Information Retrieval and Map-Reduce Implementations
Roadmap

- Introduction to information retrieval
- Basics of indexing and retrieval
- Inverted indexing in MapReduce
- Retrieval at scale
First, nomenclature...

- **Information retrieval (IR)**
  - Focus on textual information (= text/document retrieval)
  - Other possibilities include image, video, music, ...

- **What do we search?**
  - Generically, “collections”
  - Less-frequently used, “corpora”

- **What do we find?**
  - Generically, “documents”
  - Even though we may be referring to web pages, PDFs, PowerPoint slides, paragraphs, etc.
Information Retrieval Cycle

Source Selection

Query Formulation

Search

Selection

Examination

Delivery

Resource

Query

Results

Documents

Information

System discovery
Vocabulary discovery

Concept discovery
Document discovery

source reselection

www.studentsfocus.com
The Central Problem in Search

Searcher

Concepts

Query Terms
“tragic love story”

Author

Concepts

Document Terms
“fateful star-crossed romance”

Do these represent the same concepts?

www.studentsfocus.com
Abstract IR Architecture

- Query
  - Representation Function
    - Query Representation
  - Comparison Function
- Documents
  - Representation Function
    - Document Representation
  - Index
    - offline
- online
  - document acquisition (e.g., web crawling)

www.studentsfocus.com
How do we represent text?

- Remember: computers don’t “understand” anything!

- “Bag of words”
  - Treat all the words in a document as index terms
  - Assign a “weight” to each term based on “importance” (or, in simplest case, presence/absence of word)
  - Disregard order, structure, meaning, etc. of the words
  - Simple, yet effective!

- Assumptions
  - Term occurrence is independent
  - Document relevance is independent
  - “Words” are well-defined
What’s a word?

天主教教宗若望保祿二世因感冒再度住進醫院。這是他今年第二度因同樣的病因住院。

 وقال مارك ريجيف - الناطق باسم الخارجية الإسرائيلية - إن شارون قبل الدعوة وسيقوم للمرة الأولى بزيارة تونس، التي كانت لفترة طويلة المقر الرسمي لمنظمة التحرير الفلسطينية بعد خروجها من لبنان عام 1982.

Выступая в Мещанском суде Москвы экс-глава ЮКОСа заявил не совершал ничего противозаконного, в чем обвиняет его генпрокуратура России.

भारत सरकार ने आर्थिक सर्वेक्षण में वित्तीय वर्ष 2005-06 में सात फीसदी विकास दर हासिल करने का आकलन किया है और कर सुधार पर ज़ोर दिया है

日米連合で台頭中国に対処…アーミテージ前副長官提言

조재영 기자= 서울시는 25일 이명박 시장이 `행정중심복합도시' 건설안에 대해 `군대라도 동원해 막고싶은 심정'이라고 말했다는 일부 언론의 보도를 부인했다.
Sample Document

McDonald's slims down spuds

Fast-food chain to reduce certain types of fat in its french fries with new cooking oil.

NEW YORK (CNN/Money) - McDonald's Corp. is cutting the amount of "bad" fat in its french fries nearly in half, the fast-food chain said Tuesday as it moves to make all its fried menu items healthier.

But does that mean the popular shoestring fries won't taste the same? The company says no. "It's a win-win for our customers because they are getting the same great french-fry taste along with an even healthier nutrition profile," said Mike Roberts, president of McDonald's USA.

But others are not so sure. McDonald's will not specifically discuss the kind of oil it plans to use, but at least one nutrition expert says playing with the formula could mean a different taste.

Shares of Oak Brook, Ill.-based McDonald's (MCD: down $0.54 to $23.22, Research, Estimates) were lower Tuesday afternoon. It was unclear Tuesday whether competitors Burger King and Wendy's International (WEN: down $0.80 to $34.91, Research, Estimates) would follow suit. Neither company could immediately be reached for comment.

...
Information retrieval models

- An IR model governs how a document and a query are represented and how the relevance of a document to a user query is defined.

- Main models:
  - Boolean model
  - Vector space model
  - Statistical language model
  - etc
Boolean model

- Each document or query is treated as a “bag” of words or terms. Word sequence is not considered.
- Given a collection of documents $D$, let $V = \{t_1, t_2, \ldots, t_{|V|}\}$ be the set of distinctive words/terms in the collection. $V$ is called the vocabulary.
- A weight $w_{ij} > 0$ is associated with each term $t_i$ of a document $d_j \in D$. For a term that does not appear in document $d_j$, $w_{ij} = 0$.

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{|V|j}),$$
Boolean model (contd)

- Query terms are combined logically using the Boolean operators **AND, OR, and NOT**.
  - E.g., 
    
    $$(data \ AND \ mining) \ AND \ (NOT \ text)$$

- Retrieval
  - Given a Boolean query, the system retrieves every document that makes the query logically true.
  - Called **exact match**.

