KÜNSTLICHE INTELLIGENZ

Leibniz Startup & Industry Event
20. November 2018 – Lichthof des Hauptgebäudes

✓ Wissenschaftliche Highlights
✓ Visionäre Anwender
✓ Dynamische Pitches
✓ Marktplatz der Möglichkeiten

Sonderpreis für Studierende: 10 Euro

Programm und Anmeldung:
www.artificial-intelligence.ama-academy.de
Introduction to Information Retrieval
http://informationretrieval.org

IIR 1: Boolean Retrieval

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-09
Take-away

- Boolean Retrieval: Design and data structures of a simple information retrieval system
- What topics will be covered in this class?
Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview
Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).
Unstructured (text) vs. structured (database) data in 1996
Unstructured (text) vs. structured (database) data in 2006

- Data volume
- Market Cap

Google
Yahoo!
Windows
Ask
Boolean retrieval

- The Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., Caesar and Brutus
- The search engine returns all documents that satisfy the Boolean expression.

Does Google use the Boolean model?
Does Google use the Boolean model?

- On Google, the default interpretation of a query \([w_1 \ w_2 \ \ldots \ w_n]\) is \(w_1 \ \text{AND} \ w_2 \ \text{AND} \ \ldots \ \text{AND} \ w_n\).

- Cases where you get hits that do not contain one of the \(w_i\):
  - anchor text
  - page contains variant of \(w_i\) (morphology, spelling correction, synonym)
  - long queries (\(n\) large)
  - boolean expression generates very few hits

- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns matching documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.
Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview
Unstructured data in 1650: Shakespeare
Unstructured data in 1650

- Which plays of Shakespeare contain the words **Brutus and Caesar**, but not **Calpurnia**?
- One could grep all of Shakespeare’s plays for **Brutus and Caesar**, then strip out lines containing **Calpurnia**.
- **Why is grep not the solution?**
  - Slow (for large collections)
  - grep is line-oriented, IR is document-oriented
  - “**not Calpurnia**” is non-trivial
  - Other operations (e.g., find the word **Romans** near **countryman**) not feasible
Term-document incidence matrix

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANTHONY</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>BRUTUS</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CAESAR</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>CALPURNIA</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CLEOPATRA</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>MERCY</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>WORSE</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

...  

Entry is 1 if term occurs. Example: CALPURNIA occurs in Julius Caesar. Entry is 0 if term doesn’t occur. Example: CALPURNIA doesn’t occur in The tempest.
So we have a 0/1 vector for each term.

To answer the query **Brutus and Caesar and not Calpurnia**:

- Take the vectors for **Brutus**, **Caesar**, and **Calpurnia**
- Complement the vector of **Calpurnia**
- Do a (bitwise) **AND** on the three vectors
- \[110100 \text{ AND } 110111 \text{ AND } 101111 = 100100\]
### 0/1 vectors and result of bitwise operations

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Caesar</th>
<th>Cleopatra</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANTHONY</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>BRUTUS</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CAESAR</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>CALPURNIA</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CLEOPATRA</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>MERCY</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>WORSE</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

result: 1 0 0 1 0 0 0

---

**Anthony**: 
- 1
**Julius Caesar**: 
- 0
**The Tempest**: 
- 1
**Hamlet**: 
- 0
**Othello**: 
- 1
**Macbeth**: 
- 0

---

**Anthony and Caesar**: 
- 1 0 0 0 0 1
**Calpurnia**: 
- 0 0 0 0 0 0
**Cleopatra**: 
- 1 0 0 0 0 0
**Merce**: 
- 1 0 1 1 1 1
**Worse**: 
- 1 0 1 1 1 0

---

**Result**: 1 0 0 1 0 0 0
Anthony and Cleopatra, Act III, Scene ii
Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus, When Antony found Julius Caesar dead, He cried almost to roaring; and he wept When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii
Lord Polonius: I did enact Julius Caesar: I was killed i’ the Capitol; Brutus killed me.
Bigger collections

- Consider \( N = 10^6 \) documents, each with about 1000 tokens
- \( \Rightarrow \) total of \( 10^9 \) tokens
- On average 6 bytes per token, including spaces and punctuation \( \Rightarrow \) size of document collection is about \( 6 \cdot 10^9 = 6 \text{ GB} \)
- Assume there are \( M = 500,000 \) distinct terms in the collection
- (Notice that we are making a term/token distinction.)
Can’t build the incidence matrix

- $M = 500,000 \times 10^6 = \text{half a trillion } 0\text{s and } 1\text{s.}$
- But the matrix has no more than one billion 1s.
  - Matrix is extremely sparse.
- What is a better representations?
  - We only record the 1s.
Inverted Index

For each term \( t \), we store a list of all documents that contain \( t \).

