Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

Konzeption und Realisierung eines Systems zur Informationssuche in einem Dokumentenarchiv basierend auf Textinhalt und Metadaten.

Conception and Realization of an Information Retrival System for a Document Archive based on Text Content and Metadata

Annika Kremer

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Karl Hans Bläsius

Trier, Abgabedatum

Vorwort

Ein Vorwort ist nicht unbedingt nötig. Falls Sie ein Vorwort schreiben, so ist dies der Platz, um z.B. die Firma vorzustellen, in der diese Arbeit entstanden ist, oder einigen Leuten zu danken, die in irgendeiner Form positiv zur Entstehung dieser Arbeit beigetragen haben. Auf keinen Fall sollten Sie im Vorwort die Aufgabenstellung näher erläutern oder vertieft auf technische Sachverhalte eingehen.

Kurzfassung

In der Kurzfassung soll in kurzer und prägnanter Weise der wesentliche Inhalt der Arbeit beschrieben werden. Dazu zählen vor allem eine kurze Aufgabenbeschreibung, der Lösungsansatz sowie die wesentlichen Ergebnisse der Arbeit. Ein häufiger Fehler für die Kurzfassung ist, dass lediglich die Aufgabenbeschreibung (d.h. das Problem) in Kurzform vorgelegt wird. Die Kurzfassung soll aber die gesamte Arbeit widerspiegeln. Deshalb sind vor allem die erzielten Ergebnisse darzustellen. Die Kurzfassung soll etwa eine halbe bis ganze DIN-A4-Seite umfassen.

Hinweis: Schreiben Sie die Kurzfassung am Ende der Arbeit, denn eventuell ist Ihnen beim Schreiben erst vollends klar geworden, was das Wesentliche der Arbeit ist bzw. welche Schwerpunkte Sie bei der Arbeit gesetzt haben. Andernfalls laufen Sie Gefahr, dass die Kurzfassung nicht zum Rest der Arbeit passt.

The same in english.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1

Einleitung und Problemstellung

1.1 Einleitung

Nahezu jeder nutzt heutzutage E-Mail-Dienste und kennt die Problematik, dass der Posteingang sich unter der Flut täglich eintreffender Nachrichten stetig füllt, bis der Ordner unübersichtlich voll ist.
Dies wird zu einem ernsthaften Problem, sobald etwas bestimmtes darin wiedergefunden werden soll.
Die benötigte Mail war wichtig, weil sie eine bestimmte Kontaktadresse enthielt, aber wie war nochmal der Absender? Längst vergessen. Das genaue Datum? Leider ist nur noch der Monat bekannt. Wer die Suche manuell bewältigen muss, ist an dieser Stelle verloren.
Abhilfe schafft ein Information Retrieval System, mit dem man bestimmte Suchkriterien eingeben und beliebig miteinander kombinieren kann.
Das System liefert idealerweise eine Reihe passender Resultate und der Anwender braucht sich nicht selbst durch hunderte von Mails durchzuarbeiten.
In diesem Fall könnte die Person beispielsweise im Freitext nach dem Wort *„Kontaktadresse“* suchen und weitere Kriterien wie *„Datum = Juni “* hinzufügen.
Besonderheit ist das beliebige Verknüpfen: Der Anwender kann sich entscheiden, ob er nur Resultate akzeptiert, auf die Beides zutrifft, oder ob es bereits reicht, wenn eines der Kriterien erfüllt ist. Dies erlaubt eine sehr individuelle, auf die Informationsbedürfnisse des Nutzers zugeschnittene Suche.
Ein solches System ist nicht nur für das Alltagsbeispiel E-Mail-Ordner, d.h. Posteingang, Postausgang etc. wertvoll, sondern lässt sich auch auf jede andere Art von Dokumentenarchiv, dessen Dokumente sowohl Freitext als auch Metadaten beinhalten, anwenden.

1.2 Problemstellung

Ziel der Arbeit ist die Konzeption und Realisierung eines Systems zur Informationssuche (engl. Information Retrieval System) in einem Dokumentenarchiv, wobei die Dokumente von teilweise strukturierter Natur sind.
Dies bedeutet, dass sie sowohl gewöhnlichen Freitext als auch Metadaten enthalten. Der Nutzer soll spezifizieren können, in welchen Metadaten er suchen möchte,

zudem soll die Freitextsuche auswählbar sein. Alle Suchanfragen sollen hierbei beliebig mit den booleschen Operatoren *AND* (engl. und) sowie *OR* (engl. oder) verknüpfbar sein.
Hauptanwendungszweck des Systems sind E-Mail-Archive wie Posteingang und Postausgang, allerdings soll das System so flexibel sein, dass es auch auf andere teilweise strukturierte Dokumentenarchive anwendbar ist.

1.3 Teilprobleme

Aus der Aufgabenstellung ergeben sich die im folgenden beschriebenen Teilprobleme.

1.3.1 Dynamisches einlesen der Metadaten

Die genauen Metadaten sind, da das System flexibel sein soll, vor dem Ausführen des Systems noch nicht bekannt. Demnach muss das IR-System die Namen der Metadaten beim Starten des Programms dynamisch einlesen und diese dem Nutzer auf der grafischen Oberfläche anzeigen.

1.3.2 Unterscheidung Metadaten und Freitext

Damit das System die Metadaten dynamisch einlesen kann, müssen die folgenden Punkte erfüllt sein:

1. Das System muss zwischen Metadaten und Freitext unterscheiden können.
2. Metadaten setzen sich aus Name und Inhalt zusammen, weshalb beides erkannt und voneinander abgegrenzt werden muss. Im folgenden werden die Namen als „Keywords “ bezeichnet.
3. Der Inhalt kann unterschiedlichen Datentyps sein, z.B. String oder Liste, weshalb dieser bestimmt werden muss.

1.3.3 Metadatensuche

Es muss erkannt werden, welche Metadaten der Nutzer ausgewählt hat und in genau diesen muss, unter Berücksichtigung des jeweiligen Datentyps der Inhalte, gesucht werden.
Im Gegensatz zur Freitextsuche muss zu jedem Dokument vor der Suche zunächst geprüft werden, ob das entsprechende Schlüsselwort überhaupt darin auftritt.

1.3.4 Freitextsuche

Bei der Freitextsuche ist die Wortzahl weitaus größer als bei der Metadatensuche. Daraus resultieren zwei Probleme:

1. Wie kann effizient in großen Wortmengen gesucht werden?
2. Wie kann die Suche bei begrenztem Speicher bewältigt werden?

Zudem stellt sich die Frage nach einem geeigneten Verfahren, das bei längeren Anfragen auch teilweise passende Ergebnisse liefert und messen kann, wie gut die erzielten Treffer zur Nutzeranfrage passen.

1.3.5 Verknüpfung mit UND/ODER

Alle Anfragen sollen beliebig mit den logischen Operatoren *AND*, *OR* sowie *NOT* verknüpfbar sein. Dies beinhaltet die folgenden Problemstellungen:

1. Keywordsuche und Freitextsuche müssen miteinander verknüpft werden.
2. Sind meherere Keywords ausgewählt, müssen die Teilergebnisse verknüpft werden.
3. Stellt der Nutzer mehrere Anfragen, müssen die Ergebnisse der einzelnen Anfragen verknüpft werden.

1.3.6 Benutzeroberfläche

Der Nutzer benötigt eine verständliche Benutzeroberfläche, die es ihm ermöglicht, seine Suchanfragen beliebig zusammenzustellen. Hierzu muss die Oberfläche folgende grundlegenden Funktionalitäten aufweisen:

1. Das Suchverzeichnis, d.h. das Dokumentenarchiv in welchem die Suche stattfindet, muss auswählbar sein.
2. Alle im Archiv auftretenden Keywords sowie die Freitextsuche müssen auswählbar sein.
3. Logische Operatoren (AND,OR,NOT) zur Verknüpfung müssen auswählbar sein.
4. Die Suchanfrage muss für den Nutzer verständlich angezeigt werden.

2

Information Retrieval

Dieses Kapitel soll dem Leser einen Überblick über die Bedeutung des Begriffs „Information Retrieval “ vermitteln.

2.1 Bedeutung

Der englische Begriff „Information Retrieval “ lässt sich mit „Informationsrückgewinnung“ ins Deutsche übersetzen. [Aca] Hierbei wird explizit von *Rück*gewinnung gesprochen, da keine neuen Informationen erzeugt werden.
Bevor erklärt wird, wofür Informationsrückgewinnung tatsächlich steht, wird zunächst auf den Teilbegriff „Information“ eingegangen.

2.1.1 Information

Die Bedeutung des Begriffs Information ist sehr weit gefasst, was eine einheitliche Definition unmöglich macht.
Er stammt aus dem Lateinischen (informare = Gestalt geben) und heißt im Übertragenen Sinne so viel wie jemanden durch Unterweisung bilden.
Dies betont den Aspekt, dass eine Information stets einen Empfänger besitzt, welcher „gebildet“ wird. Dies kann eine Person, aber auch ein geeignetes, nach außen wirksames System sein.
Daten als Träger von Informationen müssen vom Empfänger aufgenommen und korrekt interpretiert werden, damit aus den Daten tatsächlich Informationen entstehen.
Die Informationen müssen deshalb auf irgendeine Weise dargestellt werden, z.B. durch alphabetische Zeichen, außerdem muss es hierfür einen geeigneten Träger geben. Dies kann beispielsweise ein Textdokument sein.
Information lassen sich in die folgenden drei Bestandteile zerlegen:

- Syntaktischer Teil: Ist die Struktur der Information syntaktisch zulässig? Beispiel hierfür ist die Einhaltung von Rechtschreibung und Grammatik bei Texten.
- Semantischer Teil: Welchen inhaltliche Bedeutung besitzt die Information?
- Pragmatischer Teil: Welchem Zweck dient sie?