- The retrieval results are usually quite poor because term frequency is not considered.
Boolean queries: Exact match

• The **Boolean retrieval model** is being able to ask a query that is a Boolean expression:
  – Boolean Queries are queries using *AND*, *OR* and *NOT* to join query terms
    • Views each document as a set of words
    • Is precise: document matches condition or not.
  – Perhaps the simplest model to build an IR system on
• Primary commercial retrieval tool for 3 decades.
• Many search systems you still use are Boolean:
  – Email, library catalog, Mac OS X Spotlight
Strengths and Weaknesses

- **Strengths**
  - Precise, if you know the right strategies
  - Precise, if you have an idea of what you’re looking for
  - Implementations are fast and efficient

- **Weaknesses**
  - Users must learn Boolean logic
  - Boolean logic insufficient to capture the richness of language
  - No control over size of result set: either too many hits or none
  - **When do you stop reading?** All documents in the result set are considered “equally good”
  - **What about partial matches?** Documents that “don’t quite match” the query may be useful also

www.studentsfocus.com
Assumption: Documents that are “close together” in vector space “talk about” the same things

Therefore, retrieve documents based on how close the document is to the query (i.e., similarity $\sim$ “closeness”)
Similarity Metric

- Use “angle” between the vectors:

\[
\cos(\theta) = \frac{\vec{d}_j \cdot \vec{d}_k}{||\vec{d}_j|| \cdot ||\vec{d}_k||}
\]

\[
sim(d_j, d_k) = \frac{\vec{d}_j \cdot \vec{d}_k}{||\vec{d}_j|| \cdot ||\vec{d}_k||} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,k}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2 \sqrt{\sum_{i=1}^{n} w_{i,k}^2}}}
\]

- Or, more generally, inner products:

\[
sim(d_j, d_k) = \vec{d}_j \cdot \vec{d}_k = \sum_{i=1}^{n} w_{i,j} w_{i,k}
\]
Vector space model

- Documents are also treated as a “bag” of words or terms.
- Each document is represented as a vector.
- However, the term weights are no longer 0 or 1. Each term weight is computed based on some variations of TF or TF-IDF scheme.
Term Weighting

- Term weights consist of two components
  - Local: how important is the term in this document?
  - Global: how important is the term in the collection?

- Here’s the intuition:
  - Terms that appear often in a document should get high weights
  - Terms that appear in many documents should get low weights

- How do we capture this mathematically?
  - Term frequency (local)
  - Inverse document frequency (global)
TF.IDF Term Weighting

\[ w_{i,j} = tf_{i,j} \cdot \log \frac{N}{n_i} \]

- \( w_{i,j} \) weight assigned to term \( i \) in document \( j \)
- \( tf_{i,j} \) number of occurrence of term \( i \) in document \( j \)
- \( N \) number of documents in entire collection
- \( n_i \) number of documents with term \( i \)

\[ tf_{ij} = \frac{f_{ij}}{\max\{f_{1ij}, f_{2ij}, \ldots, f_{mij}\}} \]

www.studentsfocus.com
Retrieval in vector space model

- Query \( q \) is represented in the same way or slightly differently.

- **Relevance of \( d_i \) to \( q \):** Compare the similarity of query \( q \) and document \( d_i \).

- Cosine similarity (the cosine of the angle between the two vectors)

\[
\text{cosine}(d_j, q) = \frac{\langle d_j \cdot q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{\|V\|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{\|V\|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{\|V\|} w_{iq}^2}}
\]

- Cosine is also commonly used in text clustering

www.studentsfocus.com
An Example

- A document space is defined by three terms:
  - hardware, software, users
  - the vocabulary

- A set of documents are defined as:
  - $A_1=(1, 0, 0)$, $A_2=(0, 1, 0)$, $A_3=(0, 0, 1)$
  - $A_4=(1, 1, 0)$, $A_5=(1, 0, 1)$, $A_6=(0, 1, 1)$
  - $A_7=(1, 1, 1)$, $A_8=(1, 0, 1)$, $A_9=(0, 1, 1)$

- If the Query is “hardware and software”

- what documents should be retrieved?
An Example (cont.)