<table>
<thead>
<tr>
<th>Term</th>
<th>Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 2 4 11 31 45 173 174</td>
</tr>
<tr>
<td>Caesar</td>
<td>1 2 4 5 6 16 57 132 ...</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2 31 54 101</td>
</tr>
</tbody>
</table>

\[
\vdots
\]

dictionary postings
Inverted index construction

1 Collect the documents to be indexed:
   Friends, Romans, countrymen. So let it be with Caesar . . .

2 Tokenize the text, turning each document into a list of tokens:
   Friends Romans countrymen So . . .

3 Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms:
   friend roman countryman so . . .

4 Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.
Tokenization and preprocessing

**Doc 1.** I did enact Julius Caesar: I was killed i’ the Capitol; Brutus killed me.

**Doc 2.** So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

```
Doc 1. i did enact julius caesar i was killed i’ the capitol brutus killed me
Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious
```
Generate postings

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>i’</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>hath</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
</tbody>
</table>

**Doc 1.** i did enact julius caesar i was killed i’ the capitol brutus killed me
**Doc 2.** so let it be with caesar the noble brutus hath told you caesar was ambitious
Sort postings

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>1</td>
<td>ambitious</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
<td>hath</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
<td>i'</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>2</td>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ambitious</td>
<td>2</td>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>
Create postings lists, determine document frequency

<table>
<thead>
<tr>
<th>term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>term</th>
<th>doc. freq.</th>
<th>postings lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
<td>1 → 2</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
<td>1 → 2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
<td>1 → 2</td>
</tr>
<tr>
<td>i</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>1</td>
<td>1 → 2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>1</td>
<td>1 → 2</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>1</td>
<td>1 → 2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
<td>1 → 2</td>
</tr>
<tr>
<td>told</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>you</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
<td>1 → 2</td>
</tr>
<tr>
<td>with</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Split the result into dictionary and postings file

Brutus → 1 2 4 11 31 45 173 174
Caesar → 1 2 4 5 6 16 57 132 ...
Calpurnia → 2 31 54 101
...

**dictionary**

**postings file**
Later in this course

- Index construction: how can we create inverted indexes for large collections?
- How much space do we need for dictionary and index?
- Index compression: how can we efficiently store and process indexes for large collections?
- Ranked retrieval: what does the inverted index look like when we want the “best” answer?
Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview
Simple conjunctive query (two terms)

- Consider the query: **Brutus AND Calpurnia**
- To find all matching documents using inverted index:
  1. Locate **Brutus** in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate **Calpurnia** in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists
  6. Return intersection to user
Intersecting two postings lists

\[
\text{Brutus} \quad \rightarrow \quad 1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174
\]

\[
\text{Calpurnia} \quad \rightarrow \quad 2 \rightarrow 31 \rightarrow 54 \rightarrow 101
\]

Intersection \quad \Rightarrow \quad 2 \rightarrow 31

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.
Intersecting two postings lists

**INTERSECT**($p_1, p_2$)
1. $answer \leftarrow \langle \rangle$
2. while $p_1 \neq \text{NIL}$ and $p_2 \neq \text{NIL}$
3. do if $\text{docID}(p_1) = \text{docID}(p_2)$
4. then \text{ADD}($answer, \text{docID}(p_1)$)
5. \hspace{1em} $p_1 \leftarrow \text{next}(p_1)$
6. \hspace{1em} $p_2 \leftarrow \text{next}(p_2)$
7. else if $\text{docID}(p_1) < \text{docID}(p_2)$
8. then $p_1 \leftarrow \text{next}(p_1)$
9. else $p_2 \leftarrow \text{next}(p_2)$
10. return $answer$
Query processing: Exercise

Compute hit list for ((paris AND NOT france) OR lear)
The Boolean retrieval model can answer any query that is a Boolean expression.

- Boolean queries are queries that use **AND**, **OR** and **NOT** to join query terms.
- Views each document as a **set** of terms.
- Is precise: Document matches condition or not.

