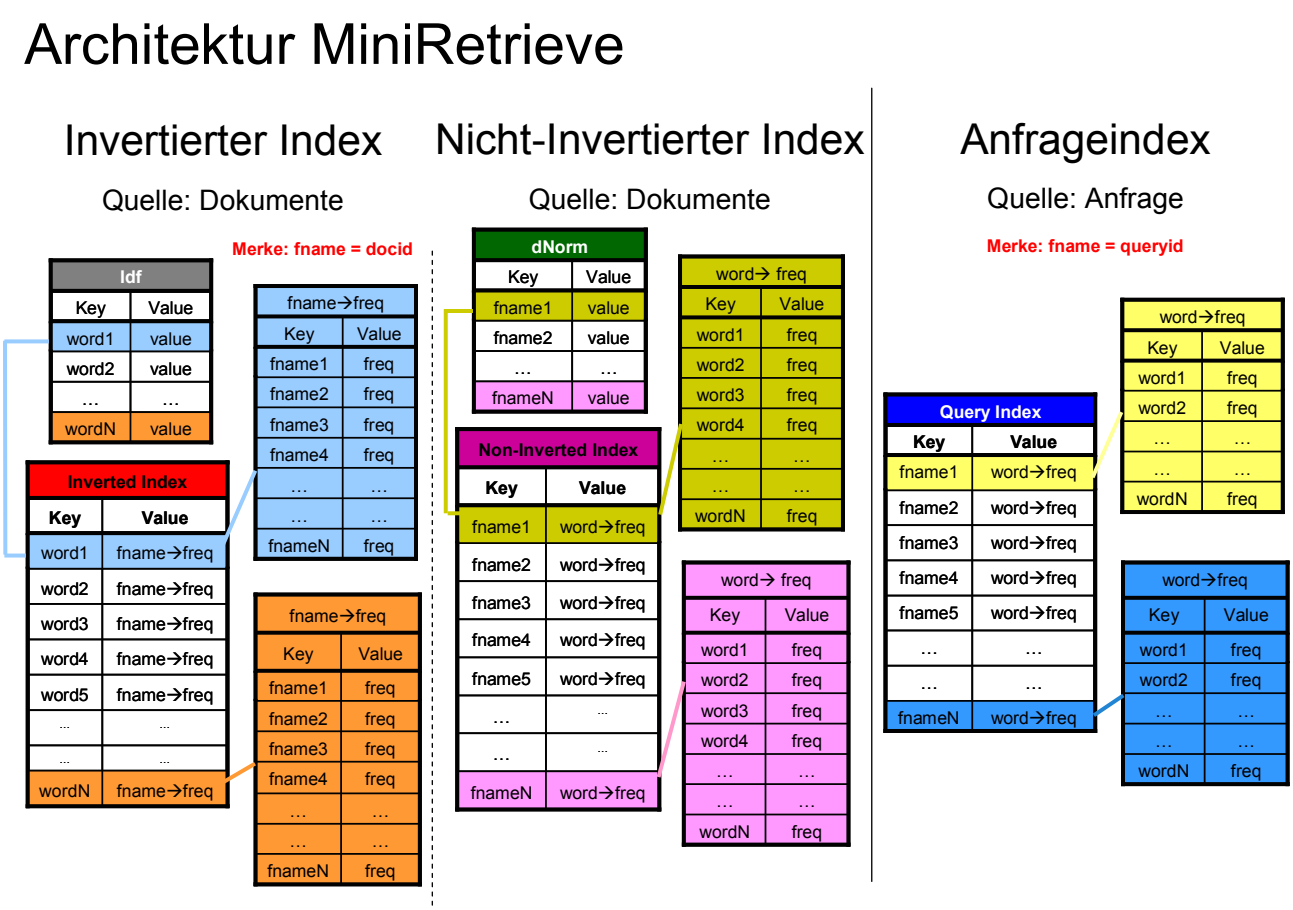
Kurs IR-Grundlagen, Praktikum 5

Ziel

In diesem Praktikum geht es um die Umsetzung eines einfachen Information Retrieval Systems. Grundlage dazu bilden die Architekturfolien und der Pseudo-Code aus der Theorie. Es geht in diesem Praktikum weniger um programmiertechnische Aspekte (wie man Dateien einliest etc.), sondern um das Verständnis, wie IR-Architektur und Gewichtungsschemata in der Praxis umgesetzt werden. Dieses Verständnis ist der Schlüssel, Limitationen von IR-Systemen zu verstehen, respektive solche Systeme optimal aufzusetzen. Das in diesem Praktikum zu erarbeitende System verwendet das Gewichtungsschema tf.idf-Cosinus (Vektorraummodell), welches wir in Kapitel 2 behandelt haben.