([PDVC06], S.314-315)

2.1.2 Information Retrieval

Nachdem bekannt ist, worum es sich bei Informationen handelt, wird im Folgenden beschrieben, worauf sich Information Retrieval bezieht.
Auch hier ist es problematisch, eine einheitliche Definition zu finden. Eine mögliche Erklärung lautet so:

Definition 2.1. *(Information Retrieval)*

Mit Information Retrieval, kurz IR, wird das Auffinden von in unstrukturierter Form vorliegender und ein Informationsbedürfnis befriedigender Materialien innerhalb großer Sammlungen bezeichnet.

Mit unstrukturierten Materialien sind hierbei meist Dokumente in Textform gemeint, es sind jedoch auch andere Formate möglich. Üblicherweise liegen die Sammlungen auf dem Computer gespeichert vor ([CDM08], S.1).

2.1.3 Unterschied zur Datenbankensuche

Information Retrieval unterscheidet sich stark von der Suche in Datenbanken.
In einer Datenbank liegen die Daten strukturiert in Form von Werttupeln bekannten Datentyps vor, was Definition 2.1 widerspricht.
Im Gegensatz zum Information Retrieval kann bei der Datenbankensuche nicht mit vagen Anfragen umgegangen werden. In der Datenbank kann zwar nach $(Miete < 300)$ gesucht werden, aber mit *„günstige Miete“* wäre das System überfordert: Wie ist günstig zu interpretieren? ([Fer03], S.10)
Ein Information Retrieval System muss solche Anfragen mit unklarer Bedeutung verarbeiten können.

2.2 Beispiel Websuche

Zum besseren Verständnis soll an dieser Stelle ein Beispiel zur Veranschaulichung gegeben werden.
Nahezu jeder benutzt im Alltag Web-Suchmaschinen. Websuche stellt eine Form Information Retrieval dar, da hier Freitext beinhaltende Dokumente (z.B. im HTML- oder pdf-Format) aufgefunden werden sollen, um das Informationsbedürfnis des Internetnutzers zu befriedigen. ([Fer03], S.6)
Möchte dieser zum Beispiel seinen nächsten Urlaub planen, könnte seine Suchanfrage *„Hotel günstig Kreta“* lauten.
Die gesuchten Informationen dienen also dem Zweck, den Urlaub zu planen.
Problematisch ist hierbei der semantische Teil der Informationen: Die inhaltliche Bedeutung der Resultate muss mit der ursprünglichen Intention des Nutzers

übereinstimmen.
Ein von der Suchmaschine geliefertes Resultat kann zwar zum syntaktischen Teil passen, da die korrekten Wörter darin auftauchen, allerdings in einem ganz anderen Kontext, sodass der Nutzer mit dem Dokument nichts anfangen kann.

2.3 Bezug zur Problemstellung

Die Aufgabe besteht darin, nach vom Nutzer auswählbaren, logisch verknüpften Kriterien innerhalb eines Dokumentenarchivs zu suchen (1.2).
Demnach ist die Definition 2.1 erfüllt, da hier Materialien innerhalb einer Sammlung, dem Dokumentenarchiv, aufgefunden werden sollen, um ein Informationsbedürfnis zu befriedigen.
Dieses Bedürfnis unterscheidet sich natürlich von Anfrage zu Anfrage, besteht aber allgemein gefasst darin, Dokumente wiederzufinden, z.B. eine bestimmte E-Mail.

2.3.1 Teilweise strukturierte Daten

Besonderheit der Problemstellung ist hierbei, dass die Dokumente teilweise strukturiert sind, d.h. es liegt zwar Freitext vor, aber zusätzlich sind Metadaten vorhanden.
Im Falle der Freitextsuche lässt sich aufgrund der unstrukturierten Textform eindeutig von Information Retrieval sprechen.
Anders sieht es bei den Metadaten aus, welche alle die folgende Syntax und damit Struktur besitzen:

(Name Inhalt)

Es liegt jedoch immer noch ein Information Retrieval Problem vor, da der Begriff auch die Suche in teilweise strukturierten (engl. semistructured) Dokumenten einschließt ([CDM08], S.1-2).
Wobei hierbei angemerkt sei, dass selbst die Metadaten nicht vollkommen strukturiert sind: Der Datentyp des Inhalts ist offen gelassen und es gibt keinerlei Vorgaben, welche Keywords in den Dokumenten auftreten müssen.
In den folgenden Kapiteln wird beschrieben, auf welche Weise ein Information Retrieval System konzipiert und realisiert werden kann, welches in der Lage ist, die Problemstellung 1.2 zu lösen.
Hierzu werden zunächst die hierfür benötigten Modelle boolesches Retrieval (Kapitel 4) sowie das Vector Space Model (Kapitel 5) erläutert.

3

Grundbegriffe

Unabhängig vom jeweiligen Modell gibt es einige Grundbegriffe, welche im Zusammenhang mit Information Retrieval Verfahren immer wieder auftauchen und die darum vorab vorgestellt werden sollen.

3.1 Anfrage

Eine Anfrage (engl. query) ist das, was der Anwender in den Computer eingibt, um sein Informationsbedürfnis zu befriedigen ([CDM08], S.5).
Anfragen können je nach Information Retrieval System vollkommen unterschiedlich strukturiert sein. Wichtig ist, dass der Nutzer weiß wie er seine Anfrage stellen muss, um mehr über das gewünschte Thema zu erfahren, was bei booleschen Modellen (siehe Kapitel 4) recht komplex werden kann.

3.2 Indexierung

Damit Dokumente eines Archivs von Information Retrieval Modellen verarbeitet werden können, müssen diese in einzelne Einheiten zerlegt werden, wobei jede Einheit mit einem eindeutigen Index versehen sein muss, um hinterher mit einem schnellen Zugriff abrufbar zu sein.
Zudem müssen auch die Dokumente selbst mit einem Index versehen werden, genannt *docID* (kurz für *document identification*), um auf deren enthaltende Einheiten schnell zugreifen zu können. Dabei handelt es sich meist um einen ganzzahligen Wert ([CDM08], S.7).
Dieses Vorgehen wird als Indexierung bezeichnet und ist unabdingbar, da ansonsten für jede Anfrage erneut über die gesamten Dokumentinhalte iteriert werden müsste, was ineffizient und für den Nutzer unzumutbar langsam wäre ([CDM08], S.3).

3.2.1 Term und Vokabular

Die indizierten Einheiten, in welche die Dokumente zerlegt werden, sind unter dem Begriff Term bekannt ([CDM08], S.3).
Terme sind im häufigsten Fall einfach Wörter eines Textes, dies muss jedoch nicht zwangsläufig der Fall sein.
Manche Systeme reduzieren Wörter beispielsweise auf deren Stammformen, um ähnliche Wörter zu einem einzigen Term zusammenzufassen. Alternativ lassen sich Wörter neben dem Wortstamm auch auf ihre grammatikalische Grundform reduzieren. Das Reduzieren der Wörter wird allgemein als Lemmatisierung oder Stemming bezeichnet.
Auf diese Weise müssen weniger Terme verwaltet werden, was den Speicherbedarf reduziert. Außerdem können leichter ähnliche Dokumente gefunden werden, da auch zum Suchbegriff verwandte Wörter zu einem Treffer führen ([Fer03], S.40-41).
Die Menge aller unterschiedlichen Terme eines Archivs wird als Vokabular bezeichnet ([CDM08], S.6).

3.2.2 Stopwords

Nicht jedes Wort wird bei der Indexierung zu einem Term: Handelt es sich um sehr häufig auftretende und zum Sinn des Textes wenig beitragende Wörter, wie z.B. „und“ oder „dann“, können diese wegfallen, um Speicherplatz zu sparen ([Fer03], S.37). Außerdem kommen durch ignorieren unwichtiger Terme weniger Dokumente infrage, was die Suche beschleunigt.

4

Boolesches Retrieval

Dieses Kapitel stellt das klassische Information-Retrieval-Verfahren boolesches Retrieval (engl.Boolean Retrieval) vor.

4.1 Eigenschaften des Verfahrens

Boolesches Retrieval überprüft Dokumente darauf, ob eine bestimmte Bedingung zutrifft.
Somit gibt es nur die Unterteilung in passende Dokumente und solche, welche diese Bedingung nicht erfüllen. Eine darüber hinausgehende Bewertung der Ergebnisse findet nicht statt, was zu einer ungeordneten Ergebnismenge führt ([Fer03], S.33). Das fehlende Ranking der Ergebnisse ist ein häufiger Kritikpunkt des Verfahrens.

4.2 Funktionsprinzip

Boolesches Retrieval basiert auf Mengenoperationen. Deshalb werden Dokumente Mengen zugeordnet, die jeweils durch bestimmte Attribute charakterisiert sind.
Dokument bezeichnet die Einheit, auf welcher das Retrieval stattfindet. Ein Dokument kann deshalb eine kleine Textmemo, aber auch ein ganzes Buchkapitel sein ([CDM08], S.4).

4.2.1 Attribut

Ein solches Attribut ist eine Abbildung, welche jedem Dokument einen Wert für dieses Attribut zuordnet. Die Abbildung erzeugt somit Attribut-Wert-Paare, was in Formel 4.1 gezeigt wird.

$$t : D \to T, t(d) = t_i \tag{4.1}$$

Hierbei bezeichnet t die Abbildung (d.h. das Attribut), D die Menge aller Dokumente und T den Wertebereich des Attributs t.

