieren, da meist die historische Aussage der Quelle im Vordergrund steht. In anderen Bereichen der digitalen Geisteswissenschaften kann die Entscheidung, welche Einheit für den Import in einen Knoten gewählt wird, durchaus anders ausfallen. So ist für Philologien die Betrachtung auf Buchstabenebene interessant[5]. Im Graphmodell ist man im Hinblick auf die Granularität des Datenmodells wesentlich flexibler als z.B. bei XML oder Standoff-Markup. So ist es beispielsweise denkbar, an einen Wortknoten eine weitere Kette von Knoten anzulagern, welche pro Knoten jeweils einen Buchstaben des Wortes und die zugehörigen Annotationen enthalten. Es handelt sich um einen buchstabenbasierten Sub-Graphen, dessen Anfang und Ende mit dem Wortknoten verbunden ist. Damit können verschiedene Granularitätsstufen in einem Modell und in einer Datenbank abgebildet werden.

[3] Vgl. @DekkerHaentjensItmorejust2017.

[4] @SchmidtInteroperableDigitalScholarly2014, 4.1 Annotations.

[5] In FuD (http://fud.uni-trier.de/) werden Texte in Standoff-Markup auf Buchstabenebene ausgezeichnet, während beim DTA-Basisformat der Fokus auf der wortbasierten Auszeichung liegt (vgl. http://www.deutschestextarchiv.de/doku/basisformat/eeAllg.html).

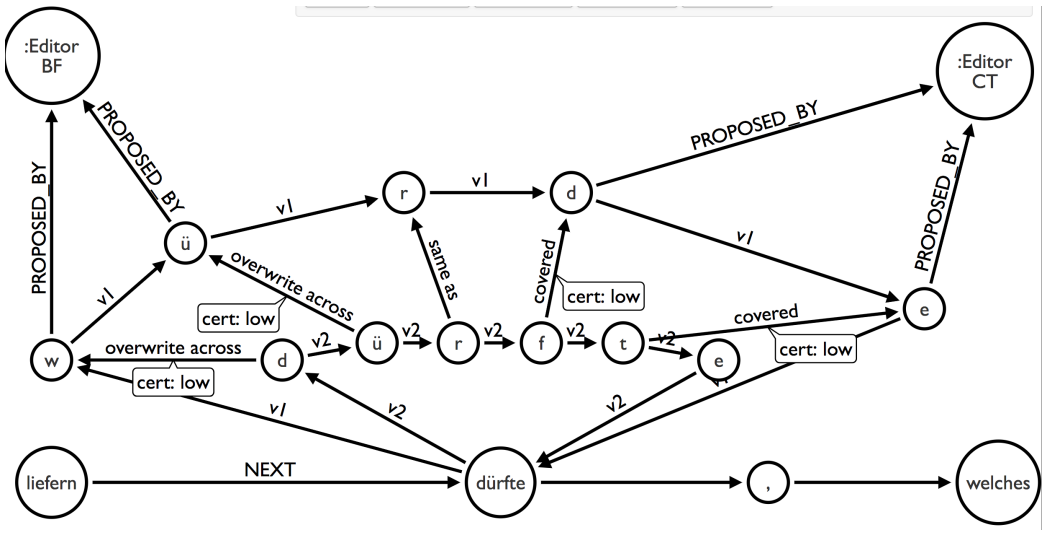

Figure 1: Granularität von Text im Graphen

2.2 Technische Vorbemerkungen

2.2.1 Die Graphdatenbank neo4j

Technische Grundlage der im Rahmen dieses Werkes vorgestellten Beispiel ist die Graphdatenbank neo4j[6]. Die Graphdatenbank gibt es unter Open-Source-Lizenz in einer Community-Edition und unter einer kommerziellen Lizenz in einer Enterprise-Edition. Für die hier vorliegenden Aufgabengebiete war die Leistungsfähigkeit der Community-Edition völlig ausreichend. Darüber hinaus hat sich neo4j in den letzten Jahren als eine der führenden Graphdatenbanken etabliert und bietet neben stabiler Funktionalität auch genormte Schnittstellen. Prinzipiell wurde darauf hingearbeitet, dass die hier vorgestellten Überlegungen zu Graphenmodellierung historisch-kritischer Editionen auch auf andere Property-Graphdatenbanken wie z.b. orientdb[7] übertragbar sind.

[6]Vgl. www.neo4j.com (abgerufen am 7.8.2017).

[7]Die Graphdatenbank orientdb (http://orientdb.com/) bietet ein für Historiker sehr interessantes Feature, da sie als Datentyp für die Propertys von Knoten auch Datumsangaben im ISO-Format zulässt. Vgl. https://orientdb.com/docs/2.2/Managing-Dates.html (abgerufen am 7.8.2017).

2.2.2 Der neo4j-XML-Importer

Für den Import der Texte wurde der neo4j-XML-Importer von Stefan Armbruster verwendet[8]. Der Importer nimmt TEI-XML-Dateien entgegen und importiert sie in die Graphdatenbank.

2.2.2.1 Import

Befehl für den Import der Patzig-XML-Datei[9]:

```
CALL apoc.xml.import('http://www.deutschestextarchiv.de/
    book/download_xml/patzig_msgermfol841842_1828',
    {createNextWorkRelationships: true})
    yield node return node;
```

Dabei werden die XML-Knoten in Graphknoten umgewandelt und verschiedene Arten von Kanten erstellt, die einerseits die Baum-Hierarchie des XMLs im Graphen abbilden und andererseits die im XML vorhandenen Textknoten miteinerander verknüpfen[10]. Dabei werden die Wörter innerhalb der XML-Textknoten werden in Ketten von Wortknoten umgewandelt. Zur Abbildung des Wurzelelement der importieren XML-Datei wird ein Knoten vom Typ `XmlDocument` angelegt. Dieser erhält die Propertys `_xmlEncoding` zur Darstellung des Encodings, `_xmlVersion` für die Xml-Version und `url` für die URL des importierten XML-Dokuments.

Mit einem weiteren cypher-Query erhalten alle der importierten Knoten die Eigenschaft `url` mit der URL des importierten XML-Dokuments. Damit lassen sich Knoten in einer Graphdatenbank mit mehreren importierten XML-Dokumenten auseinanderhalten.

```
MATCH (d:XmlDocument)-[:NEXT_WORD*]->(w:XmlWord)
```

[8]Der generische XML-Import wird momentan von Stefan Armbruster entwickelt (https://github.com/sarmbruster/ abgerufen am 23.11.2017). Es ist geplant, den Importer in die apoc-Bibliothek zu integrieren. (https://github.com/neo4j-contrib/neo4j-apoc-procedures abgerufen am 23.11.2017).

[9]Für die Vereinheitlichung des Druckbildes mussten an einigen Stellen Zeilenumbrüche in die Codebeispiele eingefügt werden, die deren direkte Ausführung behindern.

[10]In TEI-XML gibt es zwei verschiedene Arten von Elementen. Die eine Klasse dient der Klassifizierung von Text, die zweite Art bringt Varianten und zusätzlichen Text mit!!! Literaturhinweis ergänzen, Hans-Werner Bartz fragen.

```
SET w.url = d.url;
```

Mit dem nächsten cypher-Query werden die Knoten des importierten XML-Dokuments durchnummertiert und der jeweilige Wert in der Property `DtaID` abgelegt.

```
MATCH p = (start:XmlDocument)-[:NEXT*]->(end:XmlTag)
WHERE NOT (end)-[:NEXT]->() AND start.url =  'http://www.deutschestextarchiv.de/bo
WITH nodes(p) as nodes, range(0, size(nodes(p))) AS indexes
UNWIND indexes AS index
SET (nodes[index]).DtaID = index;
```

2.2.2.2 Erläuterung der entstandenen Graphstrukturen

Nach Abschluss des Imports werden jetzt die importierten Datenstrukturen erläutert. In der folgenden Tabelle werden die verschiedenen Typen von Knoten erläutert, die während des Imports erstellt wurden.

Tabelle zum Importvorgang der XML-Elemente und den entsprechenden Knoten

XML-Knoten	Graphknoten	Bemerkungen
XML-Wurzelelement	XmlDocument	Gibt es nur einmal. Es enthält Angaben zur Encodierung, zur XML-Version und die URL der importierten XML-Datei
XML-Element-Knoten	XmlTag-Knoten	Die Attribute des XML-Elements werden in entsprechende Propertys des XMLTag-Knotens in der Datenbank umgewandelt
XML-Text-Knoten	XmlWord	Jedes Wort des XML-Textknotens wird ein XmlWord-Knoten im Graphen

In der nächsten Tabelle werden die verschiedenen Kantentypen erläutert, mit einen einerseits die Serialität des XMLs (`NEXT`-Kanten), und die Hierarchie (`NEXT_SIBLING` und `IS_CHILD_OF`-Kanten), andererseits aber auch die Abfolge der Inhalte der XML-Textelemente (`NEXT_WORD`) dargestellt werden.

