Lecture Objectives

- Learn about and implement
- Boolean search
- Inverted index
- Positional index
**Indexing Process**

- **Documents acquisition**
- **Text transformation**
- **Index creation**
- **Index**

**Pre-processing output**

- **Example sentence**
- **Pre-processing is applied to text in information retrieval.**
- **Text inform retrieval includ Token Stop Word Remov Stem**

- **Add processed terms to index**
- **What is “index”?**
Indexing

- Search engines vs PDF find or grep?
  - Infeasible to scan large collection of text for every “search”
  - Find section that has: “UK and Scotland and Money”?!?
- Book Index
  - For each word, list of “relevant” pages
  - Find topic in sub-linear time
- IR Index:
  - Data structure for fast finding terms
  - Additional optimisations could be applied
**Document Vectors**

- Represent documents as vectors
  - Vector $\rightarrow$ document, cell $\rightarrow$ term
  - Values: term frequency or binary (0/1)
  - All documents $\rightarrow$ collection matrix

<table>
<thead>
<tr>
<th></th>
<th>he</th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>think</th>
<th>wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

D1: He likes to wink, he likes to drink
D2: He likes to drink, and drink, and drink
D3: The thing he likes to drink is ink
D4: The ink he likes to drink is pink
D5: He likes to wink, and drink pink ink

**Inverted Index**

- Represent terms as vectors
  - Vector $\rightarrow$ term, cell $\rightarrow$ document
  - Transpose of the collection matrix
  - Vector: inverted list

<table>
<thead>
<tr>
<th></th>
<th>he</th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>think</th>
<th>wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

D1: He likes to wink, he likes to drink
D2: He likes to drink, and drink, and drink
D3: The thing he likes to drink is ink
D4: The ink he likes to drink is pink
D5: He likes to wink, and drink pink ink
**Boolean Search**

- Boolean: exist / not-exist
- Multiword search: logical operators (AND, OR, NOT)
- Example
  - Collection: search Shakespeare's Collected Works
  - Boolean query: Brutus AND Caesar AND NOT Calpurnia

- Build a **Term-Document Incidence Matrix**
  - Which term appears in which document
  - Rows are terms
  - Columns are documents

---

**Collection 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>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>

Terms: 1 if document contains term, 0 otherwise

Query: Brutus AND Caesar AND NOT Calpurnia
Apply on rows: 110100 AND 110111 AND !(010000) = 100100
Bigger collections?

- Consider $N = 1$ million documents, each with about 1000 words.
- $n = 1M \times 1K = 1B$ words
  $\Rightarrow$ Heap’s law $\Rightarrow v \approx 500K$
- Matrix size = 500K unique terms x 1M documents = 0.5 trillion 0’s and 1’s entries!
- If all words appear in many documents $\Rightarrow \max\{\text{count(1’s)}\} = N \times \text{doc. length} = 1B$
- Actually, from Zip’s law $\Rightarrow$ 250k terms appears once!
- Collection matrix is extremely \textbf{sparse}. (mostly 0’s)

Inverted Index: Sparse representation

- For each term $t$, we must store a list of all documents that contain $t$.
  - Identify each by a \textbf{docID}, a document serial number

Dictionary

\textbf{Brutus}

\textbf{Caesar}

\textbf{Calpurnia}

Postings List

\begin{tabular}{c}
\begin{tabular}{c}
\textbf{Doc number (sorted)}
\end{tabular}
\end{tabular}

\begin{tabular}{c}
\begin{tabular}{c}
\textbf{Postings}
\end{tabular}
\end{tabular}
Inverted Index Construction

Documents to be indexed

Tokenizer

Token stream

Normaliser

Terms (modified tokens)

Indexer

Inverted index

Step 1: Term Sequence

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
Step 2: Sorting

- Sort by:
  1) Term
  2) Doc ID

Step 3: Posting

1. Multiple term entries in a single document are merged
2. Split into Dictionary and Postings
3. Doc. Frequency (df) information is added
### Inverted Index: matrix → postings

