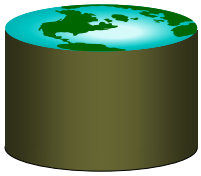
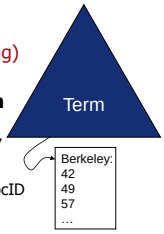
Text/Web Search II: Ranking & Crawling



Review: Simple Relational Text Index

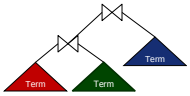
- **Create and populate a table**
  InvertedFile(term string, docID string)
- **Build a B+-tree or Hash index on InvertedFile.term**
  - Use something like "Alternative 3" index
    - Keep lists at the bottom sorted by docID
    - Typically called a "postings list"



"Berkeley Database Research"

Boolean Search in SQL

```
SELECT IB.docID
  FROM InvertedFile IB, InvertedFile ID, InvertedFile IR
 WHERE IB.docID = ID.docID AND ID.docID = IR.docID
   AND IB.term = "Berkeley"
   AND ID.term = "Database"
   AND IR.term = "Research"
ORDER BY magic_rank()
```

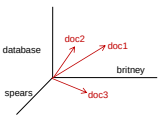


- **This time we wrote it as a join**
  - Last time wrote it as an INTERSECT
- **Recall our query plan**
  - An indexscan on each `Ix.term` "instance" in FROM clause
  - A merge-join of the 3 indexscans (ordered by docID)
- **magic_rank() is the "secret sauce" in the search engines**
  - Will require rewriting this query somewhat…

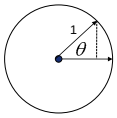
Classical IR Ranking

- **Abstraction: Vector space model**
  - We'll think of every document as a "vector"
    - Imagine there are 10,000 possible terms
    - Each document (bag of words) can be represented as an array of 10,000 counts
    - This array can be thought of as a point in 10,000-dimensional space
  - Measure "distance" between two vectors: "similarity" of two documents
- **A query is just a short document**
  - Rank all docs by their distance to the query "document"!

Classical IR Ranking



- **What's the right distance metric?**
  - Problem 1: two long docs seem more similar to each other than to short docs
    - Solution: normalize each dimension by vector's (Euclidean) length
    - Now every doc is a point on the unit sphere
  - Now: the dot-product (sum of products) of two normalized vectors happens to be cosine of the angle between them!
    - $(dj \cdot dk)/(|dj||dk|) = \cos(\theta)$
      - to see this in 2D, "rotate" so one vector is (1,0)
  - BTW: for normalized vectors, cosine *ranking* is the same as *ranking* by Euclidean distance



TF × IDF

What is the idf of a term that occurs in all of the docs? In almost no docs?

- **Counting occurrences isn't a good way to weight each term**
  - Want to favor repeated terms in this doc
  - Want to favor unusual words in this doc
- **TF × IDF (Term Frequency × Inverse Doc Frequency)**
  - For each doc d
    - DocTermRank =
      #occurrences of t in d **TF**
      × log((total #docs)/(#docs with this term)) **IDF**
  - Instead of using counts in the vector, use DocTermRank

Indexing TF × IDF

- **Let's add some more to our schema**
  - TermInfo(term string, numDocs int).
    Used to compute IDF.
    - This is a "materialized" view on the invertedFile table.
    - Write down the SQL for the view!
  - InvertedFile(term string, docID int64, DocTermRank float)
    - Why not just store TF rather than DocTermRank?

In SQL Again…

–InvertedFile (term string, docID int64, DocTermRank float)

Simple Boolean Search

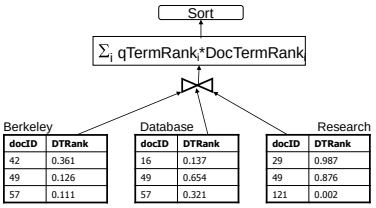
```
CREATE VIEW BooleanResult AS (
SELECT IB.docID, IB.DocTermRank as bTFIDF,
       ID.DocTermRank as dTFIDF,
       IR.DocTermRank as rTFIDF,
  FROM InvertedFile IB, InvertedFile ID, InvertedFile IR
 WHERE IB.docID = ID.docID AND ID.docID = IR.docID
   AND IB.term = "Berkeley"
   AND ID.term = "Database"
   AND IR.term = "Research");

SELECT docID,
       (<Berkeley-doctermrank>*bTFIDF +
        <Database-doctermrank>*dTFIDF +
        <Research-doctermrank>*rTFIDF>) AS magic_rank
  FROM BooleanResult
ORDER BY magic_rank;
```

Cosine similarity. Note that the query "doc" vector is a constant, can be queried up front

Really? Sort the whole Boolean Result??

