Querying

Introduction to Information Retrieval
Informatics 141 / CS 121
Donald J. Patterson

Content adapted from Hinrich Schütze
http://www.informationretrieval.org
Overview

- Boolean Retrieval
- Weighted Boolean Retrieval
- Zone Indices
- Term Frequency Metrics
- The full vector space model
Querying

From the bottom

- “Grep”
  - Querying without an index or a crawl
  - Whenever you want to find something you look through the entire document for it.
- Example:
  - You have the collected works of Shakespeare on disk
  - You want to know which play contains the words “Brutus AND Caesar”
“Grep”

“Brutus AND Caesar” is the query.

This is a boolean query. Why?

What other operators could be used?

The grep solution:

- Read all the files and all the text and output the intersection of the files
• “Grep”
  • Slow for large corpora
  • Calculating “NOT” is non-trivial
  • Some operations not feasible
    • Query: “Romans NEAR Countrymen”
  • Doesn’t support ranked retrieval
• Moving beyond grep is the motivation for the inverted index.
“Grep”
- Slow for large corpora
- Calculating “NOT” is non-trivial
- Some operations not feasible
  - Query: “Romans NEAR Countrymen”
- Doesn’t support ranked retrieval
- Moving beyond grep is the motivation for the inverted index.
Querying

Our **inverted index** is a 2-D array or Matrix

A Column For Each Document

<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>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• **Boolean Query**
  • Queries are boolean expressions
  • Search returns all documents which satisfy the expression
  • Does Google use the Boolean model?
• **Boolean Query**
  - Straightforward application of inverted index
  - where cells of inverted index are (0,1)
  - indicating presence or absence of a term
**Boolean Query**
- 0/1 vector for each term
- "Brutus AND Caesar AND NOT Calpurnia =
- Perform bitwise Boolean operation on each row:
- \[110100 \text{ AND } 110111 \text{ AND } \overline{010000} = 100100\]

<table>
<thead>
<tr>
<th>Term</th>
<th>Anthony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Querying

- **Boolean Query**
  - A big corpus means a sparse matrix
  - A sparse matrix motivates the introduction of the posting
  - Much less space to store
  - Only recording the “1” positions
• Boolean Query
  • Query processing on postings
  • Brutus AND Caesar
    • Locate the postings for Brutus
    • Locate the postings for Caesar
    • Merge the postings

Brutus: 2 4 8 16 32 64
Caesar: 1 2 3 5 8 13
Querying

- **Boolean Query**
- Merging -> walk through the two posting simultaneously
- postings sorted by doc ID

Brutus: 2 4 8 16 32 64
Caesar: 1 2 3 5 8 13

2 8
Querying

• **Boolean Query**
  • An algorithm based on postings
  • Linear in the size of the postings

\[
\text{INTERSECT}(p_1, p_2) \\
\text{1} \quad \text{answer} \leftarrow \langle \rangle \\
\text{2} \quad \text{while } p_1 \neq \text{nil} \text{ and } p_2 \neq \text{nil} \\
\text{3} \quad \text{do if } \text{docID}(p_1) = \text{docID}(p_2) \\
\text{4} \quad \text{then } \text{ADD}(\text{answer}, \text{docID}(p_1)) \\
\text{5} \quad \quad \quad \ p_1 \leftarrow \text{next}(p_1) \\
\text{6} \quad \quad \quad \ p_2 \leftarrow \text{next}(p_2) \\
\text{7} \quad \text{else if } \text{docID}(p_1) < \text{docID}(p_2) \\
\text{8} \quad \text{then } p_1 \leftarrow \text{next}(p_1) \\
\text{9} \quad \quad \quad \ \text{else } p_2 \leftarrow \text{next}(p_2) \\
\text{10} \quad \text{return } \text{answer}
\]
**Boolean Query**

- Is the algorithmic complexity better than scanning?
- Where would you put more complex formulae?

```plaintext
\text{INTERSECT}(p_1, p_2)
1\quad \text{answer} \leftarrow <>
2\quad \text{while } p_1 \neq \text{nil} \text{ and } p_2 \neq \text{nil}
3\quad \text{do if } \text{docID}(p_1) = \text{docID}(p_2)
4\quad \quad \text{then } \text{ADD}(\text{answer}, \text{docID}(p_1))
5\quad \quad p_1 \leftarrow \text{next}(p_1)
6\quad \quad p_2 \leftarrow \text{next}(p_2)
7\quad \text{else if } \text{docID}(p_1) < \text{docID}(p_2)
8\quad \quad \text{then } p_1 \leftarrow \text{next}(p_1)
9\quad \quad \text{else } p_2 \leftarrow \text{next}(p_2)
10\quad \text{return } \text{answer}
```
• **Boolean Queries**
  - Exact match
  - Views each document as a “bag of words”
  - Precise: a document matches or it doesn’t
  - Primary commercial retrieval tool for 3 decades
  - Professional searchers (e.g., lawyers) still like Boolean queries
  - No question about what you are getting
Not quite End of Chapter 1