Introduction to IR Systems: Supporting Boolean Text Search

Chapter 27, Part A

Information Retrieval

- A research field traditionally separate from Databases
  - Goes back to IBM, Rand and Lockheed in the 50's
  - G. Salton at Cornell in the 60's
  - Lots of research since then
- Products traditionally separate
  - Originally, document management systems for libraries, government, law, etc.
  - Gained prominence in recent years due to web search

IR vs. DBMS

- Seem like very different beasts:

IR	DBMS
Imprecise Semantics	Precise Semantics
Keyword search	SQL
Unstructured data format	Structured data
Read-Mostly. Add docs occasionally	Expect reasonable number of updates
Page through **top *k*** results	Generate full answer

- Both support queries over large datasets, use indexing.
  - In practice, you currently have to choose between the two.

IR's "Bag of Words" Model

- Typical IR data model:
  - Each document is just a bag (multiset) of words ("terms")
- Detail 1: "Stop Words"
  - Certain words are considered irrelevant and not placed in the bag
  - e.g., "the"
  - e.g., HTML tags like <H1>
- Detail 2: "Stemming" and other content analysis
  - Using English-specific rules, convert words to their basic form
  - e.g., "surfing", "surfed" --> "surf"

Boolean Text Search

- Find all documents that match a Boolean containment expression:

  "Windows"
  AND ("Glass" OR "Door")
  AND NOT "Microsoft"
- Note: Query terms are also filtered via stemming and stop words.
- When web search engines say "10,000 documents found", that's the Boolean search result size (subject to a common "max # returned' cutoff).

Text "Indexes"

- When IR folks say "text index"…
  - Usually mean more than what DB people mean
- In our terms, both "tables" and indexes
  - Really a logical schema (i.e., tables)
  - With a physical schema (i.e., indexes)
  - Usually not stored in a DBMS
    - Tables implemented as files in a file system
    - We'll talk more about this decision soon

A Simple Relational Text Index

- Create and populate a table
  InvertedFile(term string, docURL string)
- Build a B+-tree or Hash index on InvertedFile.term
  - Alternative 3 (<Key, list of URLs> as entries in index) critical here for efficient storage!!
    - Fancy list compression possible, too
  - Note: URL instead of RID, the web is your "heap file"!
    - Can also *cache* pages and use RIDs
- This is often called an "inverted file" or "inverted index"
  - Maps from words -> docs
- Can now do single-word text search queries!

An Inverted File

- Search for
  - "databases"
  - "microsoft"

term	docURL
data	http://www-inst.eecs.berkeley.edu/~cs186
database	http://www-inst.eecs.berkeley.edu/~cs186
date	http://www-inst.eecs.berkeley.edu/~cs186
day	http://www-inst.eecs.berkeley.edu/~cs186
dbms	http://www-inst.eecs.berkeley.edu/~cs186
decision	http://www-inst.eecs.berkeley.edu/~cs186
demonstrate	http://www-inst.eecs.berkeley.edu/~cs186
description	http://www-inst.eecs.berkeley.edu/~cs186
design	http://www-inst.eecs.berkeley.edu/~cs186
desire	http://www-inst.eecs.berkeley.edu/~cs186
developer	http://www.microsoft.com
differ	http://www-inst.eecs.berkeley.edu/~cs186
disability	http://www.microsoft.com
discussion	http://www-inst.eecs.berkeley.edu/~cs186
division	http://www-inst.eecs.berkeley.edu/~cs186
do	http://www-inst.eecs.berkeley.edu/~cs186
document	http://www-inst.eecs.berkeley.edu/~cs186

Handling Boolean Logic

- How to do "term1" OR "term2"?
  - Union of two DocURL sets!
- How to do "term1" AND "term2"?
  - Intersection of two DocURL sets!
    - Can be done by sorting both lists alphabetically and merging the lists
- How to do "term1" AND NOT "term2"?
  - Set subtraction, also done via sorting
- How to do "term1" OR NOT "term2"
  - Union of "term1" and "NOT term2".
    - "Not term2" = all docs not containing term2. Large set!!
  - Usually not allowed!
- Refinement: What order to handle terms if you have many ANDs/NOTs?