Der Attributwert t_i mit $t_i \in T$ und $i \in \mathbb{N}$ wird durch die Abbildung t dem Dokument $d \in D$ zugeordnet.

4.2.2 Anfragen

Elementare boolesche Anfrage

Ein Attribut-Wert-Paar wird auch als elementare boolesche Anfrage bezeichnet. Bei der elementaren booleschen Anfrage (t, t_1) werden zum Beispiel alle Dokumente gesucht, deren Attribut t den Wert t_1 annimmt.
Die Ergebnismenge D_{t,t_i} für eine Anfrage (t, t_i) kann demnach wie in Formel 4.2 charakterisiert werden.

$$D_{t,t_i} = \{d \in D | t(d) = t_i\} \tag{4.2}$$

Verknüpfung

Werden elementare Anfragen miteinander logisch verknüpft, so werden abhängig vom jeweiligen booleschen Operator bestimmte Mengenoperationen auf den Ergebnismengen der elementaren Anfragen ausgeführt.
Die möglichen booleschen Operatoren sind hierbei AND, OR und NOT.
(t, t_1) AND (s, s_1) bedeutet, dass alle Dokumente gesucht sind, bei denen sowohl $t(d) = t_1$ als auch $s(d) = s_1$ gilt. Die erforderliche Mengenoperation ist deshalb der Durchschnitt aus den beiden Ergebnismengen, was in Formel 4.3 gezeigt wird.

$$D_{t,t_1} \cap D_{s,s_1} \tag{4.3}$$

Wird hingegen der Operator OR verwendet, wird die Mengenoperation Vereinigung benötigt (siehe Formel 4.4), da alle Dokumente mit $t(d) = t_1$ oder $s(d) = s_1$ gesucht sind.

$$D_{t,t_1} \cup D_{s,s_1} \tag{4.4}$$

Außerdem kann der unäre Operator NOT verwendet werden, welcher das Komplement der Ergebnismenge erzeugt. Für die Anfrage NOT (t, t_1) muss erst die Menge aller Dokumente bestimmt werden, bei denen $t(d) = t_1$ zutrifft, um diese anschließend von der Gesamtmenge aller Dokumente abzuziehen. Dies wird in Formel 4.5 dargestellt.

$$D \setminus D_{t,t_1} \tag{4.5}$$

Da bei jeder Mengenoperation als Ergebnis neue Mengen entstehen, lassen sich hierauf erneut die oben beschriebenen Operatoren anwenden. Auf diese Weise können Anfragen beliebig tief geschachtelt werden ([Fer03], S.34).

4.3 Implementierungsansätze

Im folgenden Abschnitt werden die klassischen Implementierungsansätze vorgestellt, mit denen sich die soeben beschriebenen Operationen realisieren lassen.

4.3.1 Term-Inzidenz-Matrix

Eine mögliche Implementierung des booleschen Retrieval stellt die Umsetzung mittels einer Term-Dokument-Inzidenz-Matrix dar.
Dies bedeutet, dass die Zeilen der Matrix die Terme enthalten und die Spalten die Dokumente, was auch umgekehrt realisierbar ist. Genau betrachtet handelt es sich hier nicht um die Terme und Dokumente selbst, sondern deren Indizes.
Tritt Term t in Dokument d auf, so lautet der Eintrag für (t, d) der Matrix 1. Alle Einträge für nicht vorkommende Terme sind hingegen mit einer 0 versehen.
Abbildung 4.1 zeigt eine Beispielmatrix, wobei die tatsächliche Anzahl an Termen und Dokumenten in einer Sammlung weitaus größer ausfällt.
Eine Term-Inzidenz-Matrix verbraucht unnötig Speicherplatz, da sehr viele Einträge der Matrix eine 0 enthalten. Gerade bei sehr großen Sammlungen bzw. Dokumenten ist dies praktisch nicht realisierbar.

	1	2	3	4	5	6	7
Kontaktadresse	0	1	1	0	0	0	1
Seminar	1	0	1	0	1	0	0
Termin	1	1	1	0	0	0	0

Abb. 4.1. Term-Dokument-Inzidenz-Matrix. Die Zeile enthalten die Terme, die Spalten die Dokumente (hier durch deren *docID* repräsentiert). Alle Einträge mit einer 0 sind leere Einträge (Eigene Abbildung).

Verarbeitung einer Anfrage

Um eine Anfrage wie *Kontaktadresse AND Seminar AND Termin* mithilfe einer Matrix zu verarbeiten, werden einfach die entsprechenden Zeilen der Matrix genommen und bitweise logisch verknüpft, was für die obige Anfrage wie folgt aussieht:

0110001 AND
1010100 AND
1110000

0010000

Demnach wird das Dokument mit der *docID* 3 zurückgegeben. Analog funktioniert die *OR*-Verknüpfung:

0110001 OR
1010100 OR
1110000

1110101

Dieses Beispiel führt demnach zur Ergebnismenge $\{1, 2, 3, 5, 7\}$ ([CDM08], S.4).

4.3.2 Invertierte Liste

In der Regel werden zur Implementierung des booleschen Retrieval invertierte Listen verwendet ([Fer03], S.36).
Der Name basiert auf den darin gespeicherten invertierten Indizes. Diese werden deshalb als invertiert bezeichnet, weil sie vom Term zurück auf die Postion, in welcher der Term aufgetreten ist, schließen lassen.
In einer geeigneten Speicherstruktur, zum Beispiel einem Dictionary, werden zu jedem Term alle Dokumente gespeichert, in denen der Term auftritt.
Diese Dokumentlisten werden als invertierte Listen (engl. inverted lists) bezeichnet (siehe Abbildung Abbildung 4.2). Manche Implementierungen beinhalten neben dem Dokumentindex zusätzliche Informationen wie die genaue Wortpostion im Dokument.
Das Verfahren ermöglicht sehr schnelle Zugriffe, ist allerdings speicherintensiv ([Fer03], S.36). Im Vergleich zur Inzidenz-Matrix wird jedoch deutlich weniger Speicher benötigt, da die vielen leeren Einträge entfallen.

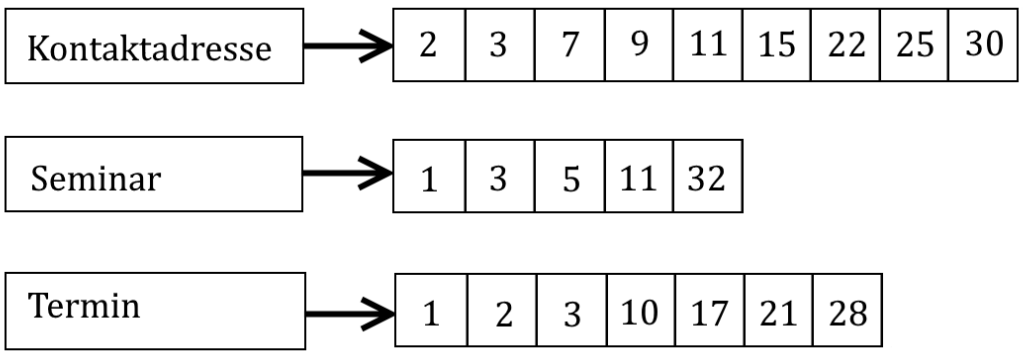

Abb. 4.2. Invertierte Listen zu Beispieltermen. Die Zahlen sind die Dokumentindizes oder auch *docID*s, in denen der jeweilige Term vorkommt (Eigene Abbildung).

Verarbeitung einer Anfrage

Hier stellt sich die Frage, wie denn nun eine boolesche Anfrage, wie in Abschnitt 4.2.2 beschrieben, mithilfe invertierter Listen umgesetzt werden kann.

Elementare Anfrage

Angenommen, es liegt eine elementare boolesche Anfrage in der Form (t, t_1) vor. In der Praxis ist meist nach dem Vorkommen eines bestimmtes Wortes gefragt. Demnach entspricht das Attribut t dem Term des gesuchten Wortes.
Da man auf dessen Vorkommen prüft, gelten für den Wertebereich $T = \{true, false\}$ und für den Attributwert $t_1 = true$.
Die Verarbeitung einer solchen elementaren Anfrage geht relativ einfach: Über den Index kann auf den Term schnell zugegriffen werden, vorausgesetzt dieser ist in der Sammlung enthalten. Trifft dies zu, kann einfach die gesamte zugehörige invertierte Liste als Resultat ausgegeben werden, da für alle enthaltenen Dokumente $t_1 = true$ gilt.

AND-Verknüpfungen

Wie sieht nun die Verarbeitung aus, wenn mehrere Anfragen miteinander verknüpft werden? Hierzu wird zunächst der AND - Operator betrachtet. Eine Anfrage liegt dann in der Form (t, t_1) AND (s, s_1) vor, wie etwa bei dem Beispiel *Kontaktadresse AND Seminar*, wobei gilt $t = Kontaktadress, t_1 = true$ sowie $s = Seminar, s_1 = true$.
Demnach werden alle Dokumente gesucht, in denen beide Terme auftauchen. Dazu wird der Durchschnitt aus den zugehörigen invertierten Listen gebildet. Betrachtet man die Abbildung 4.2, so ist der Durchschnitt für $D_{t,t_1} \cap D_{s,s_1}$ bzw. für $Kontaktadress \cap Seminar$ gleich der Ergebnisliste $\{3, 11\}$.

