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

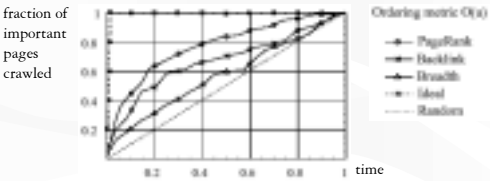
Crawling the Web

- 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

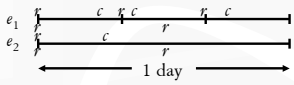


- A typical crawl (Cho et al. "Efficient Crawling through URL Reordering." *WWW7*, 1998)
  - Backlink count is the importance metric
  - But PageRank is still the best ordering metric!

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

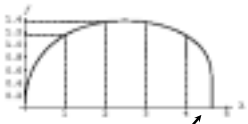
Example of uniform versus proportional



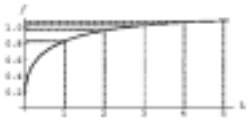
(Not Poisson)

- Say $f_1 + f_2 = 4$
  - $f_1 = 3, f_2 = 1$: expected freshness $1/2$ and $1/2$
  - $f_1 = 2, f_2 = 2$: expected freshness $3/8$ and $3/4$

Optimal refresh strategy



(a) change frequency vs. synchronization frequency for freshness optimization



(b) change frequency vs. synchronization frequency for age optimization

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)

policy	*Freshness*	*Age*
Proportional	0.12	400 days
Uniform	0.57	5.6 days
Optimal	0.62	4.3 days

Freshness and age prediction based on the real web

Indexing Web pages

All pages

All words	Page 1	Page 2	Page 3	…	Page *n*
"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
	…	…	…	…	…

- 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, …}⟩
  - ⟨"search", {3, 9, 192, 512, …}⟩
- 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

hash("database") = 0110
hash("dog") = 1100
hash("cat") = 0010

doc_1 contains "database": 0110
doc_2 contains "dog": 1100
doc_3 contains "cat" and "dog": 1110

Does doc_3 contain "database"?

☞ 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 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

page	*signature*
1	0 0 0 0 1 0 0 0
2	0 0 0 0 1 0 0 0
3	0 0 0 0 1 1 0 1 0
4	0 1 1 0 1 1 0 0
…	…
n	0 0 0 0 0 1 0 1 0

Slice 7 … Slice 0

Bit-sliced signature files

Starting to look like an inverted list again!

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 s, w, 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): $\{f_1, \ldots, f_n\}$
  - $f_i = 1$ / the number of times w_i 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

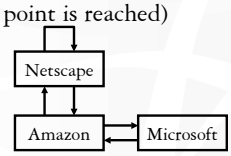
- A page with more backlinks is ranked higher
- Intuition: Each backlink is a "vote" for the page's importance
- Based on local link structure; still easy to spam

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
- Naïve PageRank
  - $N(p)$: number of outgoing links from page p
  - $B(p)$: set of pages that point to p
  - $\text{PageRank}(p) = \Sigma_{q \in B(p)} (\text{PageRank}(q) / N(q))$
  - ☞ 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 naïve PageRank

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



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

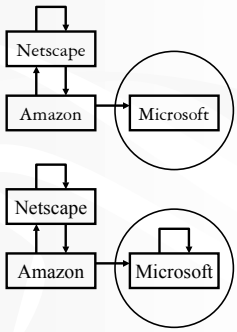
$$\begin{bmatrix} n \\ m \\ a \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0.5 \\ 1.5 \end{bmatrix}, \begin{bmatrix} 1.25 \\ 0.75 \\ 1 \end{bmatrix}, \begin{bmatrix} 1.125 \\ 0.5 \\ 1.375 \end{bmatrix}, \begin{bmatrix} 1.25 \\ 0.6875 \\ 1.0625 \end{bmatrix}, \ldots,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

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
- $\text{PageRank}(p)$ 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
- PageRank(p) =
  $d \cdot \Sigma_{q \in B(p)}$ (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

Beyond this lecture

- Inverted lists in practice contain a lot of context information

Google (1998)

Hit: 2 bytes

	Capitalization	Relative font size				
plain:	cap:1	imp:3	position: 12			within the page
In URL/title/meta tag 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

URL associated with 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