Primary commercial retrieval tool for 3 decades

Many professional searchers (e.g., lawyers) still like Boolean queries.

- You know exactly what you are getting.

Many search systems you use are also Boolean: spotlight, email, intranet etc.
Commercially successful Boolean retrieval: Westlaw

- Largest commercial legal search service in terms of the number of paying subscribers
- Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
- The service was started in 1975.
- In 2005, Boolean search (called “Terms and Connectors” by Westlaw) was still the default, and used by a large percentage of users . . .
- . . . although ranked retrieval has been available since 1992.
Information need: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company Query: “trade secret” /s disclos! /s prevent /s employe!

Information need: Requirements for disabled people to be able to access a workplace Query: disab! /p access! /s work-site work-place (employment /3 place)

Information need: Cases about a host’s responsibility for drunk guests Query: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest
Westlaw: Comments

- Proximity operators: `/3 = within 3 words, /s = within a sentence, /p = within a paragraph`
- Space is disjunction, not conjunction! (This was the default in search pre-Google.)
- Long, precise queries: incrementally developed, not like web search
- Why professional searchers often like Boolean search: precision, transparency, control
- When are Boolean queries the best way of searching? Depends on: information need, searcher, document collection, ...
Outline

1 Introduction
2 Inverted index
3 Processing Boolean queries
4 Query optimization
5 Course overview
Query optimization

- Consider a query that is an AND of $n$ terms, $n > 2$
- For each of the terms, get its postings list, then AND them together
- Example query: Brutus AND Calpurnia AND Caesar
- What is the best order for processing this query?
Example query: **Brutus AND Calpurnia AND Caesar**

Simple and effective optimization: Process in order of increasing frequency

Start with the shortest postings list, then keep cutting further

In this example, first Caesar, then Calpurnia, then Brutus
Optimized intersection algorithm for conjunctive queries

\textsc{Intersect}(\langle t_1, \ldots, t_n \rangle)

1. \textit{terms} $\leftarrow$ \textsc{SortByIncreasingFrequency}(\langle t_1, \ldots, t_n \rangle)
2. \textit{result} $\leftarrow$ postings(first(\textit{terms}))
3. \textit{terms} $\leftarrow$ rest(\textit{terms})
4. \textbf{while} \textit{terms} $\neq$ \textit{NIL} and \textit{result} $\neq$ \textit{NIL}
5. \textbf{do} \textit{result} $\leftarrow$ \textsc{Intersect}(\textit{result}, postings(first(\textit{terms})))
6. \quad \textit{terms} $\leftarrow$ rest(\textit{terms})
7. \textbf{return} \textit{result}
More general optimization

- Example query: \((\text{MADDING OR CROWD}) \text{ AND } (\text{IGNOBLE OR STRIFE})\)
- Get frequencies for all terms
- Estimate the size of each \text{OR} by the sum of its frequencies (conservative)
- Process in increasing order of \text{OR} sizes
Outline

1. Introduction
2. Inverted index
3. Processing Boolean queries
4. Query optimization
5. Course overview
Course overview

- We are done with Chapter 1 of IIR (IIR 01).
- Plan for the rest of the semester: 18–20 of the 21 chapters of IIR
- In what follows: teasers for most chapters – to give you a sense of what will be covered.
Phrase queries: “Stanford University”

Proximity queries: Gates near Microsoft

We need an index that captures position information for phrase queries and proximity queries.
IIR 03: Dictionaries and tolerant retrieval

- **BO**
  - aboard
  - about
  - boardroom
  - border

- **OR**
  - border
  - lord
  - morbid
  - sordid

- **RD**
  - aboard
  - ardent
  - boardroom
  - border
IIR 04: Index construction

splits

map phase

parser

assign

master

assign

inverter

a-f
g-p
q-z

parser

parser

parser

segment files

parser

inverter

a-f
g-p
q-z

parser

reduce phase

inverter

postings
IIR 05: Index compression

Zipf’s law
IIR 06: Scoring, term weighting and the vector space model

- Ranking search results
  - Boolean queries only give inclusion or exclusion of documents.
  - For ranked retrieval, we measure the proximity between the query and each document.
  - One formalism for doing this: the vector space model