OR-Verknüpfungen

Lautet die Anfrage hingegen (t, t_1) OR (s, s_1) bzw. *Kontaktadresse OR Seminar*, so sind alle Dokumente gesucht, in denen entweder t_1 oder s_1 oder auch beide Terme vorkommen.
Gesucht ist also die Vereinigung $D_{t,t_1} \cup D_{s,s_1}$ bzw. $Kontaktadress \cup Seminar$. Dies ist die Vereinigung der invertierten Listen beider Terme. Im Falle des Beispiels 4.2 lautet die Ergebnisliste für die Anfrage *Kontaktadresse OR Seminar* $\{1, 2, 3, 5, 7, 9, 11, 15, 22, 25, 30, 32\}$.
Für beide Listenoperationen gilt, das die Indizes in den Ergebnislisten sortiert und Duplikate entfernt werden ([CDM08], S.11).

AND NOT-Verknüpfung

Da es sich bei *NOT* um einen unären Operator handelt, könnte dieser theoretisch alleine auftreten.
Eine Anfrage der Form *NOT Seminar* kann sehr viel Laufzeit kosten, wenn die Sammlung aus vielen Dokumenten besteht: Es muss über die gesamte Sammlung iteriert werden und für jedes Dokument geprüft werden, ob es in der invertierten Liste für den Term Seminar auftaucht. Der alleinstehende NOT-Operator ist deshalb so ineffizient, dass er bei den meisten booleschen Retrieval Systemen nur im Zusammenhang mit einem binären Operator zugelassen ist.
Da die Kombination *OR NOT* keinen Sinn macht, wenn man einen Term ausschließen möchte, ist dies in der Regel *AND*.
Hierbei wird aus den beiden Listen die Differenz gebildet. Lautet die Anfrage beispielsweise *Kontaktadresse AND NOT Seminar*, so werden aus der Ergebnisliste für Kontaktadresse alle Elemente entfernt, die in der Liste für Seminar enthalten sind ([Hen08], S.174).

Komplexe Ausdrücke

Da sowohl Vereinigung als auch Durchschnitt eine neue Ergebnisliste liefern, kann auf dieser wiederum jeder Operator angewandt werden, was eine beliebig tiefe Schachtelung erlaubt. Dieser Abschnitt erklärt, wie komplex geschachtelte Ausdrücke verarbeitet werden.

Im Falle von mehreren *AND*-Operatoren, wie etwa in der Suchanfrage *Kontaktadresse AND Seminar AND Termin*, ist es effizient, zunächst die einzelnen invertierten Listen aufsteigend nach deren Länge zu sortieren und dann von links nach rechts zu verarbeiten, indem das nachfolgende *AND* auf die Ergebnisliste des vorherigen Durchschnitts angewandt wird:
(*Seminar AND Termin*) *AND Kontaktadresse*
Auf diese Weise werden die Listen, über die iteriert werden muss, möglichst klein gehalten. Besitzt die kleinste Liste beispielsweise die Länge eins, dann kann nach der ersten Iteration bereits abgebrochen werden, da zulässige Lösungen in allen drei Listen vorkommen müssen.

Bei mehreren *OR*-Operatoren werden die Ausdrücke analog von links nach rechts verarbeitet, wobei die Sortierung nach Länge hierbei keinen Vorteil bringt, da bei der Vereinigung zweier Listen ohnehin über alle Elemente iteriert werden muss. Die Verarbeitung würde demnach in der folgenden Reihenfolge erfolgen:
(*Kontaktadresse OR Seminar*) *OR Termin*

Ist die Anfrage hingegen gemischt, wie etwa in (*Kontaktadresse OR Seminar*) *AND* (*Termin OR Seminar*), werden erst die inneren Ausdrücke ausgewertet und dann aus deren Ergebnislisten der Durchschnitt gebildet ([CDM08], S.11).

Die Verarbeitung mehrerer Wörter

Boolesches Retrieval kann auch mehrere zusammengehörende Wörter verarbeiten. Über die interne Verarbeitung besteht hierbei jedoch für den Nutzer kein Einblick: Das Information Retrieval System kann so realisiert sein, dass es die aus den Wörtern der Anfrage isolierten Terme mit *OR* veknüpft, es kann diese jedoch genauso gut mit*AND* verbinden ([Hen08], S.171).

5

Das Vektorraummodell

Dieses Kapitel stellt ein weiteres klassisches Information Retrieval Verfahren, das Vektorraummodell, vor.

5.1 Funktionsprinzip

Zunächst werden alle Dokumente in Terme zerlegt und indexiert, wie es auch beim booleschen Retrieval der Fall ist. Die Ermittlung des Vokabulars ist Grundvoraussetzung für alle weiteren Schritte.
Wie der Name bereits nahelegt, basiert das Verfahren auf Vektoren.

Die Grundidee besteht darin, für jedes Dokument sowie für die Anfrage einen reellen Vektor zu erstellen und anschließend zu ermitteln, welche Dokumentvektoren am ähnlichsten zum Anfragevektor sind.
Jeder Vektor besitzt hierbei die Länge des Vokabulars, da er die Gewichte aller Terme enthält. Die Bedeutung des Gewichts wird in Abschnitt 5.4 erklärt.
Im Gegensatz zum booleschen Retrieval können die Resultate des Vektorraummodells basierend auf dem Grad der Ähnlichkeit in eine Rangfolge gebracht werden ([Fer03], S.62-63).

5.2 Vektor und Vektorraum

Da das Funktionsprinzip des Modells auf Vektoren basiert, seien an dieser Stelle die zum Verständnis notwendigen Begriffe Vektorraum und Vektor erklärt.
Vektoren stellen Elemente eines Vektorraumes dar, darum ist es notwendig letzteres zuerst zu definieren ([Jän84], S.17).
Ein solcher Vektorraum lässt sich wie in Definition 5.1 ([Jän84], S.22) angegeben beschreiben.

Definition 5.1. *(Vektorraum)*
Ein Tripel $(V, +, \cdot)$*, bestehend aus einer Menge* V*, einer Abbildung (genannt Addition)*
$+ : V \times V \to V$
$(x, y) \to x + y$
und einer Abbildung (genannt skalare Multiplikation)
$\cdot : \mathbb{R} \times V \to V$
$(\lambda, x) \to \lambda x$
heißt reeller Vektorraum*, wenn für die Abbildungen* $+$ *und* $\cdot$ *die folgenden acht Axiome gelten:*

(1) $(x + y) + y = x + (y + z) \quad \forall x, y, z \in V$
(2) $x + y = y + x \quad \forall x, y, z \in V$
(3) *Es gibt ein Element* $0 \in V$ *(genannt „Null "oder„Nullvektor ") mit* $x + 0 = x \quad \forall x \in V$
(4) *Zu jedem* $x \in V$ *gibt es ein Element* $-x \in V$ *mit* $x + (-x) = 0$
(5) $\lambda(\mu x) = (\lambda\mu)x \quad \forall \lambda, \mu \in \mathbb{R}, x \in V$
(6) $1x = x \quad \forall x \in V$
(7) $\lambda(x + y) = \lambda x + \lambda y \quad \forall \lambda \in \mathbb{R}, x, y \in V$
(8) $(\lambda + \mu)x = \lambda x + \mu x \quad \forall \lambda, \mu \in \mathbb{R}, x \in V$

Hieraus ergibt sich die Frage, wie Addition und skalare Multiplikation bei Vektoren definiert sind, was in Definition 5.2 und 5.3 ([Jän84], S.18) gezeigt wird.

Definition 5.2. *(Addition)*
Sind $(x_1, ..., x_n)$ *und* $(y_1, ..., y_n)$ *n-Tupel reeller Zahlen, so werde deren Summe durch*
$(x_1, ..., x_n) + (y_1, ..., y_n) = (x_1 + y_1, ..., x_n + y_n)$
erklärt.

.

Definition 5.3. *(Skalare Multiplikation)*
Ist $\lambda \in \mathbb{R}$ *und* $(x_1, ..., x_n) \in \mathbb{R}^n$*, so erklären wir* $\lambda(x_1, ..., x_n) = (\lambda x_1, ..., \lambda x_n) \in \mathbb{R}^n$

.

Ein Vektor $\overrightarrow{v} \in V$ ist demnach ein Element des Vektorraums $(V, +, \cdot)$, wenn Addition und skalare Multiplikation die Axiome (1) - (8) aus Definition 5.1 erfüllen. Zwei Vektoren mit unterschiedlich vielen Elementen, z.B. $\overrightarrow{v_1} = (1, 2)$ und $\overrightarrow{v_2} = (1, 2, 3)$ liegen deshalb nicht im selben Vektorraum, weil sie sich weder addieren noch multiplizieren lassen.

5.3 Definition Vektorraummodell

Das soeben beschriebene Funktionsprinzip lässt sich mathematisch mithilfe von Attributen beschreiben.