Tabelle zu den erstellen Kantentypen

Kante	Bemerkungen
:NEXT	Zeigt die Serialität der XML-Datei im Graphen
:NEXT_SIBLING	Zeigt auf den nächten Graphknoten auf der gleichen XML-Hierarchie-Stufe
:NEXT_WORD	Zeigt auf das nächste Wort in einem XML-Textknoten
:IS_CHILD_OF	Zeigt auf den in der XML-Hierarchie übergeordneten Knoten
:FIRT_CHILD_OF	Zeigt vom ersten untergeordneten auf den übergeordneten Knoten.
:LAST_CHILD_OF	Zeigt vom letzten untergeordneten auf den übergeordneten Knoten.

Die folgende Abbildung zeigt einen kleinen Ausschnitt aus der TEI-XML-Datei der Patzig-Vorlesungsmitschrift.

```
rendition="#aq">po&#x017F;thumi&#x017F;chen</hi>
Werke auf-<lb/>
gedeckt u. <subst><del rendition="#s">&#x017F;eine
Fehler</del><add
place="superlinear">die&#x017F;e</add></subst> zum
Theil erka<supplied reason="damage"
resp="#BF">n&#x0303;t,</supplied><lb/>
wen&#x0303; er redete von häßlichen u.
```

Figure 2: XML-Beispiel aus der TEI-XML-Datei der Patzig-Vorlesungsmitschrift.

Beim Import der XML-Datei in den Graphen die XML-Element-Knoten in Xml-Tag-Knoten In der Abbildung des XML-Ausschnittes sind jene Teile blau markiert, die sich auch in der Graphabbildung befinden. Aus Sicht der XML-Hierarchie befindet sich der XML-Textknoten mit dem Inhalt *gedeckt u.* auf der gleichen Ebene mit dem `<subst>`-Element. Dies wird beim

Mit dieser Modellierung lassen sich beispielsweise die von einem `<add>`-Element umfassten Wörter abfragen, in dem man ausgehend vom `add`-Knoten der `FIRT_CHILD_OF`-Kante rückwärts folgt, anschließend von diesem Knoten

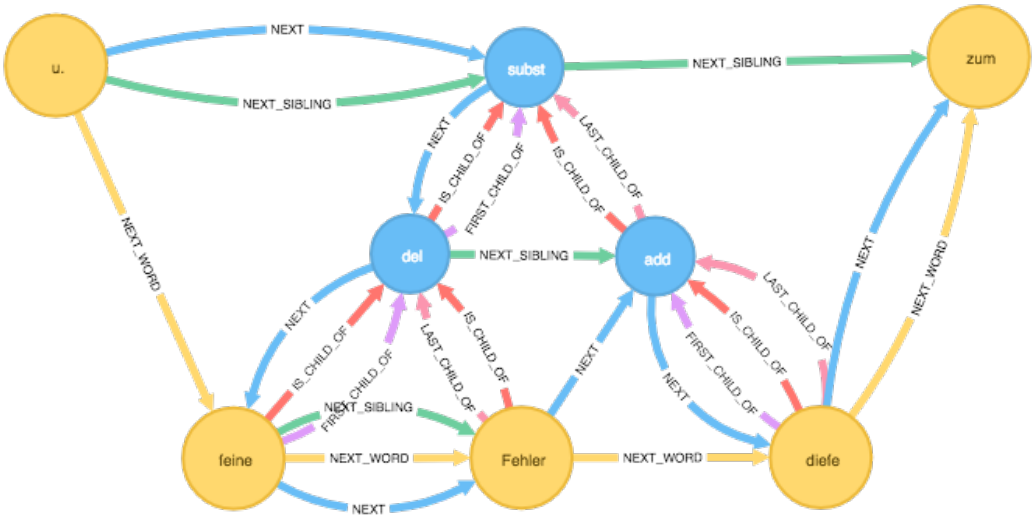

Figure 3: XML-Beispiel im Graphen.

den `NEXT_SIBLING`-Kanten so lange folgt, bis wieder die `LAST_CHILD_OF`-Kante wieder zum `add`-Knoten zurückführt. Der entsprechende cypher-Query sieht wie folgt aus:

```
MATCH
(n:XmlTag {_name:'add'})
<-[:FIRST_CHILD_OF]-(w1:XmlWord)
-[:NEXT_WORD*..5]->(w2:XmlWord)
-[:LAST_CHILD_OF]->(n)
RETURN * LIMIT 25
```

In einem zweiten Schritt kann der so entstandene Graph mit Hilfe von cypher-Querys weiter bearbeitet werden. Die Grahdatenbank neo4j ist schemafrei und somit können nun über die importieren XML-Strukturen weitere Erschließungsstrukturen gelegt werden, ohne dass ein XML-Parser sich über das nicht mehr wohlgeformte XML beschwert. Zu beachten ist bei jedem Schritt, ob wieder der Schritt zurück nach XML getätigt werden soll. Sicherlich ist es kein größeres Problem, eine in eine Graphdatenbank importierte XML-Datei wieder als solche zu exportieren. Ist der Graph aber mit weiteren Informationen angereichert, so muss geklärt werden, ob, und wenn ja wie, diese zusätzlichen Informationen in wohlgeformtes XML transferiert werden können.

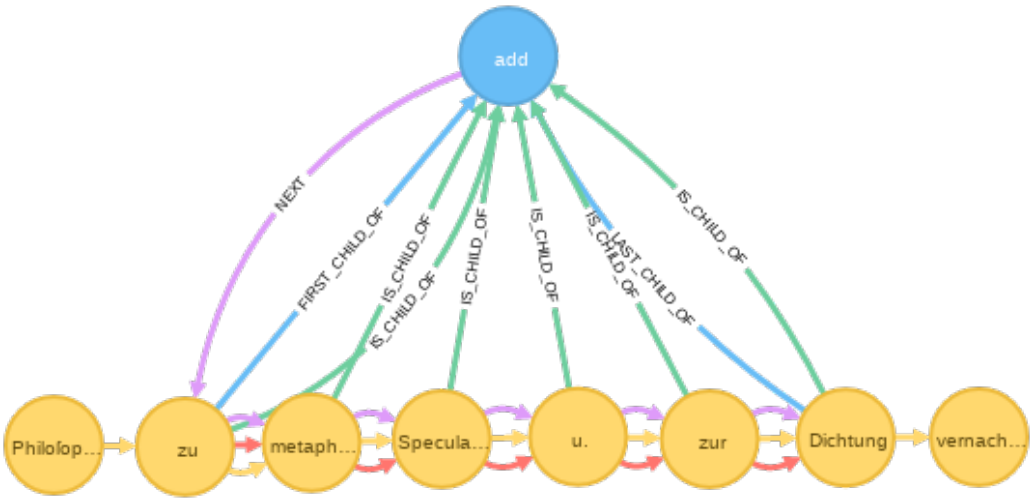

Figure 4: XML-Hierarchie eines `<add>`-Elements und der von ihm umfassten Wörter im Graphen.

3 Das DTA-Basisformat im Graphen

Das DTA-Basisformat ist ein Subset der TEI und bietet für Textphänomene jeweils nur eine Möglichkeit der Auszeichnung. Damit wird die in der TEI vorhandene Flexibilität bei der Auszeichnung eingeschränkt, um damit einen höheren Grad an Interoperabilität zu erreichen. Das DTA-Basisformat folgt den P5-Richtlinien der TEI, trifft aber eine Tag-Auswahl der für die Auszeichung historischer Texte notwendigen Elemente.

Im folgenden Abschnitt werden für ausgewählte Elemente des DTA-Basisformats mögliche Modellierungsformen im Graphen beschrieben. Zum äußeren Erscheinungsbild wird der Seitenfall sowie Spalten- und Zeilenumbrüche berücksichtigt. Bei den Textphänomenen werden Absätze, Schwer- und Unleserliches und inhaltlich werden die Kapiteleinteilung, inhaltliche Inline-Auszeichnungen und editorische Eingriffe behandelt. Für die Metadaten werden keine Modellierungsvorschläge formuliert, da diese sich sauber im XML-Baum darstellen lassen und keine Überlappungsprobleme etc. entstehen.

3.1 Strukturen des Dokuments

3.1.1 Graphenmodellierung von Zeilen

Nehmen wir als Beispiel Zeilenwechsel auf einer Seite des Patzig-Manuskripts (http://www.deutschestextarchiv.de/book/view/patzig_msgermfol841842_1828/?hl=Himalaja&p=

```
... Die<lb/>
Jnder hegen die große Verehrung vor<lb/>
dem Himalaja Gebirge. ...
```

Im Graphen sieht die Stelle wie folgt aus:

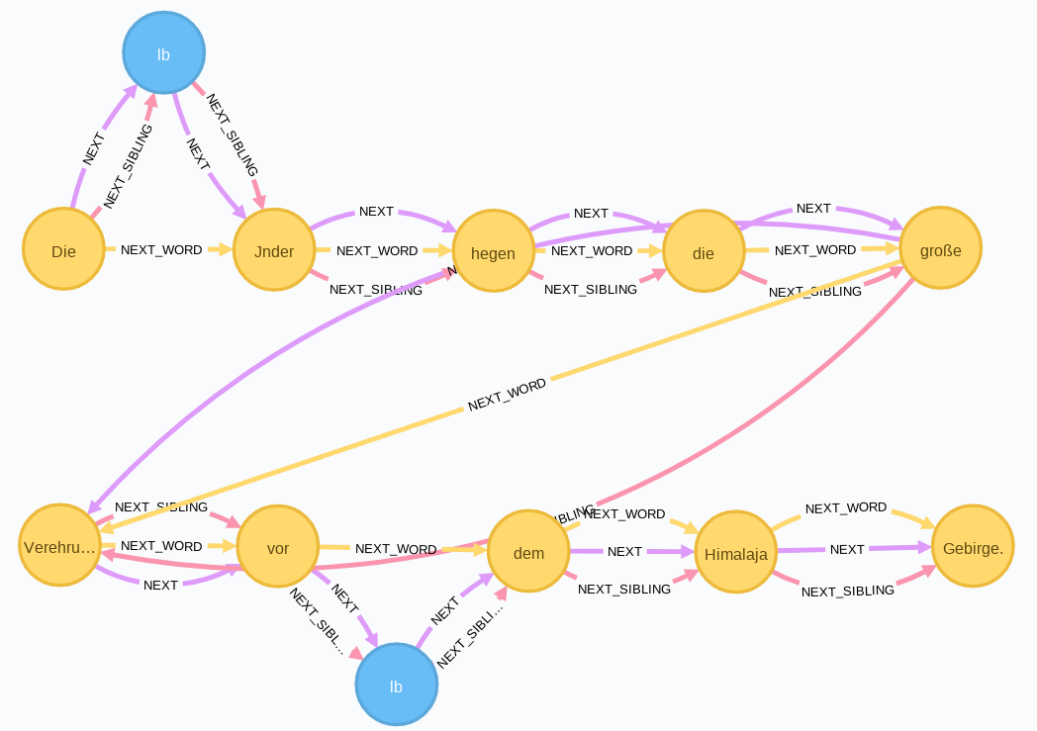

Figure 5: `<lb/>`-Element im Graphen

Das leere `<lb/>`-Element steht für die Markierung eines Zeilenanfangs (*line begins*). Der Graph soll nun so umgebaut werden, dass die Zeile durch einen

[11] URL des Beispieltextes: http://www.deutschestextarchiv.de/book/view/patzig_msgermfol841842_1828/?hl=Him abgerufen am 02.01.2018.

`line`-Knoten gekennzeichnet wird, von dem aus eine `FIRST_CHILD_OF`-Kante mit dem ersten Wort der Zeile und eine `LAST_CHILD_OF`-Kante mit dem letzten Wort der Zeile verbunden ist.

Mit dem folgenden cypher-query kommt man den auf der Abbildung sichtbaren Subgraphen:

```
MATCH (n0:XmlWord)-[:NEXT_WORD]->
(n1:XmlWord {DtaID:10272})-[:NEXT_WORD*..8]->(n2:XmlWord),
(n1)<-[:NEXT]-(t1:XmlTag {_name:'lb'}),
(n3:XmlWord {DtaID:10277})-[:NEXT]->(t2:XmlTag {_name:'lb'})
RETURN * LIMIT 20;
```

Im folgenden Schritt wird ein `line`-Knoten erzeugt, der die Zeile darstellen soll. Mit diesem werden dann das erste und das letzte Wort der Zeile verbunden.

```
MATCH (n0:XmlWord)-[:NEXT_WORD]->
(n1:XmlWord {DtaID:10272})-[:NEXT_WORD*..8]->(n2:XmlWord),
(n1)<-[:NEXT]-(t1:XmlTag {_name:'lb'}),
(n3:XmlWord {DtaID:10277})-[:NEXT]->(t2:XmlTag {_name:'lb'})
MERGE (n3)<-[:LAST_CHILD_OF]-(l:line {name:'line'})-[:FIRST_CHILD_OF]->(n1)
DETACH DELETE t1, t2
RETURN * LIMIT 20;
```

Im Graphen sieht die Stelle wie folgt aus:

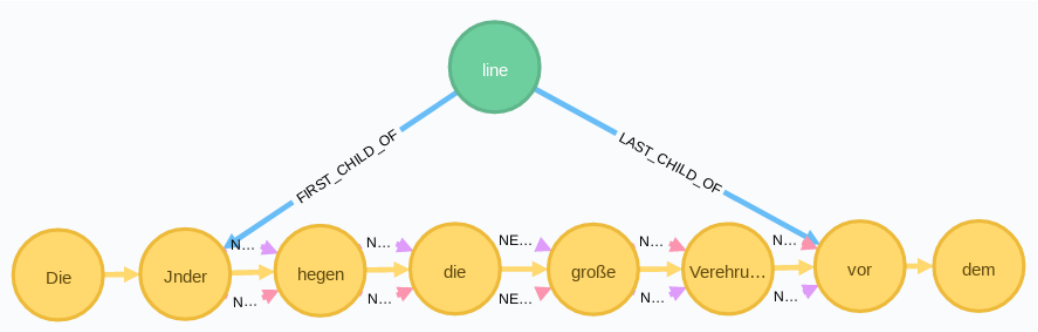

Figure 6: `<lb/>`-Element im Graphen

3.1.2 Zeilenwechsel mit Worttrennungen

Nun kommt es im Bereich der Zeilenwechsel sehr häufig zu Worttrennungen. Als Beispiel nehmen wir folgende Zeile, die sich auf der gleichen Seite wo das eben behandelte Beispiel befindet:

```
... Die Ken&#x0303;t-<lb/>
niß des Jahres durch nicht von einer Nation<lb/>
auf ...
```

Im Graphen sieht die Stelle wie folgt aus:

Mit dem folgenden cypher-query kommt man den auf der Abbildung sichtbaren Supgraphen:

```
MATCH (n0:XmlWord {DtaID:10197})-[:NEXT_WORD]->
(n1:XmlWord)-[:NEXT_WORD*..9]->(n2:XmlWord),
(n1)-[:NEXT]->(t1:XmlTag {_name:'lb'}),
(n3:XmlWord {DtaID:10207})-[:NEXT]->(t2:XmlTag {_name:'lb'})
RETURN * LIMIT 20;
```

Das `<lb/>`-Element trennt das Wort Kenntniß[12]. Im nächsten Schritt werden nun die beiden getrennten Wortknoten `Kennt-` und `niß` im zweiten Wortknoten `niß` zusammengefasst. Der erste Wortknoten `Kennt-` inkl. seiner Kanten wird gelöscht und eine neue NEXT-Kante zwischen dem `niß`-Wortknoten und dem vorhergehenden `Die`-Wortknoten erstellt. Die Informationen, an welcher Stelle das Wort getrennt war, wird in den Eigenschaften des neuen `Kenntniß`-Wortnnotens gespeichert. In der Eigenschaft `before` steht dann der Inhalt des ursprünlich ersten Wortknotens `Kennt-` und in der Eigenschaft after der Inhalt des ursprünglich zweiten Wortknotens `niß`.

Hier werden die notwendigen Cypher-Befehle angezeigt:

```
MATCH (n0:XmlWord {DtaID:10197})-[:NEXT_WORD]->
(n1:XmlWord {DtaID:10198})-[:NEXT_WORD]->
(n2:XmlWord {DtaID:10200})-[:NEXT_WORD*..8]->(n3:XmlWord {DtaID:10207}),
(n1)-[:NEXT]->(t1:XmlTag {_name:'lb'}),
(n4:XmlWord {DtaID:10207})-[:NEXT]->(t2:XmlTag {_name:'lb'})
SET n2.before = left(n1.text, size(n1.text)-1)
```

[12] Zur einfacheren Lesbarkeit wurden im Wort *Kenntniß* die Sonderzeichen normalisiert.