Ranking



docID	DTRank
42	0.361
49	0.126
57	0.111

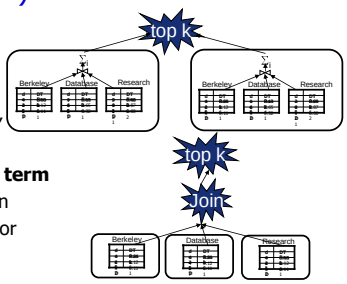
docID	DTRank
16	0.137
49	0.654
57	0.321

docID	DTRank
29	0.987
49	0.876
121	0.002

- **We'll only rank Boolean results**
  - Note: this is just a heuristic! (Why?)
    - What's a fix? Is it feasible?
  - Recall: a merge-join of the postings-lists from each term, sorted by docID
- **While merging postings lists…**
  - For each docID that matches on all terms (Bool)
    - Compute cosine distance to query
      - I.e. For all terms, Sum of (product of query-term-rank and DocTermRank)
    - This collapses the view in the previous slide
- **What's wrong with this picture??**

Parallelizing (!!)

- **Partition InvertedFile by DocID**
  - Parallel "top k"
- **Partition InvertedFile by term**
  - Distributed Join
  - top k: parallel or not?
- **Pros/cons?**
  - What are the relevant metrics?



Note that there's usually another join stage

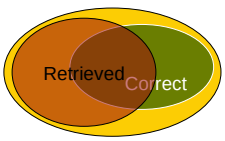
- **Docs(docID, title, URL, crawldate, snippet)**

```
SELECT title, URL, crawldate, snippet
       (<Berkeley-tfidf>*bTFIDF +
        <Database-tfidf>*dTFIDF +
        <Research-TFIDF>*rTFIDF>) AS magic_rank
  FROM BooleanResult, Docs
 WHERE BooleanResult.docID = Docs.docID
ORDER BY magic_rank;
```

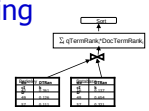
- **Typically rank *before* the join with Docs**
  - not an "interesting order"
  - so a fully parallel join with Docs
    - and/or you can replicate the Docs table

Quality of a non-Boolean Answer

- **Suppose only top k answers are retrieved**
- **Two common metrics:**
  - Precision: |Correct ∩ Retrieved| / |Retrieved|
  - Recall: |Correct ∩ Retrieved| / |Correct|



Phrase & Proximity Ranking

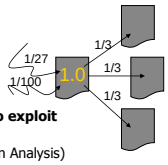


- **Query: "The Who"**
  - How many matches?
    - Our previous query plan?
  - Ranking quality?
- **One idea: index all 2-word runs in a doc**
  - "bigrams", can generalize to "n-grams"
  - give higher rank to bigram matches
- **More generally, proximity matching**
  - how many words/characters apart?
    - add a "list of positions" field to the inverted index
    - ranking function scans these two lists to compute proximate usage, cook this into the overall rank

Some Additional Ranking Tricks

- **Query expansion, suggestions**
  - Can do similarity lookups on terms, expand/modify people's queries
- **Fix misspellings**
  - E.g. via an inverted index on q-grams of letters
  - Trigrams for "misspelling" are {mis, iss, ssp, spe, pel, ell, lli, lin, ing}
- **Document expansion**
  - Can add terms to a doc before inserting into inverted file
    - E.g. in "anchor text" of refs to the doc
    - E.g. by classifying docs (e.g. "english", "japanese", "adult")
- **Not all occurrences are created equal**
  - Mess with DocTermRank based on:
    - Fonts, position in doc (title, etc.)
    - Don't forget to normalize: "tugs" doc in direction of heavier weighted terms