Attribute stellen im Vektorraummodell eine Abbildung der Dokumentenmenge D auf die reellen Zahlen $\mathbb{R}$ dar. Damit ist der Wertebereich der Attribute im Gegensatz zum booleschen Retrieval eindeutig festgelegt.
Die Definition lautet somit wie folgt:

Definition 5.4. *(Vektorraummodell mit Attributen)*

Sei $D = d_1, ..., d_n$ eine Menge von Dokumenten oder Objekten und $A = A_1, ..., A_n$ eine Menge von Attributen $A_j : D \to R$ auf diesen Objekten. Die Attributwerte $A_j(d_i) =: w_{i,j}$ des Dokuments d_i lassen sich als Gewichte auffassen und zu einem Vektor $w_i = (w_{i,1}, ..., w_{i,n}) \in \mathbb{R}^n$ zusammenfassen. Dieser Vektor beschreibt das Dokument im Vektorraummodell: Er ist seine Repräsentation und wird Dokumentvektor genannt.
Eine Anfrage wird durch einen Vektor $q \in \mathbb{R}^n$ mit Attributwerten, den Anfragevektor oder Query-Vektor, dargestellt.
Eine Ähnlichkeitsfunktion $s : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ definiere für je zwei Vektoren $x, y \in \mathbb{R}^n$ einen reellen Ähnlichkeitswert $s(x, y)$.

Diese Definition ist aufgrund der Beschreibung durch Attribute sehr allgemein gehalten. Es ist deshalb nicht definiert, welche Einheiten des Dokuments gewichtet werden. Demnach sind theoretisch auch andere Dokumentformate wie etwa Bilder mit Pixeln bzw. Pixelgruppen als Attributen möglich ([Fer03], S.63).
Praktisch gesehen machen im Rahmen dieser Arbeit jedoch andere Formate als Texte keinen Sinn. Darum wird im folgenden davon ausgegangen, dass ausschließlich Terme gewichtet werden. Demnach lässt sich die Attributmenge A spezifisch auf die Problemstellung bezogen als Menge der Terme oder Vokabular T auffassen.

5.4 Gewichte

Bei Gewichten handelt es sich, wie bereits beschrieben, um reelle Zahlenwerte.
Ein Gewicht gibt die Wichtigkeit eines Terms basierend auf dessen statistischer Häufigkeit an ([CDM08], S.100).

5.4.1 Termhäufigkeit

Die Häufigkeit, mit der ein Term t in einem Dokument d auftritt, wird als Termhäufigkeit (engl. term frequency) bezeichnet. Demnach wird die Termhäufigkeit pro Vorkommen von t in d um eins erhöht ([CDM08], S.71).
Es erscheint intuitiv logisch, dass ein Text, der das gesuchte Wort mehrmals beinhaltet wichtiger sein muss als ein Dokument, in welchem der Begriff nur ein einziges mal auftaucht.
Dies Gewichtungsschema erlaubt eine viel genauere Differenzierung als eine simple Unterscheidung zwischen true und false, wie es beim booleschen Retrieval der Fall ist.

5.4.2 Dokumenthäufigkeit

Die Termhäufigkeit stellt für sich genommen schon eine mögliche Gewichtungsmethode dar, allerdings keine besonders gute: Die Bewertung alleine aufgrund der Termhäufigkeit lässt außer Acht, dass nicht alle Terme gleich wichtig sind.
Taucht ein Term beispielsweise in jedem Dokument auf, kann es nicht besonders aussagekräftig sein. Demnach ist es sinnvoll, zusätzlich zur Termhäufigkeit $tf_{t,d}$ auch die Dokumenthäufigkeit (engl. document frequency) df_t zu bestimmen.
Diese entspricht der Anzahl Dokumente in D, welche t enthalten. Um den Einfluss nicht aussagekräftiger Terme zu reduzieren, wird das Gewicht umso stärker verringert, je größer die Dokumenthäufigkeit ausfällt ([CDM08], S.108).

5.4.3 Invertierte Dokumenthäufigkeit

Zur Reduktion des Gewichts basierend auf der Dokumenthäufigkeit wird als reduzierender Faktor die Invertierte Dokumenthäufigkeit (engl. inverse document frequency) verwendet. Die invertierte Dokumenthäufigkeit idf_t des Terms t berechnet sich wie in Formel 5.1 gezeigt.

$$idf_t = \frac{1}{df_t} \tag{5.1}$$

Oftmals werden modifizierte Formen verwendet, um die großen Werte seltener Terme durch den Logarithmus wieder zu dämpfen ([Fer03], S.68-69).
Formel 5.2 zeigt ein Beispiel für eine solche modifizierte invertierte Dokumenthäufigkeit. In der Regel beträgt die Basis des Logarithmus 10, dies spielt aber letztendlich für das korrekte Ranking der Resultate keine Rolle ([CDM08], S.108-109).

$$idf_t = log\frac{N}{df_t} \tag{5.2}$$

5.4.4 TF-IDF-Gewichtung

Die vollständige Methode zur Gewichtung einzelner Terme kombiniert Termhäufigkeit und invertierte Dokumenthäufigkeit, indem erstere mit letzterer multipliziert wird.
Alle Formeln dieses Typs werden als TF-IDF Gewichtung (engl. tf-idf weighting) bezeichnet ([Fer03], S.71).
Das Gewicht für Term t in Dokument d berechnet sich somit wie in Formel 5.3 gezeigt ([CDM08], S.109).

$$tf - idf_{t,d} = tf_{t,d} \times idf_t, \tag{5.3}$$

Verwendet man für die invertierte Dokumenthäufigkeit den unmodifizierten Wert 5.1, so ergibt sich hieraus die Berechnung:

$$tf - idf_{t,d} = \frac{tf_{t,d}}{df_t} \tag{5.4}$$

Für die modifizierte Formel 5.2 lautet die TF-IDF-Gewichtung wie folgt:

$$tf - idf_{t,d} = tf_{t,d} \times (log \frac{N}{df_t}) \tag{5.5}$$

Sei T die Menge aller Terme der Sammlung bzw. das Vokabular, dann enthält der Gewichtsvektor w_i zu einem Dokument $d_i \in D$ für jeden Term $t_j \in T$ dessen Gewicht $w_{i,j} = tf - idf_{j,i}$, sodass $w_i = (tf - idf_{1,i}, ..., tf - idf_{n,i})$ gilt.

5.5 Anfragen

Beim Vektorraummodell gibt es keine booleschen Operatoren zur Verknüpfung von Termen, weshalb Anfragen in Freitextform gestellt werden. Diese Form wird auch in der Websuche verwendet und ist darum sehr bekannt.
Da die Reihenfolge von Wörtern weder bei Anfragen noch in den Dokumenten eine Rolle spielt, lassen sich Anfragen einfach als eine Menge von Wörtern bzw. als die daraus resultierende Menge von Termen betrachten.
Ein solches Modell, das lediglich die Anzahl, nicht aber die Reihenfolge von Wörtern berücksichtigt, wird auch als *bag of words model* bezeichnet.
Da für jeden Term ein anderer Ähnlichkeitswert erzielt wird, werden die Ähnlichkeitswerte aller in der Menge enthaltenen Terme addiert, sodass pro Dokument ein Gesamtwert berechnet wird ([CDM08], S.107).
Anfragetexte werden genau wie Dokumente behandelt und die Vektoren wie in Abschnitt 5.4.4 beschrieben bestimmt ([Fer03], S.82).

5.6 Ähnlichkeitsfunktion

Grundidee des Vektorraummodells ist das Ermitteln der Ähnlichkeit zwischen Vektoren. Deshalb muss hierfür eine geeignete Ähnlichkeitsfunktion gefunden werden.

5.6.1 Euklidischer Abstand

Eine typische Distanzfunktion für Vektoren ist der euklidische Abstand, bei dem die Differenz wie in Formel 5.6 gezeigt berechnet wird ([KMOB14],S.132).

$$d(\overrightarrow{x}, \overrightarrow{y}) = \sqrt{\sum_{i=1}^{M} (x_i - y_i)^2} \tag{5.6}$$

Allerdings besitzt der euklidische Abstand den gravierenden Nachteil, dass die Länge der Vektoren eine Rolle spielt.
Liegen zwei Dokumente $d1$ und $d2$ vor und $d2$ besitzt den Inhalt von $d1$ zweimal aneinandergereiht, so wird $d2$ als ähnlicher eingestuft, aus dem einzigen Grund weil die Termhäufigkeit doppelt so groß ist und der Vektor damit die doppelte Länge hat.
Das Problem, dass zwei unterschiedlich lange Dokumente, in denen die gesuchten Terme etwa gleich verteilt sind, dennoch vollkommen verschiedene Ähnlichkeitswerte erzielen macht den euklidischen Abstand zu einer ungeeigneten Ähnlichkeitsfunktion.

5.6.2 Cosinus-Maß

Um den Einfluss der Vektorlänge zu eliminieren, wird in der Regel das Cosinus-Maß verwendet.
Das Cosinus-Maß ist das Skalarprodukt der normalisierten Vektoren ([CDM08], S.112). Ein normalisierter Vektor besitzt immer die Länge 1, da Ursprungsvektor durch die euklidische Länge dividiert wird.
Zur Berechnung des Cosinus-Maßes müssen somit sowohl das Skalarprodukt als auch die Längenberechnung eines Vektors bekannt sein, darum werden beide an dieser Stelle erklärt.