<table>
<thead>
<tr>
<th></th>
<th>he</th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>think</th>
<th>wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **D1**: He likes to wink, he likes to drink
- **D2**: He likes to drink, and drink, and drink
- **D3**: The thing he likes to drink is ink
- **D4**: The ink he likes to drink is pink
- **D5**: He likes to wink, and drink pink ink

### Inverted Index: with frequency

- **Boolean**: term → DocIDs list
- **Frequency**: term → touples (DocID,count(term)) lists

**he**: 1:2, 2:1, 3:1, 4:1, 5:1

**drink**: 1:1, 2:3, 3:1, 4:1, 5:1

**ink**: 3:1, 4:1, 5:1

**pink**: 4:1, 5:1

**thing**: 3:1

**wink**: 1:1, 5:1

*appeared in D2 3 times*
Query Processing

- Find documents matching query \{ink AND wink\}
  1. Load inverted lists for each query word
  2. Merge two postings lists → Linear merge

- Linear merge → $O(n)$
  \( n \): total number of posts for all query words

<table>
<thead>
<tr>
<th></th>
<th>3:1</th>
<th>4:1</th>
<th>5:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ink</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wink</td>
<td>1:1</td>
<td>5:1</td>
<td></td>
</tr>
</tbody>
</table>

Matches

1: \( f(0,1) \)
3: \( f(1,0) \)
4: \( f(1,0) \)
5: \( f(1,1) \)

Phrase Search

- Find documents matching query “pink ink”
  1. Find document containing both words
  2. Both words has to be a phrase

- Bi-gram Index:

He likes to wink, and drink pink ink → Convert to bigrams

He_likes likes_to to_wink wink_and and_drink drink_pink pink_ink

- Bi-gram Index, issues:
  - Fast, but index size will explode!
  - What about trigram phrases?
  - What about proximity? “ink is pink”
**Proximity Index**

- Terms positions is embedded to the inv. Index
  - Called proximity/positional index
  - Enables phrase and proximity search
  - Toubles (DocID, term position)

<table>
<thead>
<tr>
<th>Term</th>
<th>Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>he</td>
<td>1:2 2:1 3:1 4:1 5:1</td>
</tr>
<tr>
<td>drink</td>
<td>1:1 2:3 3:1 4:1 5:1</td>
</tr>
</tbody>
</table>

- **D1**: He likes to wink, he likes to drink
- **D2**: He likes to drink, and drink, and drink
- **D3**: The thing he likes to drink is ink
- **D4**: The ink he likes to drink is pink
- **D5**: He likes to wink, and drink pink ink

**Query Processing: Proximity**

- Find documents matching query “pink ink”
  1. Use Linear merge
  2. Additional step: check terms positions

- **Proximity search**: 
  \[
  \text{pos(term1)} - \text{pos(term2)} < |w| \rightarrow \#5(\text{pink, ink})
  \]

**Matches**

3: \( f(1,0) = 0 \)

4: \( f(1,1) = ? = \text{pos(ink)} - \text{pos(pink)} \rightarrow 1? \)

5: \( f(1,1) = ? = \text{pos(ink)} - \text{pos(pink)} \rightarrow 1? \)
**Proximity search: data structure**

- Possible data structure:
  - `<term: df;`
  - `DocNo: pos1, pos2, pos3`
  - `DocNo: pos1, pos2, pos3`
  - `...... >`

- Example:
  - `<be: 993427;`
  - `1: 7, 18, 33, 72, 86, 231;`
  - `2: 3, 149;`
  - `4: 17, 191, 291, 430, 434;`
  - `5: 363, 367, ...>`

---

**Practical**
Summary

• Document Vector
• Term Vector
• Inverted Index
• Collection Matrix
• Posting
• Proximity Index
• Query Processing → Linear merge

Resources

• Text book 1: Intro to IR, Chapter 1 & 2.4
• Text book 2: IR in Practice, Chapter 5
• Lab 2