**Ranking Results in IR Search**

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

- **Boolean SQL Search**
  - `SELECT IB.docID
   FROM InvertedFile IB, InvertedFile ID, InvertedFile IR
   WHERE IB.docID = ID.docID AND ID.docID = IR.docID
   AND ID.term = "Berkeley"
   AND IR.term = "Database"
   ORDER BY magic_rank();`

- **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 more similar to each other than to short docs
    - Solution: normalize each dimension by vector's (Euclidean) length
    - Problem 2: normalize 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
    - \( \cos(\theta) = \frac{d_i \cdot d_j}{|d_i||d_j|} \)
    - BTW: for normalized vectors, cosine ranking is the same as ranking by Euclidean distance (prove this to yourself for 2-d)

- **In SQL Again...**
  - `CREATE VIEW BooleanResult AS {
   SELECT IB.docID, IB.DocTermRank as bTFIDF,
   ID.DocTermRank as tTFIDF,
   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";`
  - `CREATE VIEW BooleanResult AS {
   SELECT docID,
   "<Berkeley-TFIDF> + <Database-TFIDF> + <Research-TFIDF>" AS magic_rank
   FROM BooleanResult
   ORDER BY magic_rank;`
Some Additional Ranking Tricks

- Phrases/Proximity
  - Bump exact phrase matches up the ranking
  - Give extra weight to proximate occurrences

- 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
  - Trigrams for “misspelling” are (mis, is, ap, ap, pol, all, il, lin, ing)

- Document expansion
  - Can add terms to a doc before inserting into inverted file
  - E.g. in “anchor text” of web to the doc

- Not all occurrences are created equal
  - Nobs with DocTermRank based on:
    - Fonts, position in doc (title, etc.)
    - Don’t forget to normalize “top” doc in direction of heavier weighted terms

Random Notes from the Real World

- The web’s dictionary of terms is huge. Includes:
  - Numerals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 10, 11, 12, ...
  - Codes: “CranioVertebral,” “Jalisco,” ...
  - Mispellings: “the,” “quik,” “brown,” “foux”
  - Multiple languages: “Hola,” “Bonjour,” “こんにちは,” “こんにちは” (Japanese), etc.

- Web spam
  - Try to get top-rated. Companies will help you with this!
  - Imagine how to spam TF x IDF
    - 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 short (1-2 words)
  - Users don’t notice minor inconsistencies in answers

- Big challenges in running thousands of machines, 24x7 service!

Parallelism 101: Metrics

- Speedup
  - Same task, more resources

- Scaleup
  - Bigger task, bigger resources

- Transaction scaleup
  - More tasks, bigger resources

Parallelism 101: Hardware

- Shared Memory

- Shared Disk

- Shared Nothing (Clusters)
Parallelism 101: Types of Parallelism

- Pipelined Parallelism
- Partition Parallelism

Barriers to Perfect Parallelization

- Startup
- Interference
- Skew

Relational Stuff Parallelizes Beautifully

- Dataflow: Single Instruction Multiple Data (SIMD)
- Relational, so order-independent!
- Pipelines AND Parallelizes beautifully

Data Layouts

- How to partition a table?
  - Round-robin
  - Range-partition
  - Hash partition
- Secondary indexes?
  - Partitioned with data
    - Broadcast and fetch
    - Expensive to maintain DISTINCT
  - Partitioned by key
    - Two-step lookup (latency)

Parallel Aggregation

- SUM
- AVERAGE
- MEDIAN

Parallel Sort

- Read data and range-partition it on the fly
- Sort locally
- Pipelining too!
  - Reading from disk
  - Sending over NW
  - Receiving and sorting
  - Writing runs
- **Hash Join**
  - **Piece of cake!**

  Original Relation

  - **INPUT**
  - **Partitions of R & S**
  - **Input buffer for Si**
  - **Hash table for partition Ri**
  - **B main memory buffers**
  - **Join Result**
  - **Hash table for partition Ri (k + B-1 pages)**
  - **Output buffer**
  - **Disk**

  **Join Result**

  - **B main memory buffers**
  - **Disk**

  **How about our text search query?**

  Top-K

  \[ \sum_{i} qTermRank_{i} \times DocTermRank_{i} \]

  - **Berkeley**
  - **Database**
  - **Research**

<table>
<thead>
<tr>
<th>DocID</th>
<th>TermRank</th>
<th>DocID</th>
<th>TermRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>0.24</td>
<td>09</td>
<td>0.26</td>
</tr>
<tr>
<td>04</td>
<td>0.38</td>
<td>10</td>
<td>0.35</td>
</tr>
<tr>
<td>07</td>
<td>0.11</td>
<td>19</td>
<td>0.01</td>
</tr>
</tbody>
</table>

  - **Top-K**
  - **49**
  - **49**
  - **29**

  - **DTRank**
  - **docID**
  - **0.002**
  - **121**
  - **0.876**
  - **49**
  - **0.987**
  - **29**

  - **Sigma**
  - **qTermRank**
  - **DocTermRank**

  - **merge**