Chapter 5

Ranking with Indexes
Indexes and Ranking

- Indexes are designed to support *search*
  - Faster response time, supports updates

- Text search engines use a particular form of search: *ranking*
  - Docs are retrieved in sorted order according to a score computing using the doc representation, the query, and a *ranking algorithm*

- What is a reasonable abstract model for ranking?
  - Enables discussion of indexes without details of retrieval model
Fred's Tropical Fish Shop is the best place to find tropical fish at low, low prices. Whether you're looking for a little fish or a big fish, we've got what you need. We even have fake seaweed for your fishtank (and little surfboards too).

More Concrete Model

\[ R(Q, D) = \sum_i g_i(Q) f_i(D) \]

\( f_i \) is a document feature function

\( g_i \) is a query feature function

Document

More Concrete Model

Quality Features

303.01

Document Score

Topical Features

9.7 fish
4.2 tropical
22.1 tropical fish
8.2 seaweed
4.2 surfboards

Topical Features

fish 5.2
tropical 3.4
tropical fish 9.9
cichlids 1.2
barbs 0.7

Query

14 incoming links
3 update count

Quality Features

incoming links 1.2
update count 0.9
Inverted Index

- Each index term is associated with an *inverted list*
  - Contains lists of *documents*, or lists of *word occurrences* in documents, and other information
  - Each entry is called a *posting*
  - The part of the *posting* that refers to a specific document or location is called a *pointer*
  - Each document in the collection is given a unique number
  - Lists are usually *document-ordered* (sorted by document number)
Example “Collection”

$S_1$ Tropical fish include fish found in tropical environments around the world, including both freshwater and saltwater species.

$S_2$ Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

$S_3$ Tropical fish are popular aquarium fish, due to their often bright coloration.

$S_4$ In freshwater fish, this coloration typically derives from iridescence, while saltwater fish are generally pigmented.

Four sentences from the Wikipedia entry for tropical fish
Simple Inverted Index

and 1
aquarium 3
are 3
around 4
as 2
both 1
bright 3
coloration 3
derives 4
due 3
environments 1
fish 1
fishkeepers 2
found 1
fresh 2
freshwater 1
from 4
generally 4
in 1
include 1
including 1
iridescence 4
marine 2
often 2
only 2
pigmented 4
popular 3
refer 2
referred 2
requiring 2
salt 1
saltwater 2
species 1
term 2
the 1
their 2
this 4
those 2
to 2
tropical 1
typically 4
use 2
water 1
while 4
with 2
world 1
Inverted Index with counts - supports better ranking algorithms

<table>
<thead>
<tr>
<th>Word</th>
<th>1:1</th>
<th>2:1</th>
<th>3:1</th>
<th>4:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aquarium</td>
<td>3:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>are</td>
<td>3:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>around</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>2:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>both</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bright</td>
<td>3:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coloration</td>
<td>3:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>derives</td>
<td></td>
<td>4:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>due</td>
<td></td>
<td></td>
<td>3:1</td>
<td></td>
</tr>
<tr>
<td>environments</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fish</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fishkeepers</td>
<td>2:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>found</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fresh</td>
<td>2:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>freshwater</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>generally</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>include</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>including</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iridescence</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>marine</td>
<td>2:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>often</td>
<td>2:1</td>
<td>3:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>only</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pigmented</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popular</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>refer</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>referred</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>requiring</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>salt</td>
<td>1:1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>saltwater</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>species</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>term</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the</td>
<td></td>
<td></td>
<td>1:1</td>
<td></td>
</tr>
<tr>
<td>their</td>
<td></td>
<td></td>
<td></td>
<td>3:1</td>
</tr>
<tr>
<td>this</td>
<td></td>
<td></td>
<td></td>
<td>4:1</td>
</tr>
<tr>
<td>those</td>
<td></td>
<td></td>
<td></td>
<td>2:1</td>
</tr>
<tr>
<td>to</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tropical</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>typically</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>use</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>water</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>while</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>world</td>
<td></td>
<td></td>
<td></td>
<td>1:1</td>
</tr>
</tbody>
</table>

No. of time the word occurs: Doc #
Inverted Index with Positions - Supports Proximity Matches

