Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

Information Engineering 1: Information Retrieval

Systeme, Architektur

Kapitel 3

Martin Braschler

Agenda

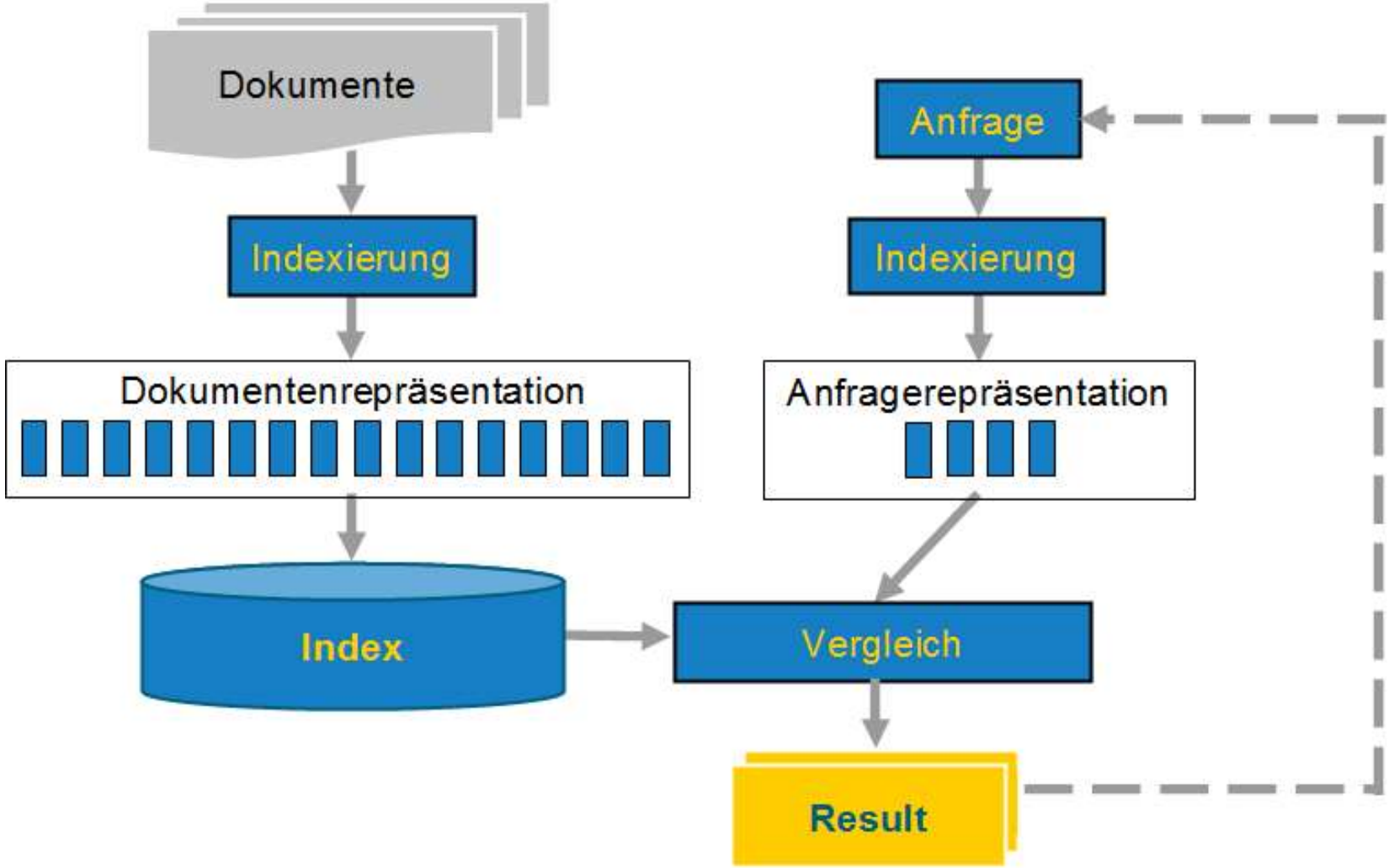
- IR Prozess
- Invertierte Liste
- Granularität
- Lokalisierung in grossen Listen
- MiniRetrieve

Fragen

- Warum kann Google 170 Terabyte[1] in 0.1 Sekunden durchsuchen?

