Lecture 10
Web Search and Introduction to Information Retrieval

Some slides due to Raghavan et al., via Dan Weld

Some Administration

- Exams back on Wednesday, we hope
- We're a few lectures ahead of the syllabus schedule
  - That lets us focus on some more fun & advanced topics
- We'll return to triggers and RSS in a few lectures; time to do something *non*-XML

2

Searching the Web

- Today, a short architectural overview
- Then onto:
  - IR issues
  - Crawling and search architecture
  - Modern search challenges

3

Searching the Web (2)

- Web Search is basically a database problem, but no one uses SQL databases
  - Every query is a top-k query
  - Every query plan is the same
  - Massive numbers of queries and data
  - Read-only data
- A search query can be thought of in SQL terms, but the engineered system is completely different

4

A Few Numbers

- 1B-100B pages
- 1 minute-1 month freshness
- 213M queries (*per day!*)
- Biggest challenges to processing a query:
  1. Result relevance
  2. Processing speed
  3. Scaling to many documents
- We'll talk about all of these, eventually

5

Search Document Model

- Think of a "web document" as a tuple with several columns:
  - Incoming link text
  - Title
  - Page content (maybe many sub-parts)
  - Unique docid
- A web search is really

```
SELECT * FROM docs WHERE
docs.text LIKE 'userquery' AND
docs.title LIKE 'userquery' AND …
ORDER BY 'relevance'
```

- Where `relevance` is very complicated…

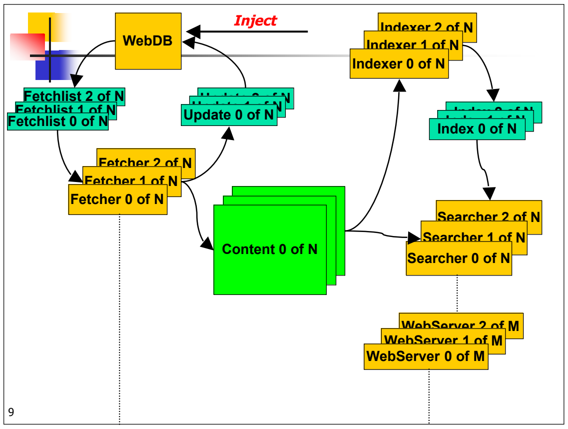
6

Search Challenges

- Three main challenges to processing a query:
  1. Result relevance
  2. Processing speed
  3. Scaling to many documents
- We'll talk about all of these, eventually
- But first…

Nutch: A Case Study

- A search engine is much more than the query system components
  - Simply obtaining the pages and constructing the index is a lot of work



Moving Parts

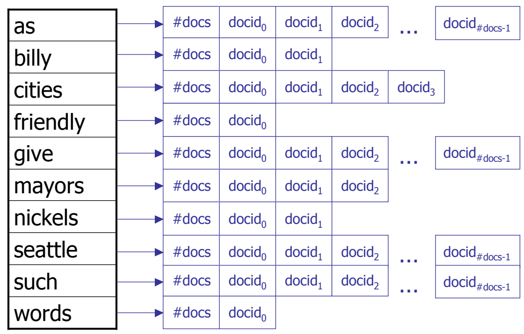
- Acquisition cycle
  - WebDB
  - Fetcher
- Index generation
  - Indexing
  - Link analysis (maybe)
- Serving results

WebDB

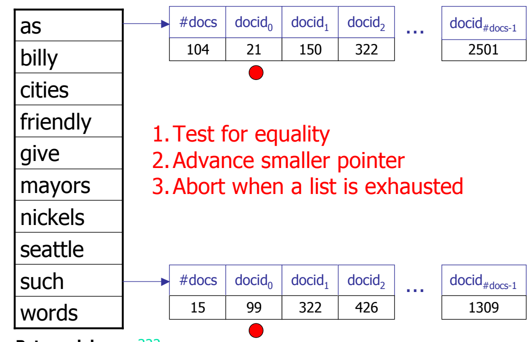
- Contains info on all pages, links
  - URL, last download, # failures, link score, content hash, ref counting
  - Source hash, target URL
- Must always be consistent
- Designed to minimize disk seeks
  - 19ms seek time x 200m new pages/mo
    = ~44 days of disk seeks!