Hypertext Ranking



- **On the web, we have more information to exploit**
  - The hyperlinks (and their anchor text)
  - Ideas from Social Network Theory (Citation Analysis)
  - "Hubs and Authorities" (Clever), "PageRank" (Google)
- **Intuition (Google's PageRank)**
  - If you are important, and you link to me, then I'm important
  - Recursive definition --> recursive computation
    1. Everybody starts with weight 1.0
    2. Share your weight among all your outlinks
    3. Repeat (2) until things converge
  - Note: computes the first eigenvector of the adjacency matrix
    - And you thought linear algebra was boring :-)
  - Leaving out some details here …
- **PageRank sure seems to help**
  - But rumor says that other factors matter as much or more
    - Anchor text, title/bold text, etc. --> much tweaking over time

Random Notes from the Real World

- **The web's dictionary of terms is HUGE. Includes:**
  - numerals: "1", "2", "3", … "987364903", …
  - codes: "gistindex_keydistance", "authlatlon", …
  - misspellings: "teh", "quik", "browne", "focs"
  - multiple languages: "hola", "bonjour", "ここんんににちちはは" (Japanese), etc.
- **Web spam**
  - Try to get top-rated. Companies will help you with this!
  - Imagine how to spam TF x IDF
    - "Stanford Stanford Stanford Stanford Stanford Stanford Stanford Stanford … Stanford lost The Big Game"
    - And use white text on a white background :-)
  - Imagine spamming PageRank…?!
- **Some "real world" stuff makes life easier**
  - Terms in queries are Zipfian! Can cache answers in memory effectively.
  - Queries are usually little (1-2 words)
  - Users don't notice minor inconsistencies in answers
- **Big challenges in running thousands of machines, 24x7 service!**

Building a Crawler

- **Duh! This is graph traversal.**

```
crawl(URL) {
    doc = fetch(URL);
    foreach href in the URL
        crawl(*href);
}
```

- **Well yes, but:**
  - better not sit around waiting on each fetch
  - better run in parallel on many machines
  - better be "polite"
  - probably won't "finish" before the docs change
    - need a "revisit policy"
  - all sorts of yucky URL details
    - dynamic HTML, "spider traps"
    - different URLs for the same data (mirrors, .. in paths, etc.)

Single-Site Crawler

- **multiple outstanding fetches**
  - each with a modest timeout
    - don't let the remote site choose it!
  - typically a multithreaded component
    - but can typically scale to more fetches/machine via a single-threaded "event-driven" approach
- **a set of pending fetches**
  - this is your crawl "frontier"
  - can grow to be quite big!
  - need to manage this wisely to pick next sites to fetch
  - what traversal would a simple FIFO queue for fetches give you? Is that good?

Crawl ordering

- **What do you think?**
  - Breadth first vs. Depth first?
  - Content driven? What metric would you use?
- **What are our goals**
  - Find good pages soon (may not finish before restart)
  - Politeness

Crawl Ordering, cont.

- **Good to find high PageRank pages, right?**
  - Could prioritize based on knowledge of P.R.
    - E.g. from earlier crawls
  - Research sez: breadth-first actually finds high P.R. pages pretty well though
    - Random doesn't do badly either
  - Other research ideas to kind of approximate P.R. online
  - Have to be at the search engines to really know how this is best done
    - Part of the secret sauce!
    - Hard to recreate without a *big* cluster and *lots* of NW

Scaling up

- **How do you parallelize a crawler?**
  - Roughly, you need to partition the frontier a la parallel join or map/reduce
  - Load balancing requires some thought
    - partition by URL prefix (domain name)? by entire URL?
- **DNS lookup overhead can be a substantial bottleneck**
  - E.g. the mapping from www.cs.berkeley.edu to 169.229.60.105
  - Pays to maintain local DNS caches at each node

More on web crawlers?

- **There is a quite detailed Wikipedia page**
  - Focus on academic research, unfortunately
  - Still, a lot of this stuff came out of universities
    - Wisconsin (webcrawler '94), Berkeley (inktomi '96), Stanford (google '99)

Resources

- **Textbooks**
  - *Managing Gigabytes*, Witten/Moffat/Bell
  - *Modern Information Retrieval,* Baeza-Yates/Ribeiro-Neto
  - *Introduction to Information Retrieval,* Manning/ Raghavan/Schütze (free online!)
- **Lecture Notes**
  - Manning/Raghavan/Schütze notes to go with text
    - Source of some material in these slides