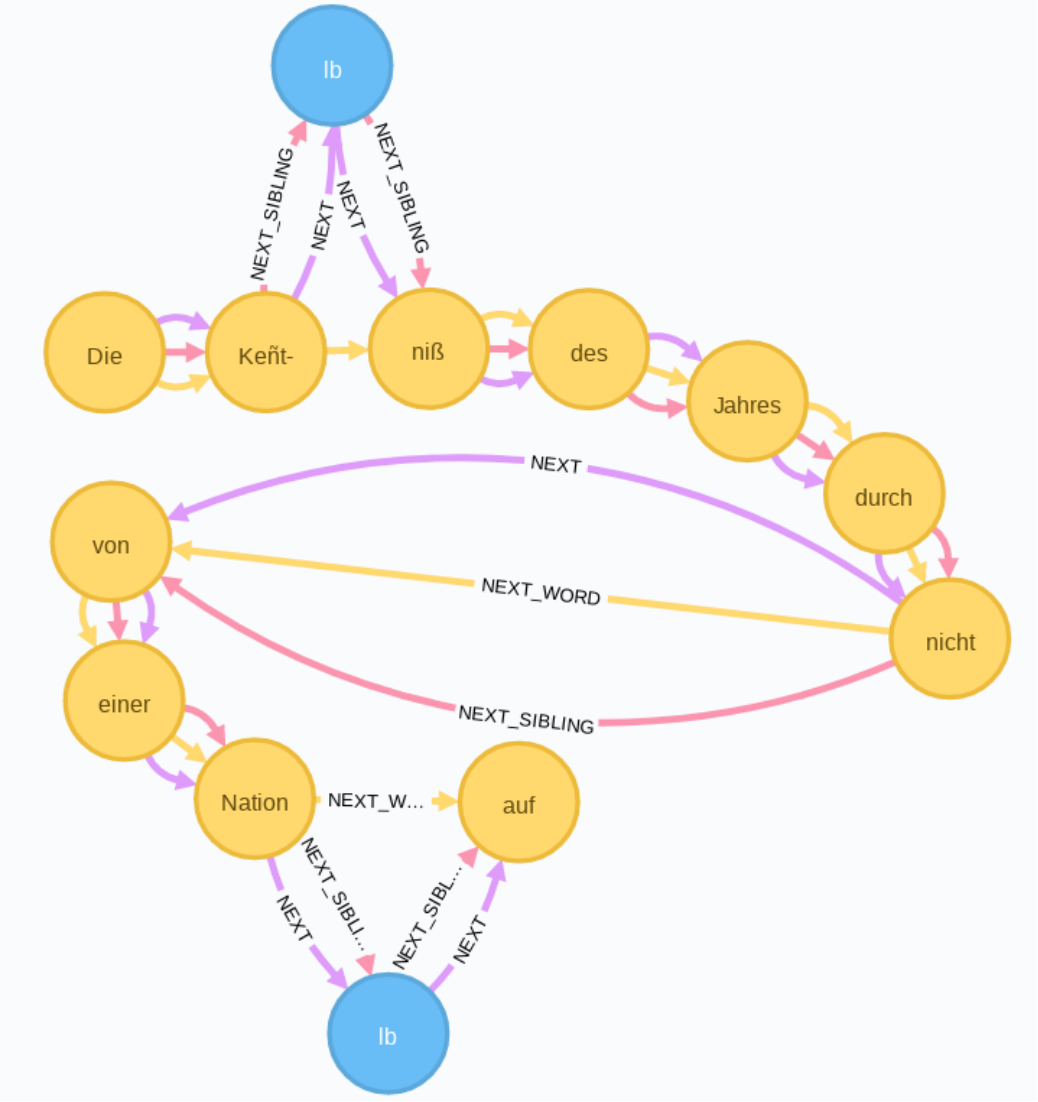

Figure 7: `<lb/>`-Element im Graphen

```
SET n2.after = n2.text
SET n2.text = left(n1.text, size(n1.text)-1)+n2.after
MERGE (n0)-[:NEXT_WORD]->(n2)
MERGE (n4)<-[:LAST_CHILD_OF]-(l:line {name:'line'})-[:FIRST_CHILD_OF]->(n2)
DETACH DELETE t1, t2, n1
RETURN * LIMIT 20;
```

Im Graphen ergibt sich anschließend folgendes Bild:

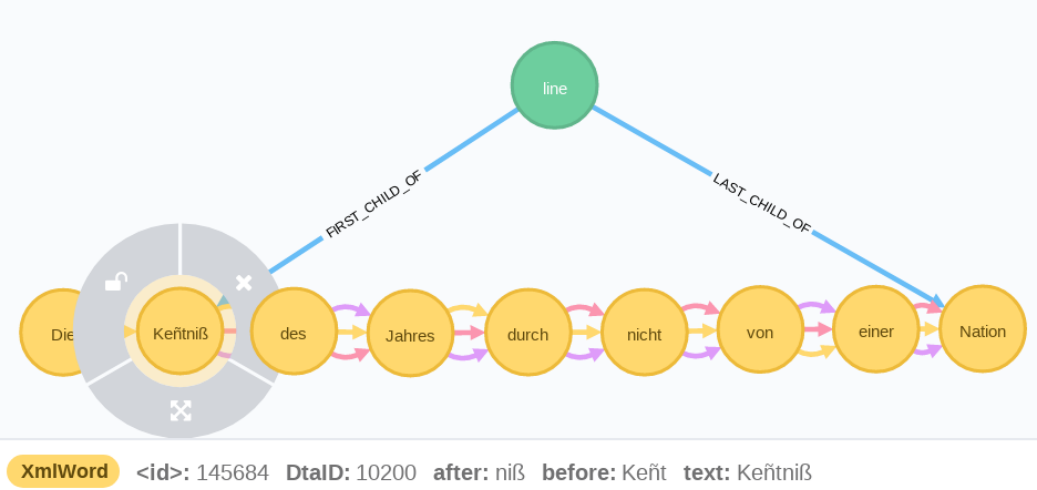

Figure 8: `<lb/>`-Element im Graphen herausgenommen, Wortknoten zusammengefasst

Am unteren Bereich der Abbildung sind in der Legende die Propertys des Wortknotens *Kentniß* hervorgehoben. Dort erkennt man die vorher vorhandenen Wortbestandteile und den neuen Wert der Property *text*.

3.1.3 Seitenzahlen und Faksimilezählung

Im DTA-Bf wird jeweils der Anfang einer Seite mit dem leeren Element `<pb>` markiert[13]. Das leere Element kann noch die Attribute `facs` für die Zählung

[13] Vgl. die Dokumentation des DTA-Basisformats unter http://www.deutschestextarchiv.de/doku/basisformat/seitenFacsNr.html abgerufen am 25.11.2017.

der Faksimileseiten und `n` für die auf der Seite ggf. angegebene Seitenzahl enthalten.

```
<pb facs="#f[Bildnummer]" n="[Seitenzahl]"/>
```

Ist eine Seitenzahl im Faksimile falsch wiedergegeben, so wird diese originalgetreu übernommen und die richtige Seitenzahl in eckigen hinzugefügt in das `n`-Attribut übernommen.

```
<pb facs="#f[Bildnummer]" n="[fehlerhafte Seitenzahl [korrigierte Seitenzahl]]"/>
```

Das `<pb/>`-Element auf den Seiten 5 und 6 aus Patzig (http://www.deutschestextarchiv.de/book/view

```
... Abwe&#x017F;enheit vom heimi&#x017F;chen Boden ent-<lb/>
<note place="left"><figure type="stamp"/><lb/>
</note>fernt hielt, der &#x017F;ich viel mit einem Volke<lb/>
<fw place="bottom" type="catch">befreun-</fw><lb/>
<pb facs="#f0006" n="2."/>
befreundete, welches durch den
...
in einzelnen großen Zügen zu ent-<lb/>
werfen</hi>.</p><lb/>
<fw place="bottom" type="catch">Nachdem</fw><lb/>
<pb facs="#f0007" n="3."/>
<p><note place="left"><hi rendition="#u">Neue&#x017F;te A&#x017F;tronomi&#x017F;c
deckungen.</hi><lb/> ...
```

Im Graphen findet man das `<pb>`-Element der Seite 6 mit folgendem Query[ˆa825]:

```
MATCH
(n1:XmlWord {DtaID:869})-[:NEXT]->
(lb1:XmlTag {_name:'lb'})-[:NEXT]->
(t2:XmlTag {_name:'fw', place:'bottom', type:'catch'})-[:NEXT_SIBLING]->
(lb2:XmlTag {_name:'lb'})-[:NEXT_SIBLING]->
(pb:XmlTag {_name:'pb'}),
(n1:XmlWord)-[nw1:NEXT_WORD]->
(n2:XmlWord)-[nw2:NEXT_WORD]->
(n3:XmlWord)-[nw3:NEXT_WORD]->
```

[14] Die Beispielseite findet sich unter http://www.deutschestextarchiv.de/book/view/patzig_msgermfol841842_1828/ abgerufen am 25.11.2017.

```
(n4:XmlWord),
(n2:XmlWord)-[:NEXT]->(t1:XmlTag {_name:'lb'})
RETURN * LIMIT 20;
```

Im Graphen ergibt sich folgendes Bild:

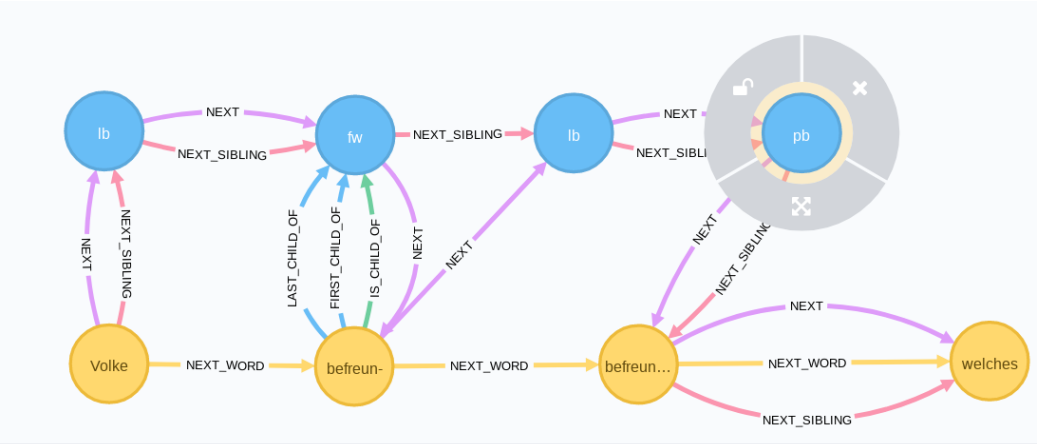

Figure 9: Der Pfad vom `<pb/>`-Element zum ersten Wort der Seite *befreundet*.