PseudoCode - Teil 1.1

```
# $ -> 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

```
# $ -> 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

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

```
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

```
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"
        }
    }
}
```

Kollektion

Wir benutzen in diesem Praktikum die Cranfield Testkollektion. Diese besteht aus 1400 Dokumenten und 225 Anfragen. Sowohl die Filenamen der Dokumente (Ordner „documents“) als auch der Anfragen (Ordner „queries“) entsprechen jeweils der eindeutigen Identifikationsnummer des Dokumentes/Anfrage.

Aufgabe

Es ist Ihnen überlassen, in welcher Programmiersprache Sie Ihr IR-System auf die Beine stellen wollen. Hilfe erhalten Sie aber auf alle Fälle in Perl und Java.
In Perl lässt sich ein funktionierendes IR-System in weniger als 150 Zeilen Code umsetzen. In Java müssen Sie dazu im Allg. etwas mehr Code schreiben (je nach Ansatz).
Es ist ein Grundgerüst in Java vorgegebenen, das Ihnen viel Programmierarbeit abnimmt. Dieses Gerüst ist simpel gestalten und erhebt keinen Anspruch auf besonders gutes Design, sondern versucht die Vorgabe aus dem Pseudo-Code möglichst direkt umzusetzen. Die Verwendung dieses Gerüstes ist selbstverständlich freiwillig. Wenn Sie den Anspruch haben, ein IR-System von Grund auf neu zu programmieren, können direkt bei Aufgabe 2 beginnen.

Aufgabe 1

Nehmen Sie das vorgegebene Grundgerüst „MiniRetrieveGrundrundgeruest.java“ und implementieren Sie darauf basierend ein funktionsfähiges IR-System. Das Grundgerüst folgt der bereits bekannten Architektur und dem Pseudo-Code. D.h. die Datenstruktur der Indizes, die sonstige Programmstruktur und auch die meisten Methoden sind vorgegeben. Studieren Sie in einem ersten Schritt das vorgegebene Grundgerüst. Versuchen Sie dieses vollständig zu verstehen. Suchen Sie die Verbindungen zur Version in Pseudo-Code.
Sobald Sie das Gerüst verstanden haben, machen Sie sich daran die fehlenden Methoden zu implementieren. Die Methodenköpfe sind dabei bereits vorgegeben.

Aufgabe 1.1 – Erstellung invertierten und nicht-invertierten Index

Implementieren Sie die Erstellung des invertierten und nicht-invertierten Indexes `createIndexes(String directory)`.

- Durchlaufen Sie dazu alle Dokumente im Ordner „documents“ der Cranfield Kollektion (Filename entspricht DokumentenId)
- Um den Inhalt eines Files auszulesen, können Sie die statische Methode „`readFile(String filename)`“ der Klasse `Utilities` benutzen.
- Tokenisieren Sie den Inhalt jeder einzelnen Datei
  - Benutzen Sie dazu die Regular Expression „\W“. Diese splittet den Dokumenteninhalt bei allen nicht-Wort-Characters (eine Vorlage dazu finden Sie bei der Tokenisierung der Anfragen).
  - Speichern Sie die Terme in beiden Indexes
    - Der nicht-/invertierte Index besitzt dazu eine fertig ausimplementierte Methode „`put(String filename, String term, Integer termFrequency)`“, die Sie dazu benutzen sollen
    - addieren Sie Häufigkeit auf, falls der Term schon im Index existiert

Instanzvariablen, die Sie dazu nutzen sollten:

1. myInvertedIndex -> Instanz des invertierten Index
2. myNonInvertedIndex -> Instanz des nicht invertierten Index

3. dNorm -> Dokumentennorm und den Normwert auszulesen
4. numberOfFiles -> um die Anzahl Files zu speichern

