Web Crawling, Indexing, and Ranking

CPS 196.3
Introduction to Database Systems

Outline

- **Crawling**
  - Download Web pages
- **Indexing**
  - Index downloaded pages to facilitate searches
- **Ranking**
  - Rank result pages so that the most relevant ones are returned first

Crawling the Web

- Start with an initial set of URL’s, and place them in a priority queue
- Repeat until some stopping condition
  - Pick a URL from the queue
  - Download the page
  - Extract the URL’s on the downloaded page
  - Place newly discovered URL’s in the queue
Prioritizing crawl

- What pages should the crawler download first (so that most of the “important” pages can be downloaded in the shortest amount of time)?

- Possible importance/ordering metrics
  - Interest driven (useful for focused crawls)
    - Textual similarity to a driving query
    - Relevance to a topic
  - Location driven (based on URL)
    - Example: .com is more useful than .org
    - Example: …/home/… is more useful than …/tmp/…
  - Popularity driven
    - Backlink count
    - Google’s PageRank

Evaluating ordering strategies

  - Backlink count is the importance metric
  - But PageRank is still the best ordering metric!

Refresh strategies

- How should the crawler refresh downloaded pages?
- Metrics
  - Freshness: 1 if up to date, 0 otherwise
  - Age: 0 if up to date, (current time – modification time) otherwise
- Possible strategies
  - Uniform: revisit all pages at the same frequency regardless of how often they change
  - Proportional: revisit a page proportionally more often as it changes more often
Example of uniform versus proportional

\[ f_1 + f_2 = 4 \]
- \( f_1 = 3, f_2 = 1 \): expected freshness 1/2 and 1/2
- \( f_1 = 2, f_2 = 2 \): expected freshness 3/8 and 3/4

(Not Poisson)

Optimal refresh strategy

It is not worthwhile trying to keep up with the pages that change too frequently relative to the resources available (Cho & Garcia-Molina. "The Evolution of the Web and Implications for an Incremental Crawler." VLDB, 2000)

Indexing Web pages

<table>
<thead>
<tr>
<th>All words</th>
<th>All pages</th>
<th>Page 1</th>
<th>Page 2</th>
<th>Page 3</th>
<th>Page 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;a&quot;</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>&quot;cat&quot;</td>
<td>0</td>
<td>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>&quot;dog&quot;</td>
<td>0</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>
<td>0</td>
</tr>
</tbody>
</table>

- Inverted lists: store the matrix by rows
- Signature files: store the matrix by columns
With compression, of course!
Inverted lists

- For each word, store an inverted list
  - (word, page-id-list)
  - ("database", \{3, 7, 142, 857, \ldots\})
  - ("search", \{3, 9, 192, 512, \ldots\})
- A vocabulary index for looking up inverted list by word
- Example: find pages containing “database” and “search”
  - Use the vocabulary index to find the two inverted lists
  - Return the page ID’s in the intersection

Signature files

- For each page, 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 page
- Some false positives; no false negatives

Bit-sliced signature files

- Motivation
  - To check if a page 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 \(w\) 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
Inverted lists versus signature files

- Inverted lists are better for most purposes (Zobel et al., "Inverted Files versus Signature Files for Text Indexing." TODS, 1998)

- Problems of signature files
  - False positives
  - Hard to use because i, u, and the hash function need tuning to work well
  - Long pages will likely have mostly 1’s in signatures
  - Common words will create mostly 1’s for their slices

- Saving grace of signature files
  - Good for
  - Good for

Ranking result pages

- A single search may return many pages
  - A user will not look at all result pages
  - Complete result may be unnecessary
  - Result pages need to be ranked

- 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

- Vocabulary: \[w_1, \ldots, w_n\]

- IDF (Inverse Document Frequency): \[f_1, \ldots, f_n\]
  - \(f_i = \frac{1}{\text{the number of times } w_i \text{ appears on the Web}}\)

- Significance of words on page \(p\): \([p_1 f_1, \ldots, p_n f_n]\)
  - \(p_i\) is the number of times \(w_i\) appears on \(p\)

- Textual similarity between two pages \(p\) and \(q\) is defined to be \([p_1 f_1, \ldots, p_n f_n] \cdot [q_1 f_1, \ldots, q_n f_n] = p_1 q_1 f_1^2 + \ldots + p_n q_n f_n^2\)
  - \(q\) could be the query text
Why weight significance by IDF?

- "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.
- Without IDF weighting, the similarity measure would be dominated by the so-called stop words.

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.
- Based on local link structure; still easy to spam.
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

- Naive PageRank
  - \( N(p) \): number of outgoing links from page \( p \)
  - \( B(p) \): set of pages that point to \( p \)
  - \( \text{PageRank}(p) = \sum_{q \in B(p)} (\text{PageRank}(q) / N(q)) \)
  - Each page \( p \) gets a boost of its importance from each page that points to \( p \)
  - Each page \( q \) evenly distributes its importance to all pages that \( q \) points to

Calculating naive PageRank

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

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

- \( \text{PageRank}(p) \) 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
- $\text{PageRank}(p) = d \cdot \sum_{q \in B(p)} (\text{PageRank}(q)/N(q)) + (1 - d)$

Intuition in the random surfer model

- A surfer occasionally gets bored and jumps to a random page on the Web instead of following a random link on the current page

Beyond this lecture

- Inverted lists in practice contain a lot of context information
- 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