Markiert ist das `<pb/>`-Element der Seite 6. Im Fuß der Abbildung werden die Propertys des Elements angezeigt. Der Textfluss wird durch den Wortknoten `befreun-` unterbrochen, der eine Kustode darstellt. Diese soll aus dem Textfluss herausgelöst und direkt mit dem letzten Wortknoten `Volke` über die neu eingeführte `catch_words`-Kante verbunden werden. Der `<fw>`, und der `<lb/>`-Knoten werden gelöscht und der letzte Wortknoten der Seite über eine neue `NEXT`-Kante mit dem `<pb/>`-Knoten verknüpft.

Hier der Query für den Umbau:

```
MATCH
(n1:XmlWord {DtaID:869})-[:NEXT]->
(lb1:XmlTag {_name:'lb'})-[:NEXT]->
(t2:XmlTag {_name:'fw', place:'bottom', type:'catch'})-[:NEXT_SIBLING]->
(lb2:XmlTag {_name:'lb'})-[:NEXT_SIBLING]->
(pb:XmlTag {_name:'pb'}),
(n1:XmlWord)-[nw1:NEXT_WORD]->
(n2:XmlWord)-[nw2:NEXT_WORD]->
```

```
(n3:XmlWord)-[nw3:NEXT_WORD]->
(n4:XmlWord),
(n2:XmlWord)-[:NEXT]->(t1:XmlTag {_name:'lb'})
DELETE nw1, nw2
DETACH DELETE t2
MERGE (n1)-[:NEXT_WORD]->(n3)
MERGE (n1)-[:CATCH_WORDS]->(n2)
MERGE (n1)-[:NEXT_WORD]->(n3)
MERGE (lb1)-[:NEXT]->(n2)
RETURN * LIMIT 20;
```

Im Graphen ergibt sich folgendes Bild:

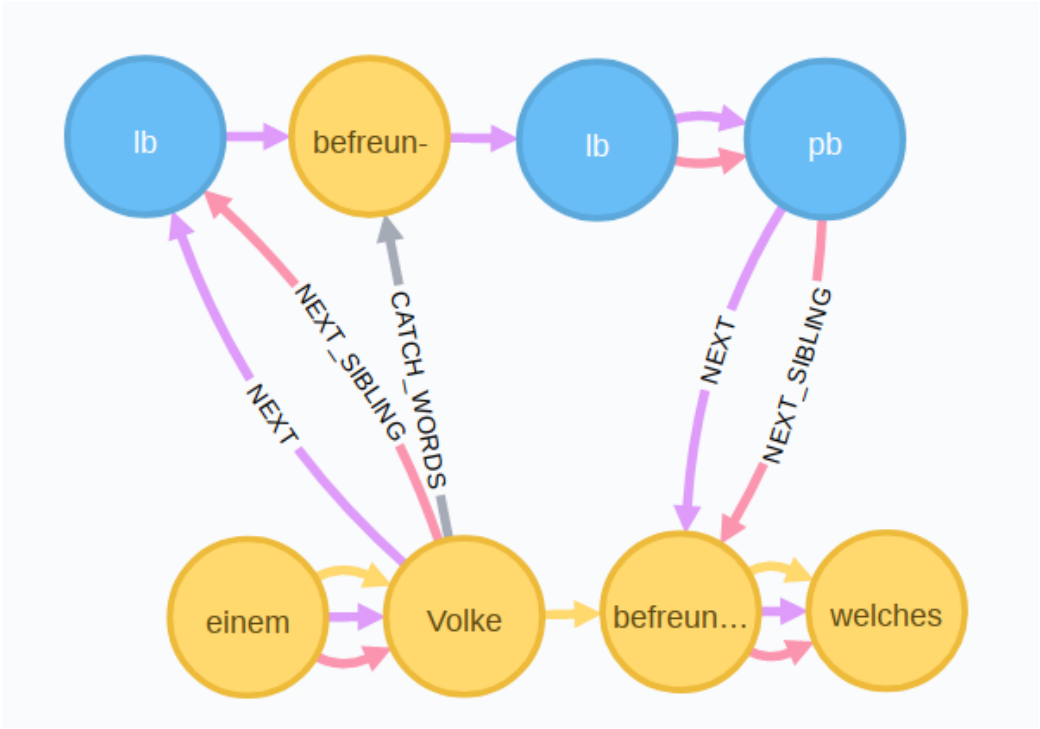

Figure 10: Die Kustode *befreun-* wird aus der `NEXT_WORD`-Textkette herausgenommen und über eine `CATCH_WORDS`-Kante mit dem Wortknoten *Volke* verknüpft.

Die Kustode ist nun nicht mehr über `NEXT_WORD`-Kanten mit dem Fließtext verknüpft, bleibt aber über die `CATCH_WORDS`-Kante mit dem letzten Wort

der Seite verbunden. In einem zweiten Schritt müssen nun die beiden <pb/>-Elementknoten zu einem neu einzuführenden page-Knoten zusammengeführt werden. Hierfür lassen wir uns im nächsten cypher-Query alle <pb/>-Knoten mit einer DtaID kleiner als 875 anzeigen, da diese vor dem <pb/>-Knoten der Seite 6 mit der DtaID 874 liegen:

```
MATCH (n:XmlTag {_name:'pb'})
WHERE n.DtaID < 875
RETURN n;
```

"n"
{"_name":"pb","facs":"#f0001","DtaID":444}
{"_name":"pb","facs":"#f0002","DtaID":445}
{"_name":"pb","facs":"#f0003","DtaID":455}
{"_name":"pb","facs":"#f0004","DtaID":558}
{"_name":"pb","facs":"#f0005","DtaID":562,"n":"1."}
{"_name":"pb","facs":"#f0006","DtaID":874,"n":"2."}

Figure 11: Tabellenansicht aller <pb/>-Knoten mit einer DtaID kleiner als 875.

Aus der Tabellenansicht ist zu entnehmen, dass Seite 5 von den <pb/>-Elementen mit der DtaID 562 und 874 eingefasst wird.

Der cypher-Query zum Einfügen des page-Knoten sieht wie folgt aus:

```
MATCH
(pb1:XmlTag {DtaID:562, _name:'pb'})-[n1:NEXT*..5]->(w1:XmlWord {DtaID:565}),
(pb2:XmlTag {DtaID:874, _name:'pb'})<-[n2:NEXT*..5]-(w2:XmlWord {DtaID:872})
MERGE
```

```
(w1)<-[:FIRST_CHILD_OF]-(page:page {facs:'#f0005', n:1})-[:LAST_CHILD_OF]->(w2)
RETURN pb1, w1, pb2, w2, page;
```

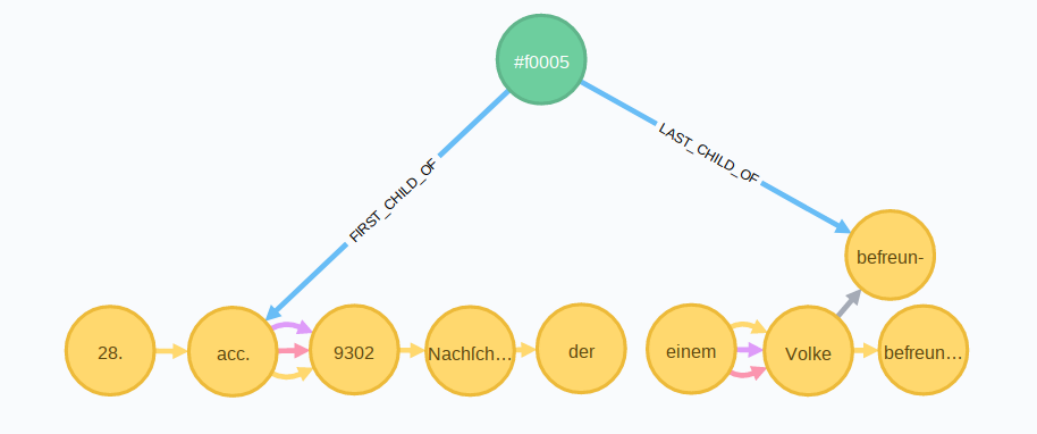

Figure 12: Die Seite wird modelliert mit dem `page`-Knoten #0005 der mit dem ersten Wort über eine `FIRST_CHILD_OF`- und mit dem letzten Wort der Seite über eine `LAST_CHILD_OF`-Kante verknüpft ist.[16]

3.1.4 Absätze

Absätze werden im DTA-Basisformat mit dem `<p>`-Element eingefasst. Im Manusskript von Patzig finden sich insgesamt 238 mit dem `<p>`-Element eingefasste Textabschnitte[17].

```
<p>Geogno&#x017F;ie, <choice><orig>Metereologie</orig><reg resp="#BF">Meteorologie</reg></choice> u. Phy&#x017F;iologie<lb/>
der <choice><abbr>Pflanz&#xFFFC;</abbr><expan resp="#BF">Pflanzen</expan></choice> haben mich den größten Theil<lb/>
meines Lebens be&#x017F;chäftigt u. &#x017F;o hoffe ich<lb/>
in die&#x017F;en Gegen&#x017F;tänden mich &#x017F;o deutlich<lb/>
zu mache, <choice><abbr>dß</abbr><expan resp="#BF">daß</expan></choice> auch die mit mindern <choice><abbr>Vorken&#x0303;t-</lb/>
ni&#x017F;&#x017F;&#xFFFC;</abbr><expan resp="#BF">Vorken&#x0303;t-<lb/>
ni&#x017F;&#x017F;en</expan></choice> meinen Vorträgen folgen kön&#x0303;en:</p><lb/>
```

Figure 13: XML-Auszug aus Patzig mit einem Absatz als Beispiel.

