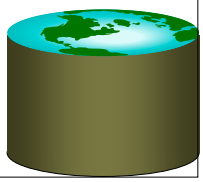
Elementary IR Systems: Supporting Boolean Text Search



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
- **Today: simple IR techniques**
  - Show similarities to DBMS techniques you already know
  - We'll skip:
    - Specialized storage tricks
    - Ranking results (hopefully later!)
    - Parallelism (hopefully later)
    - Bells and whistles (lots of little ones!)

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

- **Under the hood, not as different as they might seem**
  - But in practice, you have to choose between the 2

IR's "Bag of Words" Model

- **Typical IR data model:**
  - Each document is just a bag 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"**
  - Using English-specific rules, convert words to their basic form
  - e.g. "surfing", "surfed" --> "surf"
  - Unfortunately have to do this for each language
    - Yuck!

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.**

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**
  - Something like "Alternative 3" critical here!!
    - Keep lists of dup keys sorted by docURL
    - Fancy list compression important, too
    - Typically called a "postings list"
  - 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
    - whereas normal files map docs to the words in the doc!
- **Can now do single-word text search queries!**

An Inverted File

- **Snippets from:**
  - Class web page
  - microsoft.com
- **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
document	http://www.microsoft.com
microsoft	http://www.microsoft.com
microsoft	http://www-inst.eecs.berkeley.edu/~cs186
midnight	http://www-inst.eecs.berkeley.edu/~cs186
midterm	http://www-inst.eecs.berkeley.edu/~cs186
minibase	http://www-inst.eecs.berkeley.edu/~cs186
million	http://www.microsoft.com
monday	http://www.microsoft.com
more	http://www.microsoft.com
most	http://www-inst.eecs.berkeley.edu/~cs186
ms	http://www-inst.eecs.berkeley.edu/~cs186
msn	http://www.microsoft.com
must	http://www-inst.eecs.berkeley.edu/~cs186
necessary	http://www-inst.eecs.berkeley.edu/~cs186
need	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 postings lists!
    - Can be done via merge-join over postings lists
    - Remember: postings list per key sorted by DocURL in index
- **How to do "term1" AND NOT "term2"?**
  - Set subtraction
    - Also easy because sorted
- **How to do "term1" OR NOT "term2"**
  - Union of "term1" and "NOT term2".
    - "Not term2" = all docs not containing term2. Yuck!
  - Usually not allowed!
- **Query Optimization: what order to handle terms if you have many ANDs?**

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

Boolean Search in SQL

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

- **Really only one SQL query template in Boolean Search**
  - Single-table selects, UNION, INTERSECT, EXCEPT
- **magic_rank() is the "secret sauce" in the search engines**
  - Hopefully we'll study this later in the semester
  - Combos of statistics, linguistics, and graph theory tricks!

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
  - Postings lists sorted by (DocURL, position)
- **Post-process the results**
  - Find "Happy" AND "Days"
  - Keep results where positions are 1 off
    - Can be done during merge-join to AND the 2 lists!
- **Can do a similar thing for "term1" NEAR "term2"**
  - Position < *k* off
  - Think about refinement to merge-join…

Somewhat better compression

- InvertedFile (term string, count int, position int, docID int)
- Docs(docID int, docURL string, snippet string, …)
- Btree on InvertedFile.term
- Btree on Docs.docID

- Requires a final join step between typical query result and Docs.docID
  - Can do this lazily: cursor to generate a page full of results

Updates and Text Search

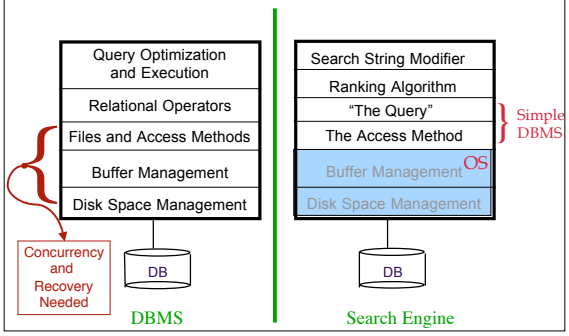
- **Text search engines are designed to be query-mostly**
  - Deletes and modifications are rare
  - Can postpone updates (nobody notices, no transactions!)
    - Updates done in batch (rebuild the index)
  - Can't afford to go offline for an update?
    - Create a 2nd index on a separate machine
    - Replace the 1st index with the 2nd!
  - So no concurrency control problems
  - Can compress to search-friendly, update-unfriendly format
  - Can keep postings lists sorted
- **For these reasons, text search engines and DBMSs are usually separate products**
  - Also, text-search engines tune that one SQL query to death!
  - The benefits of a special-case workload.

Lots more tricks in IR

- **How to "rank" the output?**
  - A mix of simple tricks works well
  - Some fancier tricks can help (use hyperlink graph)
- **Other ways to help users paw through the output?**
  - Document "clustering" (e.g. NorthernLight)
  - Document visualization
- **How to use compression for better I/O performance?**
  - E.g. making postings lists smaller
  - Try to make things fit in RAM!
- **How to deal with synonyms, misspelling, abbreviations?**
- **How to write a good webcrawler?**

- **Hopefully we'll return to some of these later**
  - See *Managing Gigabytes* for some of the details

Recall From the First Lecture



You Know The Basics!

- **"Inverted files" are the workhorses of all text search engines**
  - Just B+-tree or Hash indexes on bag-of-words
- **Intersect, Union and Set Difference (Except)**
  - Usually implemented via sorting
  - Or can be done with hash or index joins
- **Most of the other stuff is not "systems" work**
  - A lot of it is cleverness in dealing with language
  - Both linguistics and statistics (more the latter!)

Revisiting Our IR/DBMS Distinctions

- **Semantic Guarantees**
  - DBMS guarantees transactional semantics
    - If an inserting transaction commits, a subsequent 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.
    - Can even return incorrect answers for various reasons!
- **Data Modeling & Query Complexity**
  - DBMS supports any schema & queries
    - But requires you to define schema
    - And SQL is hard to figure out for the average citizen
  - IR supports only one schema & query
    - No schema design required (unstructured text)
    - Trivial (natural?) query language for simple tasks

Revisiting Distinctions, Cont.

- **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
      - Postpone any work you can to subsequent index joins
      - But make sure you can rank!