- [1] eine der Schätzungen zur Grösse des Oberflächen-Webs, heute längst veraltet, denken Sie sich eine andere hohe Zahl

IR-Prozess


Dokumente
Indexierung
Dokumentenrepräsentation
Index
Anfrage
Indexierung
Anfragerepräsentation
Vergleich
Result

Fragen

- Was für Indexe werden benötigt? Aufbau?

Invertierte Liste, ein Beispiel (nach H-P. Frei)

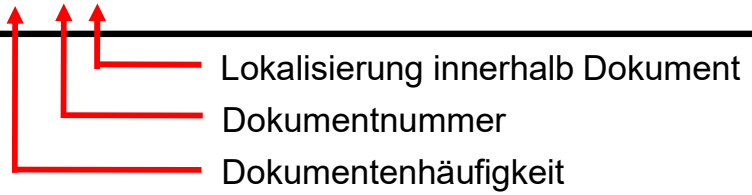
Zürcher Hochschule für Angewandte Wissenschaften

- Eine invertierte Liste besteht aus einer geordneten Liste von Merkmalen mit mindestens Information über die Häufigkeit jedes Merkmals in der Dokumentenkollektion.
- Kleine Beispielskollektion mit 6 kurzen Dokumenten:

Dok.#	Text
1	Pease porridge hot, pease porridge cold
2	Pease porridge in the pot
3	Nine days old
4	Some like it hot, some like it cold
5	Some like it in the pot
6	Nine days old

Invertierte Liste

Merkmal #	Merkmal	Merkmal und Lokalisierungsinformation
1	cold	2; (1, 6), (4, 8)
2	days	2; (3, 2), (6, 2)
3	hot	2; (1, 3), (4, 4)
4	in	2; (2, 3), (5, 4)
5	it	2; (4, 3,7), (5, 3)
6	like	2; (4, 2,6), (5, 2)
7	nine	2; (3, 1), (6, 1)
8	old	2; (3, 3), (6, 3)
9	pease	2; (1, 1,4), (2, 1)
10	porridge	2; (1, 2,5), (2, 2)
11	pot	2; (2, 5), (5, 6)
12	some	2; (4, 1,5), (5, 1)
13	the	2; (2, 4), (5, 5)



Inhalt einer invertierten Liste

- Beachte Sie, dass in diesem einfachen Beispiel die Merkmale weder gestemmt noch eine Stoppwortliste angewandt worden ist.
- Die invertierte Liste zeigt:
  - Dokumenthäufigkeit, z.B. in wie vielen Dokumenten ein Merkmal auftritt
    - (in diesem einfachen Beispiel ist es zufälligerweise immer 2 für jedes Merkmal)
  - In welchem Dokument das Merkmal auftritt
  - An welche Stelle im Dokument das Merkmal auftritt (optional)
- Eine invertierte Liste kann mehr oder weniger Information enthalten, z.B:
  - Gewichtung der Merkmale
  - Kategorisierung der Merkmale

Granularität einer invertierten Liste

- Granularität ist die Genauigkeit, zu welcher eine invertierte Liste die Lokalisierung der Merkmale festlegt.
- **Grober Index**: z.B. nur Blöcke von Texten in welchen mehrere Dokumente gespeichert sein können
- **Mittlerer Index**: z.B. die Lokalisierung sind als Dokumentnummern gespeichert
- **Feiner Index**: Index liefert ein Satz, Wort oder sogar ein Byte zurück

Fragen

- Was für einen Einfluss hat die Granularität auf den Speicherbedarf und die Rechenkomplexität?