Da das `<p>`-Element im Unterschied zu den leeren Elementen wie `pb` oder `lb` ein öffnendes und schließendes Tag hat, wird beim Import der TEI-

[16] Die Darstellung der Wortkette ist zwischen den Wortknoten *der* und *einem* zu Gunsten der Übersichtlichkeit gekürzt.

[17] Die Anzahl der `<p>`-Elemente im Graph erhält man mit der Abfrage MATCH (n:XmlTag {_name:'p'}) RETURN count(n);

Xml-Datei durch den Importer schon ein `p`-Knoten erstellt, der mit einer `FIRST_CHILD_OF`-Kante mit dem ersten Wort des Absatzes und mit einer `LAST_CHILD_OF`-Kante mit dem letzten Wort des Absatzes verknüpft ist.

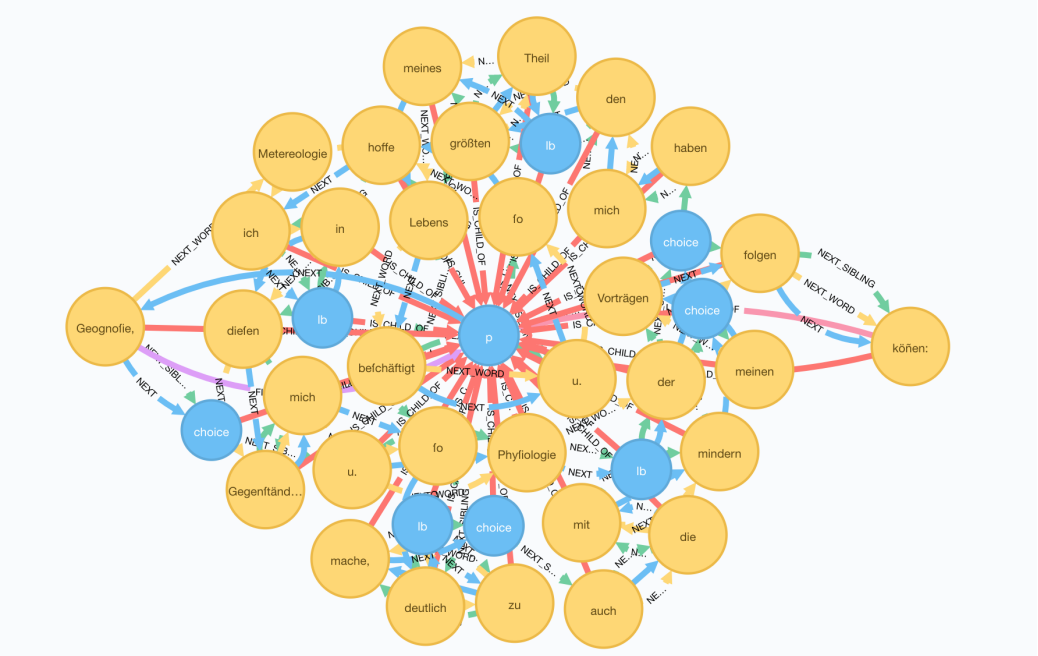

Figure 14: Ein Teil des gleichen Absatzes aus Patzig im Graphen.

Alle Wörter eines Absatzes sind darüber hinaus über `NEXT_SIBLING`-Kanten in der Textreihenfolge verknüpft.

3.1.5 Kapiteleinteilung

Im DTA-Basisformat wird bei der Transkiption von Büchern die Kapiteleinteilung mit verschachtelten `div`-Element vorgenommen. Das im `div`-Element erlaubte @n-Attribut gibt die Strukturebene an. Über das @type-Attribut kann der Typ des Kapitels näher spezifiziert werden. Eine Liste der möglichen Werte für das Attribut findet sich unter http://deutschestextarchiv.de/doku/basisformat/div.html.

Für Manuskripte, wie der hier behandelten Vorlesungsmitschrift von Patzig gibt es unter http://deutschestextarchiv.de/doku/basisformat/msKapitel.html

noch zwei zusätzliche Werte für das @type-Attribut, nämlich *session* für Vorlesungsmitschriften und *letter* für Briefe.

Mit folgendem cypher-Query erhalten wir die in Patzig verwendeten Werte für das @type-Attribut des `div`-Elements.

```
MATCH (n:XmlTag {_name:'div'})
RETURN n.type, count(n.type) AS Anzahl ORDER BY Anzahl DESC;
```

n.type	Anzahl
session	62
null	0

Es sind also insgesamt 62 Kapitel vom Typ *session* (Vorlesungsmitschrift) enthalten. Mit folgendem cypher-Query wird die Kapitelstruktur der ersten Kapitel und der darunter liegenden Ebenen bis zum jeweils ersten und letzten Wort des Kapitels angezeigt.

```
MATCH
p1 = shortestPath(
    (div:XmlTag {_name:'div'})<-[:FIRST_CHILD_OF*..20]-(w1:XmlWord)),
p2 = shortestPath(
    (div:XmlTag {_name:'div'})<-[:LAST_CHILD_OF*..20]-(w2:XmlWord))
RETURN p1,p2 LIMIT 20;
```

Mit dem folgenden cypher-Query wird das erste Wort des Kapitels über eine `FIRST_CHILD_OF`-Kante und das letzte Wort des Absatzes über eine `LAST_CHILD_OF`-Kante mit dem `div`-Knoten verbunden. Um die neu erstellen Kanten von den vom Importer erstellen zu unterscheiden erhalten diese die Proptery *type* mit dem Wert *graph*. Um die `div`-Knoten von den anderen `XmlTag`-Knoten unterscheiden zu können erhalten sie das zusätzliche Label *Session*.

```
MATCH
p1 = shortestPath(
    (div:XmlTag {_name:'div'})<-[:FIRST_CHILD_OF*..20]-(w1:XmlWord)
    ),
p2 = shortestPath(
    (div:XmlTag {_name:'div'})<-[:LAST_CHILD_OF*..20]-(w2:XmlWord)
```

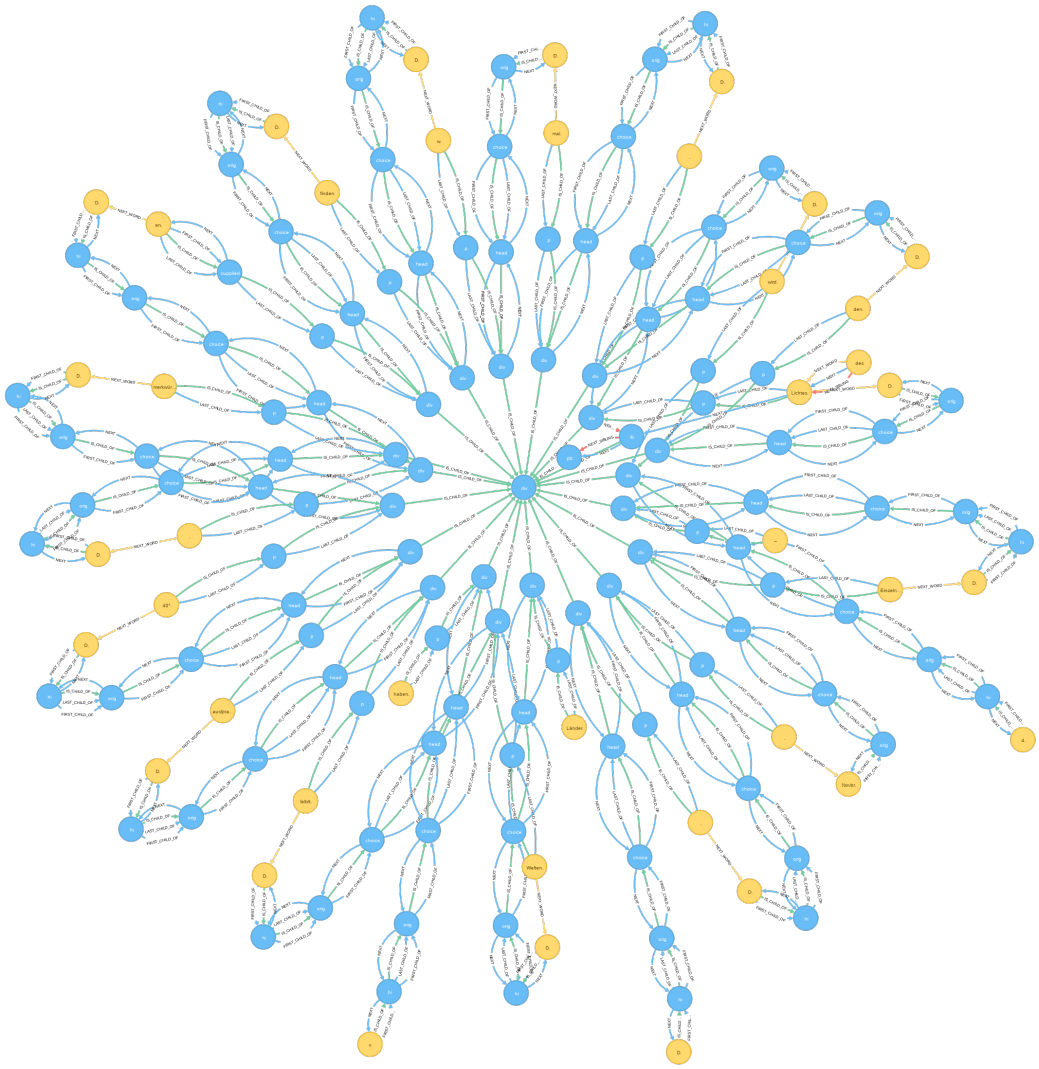

Figure 15: Struktur der ersten Kapitel mit dem jeweils ersten und letzten Wort.

```
    )
MERGE (w1)-[:FIRST_CHILD_OF {type:'graph'}]->
    (div)<-[:LAST_CHILD_OF {type:'graph'}]-(w2)
SET div:Session
RETURN * LIMIT 20;
```