Euklidische Länge

Sei $\overrightarrow{x}$ ein Vektor, dann wird seine euklidische Länge wie in Formel 5.7 angegeben berechnet.

$$|\overrightarrow{x}| = \sqrt{\sum_{i=1}^{M} x_i^2} \tag{5.7}$$

Skalarprodukt

Das Skalarprodukt zweier Vektoren $\overrightarrow{x}$ und $\overrightarrow{y}$ wird wie in Formel 5.8 gezeigt berechnet.

$$\overrightarrow{x} \cdot \overrightarrow{y} = \sum_{i=1}^{M} x_i y_i \tag{5.8}$$

Funktion

Das Cosinus-Maß multipliziert die Vektoren und dividiert sie durch deren Länge, sodass die Ähnlichkeitsfunktion von der Länge unbeeinflusst ist, was in Formel 5.9 gezeigt wird ([CDM08], S.111).

$$sim(\overrightarrow{x}, \overrightarrow{y}) = \frac{\overrightarrow{x} \cdot \overrightarrow{y}}{|\overrightarrow{x}| \cdot |\overrightarrow{y}|} \tag{5.9}$$

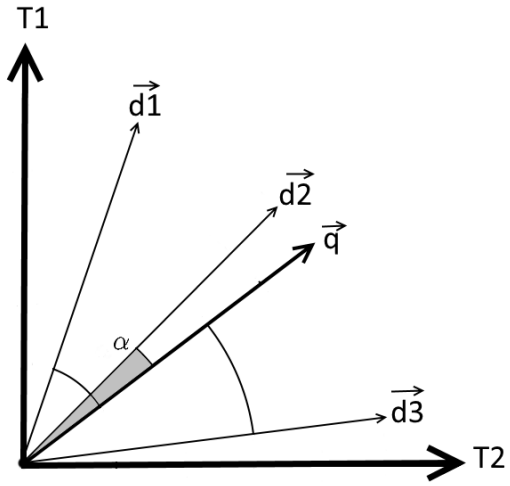

Abb. 5.1. Vektorraum mit den Termen $T1$ und $T2$ als Achsen, drei Dokumentvektoren und dem Anfragevektor. Das Cosinus-Maß liefert als ähnlichstes Dokument d_2 (Eigene Abbildung, basierend auf [SB10], S.55).

Zum besseren Verständnis des Cosinus-Maßes zeigt die Beispielabbildung 5.1 einen Vektorraum mit zwei Termen als Achsen, sodass der Raum sich zweidimensional darstellen lässt. In der Realität gibt es meist weitaus mehr Achsen, da das Vokabular tausende Terme beinhalten kann.
Der Vektorraum beinhaltet insgesamt drei Dokumentvektoren $\overrightarrow{d_i}$ sowie den Anfragevektor $\overrightarrow{q}$ als Elemente, wobei alle Vektoren normalisiert sind.
Unter Verwendung des Cosinus-Maßes ergibt sich für $sim(\overrightarrow{d_2}, \overrightarrow{q}) = cos(\alpha)$ der höchsten Wert, da α der kleinste aller Winkel zwischen einem Dokumentvektor $\overrightarrow{d_i}$ und $\overrightarrow{q}$ ist. Hier wird deutlich, dass lediglich der Winkel, nicht aber die Länge relevant ist.
Damit wird $d2$ als erstes Ergebnisdokument ausgegeben ([SB10], S.55-56).

6

Bewertung eines Information Retrieval Systems

6.1 Precisison

6.2 Recall

7

Implementierung

In diesem Kapitel wird beschrieben, auf welche Weise das Information Retrieval System dieser Arbeit in der Programmiersprache Lisp realisiert wurde.

7.1 Teilweise strukturierte Dokumente

Besonderheit der Problemstellung ist das Vorliegen der Dokumente in semistrukturierter Form (siehe 1.2.
Dies bedeutet, dass zwei unterschiedliche Teilprobleme zu lösen sind: Zum einen die Keywordsuche, welche sich auf die Suche in strukturierten Metadaten bezieht und zum anderen die Freitextsuche. Es liegt nahe, beides getrennt zu lösen, da die Suchen unterschiedliche Anforderungen besitzen.
Bevor erklärt wird, wie die beiden Verfahren jeweils realisiert wurden, ist es wichtig, zunächst eine Vorstellung zu haben, wie die zu durchsuchenden Dokumente des Archivs beschaffen sind, weshalb Abbildung 7.1 ein Beispiel zeigt.

Listing 7.1. Beispieldokument

```
1
2
3   (absender ("<MaxMuster@muster-mail.de>"))
4   (Betreff (" Umfrage"))
5   (datum ("Wed, 22 Jun 2017 07:47:51 +0200"))
6   (anzahlAnhaenge 0)
7   (Termin nil)
8   (ABSENDER "Max_Muster")
9
10  (ABSENDER-MAIL-ADRESSE "MaxMuster@muster-mail.de")
11  (EMPFAENGER ("doe>> John Doe"))
12  (EMPFAENGER-MAIL-ADRESSEN ("johnd@muster-mail.de"))
13  (BETREFF "Umfrage")
14  (EMAIL-TYP "sent")
15  (QUELLBOXART "SENT")
16
17  Hallo John,
18
19  ich werde dir die Umfragenformulare schnellstmöglich per Post
20  zukommen lassen.
21
22
23  Viele Grüße
24  Max Muster
```

In der Praxis enthalten die Dokumente oft weitaus mehr Keywords, deren Inhalt sich auch über mehrere Zeilen erstrecken kann, sowie Kommentare, welche vom System als Freitext interpretiert werden.
In diesem Beispiel handelt es sich zwar um eine E-Mail, die Keywords können jedoch inhaltlich vollkommen unterschiedlich ausfallen. Allen Dokumenten gemeinsam ist die grundlegende Struktur aus 7.2.

Listing 7.2. Dokumentstruktur

```
1    (Keywordname_1 Inhalt)
2    ....
3    (Keywordname_n Inhalt)
4
5    Freitext
```

7.2 Vorverarbeitung

Splitten in Keywords und Text

7.3 Keywordsuche

Für die Keywordsuche ergeben sich zwei Schritte:

1. In welchen Dokumenten tritt das Keyword auf?
2. Welches dieser Dokumente besitzt einen mit der Anfrage übereinstimmenden Keywordinhalt?

7.3.1 Schritt 1

Der erste Punkt ist notwendig, da es keine Vorgaben bezüglich des Auftretens von Keywords gibt. Ein Keyword kann in einem einzigen Dokument, aber auch in der gesamten Sammlung vorkommen.
Da hier auf eine klar definierte Bedingung hin überprüft wird, bietet sich das boolesche Retrieval an.
Sei K die Gesamtmenge aller möglichen Keywords, dann lautet die Ergebnismenge $D_{keyword_i}, true = \{d \in D | keyword_i(d) = true\}$ mit $keyword_i \in K$. Dabei ist $keyword_i(d) = true$, wenn Dokument d das $keyword_i$ als Keywordnamen (siehe Struktur 7.2) enthält.
Da es sich hierbei um gewöhnliches boolesches Retrieval handelt, wurden die Implementierungsansätze für boolesches Retrieval betrachtet.
Eine Term-Inzidenz-Matrix (siehe Abschnitt 4.3.1) wurde sofort verworfen, da sie zu viel Speicher benötigt.
Invertierte Listen (siehe Abschnitt 4.3.2) bieten hingegen eine hervorragende Möglichkeit, zu jedem Keyword die Dokumente zu speichern, in welchen es auftritt.
Auch wenn der Begriff „Invertierte Liste“ lautet, ist die Wahl der zugrunde liegenden Speicherstruktur frei.

In dieser Arbeit wurde sich für die in der Programmiersprache Lisp vorhandenen Hash Tables entschieden.
Die Keywordnamen werden hierbei zu den Keywords der Hash-Table und die Liste der zugehörigen *docID*s zum Inhalt des Hasheintrags.
Grund hierfür ist das effiziente Zugreifen auf Hasheinträge, was das Iterieren über Listen erspart.

7.3.2 Schritt 2

Wird zum Beispiel nach *Absender* = *Klaus* gesucht, führt das Ausführen von Schritt 1, d.h. das Zugreifen auf den Hasheintrag zum Key *Absender*, erst einmal nur zu einer Liste von in frage kommender Dokumente.
Deshalb wurde die ursprüngliche Implementierung durch Invertierte Listen modifiziert, indem statt der *docID* eine komplexe Datenstruktur gespeichert wird.
Lisp bietet hierzu Structs, was verwendet wurde um zusätzlich zur *docID* den Keywordinhalt sowie den Typ des Inhalts, z.B. String oder Number, in einem Struct zusammengefasst zu speichern.
Das bietet den Vorteil, auf die enthaltenen slots (Variablen im Struct) gezielt per Name zugreifen zu können.
Das Speichern der Keywordinhalte als Teil der Hasheinträge stellt im Normalfall kein Problem dar, da die Inhalte nur sehr selten über eine Zeile hinausgehen. Ein Zerlegen in Terme wie beim Freitext ist darum nicht notwendig und wurde, um Speicher zu sparen, unterlassen.
Als Resultat der modifizierten Form invertierter Listen lässt sich einfach über die zum Keyword gespeicherten Structs iterieren und diese werden jeweils nach dem Suchbegriff bzw. den Suchbegriffen durchsucht.

Typspezifische Suche

Es stellt sich die Frage, was das Speichern des Typs zusätzlich zum Inhalt bringt. Die Keywordinhalte können unterschiedlichen Typs sein, was über die Art der Suche entscheidet:

1. **String**: Der Keywordinhalt wird mit der vordefinierten Lisp-Funktion search durchsucht. Die Suche ist erfolgreich, wenn die gesuchte Zeichenkette an einer beliebigen Stelle im Keyword auftaucht.
2. **Number**: Ist der Inhalt eine Zahl, wird die Suchanfrage (die stets als String übergeben wird) wenn möglich zum Datentyp Number konvertiert. Hierbei sind ausgeschriebene Zahlen von null bis zwölf auch konvertierbar. Ist kein Konvertieren möglich, schlägt die Suche sofort fehl, da der Inhalt nicht zur Anfrage passen kann.
3. **Liste**: Eine Liste (Datentyp Cons in Lisp) wird rekursiv durchsucht, um alle darin enthaltenen Elemente typspezifisch zu durchsuchen, d.h. darin enthaltene Strings, Zahlen und Unterlisten.

