Ranking Results in IR Search

### Boolean Search in SQL

```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";
```

- This time we wrote it as a join
- Last time we wrote it as an INTERSECT
- Recall our query plan
  - An indexscan on each table "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
- What’s the right distance metric?
  - Problem 1: two long docs seem similar to each other than to short docs
    - Solution: normalize each dimension by each of its components by vector’s length
  - Now, the dot-product (sum of products) of two normalized vectors happens to be cosine of angle between them

### In SQL Again...

```sql
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");  
```

- InvertedFile (term string, docID int64, DocTermRank float)
- Cosine similarity
  - Note that the query is a constant
  - The cosine similarity, not the query doc vector is a constant

---

**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”**

TF * IDF

- 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 / \log \text{total #docs} / \text{docs with this term} / IDF
  - Instead of using counts in the vector, use DocTermRank
- Let’s add some more to our schema
  - TermString(term string, numDocs int) – used to compute IDF
  - InvertedFile(term string, docID int64, DocTermRank float)
We'll only rank Boolean results
- Note: this is just a heuristic! (Why?)
- 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 (Boo!)
  - 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

Some Additional Ranking Tricks
- Phrases/Proximity
  - Ranking function can incorporate position
- Query expansion, suggestions
  - Can keep a similarity matrix on terms, and expand/modify people's queries
- Fix misspellings
  - E.g. via an inverted index on n-grams
  - N-grams for "misspelling" are {mis, is, isp, etc.}
- Document expansion
  - Can add terms to a doc before inserting into inverted file
    - E.g. in "anchor text" of refs to the doc
- Not all occurrences are created equal
  - Mess with DocTermRank based on:
    - Fonts, position in doc
    - 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)
  - Comes 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 principal eigenvector of the adjacency matrix
- 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: "transValueIsNull", "palloc"
  - Misspellings: "teh", "quik", "browne", "focs"
  - Multiple languages: "hola", "bonjour", "__________" (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... The Big Game" (Google)
  - And use white text on a white background :-)
- 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 a 24x7 service!
  - We discuss some of this in CS262A