Fetcher

- Fetcher is very stupid. Not a "crawler"
- Divide "to-fetch list" into *k* pieces, one for each fetcher machine
- URLs for one domain go to same list, otherwise random
  - "Politeness" w/o inter-fetcher protocols
  - Can observe robots.txt similarly
  - Better DNS, robots caching
  - Easy parallelism
- Two outputs: pages, WebDB edits

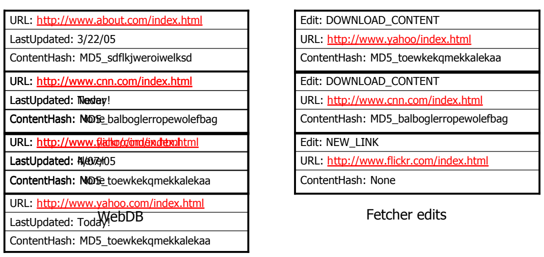


Query: **such as**



Returned docs: 322

WebDB/Fetcher Updates

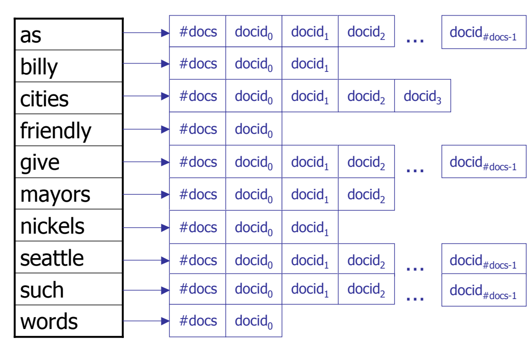


Indexing

- Iterate through all *k* page sets in parallel, constructing inverted index
- Creates a "searchable document" like we saw earlier:
  - URL text
  - Content text
  - Incoming anchor text

Serving Results - speed

- You could just run grep, but:
  - Each query will need to touch each document;
- Key to fast processing is the *inverted index*
- Basic idea is: for each word, list all the documents where that word can be found



Query: **such as**

as
billy
cities
friendly
give
mayors
nickels
seattle
such
words

as →

#docs	docid$_0$	docid$_1$	docid$_2$	…	docid$_{\#docs-1}$
104	21	150	322		2501

1. Test for equality
2. Advance smaller pointer
3. Abort when a list is exhausted

such →

#docs	docid$_0$	docid$_1$	docid$_2$	…	docid$_{\#docs-1}$
15	99	322	426		1309

Returned docs: 322

Challenges

- Three problems in life:
  1. Result relevance
  2. Processing speed
  3. Scaling to many documents

20

How are pages ranked?

- IR is largely study of how/why to prefer one page over another
  - Boolean retrieval
  - Vector-space model
  - Cosine distance
  - Assessing rank quality

21

Boolean Retrieval

- For each doc, two possible outcomes of query processing
  - TRUE or FALSE
  - "exact match" retrieval
  - Simplest form of ranking, used to be common
- Query specified with Boolean operators
  - AND, OR, NOT
  - Proximity operators (NEAR) also possible

22

Query

- Which plays of Shakespeare contain the words ***Brutus*** AND ***Caesar*** but NOT ***Calpurnia***?

23

Term-document incidence

	Tempest	Hamlet	Othello	Macbeth
Antony	0	0	0	1
Brutus	0	1	0	0
Caesar	0	1	1	1
Calpurnia	0	0	0	0
Cleopatra	0	0	0	0
mercy	1	1	1	1
worser	1	1	1	0

1 if **play** contains **word**,
0 otherwise

24

Beyond Term Search

- Phrases?
- Proximity: Gates NEAR Microsoft
  - Index should capture position info
- Zones in documents:
  Find (***author=Ullman***) AND
  (***text contains automata***)

25

Ranking Search Results

- Boolean queries simply *include* or *exclude* a document from results
- That's fine with few hits

Results **1** - **10** of about **48,500,000** for **shakespeare**. (**0.28** seconds)

- Boolean is a good first pass, but we need to prefer some documents over others

26

Hit Counting

- We could simply measure the size of the overlap between the document and the query
  - How many query words "hit"?
- But what about:
  - Term frequency in document
  - Term scarcity in collection
  - Length of documents

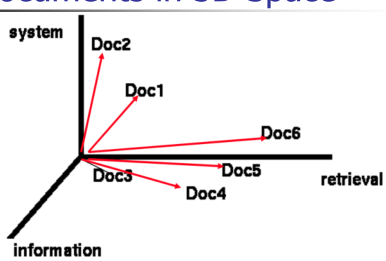
27

Documents as Vectors

- Each doc *j* can be viewed as a vector of *tf* values, one component for each term
- We thus have a *vector space*
  - Terms are axes
  - A doc is a point in the space
  - Space is hugely multidimensional. Can easily have 20,000+ dimensions

28

Documents in 3D Space



- One assumption: documents that are "close together" in space are also close in meaning