7.4 Freitextsuche

Die Freitextsuche gestaltet sich komplexer, da hier längere Texte durchsucht werden müssen. Der Einsatz von vordefinierten Funktionen wie search kommt darum aus Effizienzgründen nicht infrage.
Stattdessen muss der Text erst in Terme zerlegt werden, um ihn verarbeiten zu können. Hierbei wurde das Lisp-Package split-sequence ([Ion16]) verwendet, um die Dokumente zeilenweise in Wörter zu zerlegen.
Da bei einer Freitextsuche ist ein Verfahren notwendig, das auch teilweise Treffer und ein Ranking der Ergebnisse ermöglicht. Die Suchanfragen sind hier in der Regel länger und sollen auch dann zu Resultaten führen, wenn nur ein paar der Begriffe auftreten. Das Ranking gibt Auskunft darüber, wie hoch der Grad der Übereinstimmung ist.
Da boolesches Retrieval weder Ranking noch Teiltreffer ermöglicht, fällt die Wahl hier auf das Vektorraummodell (siehe 5).

8

Die Benutzeroberfläche

Dieses Kapitel soll die Konzeption der Oberfläche begründen und deren Nutzung erläutern.

8.1 Anforderungen

Die Benutzeroberfläche muss die folgenden, aus der Problemstellung resultierenden Funktionalitäten erfüllen:

1. Wählen, Festlegen und Ändern des Suchverzeichnisses
2. Auswählen der Keywords
3. Auswählen der Freitextsuche
4. Eingabe des Suchtextes
5. Logische Operatoren zur externen Verknüpfung zur Verfügung stellen
6. Logische Operatoren zur internen Verknüpfung zur Verfügung stellen
7. Anzeigen der Anfrage
8. Starten der Suche
9. Zurücksetzen der Anfrage
10. Anzeige der Resultate

Neben den oben genannten inhaltlichen Anforderungen sind noch weitere Punkte bezüglich Anwenderfreundlichkeit zu berücksichtigen:

1. Übersichtlichkeit
2. Intuitive Bedienbarkeit, d.h. der Nutzer soll möglichst wenig nachdenken müssen

Die Realisierung der soeben beschriebenen Anforderungen wird in den folgenden Abschnitten erläutert.

8.2 Grundaufbau der Oberfläche

Die Oberfläche wurde, um dem Nutzer eine gewisse Übersichtlichkeit zu bieten, in drei Bereiche gegliedert, die in Abbildung 8.1 gezeigt sind.

Der oberste Bereich beinhaltet die Verzeichnisauswahl, da dies der erste Schritt ist, der vom Anwender ausgeführt werden muss.
In der Mitte befindet sich die Anzeige der Gesamtanfrage, weil sie sich dort sofort im Blickfeld des Nutzers befindet.
Die Editierung der Teilanfragen wurde im unteren Teil der Oberfläche untergebracht, weil sich hier die meisten Bedienungselemente befinden, weshalb jede andere Position unweigerlich Einbußen bezüglich Übersichtlichkeit zur Folge hätte.
Funktional zusammengehörige Elemente wurden hierbei stets nah beieinander angeordnet, was dem Gesetz der Nähe entspricht.
Dieses Gesetz besagt, dass Elemente, die nah zusammen liegen, vom Anwender als zusammengehörig wahrgenommen werden ([Hof17], S.17).

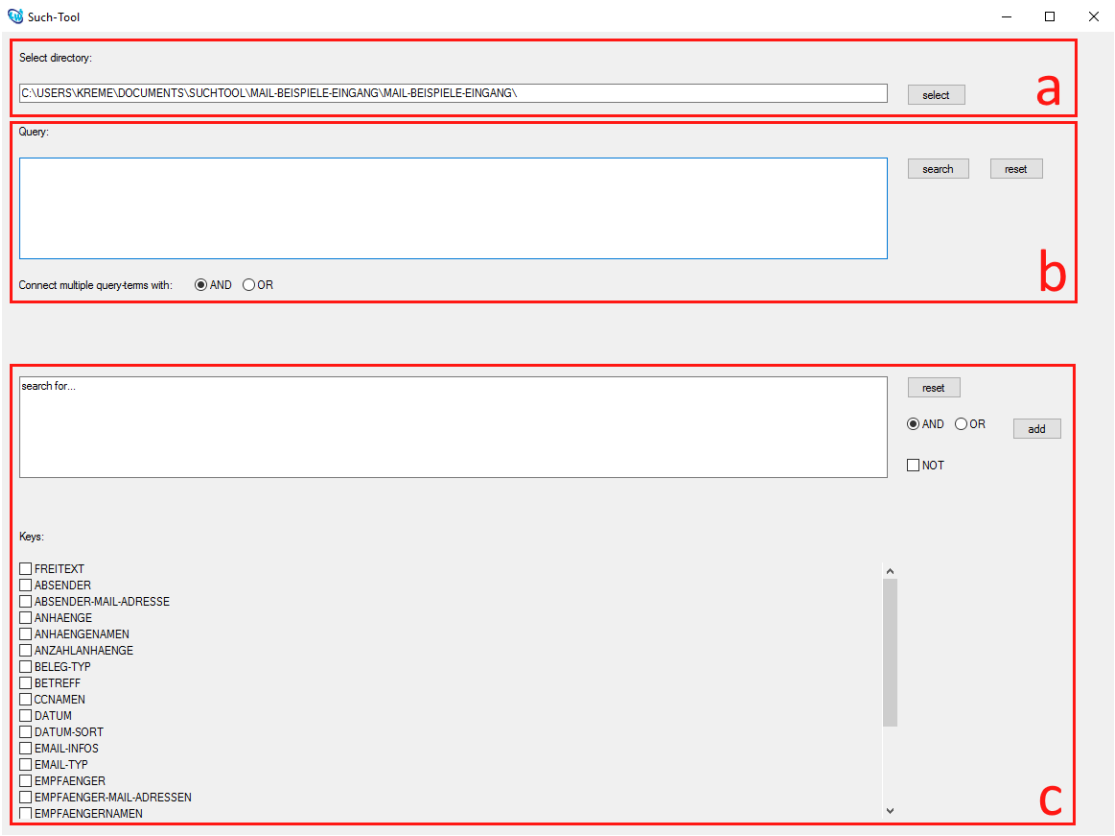

Abb. 8.1. Gliederung der Benutzeroberfläche. Teil *a* beinhaltet die Verzeichnisauswahl, Teil *b* die gesamte Suchanfrage und Teil*c* die Eingabe der Teilanfrage (Eigene Abbildung).

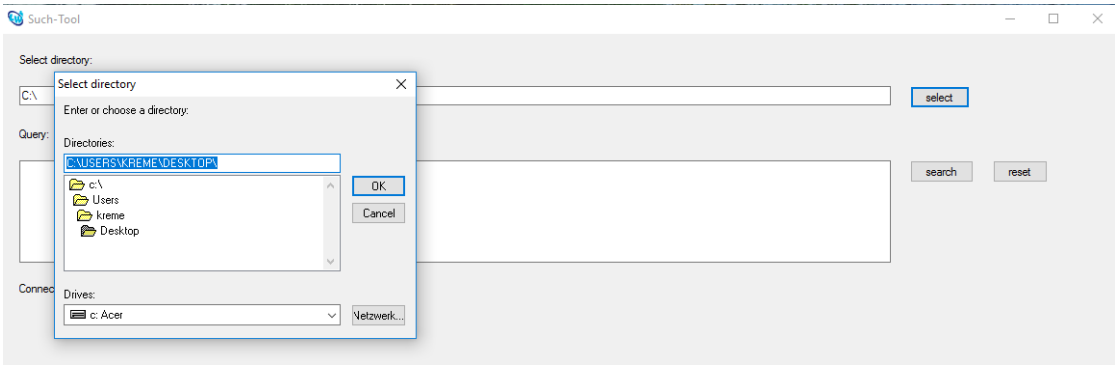

Abb. 8.2. Verzeichnisauswahl (Eigene Abbildung)

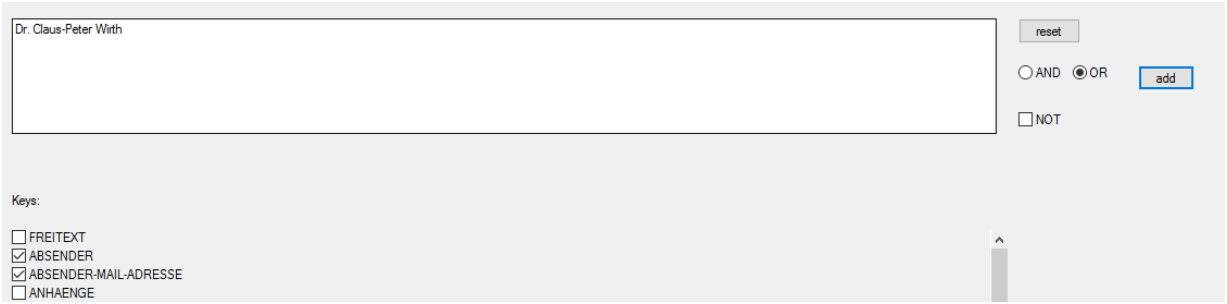

Abb. 8.3. Keywords auswählen und intern verknüpfen, hier OR (eigene Abbildung)