Aufgabe 1.2 – Berechnungen IDF und Normen

Implementieren Sie Methode `calculateIdfAndNorms()`

- Durchlaufen Sie alle Dokumente
  - Durchlaufen Sie pro Dokument alle Terme und berechnen Sie den IDF und die Dokumentennorm
    - Füllen Sie die berechneten Werte in die vorgegeben HashMaps „dNorm“ und „idf“ ab

Instanzvariablen, die Sie dazu nutzen sollten:

1. myInvertedIndex -> Instanz des invertierten Index
2. myNonInvertedIndex -> Instanz des nicht-invertierten Index
3. dNorm -> Dokumentennorm um den Normwert pro Dokument zu speichern
4. idf -> Idf-Wert, den man pro Term in den Hash speichert
5. numberOfFiles ->Anzal der Dokumente in der Kollektion

Aufgabe 1.3 – Normalisierung der Vektoren

Implementieren Sie die Normalisierung der Vektoren
`normalizeVectors(NonInvertedIndex myNonInvertedIndex)`

- Lesen Sie die Norm der Dokumente aus
- Multiplizieren Sie den Wert des Akkumulators vor der Division noch mit Faktor 1000
- Dividieren Sie den Wert des Akkumulators durch das Produkt von Dokument- und Anfragenorm
- Speichern Sie den neu berechneten Wert wieder im Akkumulator

Instanzvariablen, die Sie dazu nutzen sollten:

1. accuHash -> Instanz des Akkumulators um Werte auszulesen und neu berechnete zu speichern
2. dNorm -> Dokumentennorm
3. qNorm -> Anfragenorm mit dem aktuellen Normwert

Aufgabe 1.4 – Berechnungen des Akkumulators

Implementieren Sie die Methode `processQueries()`noch fertig aus

- Iterieren Sie über alle Terme der Anfrage
  - Berechnen Sie den „Idf“ des Terms, falls dieser noch nicht existiert
  - Berechnen Sie mit „b“ die AnfrageNorm „qNorm“
  - Überprüfen Sie, ob der Anfrageterm im invertierten Index vorhanden ist. Falls ja:
    - Iterieren Sie über alle Dokumente, in denen der Anfrageterm vorkommt.
    - Berechnen sie mit „a“ und „b“ den Akkumulatorenwert und speichern diesen entsprechend

Instanzvariablen, die Sie dazu nutzen sollten:

- myInvertedIndex -> Instanz des invertierten Indexes, den Sie bei der Indexierung erstellt haben
- accuHash -> Instanz des Akkumulators zur Speicherung der berechneten Werte
- qNorm -> Anfragenorm, um den Normwert zu speichern
- idf -> Speicherung Idf-Wert für Anfrageterme, die noch nicht existieren

Aufgabe 2 - Für die ambitionierten Studenten

Implementieren Sie gemäss Architektur und Pseudo-Code ihr Retrieval System von Grund auf neu. Falls Sie wollen, können Sie den prozeduralen Pseudo-Code in ein geeignetes objektorientiertes Design bringen. In erster Linie geht es aber darum, das System IR-technisch korrekt zu implementieren (keine Programmieraufgabe im eigentlichen Sinn).

Aufgabe 3 – Erweiterung IR-System (für Fanatiker)

Falls Sie Lust haben, Ihr kleines IR-System weiter auszubauen, überlegen Sie sich, wie Sie dieses noch optimieren können. Wie kann man Stoppworte/Stemming einbauen? Oder finden Sie Synergien mit dem Semesterbeitrag?

Kontrolle

Um Ihren Programmcode auf Richtigkeit zu verifizieren, vergleichen sie ihr Resultat mit dem Resultfile der Musterlösung „top10ResultsOutput.txt“, das im Trec-Format vorliegt. Durch Rundungsfehler kann Ihr Resultat ein klein wenig abweichen, sollte aber die gleiche Rangierung der Dokumente zurückgeben. Pro Anfrage finden Sie jeweils die ersten zehn Resultate.

Trec-Format:

- Spalte 1 : Anfragenummer
- Spalte 2 : Konstante „Q0“
- Spalte 3 : Rangierung
- Spalte 4 : Dokumentennummer
- Spalte 5 : RSV-Value
- Spalte 6 : Systemname