Granularität

- Eine invertierte Liste speichert eine hierarchische Menge von Zeigern, wie:
  - Textblock
  - Dokumentnummer
  - Bandnummer
  - Kapitelnummer
  - Paragraphennummer
  - Satznummer
  - Wortnummer
- Die Granularität wird gemäss den Anforderungen bestimmt. Je feiner die Granularität, desto rechen- und speicherintensiv wird es.
- Bereits ein relativ einfacher Index benötigt 50% bis 100% vom Speicherbedarf der ursprünglichen Dokumentkollektion.

Boolesches Retrieval mittels invertierter Liste

- Resultat wird aus der invertierten Liste generiert
  - Einfache Anfrage, besteht aus Merkmal t:
    - Hole alle Dokumente in der invertierten Liste von t
  - t_1 **and** t_2 **and** … **and** t_n:
    - Hole Dokumente in der Schnittmenge von n invertierten Listen t_1 …. t_n
  - t_1 **or** t_2 **or** … **or** t_n :
    - Hole Dokumente in der Vereinigungsmenge von n invertierten Listen t_1 …. t_n
  - **not** t:
    - Hole Dokumente in der Komplementärmenge (selten unterstützt! → wieso? Abhilfe?)

Lokalisierungsmerkmale in Listen

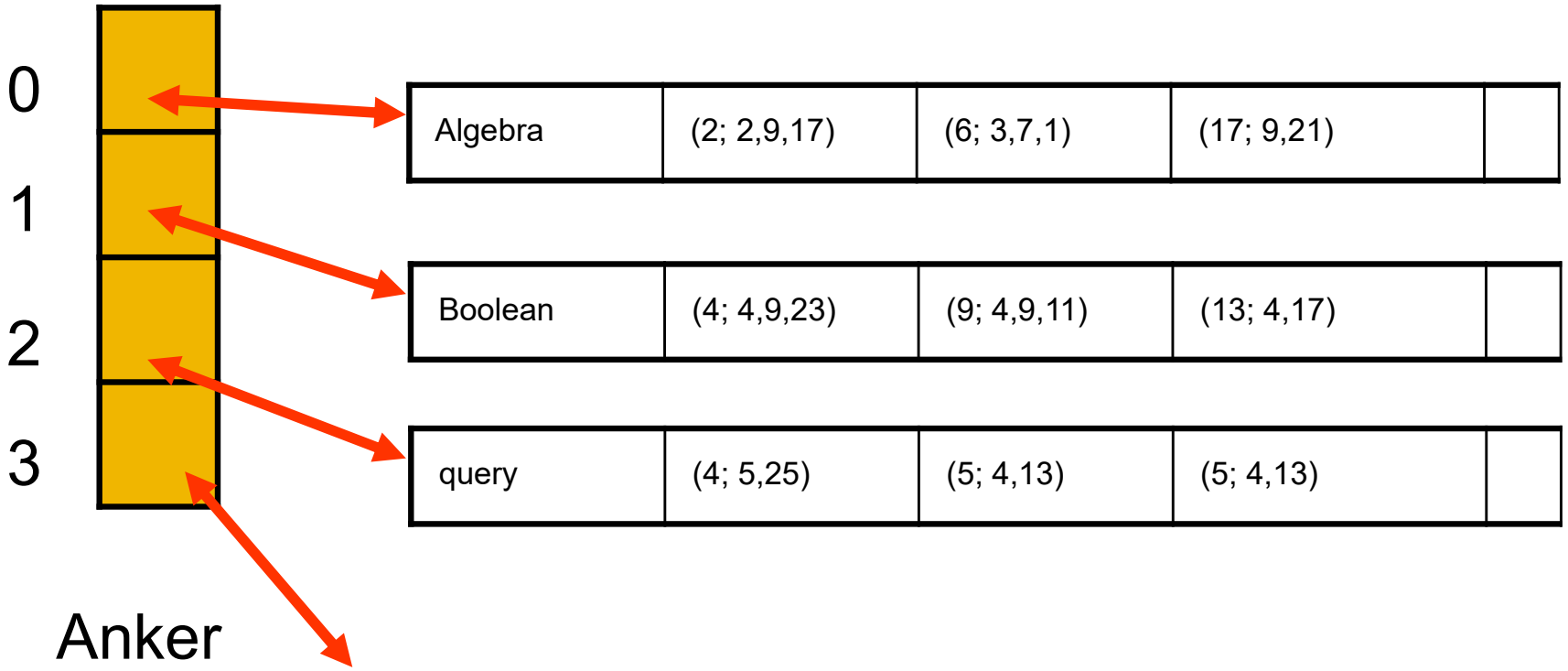
- Eine invertierte Liste ist eine extrem grosse Liste mit allen Merkmalen aus allen Dokumenten als Einträgen. Wenn Anfragen verarbeitet werden, müssen diese Merkmale aufgefunden werden:
  - Speichere die Merkmale in einer alphabetischen Liste:
    - Durchlaufe die geordnete Liste, z.B.: binäre Suche
  - Speichere die Merkmale in einer Hash-Tabelle:
    - Die Lokalisierung der Merkmale in einer Hash-Tabelle ist sehr schnell.