Position in the doc

Doc #
Proximity Matches

- Matching phrases or words within a window
  - e.g., "tropical fish", or “find tropical within 5 words of fish”
- Word positions in inverted lists make these types of query features efficient
  - e.g.,

  tropical 1,1 1,7 2,6 2,17 3,1
  fish 1,2 1,4 2,7 2,18 2,23 3,2 3,6 4,3 4,13
MapReduce

- Distributed programming framework that focuses on data placement and distribution
- A programming model (code) for processing & generating large data sets
- **Mapper** (or the *map* function)
  - Transforms a list of items (key/value pairs) into another list of items (intermediate key/value pairs) of the same length
- **Reducer** (or the *reduce* function)
  - Transforms/merges a list of items (immediate key/value pairs) into a single item (with the same intermediate key)
- Many mapper & reducer tasks on a cluster of machines
Mappers and Reducers

- Map-Reduce job
  - *Map* function (inputs $\rightarrow$ key-value pairs)
  - *Reduce* function (key & list of values $\rightarrow$ outputs)

- Map and Reduce Tasks apply Map or Reduce function to (typically) many of their inputs
  - Unit of parallelism

- *Mapper* = application of the *Map* function to a single input

- *Reducer* = application of the *Reduce* function to a single key-(list of values) pair
MapReduce

- In 2003 a system was built at Google to simplify construction of the inverted index for handling searches
- Example. Counting the *number of occurrences* of each word in a large collection of documents

Map(String key, String value):
   // key: document name
   // value: document contents
   for each word w in value
     EmitIntermediate(w, "1")

Reduce(String key, Iterator values):
   // key: a word
   // values: a list of counts
   int result = 0;
   for each v in values
     results += ParseInt(v);
   Emit(AsString(results));
Inverted Index Creation

- Input: Large number of text documents
- Output: Postings lists for every term in the collection
  - For every word, all documents that contain the word & the positions

```
I saw the cat on the mat
http://www.cat.com/
```

```
I saw the dog on the mat
http://www.dog.com/
```

- I ➔ http://www.cat.com, 0 ➔ http://www.dog.com, 0
- saw ➔ http://www.cat.com, 1 ➔ http://www.dog.com, 1
- the ➔ http://www.cat.com, 2 ➔ http://www.dog.com, 2
- mat ➔ http://www.cat.com, 6 ➔ http://www.dog.com, 6
Inverted Index Creation

Solution to the problem:

- **Mapper:** For each word in a doc, generates (word, [URL, position])
- **Reducer:** Aggregate all the information on the same word

```pseudo
// Pseudo-code for “inverted index”:
Map(String key, String value):
    // key: document URL
    // value: document contents
    vector words = tokenize(value)
    for position from 0 to len(words):
        EmitIntermediate(w, {key, position});

Reduce(String key, Iterator values):
    // key: a word
    // values: a list of {URL, position} tuples
    postings_list = [];
    for each v in values:
        postings_list.append(v);
    sort(postings_list); // Sort by URL, position
    Emit(key, AsString(postings_list));
```
Inverted Index Creation

- Inverted index combiners:
  - **Combiners** reduces the number of intermediate outputs, aggregating all occurrences of document words.
MapReduce

- MapReduce automatically parallelizes & executes a program on a large cluster of commodity machines

- The runtime system
  - Partitioning the input data
  - Scheduling the program’s execution
  - Handling machine failures
  - Managing required inter-machine communication

- A MapReduce computation processes many terabytes of data on hundreds/thousands of machines

- More than 100,000 MapReduce jobs are executed on Google’s clusters every day
MapReduce

- Basic process
  - Map stage which transforms data records into pairs, each with a **key** and a **value**
  - **Shuffle** uses a hash function so that all pairs with the **same key** end up next to each other and on the same machine
  - **Reduce** stage processes records in batches, where all pairs with the **same key** are processed at the same time

- **Idempotence** of Mapper & Reducer provides fault tolerance
  - multiple operations on same input gives same output
MapReduce
Example: Natural Join

- Join of $R(A, B)$ with $S(B, C)$ is the set of tuples $(a, b, c)$ such that $(a, b)$ is in $R$ and $(b, c)$ is in $S$

- Mappers need to send $R(a, b)$ and $S(b, c)$ to the same reducer, so they can be joined there

- Mapper output: $key = B$-value, $value = relation$ and other component ($A$ or $C$)