Abb. 8.4. Keywords mit Freitext verknüpfen (eigene Abbildung)

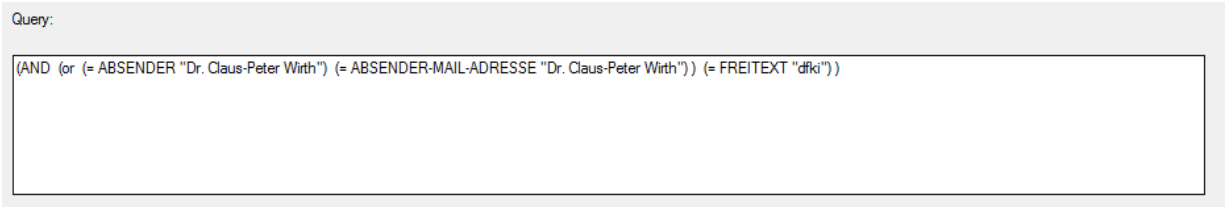

Abb. 8.5. Display (eigene Abbildung)

8.3 Verzeichnisauswahl

8.4 Suchanfrage

8.5 Ergebnis

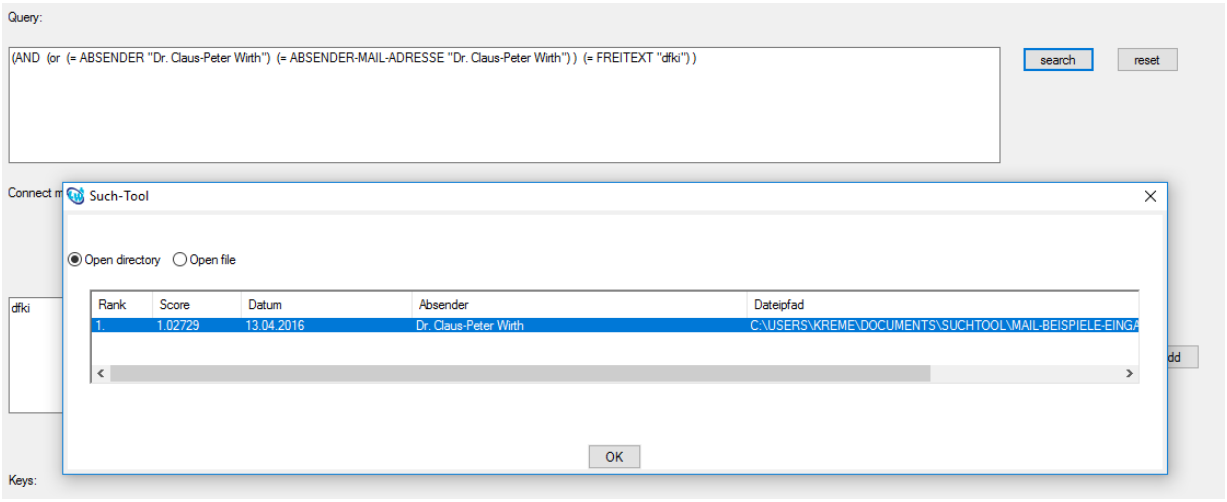

Abb. 8.6. Resultatfenster für obige Anfrage (eigene Abbildung)

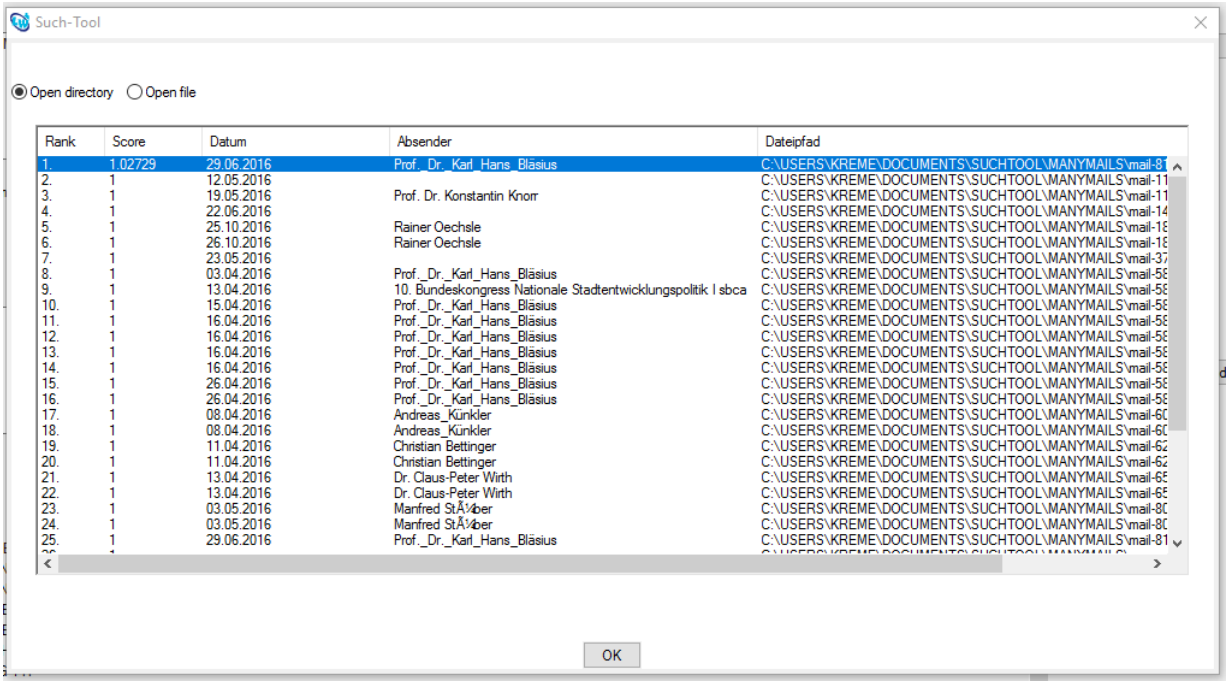

Abb. 8.7. Scrollbare Anzeige für viele Treffer (eigene Abbildung)

9

Zusammenfassung und Ausblick

In diesem Kapitel soll die Arbeit noch einmal kurz zusammengefasst werden. Insbesondere sollen die wesentlichen Ergebnisse Ihrer Arbeit herausgehoben werden. Erfahrungen, die z.B. Benutzer mit der Mensch-Maschine-Schnittstelle gemacht haben oder Ergebnisse von Leistungsmessungen sollen an dieser Stelle präsentiert werden. Sie können in diesem Kapitel auch die Ergebnisse oder das Arbeitsumfeld Ihrer Arbeit kritisch bewerten. Wünschenswerte Erweiterungen sollen als Hinweise auf weiterführende Arbeiten erwähnt werden.

Literaturverzeichnis

Aca. *Academic - Academic dictionaries and encyclopedias, Universal Lexikon.* http://universal_lexikon.deacademic.com/253489/Informationsr%C3%BCckgewinnung, letzter Zugriff am 05.06.2017.

CDM08. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: *Introduction to Information Retrieval.* Cambridge University Press, 2008.

Fer03. Ferber, Reginald: *Information Retrieval, Suchmodelle und Data-Mining-Verfahren für Textsammlungen und das Web.* dpunkt.verlag, 2003.

Hen08. Henrich, Andreas: *Information Retrieval 1 -Grundlagen, Modelle und Anwendungen*, 2008. https://www.uni-bamberg.de/fileadmin/uni/fakultaeten/wiai_lehrstuehle/medieninformatik/Dateien/Publikationen/2008/henrich-ir1-1.2.pdf, letzter Zugriff am 10.06.2017 .

Hof17. Hofer, Andreas: *User-Interface-Design WS 2016/17 - 2 Gestalten.* Vorlesungsskript, FH-Trier, 2017.

Ion16. Ionescu, Stelian: *split-sequence Package*, 2016. https://github.com/sharplispers/split-sequence/blob/master/split-sequence.lisp, letzter Zugriff am 23.06.2017.

Jän84. Jänich, Klaus: *Lineare Algebra - Ein Skriptum für das erste Semester.* Springer-Verlag, 1984.

KMOB14. Kweku-Muata Osei-Bryson, Ojelanki Ngwenyama: *Advances in Research Methods for Information Systems Research - Data Mining, Data Envelopment Analysis, Value Focused Thinking.* Nummer 34 in *Integrated Series in Information Systems.* Springer Verlag, 2014.

PDVC06. Prof. Dr. Volker Claus, Prof. Dr. Andreas Schwill: *Duden - Informatik A-Z, Fachlexikon für Studium, Ausbildung und Beruf.* Dudenverlag, 2006.

SB10. Stefan Büttcher, Charles L.A.Clarke, Gordon V.Cormack: *Information Retrieval - Implementing and Evaluating Search Engines.* The MIT Press - Massachusetts Institute of Technology, 2010.

A

Glossar

DisASTer	DisASTer (Distributed Algorithms Simulation Terrain), A platform for the Implementation of Distributed Algorithms
DSM	Distributed Shared Memory
AC	Linearisierbarkeit (atomic consistency)
SC	Sequentielle Konsistenz (sequential consistency)
WC	Schwache Konsistenz (weak consistency)
RC	Freigabekonsistenz (release consistency)

B

Erklärung der Kandidatin / des Kandidaten

☐ Die Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen- und Hilfsmittel verwendet.

☐ Die Arbeit wurde als Gruppenarbeit angefertigt. Meine eigene Leistung ist ...

Diesen Teil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser: ...

Datum	Unterschrift der Kandidatin / des Kandidaten