Web Searching & Indexing

CompSci 316
Introduction to Database Systems

Announcements (Thu. Dec. 6)

- Homework #4 sample solution will be emailed by this weekend
- Course evaluation is online and due by tomorrow
- Project demo of Mobile Pay today from Michael, Kevin, Derek
- Final exam 2-5pm Dec. 12
  - Open book, open notes; focus on the second half
  - Extended office hours
    - Tue.: 12pm-2pm, 3-4pm; Wed.: 10-11:30am, 1-2pm

Outline

- Keyword search
- Ranking research results
  - A single search may return many pages
  - A user will not look at all result pages
  - Complete result may be unnecessary
Keywords × documents

<table>
<thead>
<tr>
<th>All keywords</th>
<th>Document 1</th>
<th>Document 2</th>
<th>Document 3</th>
<th>Document 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;cat&quot;</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&quot;database&quot;</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>&quot;dog&quot;</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>&quot;search&quot;</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 means keyword appears in the document; 0 means otherwise

- Inverted lists: store the matrix by rows
- Signature files: store the matrix by columns

Inverted lists

- Store the matrix by rows
- For each keyword, store an inverted list
  - `(keyword, doc-id-list)`
  - ("database", {3, 7, 142, 857, …})
  - ("search", {3, 9, 192, 512, …})
  - It helps to sort `doc-id-list` (why?)
- Vocabulary index on keywords
  - B⁺-tree, hash-based, or trie (later)
- How large is an inverted list index?

Using inverted lists

- Documents containing "database"
  - Use the vocabulary index to find the inverted list for "database"
  - Return documents in the inverted list
- Documents containing "database" AND "search"
- OR? NOT?
What are “all” the keywords?

- All sequences of letters (up to a given length)?
  - … that actually appear in documents!
- All words in English?
- Plus all phrases?
  - Alternative: approximate phrase search by proximity
- Minus all stop words
  - They appear in nearly every document, e.g., a, of, the, it
  - Not useful in search
- Combine words with common stems
  - Example: database, databases
  - They can be treated as the same for the purpose of search

Frequency and proximity

- Frequency
  - \( \{ \text{keyword}, \{ \text{doc-id}, \text{number-of-occurrences} \}, \{ \text{doc-id}, \text{number-of-occurrences} \}, \ldots \} \)  
- Proximity (and frequency)
  - \( \{ \text{keyword}, \{ \text{doc-id}, \{ \text{position-of-occurrence}_1, \text{position-of-occurrence}_2, \ldots \} \}, \{ \text{doc-id}, \{ \text{position-of-occurrence}_1, \ldots \} \}, \ldots \} \)  
  - When doing AND, check for positions that are near

Signature files

- Store the matrix by columns and compress them
- For each document, store a \( w \)-bit signature
- Each word is hashed into a \( w \)-bit value, with only \( s < w \) bits turned on
- Signature is computed by taking the bit-wise OR of the hash values of all words on the document

<table>
<thead>
<tr>
<th>hash(database) = 0110</th>
<th>doc_1 contains “database” 0110 &quot;database”? hash(dog) = 1100</th>
<th>doc_2 contains “dog” 1100 hash(cat) = 0010</th>
<th>doc_3 contains “cat” and “dog” 1110</th>
</tr>
</thead>
</table>

- Some false positives; no false negatives
Bit-sliced signature files

Motivation
- To check if a document contains a word, we only need to check the bits that are set in the word’s hash value.
- So why bother retrieving all \( w \) bits of the signature?
- Instead of storing \( n \) signature files, store \( w \) bit slices.
- Only check the slices that correspond to the set bits in the word’s hash value.
- Start from the sparse slices.

<table>
<thead>
<tr>
<th>Doc</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 0 1 0 0 1 0 0</td>
</tr>
<tr>
<td>2</td>
<td>0 0 0 1 0 0 1 0 0</td>
</tr>
<tr>
<td>3</td>
<td>0 0 0 1 1 0 1 0 0</td>
</tr>
<tr>
<td>4</td>
<td>0 1 1 0 1 1 0 0 0</td>
</tr>
<tr>
<td>…</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0 0 0 1 0 1 0 0 0</td>
</tr>
</tbody>
</table>

Starting to look like an inverted list again!

Inverted lists versus signatures

Inverted lists better for most purposes (TODS, 1998)

Problems of signature files
- False positives
- Hard to use because \( t, w, \) and the hash function need tuning to work well
- Long documents will likely have mostly 1’s in signatures
- Common words will create mostly 1’s for their slices
- Difficult to extend with features such as frequency, proximity

Saving grace of signature files

Ranking result pages

Possible ranking criteria

Based on content
- Number of occurrences of the search terms
- Similarity to the query text

