

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

The screenshot shows a search engine interface with several search results. The search query entered is "database AND search". The results include:

- The Internet Movie Database (IMDb)...
- ... Search the Internet Movie Database. For more search options, please visit Search central...
- Images | Google...
- CPS 216: Advanced Database Systems (Fall 2008)
 - Course Information
 - Course Description / Time and Place / Books
 - Resources: Staff...
- Association for Computing Machinery
 - Founded in 1947, it is the world's educational and scientific computing society. Today, our members—...

At the bottom of the screenshot, there is a search bar containing the text "database AND search" and a "Search" button.

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

Keywords × documents

4

All keywords

All documents

Document 1 Document 2 Document 3 Document #

"a"	1	1	1	...	1
"cat"	1	1	0	...	0
"database"	0	0	1	...	0
"dog"	0	1	0	...	1
"search"	0	0	1	...	0
...

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

5

- ❖ Store the matrix by rows
- ❖ For each keyword, store an inverted list
 - $\langle \text{keyword}, \text{doc-id-list} \rangle$
 - $\langle \text{"database"}, \{3, 7, 142, 857, \dots\} \rangle$
 - $\langle \text{"search"}, \{3, 9, 192, 512, \dots\} \rangle$
 - 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

6

- ❖ 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?

What are “all” the keywords? 7

- ❖ 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 8

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

Signature files 9

- ❖ 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

Does doc_3 contain "database"?

$hash(\text{"database"}) = 0110$	doc_1 contains "database": 0110	"database"?
$hash(\text{"dog"}) = 1100$	doc_2 contains "dog": 1100	
$hash(\text{"cat"}) = 0010$	doc_3 contains "cat" and "dog": 1110	

- ☞ Some false positives; no false negatives

Bit-sliced signature files

10

❖ 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

doc	Signature
1	00001000
2	00001000
3	00011010
4	01101100
...	...
N	00001010

↓ ↓ ↓ ↓ ↓
Slice 7 ... Slice 0

Bit-sliced signature files

Starting to look like an inverted list again!

Inverted lists versus signatures

11

❖ Inverted lists better for most purposes (*TODS*, 1998)

❖ Problems of signature files

- False positives
- Hard to use because s , 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

12

❖ 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

13

- ❖ Terms $\{t_1, \dots, t_n\}$ and documents $D = \{d_1, d_2, \dots\}$
- ❖ IDF (Inverse Document Frequency) of t_j :
 - $\text{idf}_j = -\log(\# \text{ of docs in } D \text{ containing } t_j / |D|)$
- ❖ TF (Term Frequency) of t_j in d_j :
 - $\text{tf}_{i,j} = \frac{(\# \text{ of times } t_j \text{ appears in } d_j)}{(\# \text{ of term appearances in } d_j)}$
- ❖ TF-IDF weight vector of d_j :
 - $\mathbf{w}_j = \langle \text{tf}_{1,j}\text{idf}_1, \dots, \text{tf}_{n,j}\text{idf}_n \rangle$
- ❖ Textual similarity between two docs d_j and d_k can be measured by the normalized dot product of these vectors, i.e.:
 $(\mathbf{w}_j \cdot \mathbf{w}_k) / (\|\mathbf{w}_j\|_2 \cdot \|\mathbf{w}_k\|_2)$
 $= (\sum_i \text{tf}_{i,j} \text{tf}_{i,k} \text{idf}_i^2) / (\text{sqrt}(\sum_i \text{tf}_{i,j}^2 \text{idf}_i^2) \text{sqrt}(\sum_i \text{tf}_{i,k}^2 \text{idf}_i^2))$
 - One "doc" could be the query text

Why weigh significance by IDF?

14

Problems with content-based ranking

15

- ❖ 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

16

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

Google’s PageRank

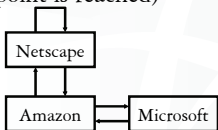
17

- ❖ 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

18

- ❖ 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)



$$\begin{bmatrix} n \\ m \\ a \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0.5 \\ 0 & 0 & 0.5 \\ 0.5 & 1 & 0 \end{bmatrix} \begin{bmatrix} n \\ m \\ a \end{bmatrix}$$

$$\begin{bmatrix} n \\ m \\ a \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.5 \\ 1.5 \end{bmatrix}, \begin{bmatrix} 1.25 \\ 0.75 \\ 1 \end{bmatrix}, \begin{bmatrix} 1.125 \\ 0.5 \\ 1.375 \end{bmatrix}, \begin{bmatrix} 1.25 \\ 0.6875 \\ 1.0625 \end{bmatrix}, \dots, \begin{bmatrix} 1.2 \\ 0.6 \\ 1.2 \end{bmatrix}$$

Random surfer model

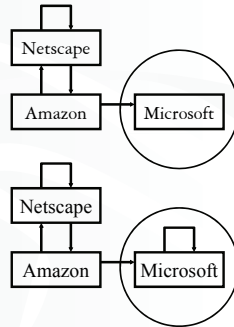
19

- ❖ 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

20

- ❖ 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

21

- ❖ d : decay factor
- ❖ 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 jump to a random page on the Web instead of following a random link on the current page

Google (1998)

22

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

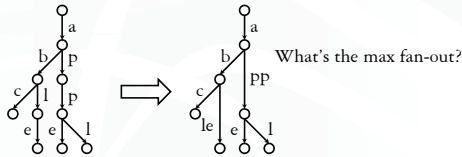
	Hit: 2 bytes	Relative Capitalization	font size	
In URL/title/meta tag	plain: cap:1	imp:3	position: 12	Within the page
In anchor text	fancy: cap:1	imp = 7	type: 4 position: 8	Within the page
In anchor text	anchor: cap:1	imp = 7	type: 4 hash:4 pos: 4	Within the anchor

- ❖ 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

23

- ❖ 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

24

Index all suffixes of a large string in a compact trie

- ☞ Can support arbitrary substring matching
- ❖ Internal nodes have fan-out ≥ 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 ²⁵

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 ²⁶

- ❖ 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, ...