Hash-Datei

- Adressierung im Hash:
  - Die Adresse der Merkmale ist mit einem „Schlüssel“ t_i verbunden auf welche mittels einer hash-Funktionen $h(t_i)$ zugegriffen wird.
- Die Hash-Funktion h erstellt Adressen, die zu den Merkmalen zeigen.
- Idealerweise
  - sollte die Hash-Funktion h die Adressen gleichmässig über den verfügbaren Speicherraum verteilen.
  - sollten zwei Merkmale t_i und t_j, die unterschiedlich sind ($t_i \neq t_j$) keine identischen Adressen produzieren, d.h. es sollte $h(t_i) \neq h(t_j)$ sein.
- **Kollisionen** sind trotzdem **unvermeidbar** und müssen entsprechend behandelt werden.

Hash-Datei Organisation

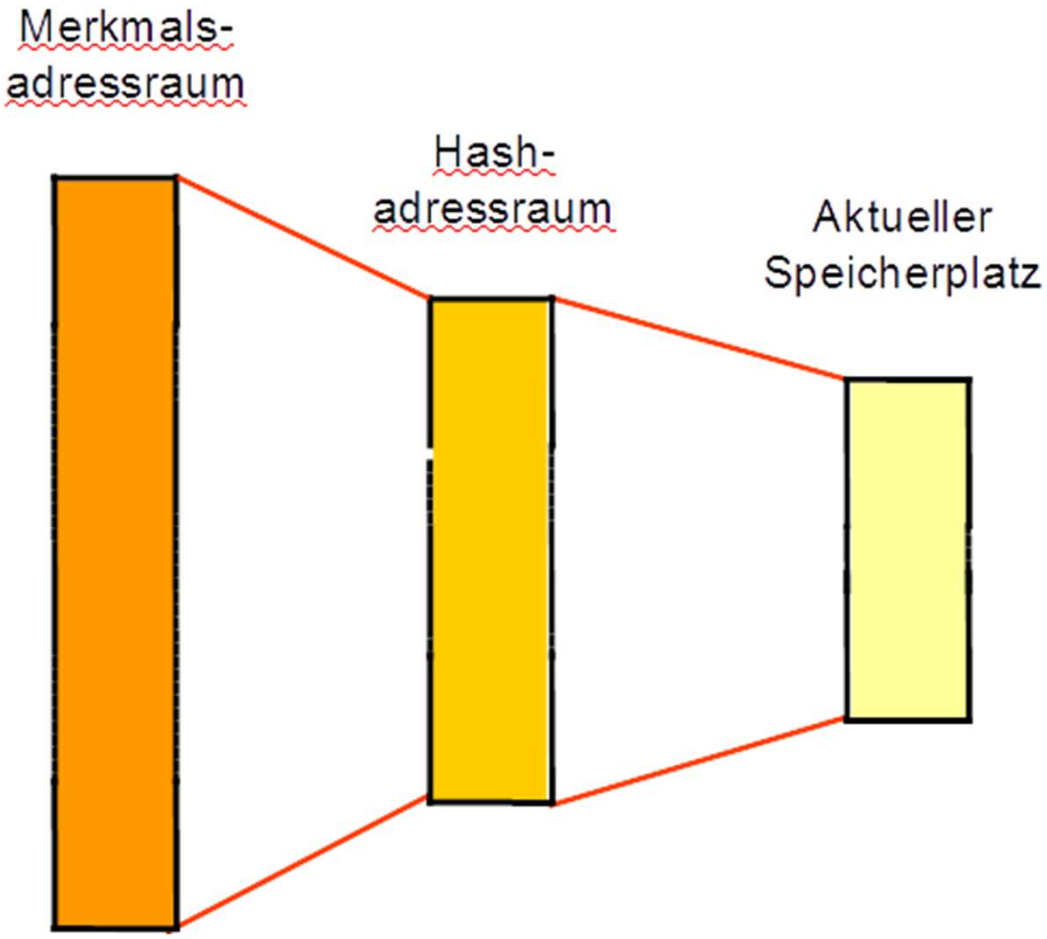
- Im Generellen, zeigt eine Funktion h(t) auf einen Anker der invertierten Liste:



Hash-Funktion Beispiele (nach H.-P. Frei)

- Annahme: Verfügbarer Adressraum ist 2^m
- Drei Beispiele von möglichen Hash-Funktionen:
  - Man nehme das Quadrat von der binären Repräsentation von t_i; selektiere m Bits von der Mitte des Resultates
  - Schneide die binäre Repräsentation von t_i in Stücke von m Bits und addiere diese zusammen. Selektiere die m niedrigstwertigen Bits der Summe als Hash-Adresse.
  - Dividiere die entsprechende Zahl von t_i durch die Länge des verfügbaren Speichers 2^m und verwende den Rest als Hash-Adresse.

Speicherplatzbedarf



Umso kleiner der Speicherplatz wird, desto wahrscheinlicher werden Kollisionen. Die Kollisionen müssen behandelt werden!

Lösen von Kollisionen

- Es gibt grundsätzlich zwei Methoden um mit Kollisionen umzugehen:
- **Lineare Methode**: vermeide den Adressraum der bereits besetzt ist
  - suche den nächst freien Adressraum, wo die Kollision auftrat
  - Suchstrategie:
    - gehe zum nächst freien Adressraum
    - verschiebe um eine fixe Anzahl von Adressen
  - Das Ziel besteht immer darin die Merkmale, für welche die Kollision auftraten, so nahe als möglich bei den ursprünglichen Hash-Adressen zu speichern.
- **Verkettete Kollisionsdateien**: Verknüpfe Dateien mit Zeigern
  - Zeiger auf einen anderen freien Adressraum im Speicherplatz
  - Stelle zusätzlichen Speicherplatz bereit für Kollisionsdateien

Schlussfolgerung

- Es gibt verschiedene Datenstrukturen um Indexe zu speichern.
- Die Auswahl der Datenstruktur hängt von der jeweiligen Situation ab.
- Die Granularität hängt von den verfügbaren Rechnerressourcen und der Suchzeit ab.
- Die Auswahl einer guten Hash-Funktion ist massgebend für effizientes Hashing.
- Kollisionen können nicht vermieden werden; Sie müssen ordnungsgemäss gelöst werden (verschiedene Methoden für unterschiedliche Situationen)

