Inverted Files for Text Search Engines

Justin Zobel, Alistair Moffat

Inhalt

- Einführung
- Index
  - Inverted Files
  - Indexkonstruktion
  - Indexverwaltung
- Optimierung
  - Indexkompression
- Weitere Ansätze zur Indexierung

Einführung

Wie suchen wir?

1) Angabe einer Frage, meist

- Schlagwörter: „Albert Einstein“, „Katzenklo“

2) Durchsuchen Liste mit Antworten

3) Folgen der Links

Bei Unzufriedenheit →

- Frage modifizieren
- Suchdomäne beschränken
- Einbeziehen od. Auslassen von Ausdrücken

Allgemeines

- Suchmaschinen:
  - Werkzeuge um bestimmte Dokumente in Sammlungen zu finden
  - Liefern leichten Zugang zu Informationen
  - Viele Suchmaschinen sind Indexbasiert

- Index? – …

Was ist ein Index?

- Datenstruktur für eine effiziente Suche

- **2 Varianten:**

1) word-level-Index
   - Abspeicherung der Wortpositionen
   - Zeigt an, wo der Ausdruck im Dokument vorkommt

2) document-level-Index
   - Anzeige darüber, ob Ausdruck in einem Dokument vorkommt
   - keine Information, wo der Ausdruck erscheint

- **Effizienteste Indexstruktur: Inverted File**

Inverted Files- Aufbau

- Sammlung von Listen
- Abspeicherung aller Terme die in Dokumenten vorkommen
- Warum ***Invertiert***?
- Üblich: Dokument → Term

  - jedem Dokument Zuordnung einer Menge von Termen

- Hier: Term → Dokument

  - jedem Term Zuordnung einer Liste von Informationen über das Vorkommen des Terms in Dokumenten

Inverted Files- Beispiel

- Dok.1: Die Hauptstadt von Deutschland heißt Berlin.
- Dok.2: Berlin ist die Hauptstadt von Deutschland
- Dok.3: Die Hauptstädte von Deutschland bzw. Spanien heißen Berlin bzw. Madrid
- **Anfrage: Hauptstadt Deutschland**

Erweiterung:

Stopping

Stemming

Wordpositionen

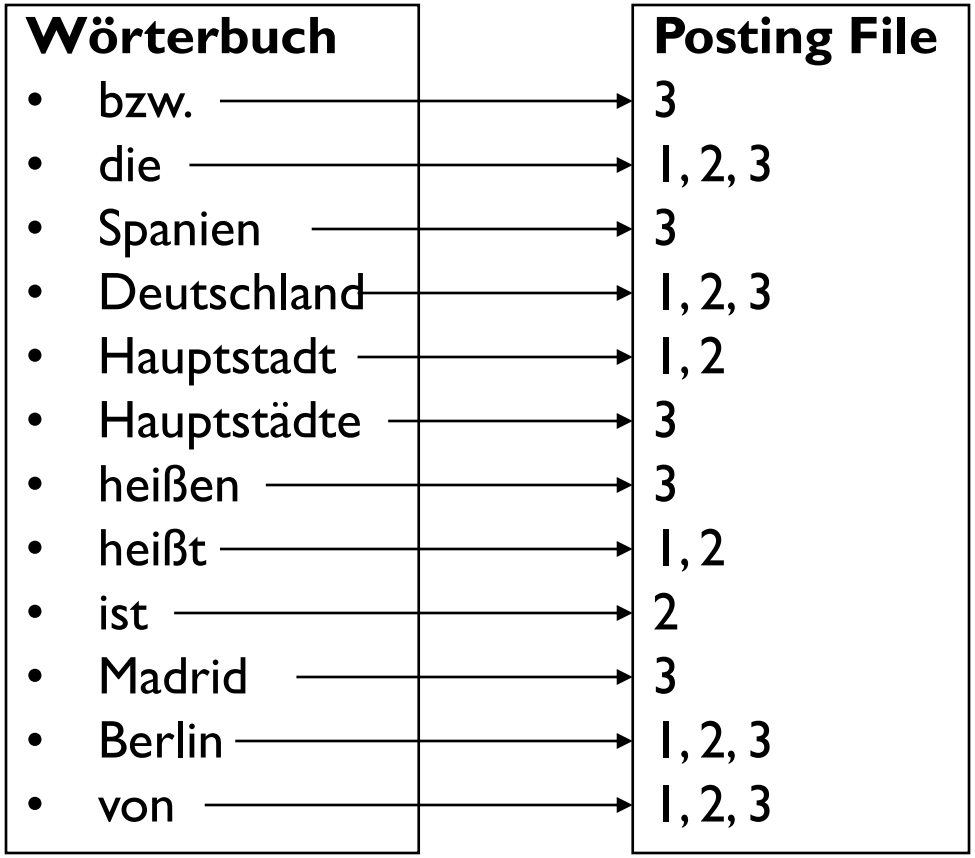
Inverted File:

- bzw. →3
- die→1,2,3
- Spanien→3
- Deutschland→1,2,3
- Hauptstadt→1,2
- Hauptstädte→ 3
- heißen→3
- heißt→1
- ist→2
- Madrid→3
- Berlin→1,2,3
- von→1,2,3

Inverted Files- Implementierung

Besteht aus:

- **Wörterbuch** (Vocabulary): Menge aller vork. Terme
- Meist im Speicher gehalten→Schnell
- **Posting File**: Speichert noch zusätzliche (anwendungsabhängige) Information, z.B.: TermVorkommen, Wortposition, ...