Based on link structure
- Backlink count
- PageRank

And more…
Textual similarity

- Terms \{t_1, ..., t_n\} and documents \(D = \{d_1, d_2, ...\}\)
- IDF (Inverse Document Frequency) of \(t_i\):
  - \(\text{idf}_i = -\log(\# \text{ of docs in } D \text{ containing } t_i)/|D|\)
- TF (Term Frequency) of \(t_i\) in \(d_j\):
  - \(\text{tf}_{ij} = \# \text{ of times } t_i \text{ appears in } d_j\)
- TF-IDF weight vector of \(d_j\):
  - \(\mathbf{w}_j = (\text{tf}_{j1}\text{idf}_1, ..., \text{tf}_{jn}\text{idf}_n)\)
- Textual similarity between two docs \(d_j\) and \(d_k\) can be measured by the normalized dot product of these vectors, i.e.:
  \[
  \frac{\mathbf{w}_j \cdot \mathbf{w}_k}{||\mathbf{w}_j|| \cdot ||\mathbf{w}_k||} = \frac{\sum_j \sum_k (\text{tf}_{j1}\text{idf}_1 \cdot \text{tf}_{k1}\text{idf}_1 + ... + \text{tf}_{jn}\text{idf}_n \cdot \text{tf}_{kn}\text{idf}_n)}{\left(\sum_j \sum_k \text{tf}_{j1}\text{idf}_1^2 + ... + \text{tf}_{jn}\text{idf}_n^2\right)^{1/2} \cdot \left(\sum_k \sum_j \text{tf}_{k1}\text{idf}_1^2 + ... + \text{tf}_{kn}\text{idf}_n^2\right)^{1/2}}
  \]
- One “doc” could be the query text

Why weigh significance by IDF?

- Without IDF weighting, the similarity measure would be dominated by the stop words
- “the” occurs frequently on the Web, so its occurrence on a particular page should be considered less significant
- “engine” occurs infrequently on the Web, so its occurrence on a particular page should be considered more significant

Problems with content-based ranking

- Many pages containing search terms may be of poor quality or irrelevant
  - Example: a page with just a line “search engine”
- Many high-quality or relevant pages do not even contain the search terms
  - Example: Google homepage
- Page containing more occurrences of the search terms are ranked higher; spamming is easy
  - Example: a page with line “search engine” repeated many times
Backlink

- A page with more backlinks is ranked higher
- Intuition: Each backlink is a "vote" for the page’s importance

Google’s PageRank

- Main idea: Pages pointed by high-ranking pages are ranked higher
  - Definition is recursive by design
  - Based on global link structure; hard to spam
- Naïve PageRank
  - $F(p)$: set of pages that page $p$ points to
  - $B(p)$: set of pages that point to $p$
  - $\text{PageRank}(p) = \sum_{q \in B(p)} \text{PageRank}(q)/|F(q)|$
  - Each page gets a boost from every page pointing to it
  - Each page distributes its importance evenly to pages it points to

Calculating naïve PageRank

- Initially, set all PageRank’s to 1; then evaluate
  $\text{PageRank}(p) \leftarrow \sum_{q \in B(p)} \text{PageRank}(q)/|F(q)|$
  repeatedly until the values converge (i.e. a fixed point is reached)

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
0.5 & 0 & 0.5 \\
0 & 0 & 0.5 \\
0.5 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
\]

Yahoo
Amazon
Microsoft

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1 \\
0.5 \\
1.5
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.75 \\
1.375
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]

\[
\begin{bmatrix}
Y \\
M \\
A
\end{bmatrix}
= \begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\begin{bmatrix}
1.25 \\
0.6875 \\
1.0625
\end{bmatrix}
\]
Random surfer model

- A random surfer
  - Starts with a random page
  - Randomly selects a link on the page to visit next
  - Never uses the “back” button

- PageRank($p$) measures the probability that a random surfer visits page $p$

Problems with the naïve PageRank

- Dead end: a page with no outgoing links
  - A dead end causes all importance to “leak” eventually out of the Web
- Spider trap: a group of pages with no links out of the group
  - A spider trap will eventually accumulate all importance of the Web

Practical PageRank

- $d$: decay factor
- $\text{PageRank}(p) = (d \cdot \sum_{q \in B(p)} \frac{\text{PageRank}(q)}{|F(q)|}) + (1 - d)$
- Intuition in the random surfer model
  - A surfer occasionally gets bored and jump to a random page on the Web instead of following a random link on the current page
Google (1998)

- Inverted lists in practice contain a lot of context information

<table>
<thead>
<tr>
<th>Hit 2 bytes</th>
<th>Capitalization</th>
<th>Relative font size</th>
</tr>
</thead>
<tbody>
<tr>
<td>plan:</td>
<td>caps:</td>
<td>type:</td>
</tr>
<tr>
<td>fancy:</td>
<td>cap:</td>
<td>type:</td>
</tr>
<tr>
<td>anchor:</td>
<td>cap:</td>
<td>type:</td>
</tr>
</tbody>
</table>

- PageRank is not the final ranking
  - Type-weight: depends on the type of the occurrence
    - For example, large font weights more than small font
  - Count-weight: depends on the number of occurrences
    - Increases linearly first but then tapers off
  - For multiple search terms, nearby occurrences are matched together and a proximity measure is computed
    - Closer proximity weights more

Summary

- Index documents for substring queries
  - Inverted lists, signature files—index “words”
  - Other approaches (not covered): suffix tree, Pat tree, suffix array—index all suffixes
- Web search and information retrieval go beyond substring queries
  - TF-IDF, PageRank, …