MiniRetrieve

- Im folgenden wird der Pseudo-Code für ein minimales IR-Systemchen dargestellt (150 Zeilen Perl mit grosszügigem Spacing/Leerzeilen)
- Dieses implementiert nur ein minimales Indexing (Tokenisierung, ggf. Gross-/Kleinschreibung)
- Es implementiert die tf.idf-Cosinus-Formel für das Ranking
- Es hält alle Datenstrukturen im Arbeitsspeicher
- Kann tausende von Dokumenten indexieren
- Liefert sinnvolle Resultate
- Kann im Batch-Modus Anfragen abarbeiten
- Es vermittelt die Grundlagen, wie Abfragen funktionieren, und warum gewisse Vorgänge effizient/ineffizient sind

Gewichtungsformel RSV

$$a_{i,j} := ff(\varphi_i, d_j) * idf(\varphi_i)$$

$$b_i := ff(\varphi_i, q) * idf(\varphi_i)$$

$$RSV(q, d_j) := \frac{\sum_{\varphi_i \in \Phi(q) \cap \Phi(d_j)} a_{i,j} * b_i}{\sqrt{\sum_{\varphi_i \in \Phi(d_j)} a_{i,j}^2} * \sqrt{\sum_{\varphi_i \in \Phi(q)} b_i^2}}$$

- Wobei:
  - RSV = retrieval status value
  - ff = feature frequency
  - idf = inverse document frequency
  - d = document
  - q = query
  - φ = term

Architektur MiniRetrieve

Zürcher Hochschule für Angewandte Wissenschaften

Invertierter Index

Quelle: Dokumente

Merke: fname = docid

Idf	
Key	Value
word1	value
word2	value
…	…
wordN	value

Inverted Index	
Key	Value
word1	fname→freq
word2	fname→freq
word3	fname→freq
word4	fname→freq
word5	fname→freq
…	…
…	…
wordN	fname→freq

fname→freq	
Key	Value
fname1	freq
fname2	freq
fname3	freq
fname4	freq
…	…
…	…
fnameN	freq

fname→freq	
Key	Value
fname1	freq
fname2	freq
fname3	freq
fname4	freq
…	…
…	…
fnameN	freq

Nicht-Invertierter Index

Quelle: Dokumente

dNorm	
Key	Value
fname1	value
fname2	value
…	…
fnameN	value

Non-Inverted Index	
Key	Value
fname1	word→freq
fname2	word→freq
fname3	word→freq
fname4	word→freq
fname5	word→freq
…	…
…	…
fnameN	word→freq

word→ freq	
Key	Value
word1	freq
word2	freq
word3	freq
word4	freq
…	…
…	…
wordN	freq

word→ freq	
Key	Value
word1	freq
word2	freq
word3	freq
word4	freq
…	…
…	…
wordN	freq

Anfrageindex

Quelle: Anfrage

Merke: fname = queryid

Query Index	
Key	Value
fname1	word→freq
fname2	word→freq
fname3	word→freq
fname4	word→freq
fname5	word→freq
…	…
…	…
fnameN	word→freq

word→freq	
Key	Value
word1	freq
word2	freq
…	…
…	…
wordN	freq

word→freq	
Key	Value
word1	freq
word2	freq
…	…
…	…
wordN	freq

PseudoCode - Teil 1.1

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

```
# $ -> Variable , @ -> Array , % -> Hash
indexing of documents and queries{
    foreach $doc in @documents {    #/create inverted and non-inverted index
      get @terms by tokenizing $doc;
      foreach $term in @terms {
            %invIndex{$term}{$doc} += 1 # increment frequency in inverted index
            %nonInvIndex{$doc}{$term} += 1 # increment frequency in non-inv index
      }
    }
    foreach $query in @queries{    #create index of queries
      get @terms by tokenizing $query;
      foreach $term in @terms {
            %queries{$query}{$term} += 1 # increment frequency
      }
    }
}
```

PseudoCode - Teil 1.2

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

```
# $ -> Variable  ,  @ -> Array  ,  % -> Hash
calculate all idfs and document normalizers{
      foreach $doc in %nonInvIndex{
        %dNorm{$doc} = 0;
        foreach $word in %nonInvIndex{$doc} {
              %idf{$word} = log ( (1 + totalNummerOfDocuments) / ( 1 + documentFrequency) )
              $a = %nonInvIndex{$doc}{$word} * %idf{$word}
              %dNorm{$doc} += $a * $a;
        }
        %dNorm {$doc} = Math.sqrt(%dNorm{$doc}
      }
}
```