29

Vector Space Query Model

1. Treat a query as a short document
2. Sort documents by increasing distance (decreasing similarity) to the query document
3. Easy to compute, as both query & doc are vectors

- First used in Salton's SMART system (1970). Now used by almost every IR system

30

Vector Representation

- Docs & Queries are vectors
- Pos'n 1 corresponds to term 1. Pos'n *t* corresponds to term *t*
- Weight of term stored in each pos'n

$$D_i = w_{d_{i1}}, w_{d_{i2}}, \ldots, w_{d_{it}}$$

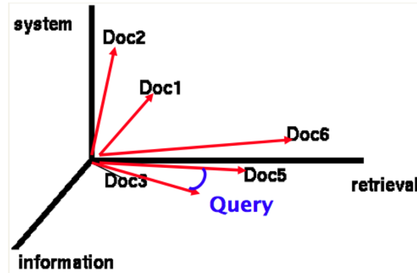
$$Q = w_{q1}, w_{q2}, \ldots, w_{qt}$$

$$w = 0 \text{ if a term is absent}$$

31

Documents in 3D Space

- Term weights indicate length of document vector along a dimension



32

Computing Weights

- Which word is more indicative of document similarity?
  - "Book" or "Rumplestiltskin"?
  - Need to consider **document frequency** - how often a word appears in doc collection
- Which doc is a better match for the query "kangaroo"?
  - One with a single mention of Kangaroos… or a doc that mentions it 10 times?
  - Need to consider **term frequency** - how many times the word appears in current document

33

TF x IDF

- "Term-Frequency" x "Inverse Document Frequency"

$$W_{ik} = tf_{ik} * \log(N / n_k)$$

- T_k = term k in document D_i
- tf_{ik} = freq of term T_k in doc D_i
- idf_k = inverse doc freq of term T_k in C
  $idf_k = \log(\frac{N}{n_k})$
- N = total # docs in collection C
- N_k = # docs in C that contain T_k

34

Inverse Document Frequency

- IDF provides high values for rare words, low values for common words

$$\log\left(\frac{10000}{10000}\right) = 0$$

$$\log\left(\frac{10000}{5000}\right) = 0.301$$

$$\log\left(\frac{10000}{20}\right) = 2.698$$

$$\log\left(\frac{10000}{1}\right) = 4$$

35

TF-IDF normalization

- Normalize term weights
  - Longer docs not given more weight
  - Force all values within [0,1]

$$w_{ik} = \frac{tf_{ik} \log(N / n_k)}{\sqrt{\sum_{k=1}^{t} (tf_{ik})^2 [\log(N / n_k)]^2}}$$

36

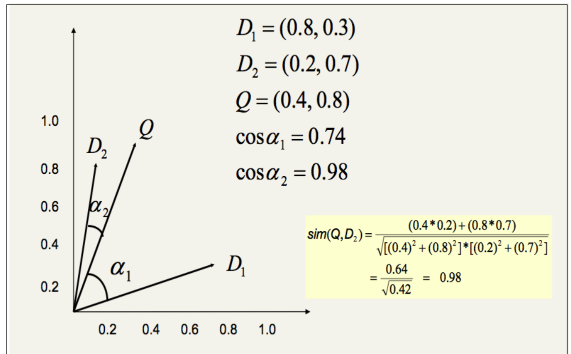
Vector space similarity

- Now, the similarity of two docs is:

$$Sim(Di,Dj) = \sum_{k=1}^{t} w_{ik} * w_{jk}$$

- Also called the cosine, or normalized inner product
(normalization done when computing term weights)
- Recall that cosine:
  - Depends on two adjacent vector lengths
  - =1 when angle is zero (points are identical)
  - Smaller when angle is greater

37



38

Computing a similarity score

- Say we have query vector Q = (0.4,0.8)
- Also, document D2 = (0.2,0.7)
- What is the result of the similarity computation?

$$sim(Q,D_2) = \frac{(0.4*0.2)+(0.8*0.7)}{\sqrt{[(0.4)^2+(0.8)^2]*[(0.2)^2+(0.7)^2]}}$$

$$= \frac{0.64}{\sqrt{0.42}} = 0.98$$

39

To Think About

- How does this ranking algorithm behave?
  - Make a set of hypothetical documents consisting of terms and their weights
  - Create some hypothetical queries
  - How are docs ranked, depending on weights of the terms and the queries' terms?

40

Summary: Vector Spaces

- User's query treated as short document
- Query is in same space as docs
- Easy to measure a doc's dist. to query
- Obvious extension from simple Boolean world

41