  - Example: $R(1, 2) \rightarrow (2, (R, 1))$
  - $S(2, 3) \rightarrow (2, (S, 3))$
Mapping Tuples

- **$R(1, 2)$**
  - Mapper for $R(1, 2)$
  - $(2, (R, 1))$

- **$R(4, 2)$**
  - Mapper for $R(4, 2)$
  - $(2, (R, 4))$

- **$S(2, 3)$**
  - Mapper for $S(2, 3)$
  - $(2, (S, 3))$

- **$S(5, 6)$**
  - Mapper for $S(5, 6)$
  - $(5, (S, 6))$
Grouping Phase

- There is a reducer for each key.
- Every key-value pair generated by any mapper is sent to the reducer for its key.
Mapping Tuples

Mapper for $R(1, 2)$

$2 \mapsto (2, (R, 1))$

Mapper for $R(4, 2)$

$2 \mapsto (2, (R, 4))$

Mapper for $S(2, 3)$

$2 \mapsto (2, (S, 3))$

Mapper for $S(5, 6)$

$5 \mapsto (5, (S, 6))$

Reducer for $B = 2$

Reducer for $B = 5$
Constructing Value-Lists

- The input to each *reducer* is organized by the system into a pair:
  - The *key*
  - The list of *values* associated with that *key*

\[(2, [(R, 1), (R, 4), (S, 3)])\] → Reducer for \(B = 2\)

\[(5, [(S, 6)])\] → Reducer for \(B = 5\)
The Reduce Function for Join

- Given key $b$ and a list of values that are either $(R, a_i)$ or $(S, c_j)$, output each triple $(a_i, b, c_j)$

  - Thus, the number of outputs made by a reducer is the product of the number of $R$'s on the list and the number of $S$'s on the list

(2, [(R, 1), (R, 4), (S, 3)])  ➔  Reducer for $B = 2$  ➔  (1, 2, 3), (4, 2, 3)

(5, [(S, 6)])  ➔  Reducer for $B = 5$  ➔  
The Drug-Interaction Problem

- Data consists of records for 3,000 drugs
  - List of patients taking the drugs, dates, and diagnoses.
  - About 1MB of data per drug

- Problem is to find drug interactions
  - Example. two drugs that when taken together increase the risk of heart attack

- Must examine each pair of drugs and compare their data
The first attempt used the following plan:

- **Key** = set of two drugs \{ i, j \}
- **Value** = the record for one of these drugs

- Given drug \textit{i} and its record \( R_i \), the mapper generates all key-value pairs (\{ i, j \}, \( R_i \)), where \( j \) is any other drug besides \( i \)

- Each \textit{reducer} receives its key and a list of the two records for that pair: (\{ i, j \}, [\( R_i \), \( R_j \)])
### Example: Three Drugs

<table>
<thead>
<tr>
<th>Mapper for Drug 1</th>
<th>{1, 2}</th>
<th>Drug 1 data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{1, 3}</td>
<td>Drug 1 data</td>
</tr>
<tr>
<td>Mapper for Drug 2</td>
<td>{1, 2}</td>
<td>Drug 2 data</td>
</tr>
<tr>
<td></td>
<td>{2, 3}</td>
<td>Drug 2 data</td>
</tr>
<tr>
<td>Mapper for Drug 3</td>
<td>{1, 3}</td>
<td>Drug 3 data</td>
</tr>
<tr>
<td></td>
<td>{2, 3}</td>
<td>Drug 3 data</td>
</tr>
</tbody>
</table>
Example: Three Drugs

Mapper for Drug 1

- \{1, 2\} Drug 1 data
- \{1, 3\} Drug 1 data

Mapper for Drug 2

- \{1, 2\} Drug 2 data
- \{2, 3\} Drug 2 data

Mapper for Drug 3

- \{1, 3\} Drug 3 data
- \{2, 3\} Drug 3 data

Reducer for \{1, 2\}

Reducer for \{1, 3\}

Reducer for \{2, 3\}
Example: Three Drugs

- **{1, 2}**
  - Drug 1 data
  - Drug 2 data
  - Reducer for {1, 2}

- **{1, 3}**
  - Drug 1 data
  - Drug 3 data
  - Reducer for {1, 3}

- **{2, 3}**
  - Drug 2 data
  - Drug 3 data
  - Reducer for {2, 3}