Boolean Search in SQL

"Windows" AND ("Glass" OR "Door") AND NOT "Microsoft"

```
(SELECT docURL FROM InvertedFile
  WHERE word = "windows"
 INTERSECT
 SELECT docURL FROM InvertedFile
  WHERE word = "glass" OR word =
"door")
EXCEPT
SELECT docURL FROM InvertedFile
 WHERE word="Microsoft"
ORDER BY relevance()
```


Boolean Search in SQL

- Really only one SQL query in Boolean Search IR:
  - Single-table selects, UNION, INTERSECT, EXCEPT
- relevance () is the "secret sauce" in the search engines:
  - Combos of statistics, linguistics, and graph theory tricks!
  - Unfortunately, not easy to compute this efficiently using typical DBMS implementation.

Computing Relevance

- Relevance calculation involves how often search terms appear in doc, and how often they appear in collection:
  - More search terms found in doc → doc is more relevant
  - Greater importance attached to finding *rare* terms
- Doing this efficiently in current SQL engines is not easy:
  - "Relevance of a doc wrt a search term" is a function that is called once per doc the term appears in (docs found via inv. index):
    - For efficient fn computation, for each term, we can store the # times it appears in each doc, as well as the # docs it appears in.
    - Must also sort retrieved docs by their relevance value.
    - Also, think about Boolean operators (if the search has multiple terms) and how they affect the relevance computation!
  - An object-relational or object-oriented DBMS with good support for function calls is better, but you still have long execution path-lengths compared to optimized search engines.

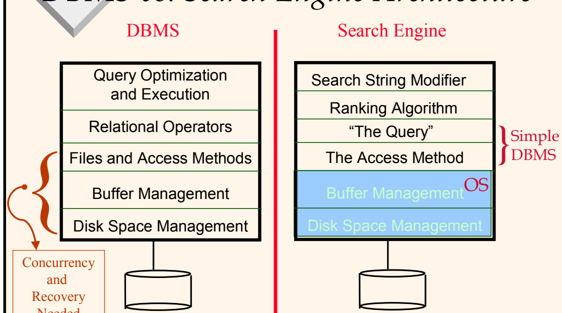
Fancier: Phrases and "Near"

- Suppose you want a phrase
  - E.g., "Happy Days"
- Different schema:
  - InvertedFile (term string, count int, position int, DocURL string)
  - Alternative 3 index on term
- Post-process the results
  - Find "Happy" AND "Days"
  - Keep results where positions are 1 off
    - Doing this well is like *join processing*
- Can do a similar thing for "term1" NEAR "term2"
  - Position < *k* off

Updates and Text Search

- Text search engines are designed to be query-mostly:
  - Deletes and modifications are rare
  - Can postpone updates (nobody notices, no transactions!)
    - Updates done in batch (rebuild the index)
  - Can't afford to go off-line for an update?
    - Create a 2nd index on a separate machine
    - Replace the 1st index with the 2nd!
  - So no concurrency control problems
  - Can compress to search-friendly, update-unfriendly format
- Main reason why text search engines and DBMSs are usually separate products.
  - Also, text-search engines tune that one SQL query to death!

DBMS vs. Search Engine Architecture



IR vs. DBMS Revisited

- Semantic Guarantees
  - DBMS guarantees transactional semantics
    - If inserting Xact commits, a later query *will see* the update
    - Handles multiple concurrent updates correctly
  - IR systems do not do this; nobody notices!
    - Postpone insertions until convenient
    - No model of correct concurrency
- Data Modeling & Query Complexity
  - DBMS supports any schema & queries
    - Requires you to define schema
    - Complex query language hard to learn
  - IR supports only one schema & query
    - No schema design required (unstructured text)
    - Trivial to learn query language

IR vs. DBMS, Contd.

- Performance goals
  - DBMS supports general SELECT
    - Plus mix of INSERT, UPDATE, DELETE
    - General purpose engine must always perform "well"
  - IR systems expect only one stylized SELECT
    - Plus delayed INSERT, unusual DELETE, no UPDATE.
    - Special purpose, must run super-fast on "The Query"
    - Users rarely look at the full answer in Boolean Search

Lots More in IR …

- How to "rank" the output? I.e., how to compute relevance of each result item w.r.t. the query?
  - Doing this well / efficiently is hard!
- Other ways to help users paw through the output?
  - Document "clustering", document visualization
- How to take advantage of hyperlinks?
  - Really cute tricks here!
- How to use compression for better I/O performance?
  - E.g., making RID lists smaller
  - Try to make things fit in RAM!
- How to deal with synonyms, misspelling, abbreviations?
- How to write a good web crawler?