Inverted Files Vor- und Nachteile

Vorteile:

- Schneller Zugriff auf Terme, Dokumente
  →Kurze Suchzeit

- Verwaltung zusätzlicher Informationen (Position im Satz)

Nachteile:

- Hoher Speicheraufwand
- Aufbau und Aktualisierung aufwendig

Techniken um Suche effektiv zu machen

- Stemming → auf Stammwörter reduzieren:
  neues → neu, heißt → heiß

- Stopping → Stopwörter(die, ist, und, bzw..) raus
- Thesaurus: Zusammenfassung *unterschiedlicher* Worte *gleicher* Bedeutung

Vorteil:

Platzsparend

Suche effektiv → Index kleiner

Nachteil:

Ergebnis wird vielleicht schlechter

Indexkonstruktion

- Die Indexkonstruktion (Invertierung) besteht aus folgenden Schritten:

1. Parsen der Dokumente
2. Sortieren der Listen
3. Zusammenfassen doppelter Einträge

Mögliche Probleme:

- Hoher Speicherverbrauch
- Lange Sortierzeit

Indexkonstruktion

1) Parsen der Dokumente

- Durchlaufen der zu indizierenden Dokumente von vorne bis hinten
- Anlegen einer Liste der gefundenen Wörter mit der dazugehörigen Dokumentnr. (Stopwörter ignorieren)
- Liste noch unsortiert
- Doppelte Einträge noch nicht zusammengefasst

Indexkonstruktion

2) Sortieren der Listen

- Alphabetische Sortierung der Liste mittels Sortieralgorithmus
- Doppelte Einträge werden noch nicht entfernt

3) Zusammenfassen doppelter Einträge

- Zusammenfassung doppelt vorkommender Wörter

Indexverwaltung

- Updates von invertierten Listen meist Neu Einfügungen

  → Löschen seltener
- Kurze invertierte Listen in Speicher fester Größe
- Lange Listen in zusammenhängende Speicherbereiche variabler Größe

Indexverwaltung

- Jede invertierte Liste beginnt als kurze Liste in einem Bucket

→Beim Überlauf längste darin enthaltene Liste wird zu langer Liste

- Was machen wir mit zu langen Listen?

Optimierung

Schon kennengelernt:

- Stemming
- Stopping

- Indexkompression mittels
  - Golomb-Code
  - Rice-Code
  - Binäre Kodierung

Vorteil:

1. Weniger als 50% der Textgröße
2. Auswertungszeit von Fragen geringer
3. Erstellung des Indexes in kurzer Rechenzeit

Indexkompression

- Zeit- und Platzsparende Speicherung von Daten bzw. Integer auf einem Computer
- Zugriffszeit auf Daten möglichst kurz

- Viele Ansätze zur Indexkompression
- Golomb-Code
- Rice-Code: Spezialfall des Golomb-Codes
- Unäre, Binäre Kodierung
- ...

Golomb-Code

- Darstellungsform für alle positiven Zahlen
- Abspeicherung von Zahlen unbekannter Größe

- Speicher sparsam
- einfach umzusetzen
- schnelle Kodierung

Beispiel zum Golomb-Code

- Teil 1:
- Berechne q der Zahl n
- Quotient wird unär ausgegeben
- q besteht nur aus „1“
- Am Ende Anhängen „0“

- n = Zahl
- q = Quotienten
- b = Parameter

$$q = \left\lfloor \frac{n-1}{b} \right\rfloor$$

- Teil 2:
- wird als Binärzahl aufgeschrieben

- n = Zahl
- q = Quotienten
- b = Parameter
- r = Rest

$$r = n - (q * b) - 1$$

Weitere Ansätze zur Indexierung

Suffix Array

- Array, welches Suffixe eines Strings in lexikographischer Reihenfolge angibt

  → erlaubt binäre Suche
- Nutzbar als Index, lokalisiert jeden Substring innerhalb des Strings

- Text lange Zeichenkette
- jedes Teilstück des Textes **ist eindeutig** durch seine Position bestimmt
- Positionen im Text frei wählbar

Weitere Ansätze zur Indexierung

Signature Files

- Effiziente Herausfilterung von Einträgen einer Datenbasis, welche eine bestimmte Bedingung erfüllen
- große Textbestände, innerhalb kurzer Zeit nach Schlüsselwörtern durchsuchbar
- Signaturen werden durch die Abbildung von Wörtern auf Bitstrings mittels Hash-Funktion erzeugt
- Speicherung der Dokumente in einer Textdatei, Signaturen in einer separaten Datei → Signature File

Aufbau:

- Text wird in gleichgroße Blöcke unterteilt
- jeder Block repräsentiert ein Stück des Textes

Das war´s ☺

Danke für eure Aufmerksamkeit!