- Key challenge in ranked retrieval: evidence accumulation for a term in a document
  - 1 vs. 0 occurrence of a query term in the document
  - 3 vs. 2 occurrences of a query term in the document
  - Usually: more is better
  - But by how much?
  - Need a scoring function that translates frequency into score or weight
IIR 07: Scoring in a complete search system

Documents

Parsing
Linguistics

Indexers

user query

Free text query parser

Spell correction

Scoring and ranking

Metadata in
zone and
field indexes

Inexact
top K
retrieval

Tiered inverted
positional index

k-gram

Indexes

Scoring parameters

MLR

training set

Results
page

Indexers

Document cache

Documents
IIR 08: Evaluation and dynamic summaries

**Manitoba - Wikipedia, the free encyclopedia**

Manitoba's capital and largest city, Winnipeg, .... According to Environment Canada, Manitoba ranked first for clearest skies year round, and ranked second ...

**Geography - History - Demographics - Economy**
en.wikipedia.org/wiki/Manitoba - Cached - Similar

**List of cities in Canada - Wikipedia, the free encyclopedia**

Cities and towns in Manitoba. See also: List of communities in Manitoba .... Dartmouth - formerly the second largest city in Nova Scotia, now a Metropolitan ...
en.wikipedia.org/wiki/List_of_cities_in_Canada - Cached - Similar

**Canadian Immigration Information - Manitoba**

The largest city in the province is the capital, Winnipeg, with a population exceeding 706900. The second largest city is Brandon. Manitoba has received ...
www.canadavisa.com/about-manitoba.html - Cached - Similar

**CBC Manitoba | EAL**

Lesson 57: Brandon - Manitoba's Second Largest City. For Teachers; For Students. Step One Open the Lesson: PDF (194kb) PDF WORD (238kb) Microsoft Word ...
www.cbc.ca/manitoba/.../lesson-57-brandon---manitobas-second-largest.html - Cached
IIR 09: Relevance feedback & query expansion
This is a one-state probabilistic finite-state automaton — a unigram language model — and the state emission distribution for its one state $q_1$. STOP is not a word, but a special symbol indicating that the automaton stops. frog said that toad likes frog STOP

$$P(\text{string}) = 0.01 \cdot 0.03 \cdot 0.04 \cdot 0.01 \cdot 0.02 \cdot 0.01 \cdot 0.2$$

$$= 0.0000000000048$$
Text classification = assigning documents automatically to predefined classes

Examples:
- Language (English vs. French)
- Adult content
- Region
IIR 14: Vector classification
IIR 15: Support vector machines

Support vectors

Maximize margin
IIR 16: Flat clustering

1. **Jag-lovers - THE source for all Jaguar information**
   ... Internet! Serving Enthusiasts since 1993 The Jag-lovers Web Currently with 40661 members The Premier Jaguar Cars web resource for all enthusiasts Lists and Forums Jag-lovers originally evolved around its ...
   www.jag-lovers.org - Open Directory 2, Wisenut 8, Ask Jeeves 8, MSN 9, Looksmart 12, MSN Search 18

2. **Jaguar Cars**
   [... ] redirected to www.jaguar.com
   www.jaguarcars.com - Looksmart 1, MSN 2, Lycos 3, Wisenut 6, MSN Search 9, MSN 29

3. **http://www.jaguar.com/**
   www.jaguar.com - MSN 1, Ask Jeeves 1, MSN Search 3, Lycos 9

4. **Apple - Mac OS X**
   Learn about the new OS X Server, designed for the Internet, digital media and workgroup management. Download a technical factsheet.
   www.apple.com/macosx - Wisenut 1, MSN 3, Looksmart 26
IIR 17: Hierarchical clustering

http://news.google.com
Latent Semantic Indexing

\[ \text{Words: } X = W \]

\[ \text{Contexts: } m \times m \]

\[ \text{w X c} \quad \text{w X m} \]
IIR 19: The web and its challenges

- Unusual and diverse documents
- Unusual and diverse users and information needs
- Beyond terms and text: exploit link analysis, user data
- How do web search engines work?
- How can we make them better?
IIR 21: Link analysis / PageRank
Take-away

- Boolean Retrieval: Design and data structures of a simple information retrieval system
- What topics will be covered in this class?
Resources

- Chapter 1 of IIR
- http://cislmu.org
  - course schedule
  - information retrieval links
  - Shakespeare search engine