CS315
Introduction to Information Retrieval

Boolean Search

Intro to IR, Boolean Search

- Basic terminology
- The need for unstructured text search
- Boolean Retrieval Model
- Algorithms for compressing data
- Algorithms for answering Boolean queries
- Read Chapter 1 of MRS

Information Retrieval (IR)

- 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

1996
Unstructured (text) vs. structured (database) data

Google

Unstructured Data in 1680

- You have all the Shakespeare plays stored in files
- Which plays contain the word Caesar?
  - Your algorithm here:
- Which plays of Shakespeare contain the words Brutus AND Caesar?

Processing unstructured data

- Which plays of Shakespeare contain the words Brutus AND Caesar but NOT Calpurnia?
  - One could grep all of Shakespeare’s plays
  - then strip out lines containing Calpurnia, but...
- It is slow! (for large corpora)
  - Computing NOT Calpurnia is non-trivial
- Other operations not feasible
  - (e.g., find Romans near countrymen)
- Ranked retrieval (find “best” documents to return) also not possible

“Term-document incidence” matrix

<table>
<thead>
<tr>
<th>Terms</th>
<th>Antony 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>Antony</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>1</td>
<td>1</td>
<td>1</td>
<td>0</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>worse</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 if file contains word, 0 otherwise
### Incidence vectors

<table>
<thead>
<tr>
<th>m terms</th>
<th>N documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony and Cleopatra</td>
<td>Antony</td>
</tr>
<tr>
<td>Antony</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
</tr>
<tr>
<td>war</td>
<td>1</td>
</tr>
</tbody>
</table>

- So we have a 0/1 binary vector for each term
- How do you answer the query?
- Take the vectors for *Brutus*, *Caesar* and (complement) *Calpurnia* \( \triangleright \) bitwise AND.

\[
110100 \text{ AND } 110111 \text{ AND } 101111 \Rightarrow 100100.
\]

*Brutus AND Caesar but NOT Calpurnia*

### Answers to query

#### Antony 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.

### How good are the retrieved docs?

- **Precision**: Fraction of retrieved docs that are relevant to user’s information need
- **Recall**: Fraction of relevant docs in collection that are retrieved
- More precise definitions and measurements to follow in later lectures
- For now, let’s focus on the techniques of retrieval

### Space requirements

- Say, \( N = 1 \text{M documents, each with about 1K terms.} \)
  How many terms in all documents (corpus)?
- Average 6 bytes/term including spaces/punctuation. How many GB of data in corpus?
Does the matrix fits in memory?

- Say there are $m = 500K$ distinct terms among these.
- How many 0’s and 1’s in the Term-doc incidence matrix?
  - 500K x 1M matrix has 500 Billion 0’s and 1’s.
  - But matrix is extremely sparse
    - it has no more than 1 Billion 1’s.
- What if we only record the 1 positions.
  What is a better representation than a matrix?

"Inverted" Index

- For each term $T$, we must store a listing of all document id’s that contain $T$.
- Do we use an array or a linked list for this?

Brutus

Calpurnia

Caesar

What happens if the word Caesar is added to document 14?

Inverted index

- Linked lists generally preferred to arrays
  - +1
  - -1
  - a

Brutus

Calpurnia

Caesar

Dictionary

Posting lists

Sorted by docID

Inverted index construction

Documents to be indexed. Tokens, Romans, countrymen.

Tokenizer

Linguistic modules

Modified tokens.

Inverted index.

friend

roman

countryman

Documents to be indexed. Friends, Romans, countrymen.

Tokenizer

Linguistic modules

More on these later.
Indexer step 1: Token sequence

- **INPUT**: Sequence of pairs:
  - (Modified token, Document ID)

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

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

Indexer step 2: Sort

- **Sort by terms**.
- Then by docID

Core indexing step.

Indexer step 3: Dictionary & Postings

- **Merge** multiple term entries in a single document
  - Note two entries for caesar!
  - We merge PER DOCUMENT.

- **Add frequency information**.

Why frequency? Will discuss later.

We just built an index...

- **How do we process a query?**
- **For later**: what kinds of queries can we process?

Today’s focus
Query processing w/ inverted index: AND

Consider processing the query:

\textit{Brutus AND Caesar}

- Locate \textit{Brutus} in the Dictionary; Retrieve its postings.
- Locate \textit{Caesar} in the Dictionary; Retrieve its postings.
- Intersect the two postings:

\[ \begin{array}{cccccccc}
2 & 4 & 8 & 16 & 32 & 64 & 128 \\
1 & 2 & 3 & 5 & 8 & 13 & 21 & 34 \\
\end{array} \]

Walk through the two postings simultaneously.

Time complexity?
- \( \_ \_ \_ \_ \_ \_ \_ \) in the total number of postings entries

If the list lengths are \( x \) and \( y \), the merge takes \( O(x+y) \) operations.

**Crucial:** postings sorted by docID.

The Intersection ("merging")

Boolean queries: Exact match

- The Boolean retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries use \textit{AND}, \textit{OR} and \textit{NOT} to join query terms
  - Views each document as a set of words
  - Is precise: document matches condition or not.
  - Perhaps the simplest model to build an IR system on
  - Primary commercial retrieval tool for 3 decades.
  - Many search systems you still use are Boolean:
    - Email, library catalog, Mac OS X Spotlight

\begin{verbatim}
INTERSECT(p1, p2)
1  answer <- \langle \rangle
2  while \( p1 \neq \text{NIL} \) and \( p2 \neq \text{NIL} \)
3    do if \( \text{docID}(p1) = \text{docID}(p2) \)
4      then ADD(answer, \text{docID}(p1))
5        p1 <- next(p1)
6        p2 <- next(p2)
7      else if \( \text{docID}(p1) < \text{docID}(p2) \)
8        then p1 <- next(p1)
9      else p2 <- next(p2)
10  return answer
\end{verbatim}
Query Optimization

What's the best order for processing pairs of postings?
- (Brutus AND Calpurnia) AND Caesar
- (Calpurnia AND Caesar) AND Brutus
- (Brutus AND Caesar) AND Calpurnia

<table>
<thead>
<tr>
<th>Brutus</th>
<th>Calpurnia</th>
<th>Caesar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Query: Brutus AND Calpurnia AND Caesar

Process in order of increasing frequency:
- start with smallest set, then keep cutting further.

This is why we kept freq in dictionary

Execute the query as (Caesar AND Brutus) AND Calpurnia.

More General Optimization

Query:
- (madding OR crowd) AND (ignoble OR strife)

Get freq’s for all terms.
- Estimate the size of each OR by the sum of its freq’s (conservative).
- Process in increasing order of OR sizes.