3.1.6 Zusammenfassung

In diesem Kapitel wurden exemplarisch die XML-Strukturen für Zeilen (`lb`), Seiten (`pb`), Absätze (`p`) und Kapitel (`div`) in Graphstrukturen überführt, in denen jedes Element nur noch aus einem Knoten besteht. Mit diesem Knoten wird jeweils das erste und das letzte betroffene Wort mit einer `FIRST_CHILD_OF`- und einer `LAST_CHILD_OF`-Kante verknüpft. Damit entstehen offensichtlich überlappende Strukturen, was im Graphen aber kein Problem darstellt.

3.2 Editorische Eingriffe

3.2.1 Hinzufügungen und Tilgungen

Die Elemente `<add>` und `<del>` werden für Kennzeichnung von Tilgungen und Hinzufügungen des Autors oder von späteren Bearbeitern verwendet.

3.2.1.1 `<add>`-Element

Dabei können die Umstände der Änderungen beim `<add>`-Element mit dem @place-Attribut näher beschrieben, welches die in der folgenden Tabelle angegebenen Werte annehmen darf[18]:

Element	@place-Wert	Bedeutung
`<add>`	superlinear	über der Zeile eingetragen
`<add>`	sublinear	unter der Zeile eingetragen
`<add>`	intralinear	innerhalb der Zeile eingetragen
`<add>`	across	über den ursprünglichen Text geschrieben

[18] Vgl. hierzu http://deutschestextarchiv.de/doku/basisformat/msAddDel.html.

Element	@place-Wert	Bedeutung
`<add>`	left	am linken Rand eingetragen
`<add>`	right	am rechten Rand eingetragen

Mit folgenden cypher-Query erhält man die Verteilung der Attributwerte.

```
MATCH (n:XmlTag {_name:'add'})
RETURN n.place, count(n.place) AS Anzahl ORDER BY Anzahl DESC;
```

n.place	Anzahl
across	436
superlinear	268
intralinear	60
left	16
sublinear	2

3.2.1.2 `<del>`-Element

Die mit dem `<del>`-Element gekennzeichneten Tilgungen können mit dem @rendition-Attribut näher beschrieben werden, dessen mögliche Werte in der folgenden Tabelle angegeben sind[19].

Element	@rendition-Wert	Bedeutung
`<del>`	#ow	Tilgung durch Überschreibung des ursprünglichen Textes
`<del>`	#s	Tilgung durch Streichung
`<del>`	#erased	Tilgung durch Radieren, Auskratzen

Mit folgenden cypher-Query erhält man die Verteilung der Attributwerte.

```
MATCH (n:XmlTag {_name:'add'})
RETURN n.rendition, count(n.rendition) AS Anzahl
ORDER BY Anzahl DESC;
```

[19] Vgl. hierzu http://deutschestextarchiv.de/doku/basisformat/msAddDel.html.

n.rendition	Anzahl
#ow	436
#s	268
#erased	60

3.2.1.3 Umbau von <add>- und <del>-Elementen in einer <subst>-Umgebung

Der Umbau wird an einem Beispieltext der Seite 32 des Patzig-Manuskripts durchgeführt[20].

```
rendition="#aq">po&#x017F;thumi&#x017F;chen</hi>
Werke auf-<lb/>
gedeckt u. <subst><del rendition="#s">&#x017F;eine
Fehler</del><add
place="superlinear">die&#x017F;e</add></subst> zum
Theil erka<supplied reason="damage"
resp="#BF">n&#x0303;t,</supplied><lb/>
wen&#x0303; er redete von häßlichen u.
```

Figure 16: -Beispiel in der XML-Ansicht.

Im Graphen findet man die entsprechende Stelle mit folgendem cypher-Query.

```
MATCH
(w1:XmlWord)-[r1:NEXT_WORD]->
(w2:XmlWord)-[r2:FIRST_CHILD_OF]->
(t1)-[r3:FIRST_CHILD_OF]->
(s:XmlTag {_name:'subst', DtaID:8248})
<-[r4:LAST_CHILD_OF]-(t2)
<-[r5:LAST_CHILD_OF]-(w4:XmlWord)
-[r6:NEXT_WORD]->(w5:XmlWord),
(w2)-[r7:NEXT_WORD]->(w3)-[r8:NEXT_WORD]->(w4)
RETURN *;
```

Der Query gruppiert sich um den `s`-Knoten, der das `subst`-Element darstellt und es über die DtaID identifiziert. Vom `s`-Knoten ausgehend,

[20] Vgl. http://www.deutschestextarchiv.de/book/view/patzig_msgermfol841842_1828/?hl=zum&p=32.

folgt der Pfad einerseits über `FIRST_CHILD_OF`-Kanten zum `n3`-Knoten (add-Element) und zum `n2`-Knoten, der schließlich das Wort *seine* darstellt. Über die `LAST_CHILD_OF`-Kante geht es zum `n4`-Knoten (del-Element) zum `n5`-Wortknoten, der das Wort *diese* darstellt. Im zweiten Teil des MATCH-Befehls wird der Pfad zwischen dem Wort *seine* und *diese* ermittelt und schließlich alles ausgegeben.

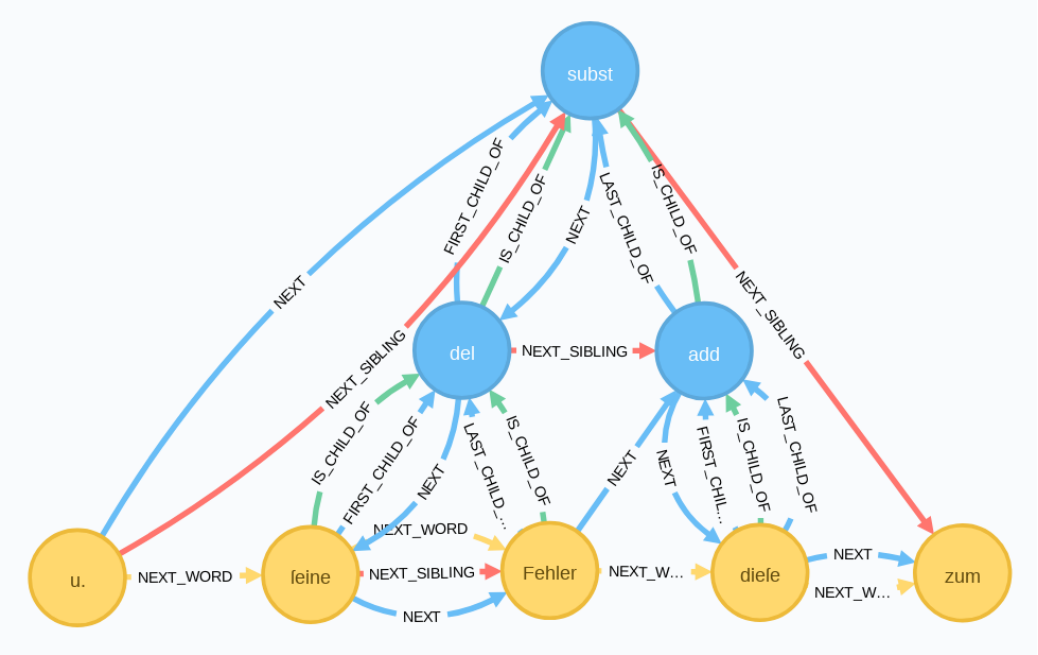

Figure 17: -Beispiel in der Graph-Ansicht.

cyper-Query für den umgebaut

```
MATCH
(w1:XmlWord)-[r1:NEXT_WORD]->
(w2:XmlWord)-[r2:FIRST_CHILD_OF]->
(t1)-[r3:FIRST_CHILD_OF]->
(s:XmlTag {_name:'subst', DtaID:8248})
<-[r4:LAST_CHILD_OF]-(t2)
<-[r5:LAST_CHILD_OF]-(w4:XmlWord)
-[r6:NEXT_WORD]->(w5:XmlWord),
(w2)-[r7:NEXT_WORD]->(w3)-[r8:NEXT_WORD]->(w4)
DELETE r1, r8
```

```
SET r8.variant_type='add'
CREATE (w1)-[:NEXT_WORD{variant_type:'add'}]->(w4)
CREATE (w1)-[:NEXT_WORD{variant_type:'del'}]->(w2)
CREATE (w3)-[:NEXT_WORD{variant_type:'del'}]->(w5)
SET r7.variant_type='del'
RETURN *;
```

