Web Searching & Indexing

CPS 116
Introduction to Database Systems

Announcements (December 3)
- Homework #4 sample solution available
- Course project demo period continues!
- Final exam next Tuesday, Dec. 8, 9am-12pm
  - Again, open book, open notes
  - Focus on the second half of the course
  - Sample final solution available
- I will be running a final review session next Monday, Dec. 7, 4-5pm

Keyword search

What are the documents containing both "database" and "search"?

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

Keywords × documents

\[
\begin{array}{cccc}
\text{All documents} & \text{Document 1} & \text{Document 2} & \text{Document 3} \\
\hline
\text{All keywords} & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \end{bmatrix} \\
\text{"a"} & \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \end{bmatrix} \\
\text{"cat"} & \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 \end{bmatrix} \\
\text{"database"} & \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 \end{bmatrix} \\
\text{"dog"} & \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 \end{bmatrix} \\
\text{"search"} & \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 \end{bmatrix} \\
\end{array}
\]

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

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"
  - Return documents in the intersection of the two inverted lists
- OR? NOT?
  - Union and difference, respectively
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
  - \( \{(\text{keyword}, \{(\text{doc-id}, \text{number-of-occurrences})\}), \ldots \} \)
Textual similarity

- Terms \( \{ t_1, \ldots, t_n \} \) and documents \( D = \{ d_1, d_2, \ldots \} \)
- IDF (Inverse Document Frequency) of \( t_i \):
  \[ \text{idf}_i = -\log \left( \frac{\# \text{ of docs in } D \text{ containing } t_i}{|D|} \right) \]
- TF (Term Frequency) of \( t_i \) in \( d_j \):
  \[ \text{tf}_{i,j} = \frac{\# \text{ of times } t_i \text{ appears in } d_j}{\# \text{ of term appearances in } d_j} \]
- TF-IDF weight vector of \( d_j \):
  \[ \text{w}_j = (\text{tf}_{1,j}, \text{idf}_1, \ldots, \text{tf}_{n,j}, \text{idf}_n) \]
- Textual similarity between two docs \( d_j \) and \( d_k \) can be measured by the normalized dot product of these vectors, i.e.:
  \[ \frac{( \text{w}_j \cdot \text{w}_k )}{\sqrt{\sum \text{tf}_{i,j}^2 \cdot \sum \text{tf}_{i,k}^2}} \]

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

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 \( p \)
  - \( \text{PageRank}(p) = \sum_{q \in B(p)} \left( \frac{\text{PageRank}(q)}{|F(q)|} \right) \)
  \( \mathrm{a} \): Each page gets a boost from every page pointing to it
  \( \mathrm{b} \): Each page distributes its importance evenly to pages it points to

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
  - Create lots of pages that point to a particular page

Calculating naïve PageRank

- Initially, set all PageRank’s to 1; then evaluate
  \[ \text{PageRank}(p) \leftarrow \sum_{q \in B(p)} \left( \frac{\text{PageRank}(q)}{N(q)} \right) \]
  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

- 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

Google (1998)

- Inverted lists in practice contain a lot of context information
  - Capitalization
  - Relative font size
  - In URL/title/meta tag
  - In anchor text
  - Within the page
  - Within the anchor

- 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

Trie: a string index

- A tree with edges labeled by characters

- A node represents the string obtained by concatenating all characters along the path from the root

- Compact trie: replace a path without branches by a single edge labeled by a string

Suffix tree

Index all suffixes of a large string in a compact trie

- Can support arbitrary substring matching
- Internal nodes have fan-out $\geq 2$ (except the root)
- No two edges out of the same node can share the same first character

To get linear space

- Instead of inlining the string labels, store pointers to them in the original string
- Bad for external memory
Patricia trie, Pat tree, String B-tree

A Patricia trie is just like a compact trie, but
- Instead of labeling each edge by a string, only label by the first character and the string length
- Leaves point to strings
- Faster search (especially for external memory) because of inlining of the first character
- But must validate answer at leaves for skipped characters

- A Pat tree indexes all suffixes of a string in a Patricia trie
- A String B-tree uses a Patricia trie to store and compare strings in B-tree nodes

Summary

- General tree-based string indexing tricks
  - Trie, Patricia trie, String B-tree
  - Two general ways to index for substring queries
    - Index words: inverted lists, signature files
    - Index all suffixes: suffix tree, Pat tree, suffix array (not covered)
  - Web search and information retrieval go beyond substring queries
    - TF-IDF, PageRank, …