Architektur MiniRetrieve

Zürcher Hochschule
für Angewandte Wissenschaften

zh
aw

accumulator

accu	
Key	**Value**
doc 1	acc. value
doc 2	acc. value
doc 3	acc. value
doc 4	acc. value
…	…
…	…
doc N	acc. value

Der Akkumulator summiert das Produkt von idf und Termhäufigkeit für jedes Word, das im entsprechenden Dokument vorkommt.

dNorm

dNorm	
Key	**Value**
fname1	value
fname2	value
fname3	value
fname4	value
fname 5	value
…	…
…	…
fnameN	value

Dokumentennorm „dNorm“ wird für alle Dokumente vorberechnet

idf

idf	
Key	**Value**
word1	value
word2	value
word3	value
word4	value
word5	value
…	…
…	…
wordN	value

Der idf (Inverse document frequency) wird für alle Wörter in allen Dokumenten vorberechnet, ggf. auch für Anfrageterm mit df=0.

Gewichtungsformel RSV

$$a_{i,j} := ff(\varphi_i, d_j) * idf(\varphi_i)$$

$$b_i := ff(\varphi_i, q) * idf(\varphi_i)$$

$$RSV(q, d_j) := \frac{\sum_{\varphi_i \in \Phi(q) \cap \Phi(d_j)} a_{i,j} * b_i}{\sqrt{\sum_{\varphi_i \in \Phi(d_j)} a_{i,j}^2} * \sqrt{\sum_{\varphi_i \in \Phi(q)} b_i^2}}$$

%accu

%dNorm

$qNorm

- Wobei:
  - RSV = retrieval status value
  - ff = feature frequency
  - idf = inverse document frequency
  - d = document
  - q = query
  - φ = term

PseudoCode - Teil 2.1

Zürcher Hochschule für Angewandte Wissenschaften

```
process queries{
    foreach $query in %queryIndex {
        $qNorm = 0
        create new %accu
        foreach $queryWord in %queryIndex{$query} {  # process all query terms
            if(! %idf{$queryWord}{
                %idf{$queryWord} = log( 1 + totalNummerOfDocuments);
            }
            $b = %queries{$query}{$queryWord}  *  %idf{$queryWord}
             $qNorm += ($b * $b)
            if( %invIndex{$queryWord} is definded ) { # if query term occurs in collection
                foreach $document in @invindex{$queryWord} {
# document scores are added up in accumulators. filename serves as document identifier
                    $a = %invIndex{$queryWord}{$document} * $idf
                    %accu{$document} +=  ( $a * $b );
                }
            }
        }
        #….. 2. Teil ……
    }
}
```

PseudoCode - Teil 2.2

Zürcher Hochschule
für Angewandte Wissenschaften

```
process queries{
    foreach $query in %queryIndex {
        #….. 1. Teil ……
         $qNorm = Math.sqrt( $qNorm )
        foreach $document in %accu {  # normalize length of vectors
                %accu{$document} /= (%dNorm{$document}) * $qNorm )
        }
        set @results = sort %accu by values  # sort and return 1000 best
    results
        foreach $result in @results{
                print "$queryid Q0 $document $rank $accuValue"
        }
    }
}
```

Optimierungen

- Sortieren beschleunigen
- Lange Postinglisten abschneiden
- Manche Gewichtungsschemata brauchen keinen NonInvIndex
- Rel. Feedback braucht einen NonInvIndex

Verständnisfrage

- Warum bieten IR-Systeme im Allgemeinen keine Wildcards?

Verständnisfrage

- Was bedeutet es, den Index zu updaten?
- Was bedeutet es, Dokumente zu löschen?