Das Ergebnis erhält man über den folgenden Query.

```
MATCH
(w1:XmlWord)-[r1:NEXT_WORD]->
(w2:XmlWord)-[r2:FIRST_CHILD_OF]->
(t1)-[r3:FIRST_CHILD_OF]->
(s:XmlTag {_name:'subst', DtaID:8248})
<-[r4:LAST_CHILD_OF]-(t2)
<-[r5:LAST_CHILD_OF]-(w4:XmlWord)
-[r6:NEXT_WORD]->(w5:XmlWord),
(w2)-[r7:NEXT_WORD]->(w3)-[r8:NEXT_WORD]->(w4)
```

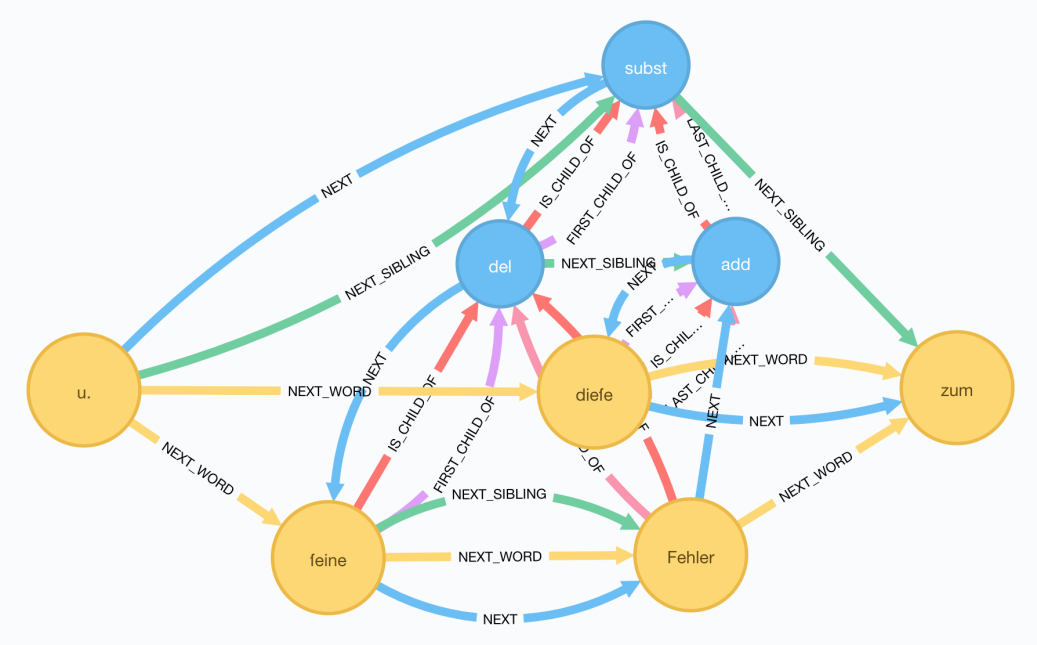

Figure 18: <subst>-Beispiel nach dem Graph-Umbau.

3.2.2 <choice>-Element

3.2.3 <sic> und <corr>-Elemente

4 Anhang

4.1 cypher-Befehle für den Import der Mitschrift von Patzig

Mit den folgenden Befehlen wird die Humboldt-Mitschrift von Patzig in die Graphdatenbank importiert, jedem Knoten zur Identifikation die DTA-URL als Propterty mitgegeben und die Knoten durchnummeriert. Die Nummerierung ist für das wiederholte Auffinden der in diesem Beitrag behandelten Textstellen notwendig.

```
// Alles löschen
MATCH(n) DETACH DELETE n;

// Patzig importieren
call
apoc.xml.import('http://www.deutschestextarchiv.de/book/download_xml/patzig_msgerm
yield node return node;

// URL von Dokument auf alle Wort-Knoten kopieren:
match (d:XmlDocument)-[:NEXT_WORD*]->(w:XmlWord)
set w.url = d.url;

// Knoten durchzählen
MATCH p = (start:XmlDocument)-[:NEXT*]->(end:XmlTag)
WHERE NOT (end)-[:NEXT]->() AND start.url = 'http://www.deutschestextarchiv.de/boo
WITH nodes(p) as nodes, range(0, size(nodes(p))) AS indexes
UNWIND indexes AS index
SET (nodes[index]).DtaID = index;
```

4.2 Liste aller im Patzig-Manusskript vorkommenden Elemente sortiert nach Häufigkeit

cypher-Query zur Erstellung der Tabelle:

```
MATCH (n:XmlTag)
RETURN n._name,
count(n._name) AS Anzahl
ORDER BY Anzahl DESC;
```

Element	Anzahl
lb	16075
hi	3768
choice	2184
expan	1856
abbr	1856
supplied	1517
persName	925
note	914
add	782
del	644
unclear	526
subst	516
pb	422
gap	419
fw	357
p	238
metamark	234
reg	221
orig	221
corr	107
sic	107
div	64
head	63
item	51
rendition	22
space	20
ref	19

Element	Anzahl
bibl	15
list	13
idno	12
surname	12
forename	12
figure	10
milestone	9
orgName	7
resp	6
respStmt	6
editor	6
cell	6
measure	4
titlePart	4
classCode	4
handNote	4
date	2
edition	2
titleStmt	2
editionStmt	2
publisher	2
publicationStmt	2
title	2
author	2
pubPlace	2
row	2
biblFull	1
fileDesc	1
editorialDecl	1
text	1
physDesc	1
back	1
sourceDesc	1
address	1
country	1
textClass	1

Element	Anzahl
titlePage	1
repository	1
encodingDesc	1
front	1
availability	1
msDesc	1
addrLine	1
byline	1
body	1
typeDesc	1
teiHeader	1
docTitle	1
licence	1
msIdentifier	1
profileDesc	1
tagsDecl	1
email	1
TEI	1
table	1
langUsage	1
docAuthor	1
extent	1
language	1
handDesc	1

4.3 Weitere Texte

4.3.1 Dokument vorbereiten

4.3.1.1 `<lb/>`-GraphElemente erstellen

Finde alle lb-Elemente, die direkt an einem Wortknoten stehen und ein Wort trennen.

```
MATCH (w0:XmlWord)-[:NEXT]->(n:XmlTag {_name:'lb'})-[:NEXT]->(w1:XmlWord)
```

```
WHERE w0.text =~ '.*-'
RETURN *;
```

```
MATCH (w0:XmlWord), (n:XmlTag {_name:'lb'}),
p1=shortestPath((w0)-[:NEXT*..1]->(n))
//p1=shortestPath((w0)-[:NEXT*..2]->(n)-[:NEXT*..2]->(w1))
WHERE w0.text =~ '.*-'
RETURN p1;
```

<lb/>-Elemente per Hand entfernen, die Wörter trennen:

```
MATCH
(n2:XmlWord)-[:NEXT_WORD]->
(n3:XmlWord)-[:NEXT_WORD]->
(n4:XmlWord),
(n3)-[:NEXT]->(t1:XmlTag{_name:'lb'})-[:NEXT]->(n4)
WHERE n3.text =~ '.*-'
SET t1.before = n3.text,
t1.after = n4.text,
n4.text = left(n3.text,
size(n3.text)-1)+n4.text
CREATE (n2)-[:NEXT_WORD]->(n4)
CREATE (n2)-[:NEXT]->(t1)
CREATE (n2)-[:NEXT_SIBLING]->(t1)
DETACH DELETE n3
RETURN *;
```

```
MATCH
(n2:XmlWord)-[:NEXT_WORD]->
(n3:XmlWord)-[:NEXT_WORD]->
(n4:XmlWord),
(n3)-[:NEXT]->(t1:XmlTag{_name:'lb'})-[:NEXT]->(n4)
CREATE (n2)-[:NEXT_WORD]->(n4)
CREATE (n2)-[:NEXT]->(t1)
CREATE (n2)-[:NEXT_SIBLING]->(t1)
DETACH DELETE n3
RETURN *;
```

```
// lb-GraphElemente erstellen
```

```
MATCH p=(t1:XmlTag {_name:'lb'})-[:NEXT]->(n1:XmlWord)-[:NEXT_WORD*..20]->(n2:XmlW
CREATE (gt:GraphElement {_name:'lb'})
MERGE (gt)-[:FIRST_CHILD_OF]->(n1)
MERGE (gt)-[:LAST_CHILD_OF]->(n2)
RETURN *;
```

4.3.1.2 `<lb/>`-Elemente umwandeln

4.3.1.2.1 `<lb/>`-Elemente ohne Worttrennungen umwandeln

4.3.1.3 `<fw>`-Elemente aus der Textkette rausnehmen

Im Patzig-Manusskript wird am Ende jeder Seite das erste Wort der folgenden Seite vermerkt um neben der Seitennummerierung auch einen inhaltlichen Anhaltspunkt für die Reihenfolge der Seiten zu geben. Am Ende der Seite 6 befindet sich das Wort `Nachdem`. Mit Graph-Refactoring wird nun dieses Wort aus der Textkette herausgenommen, verbleibt aber als Information im Graphen. Zeit