- **In Boolean query matching:**
  - document A4, A7 will be retrieved (“AND”)
  - retrieved: A1, A2, A4, A5, A6, A7, A8, A9 (“OR”)

- **In similarity matching (cosine):**
  - q=(1, 1, 0)
  - $S(q, A1)=0.71$, $S(q, A2)=0.71$, $S(q, A3)=0$
  - $S(q, A4)=1$, $S(q, A5)=0.5$, $S(q, A6)=0.5$
  - $S(q, A7)=0.82$, $S(q, A8)=0.5$, $S(q, A9)=0.5$
  - Document retrieved set (with ranking)=
    - {A4, A7, A1, A2, A5, A6, A8, A9}
Constructing Inverted Index (Word Counting)

Documents → Bag of Words → Inverted Index

- case folding, tokenization, stopword removal, stemming
- syntax, semantics, word knowledge, etc.

www.studentsfocus.com
Stopwords removal

• Many of the most frequently used words in English are useless in IR and text mining – these words are called *stop words*.
  – the, of, and, to, ....
  – Typically about 400 to 500 such words
  – For an application, an additional domain specific stopwords list may be constructed

• Why do we need to remove stopwords?
  – Reduce indexing (or data) file size
    • stopwords accounts 20-30% of total word counts.
  – Improve efficiency and effectiveness
    • stopwords are not useful for searching or text mining
    • they may also confuse the retrieval system.

www.studentsfocus.com
Stemming

- Techniques used to find out the root/stem of a word.
  E.g.,
  - user engineering
  - users engineered
  - used engineer
  - using
  
  - stem: use engineer

Usefulness:

- improving effectiveness of IR and text mining
  - matching similar words
  - Mainly improve recall

- reducing indexing size
  - combing words with same roots may reduce indexing size as much as 40-50%.
Basic stemming methods

Using a set of rules. E.g.,

• remove ending
  – if a word ends with a consonant other than s, followed by an s, then delete s.
  – if a word ends in es, drop the s.
  – if a word ends in ing, delete the ing unless the remaining word consists only of one letter or of th.
  – If a word ends with ed, preceded by a consonant, delete the ed unless this leaves only a single letter.
  – ......

• transform words
  – if a word ends with “ies” but not “eies” or “aies” then “ies --> y.”
Inverted index

- The inverted index of a document collection is basically a data structure that
  - attaches each distinctive term with a list of all documents that contains the term.

- Thus, in retrieval, it takes constant time to
  - find the documents that contains a query term.
  - multiple query terms are also easy handle as we will see soon.
An example

Example 3: We have three documents of \( id_1 \), \( id_2 \), and \( id_3 \):

\[
\begin{align*}
id_1: & \text{ Web mining is useful.} \\
& 1 \quad 2 \quad 3 \quad 4 \\
\hline
id_2: & \text{ Usage mining applications.} \\
& 1 \quad 2 \quad 3 \\
\hline
id_3: & \text{ Web structure mining studies the Web hyperlink structure.} \\
& 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8
\end{align*}
\]

<table>
<thead>
<tr>
<th>Applications</th>
<th>Hyperlink</th>
<th>Mining</th>
<th>Structure</th>
<th>Studies</th>
<th>Usage</th>
<th>Useful</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>( id_2 )</td>
<td>( id_3 )</td>
<td>( id_1, id_2, id_3 )</td>
<td>( id_3 )</td>
<td>( id_3 )</td>
<td>( id_2 )</td>
<td>( id_1 )</td>
<td>( id_1, id_3 )</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{Applications: } & \langle id_2, 1, [3] \rangle \\
\text{Hyperlink: } & \langle id_3, 1, [7] \rangle \\
\text{Mining: } & \langle id_1, 1, [2] \rangle, \langle id_2, 1, [2] \rangle, \langle id_3, 1, [3] \rangle \\
\text{Structure: } & \langle id_3, 2, [2, 8] \rangle \\
\text{Studies: } & \langle id_3, 1, [4] \rangle \\
\text{Usage: } & \langle id_2, 1, [1] \rangle \\
\text{Useful: } & \langle id_1, 1, [4] \rangle \\
\text{Web: } & \langle id_1, 1, [1] \rangle, \langle id_3, 2, [1, 6] \rangle
\end{align*}
\]

(A) (B)

Fig. 6.7. Two inverted indices: a simple version and a more complex version
Search using inverted index

Given a query \( q \), search has the following steps:

- **Step 1 (vocabulary search):** find each term/word in \( q \) in the inverted index.
- **Step 2 (results merging):** Merge results to find documents that contain all or some of the words/terms in \( q \).
- **Step 3 (Rank score computation):** To rank the resulting documents/pages, using,
  - content-based ranking
  - link-based ranking
Inverted Index: Boolean Retrieval

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

- blue
- cat
- egg
- fish
- green
- ham
- hat
- one
- red
- two

www.studentsfocus.com
Boolean Retrieval

- To execute a Boolean query:
  - Build query syntax tree
  - For each clause, look up postings
  - Traverse postings and apply Boolean operator

- Efficiency analysis
  - Postings traversal is linear (assuming sorted postings)
  - Start with shortest posting first

www.studentsfocus.com
Query processing: AND

• Consider processing the query:

  Brutus AND Caesar

  – Locate Brutus in the Dictionary;
    • Retrieve its postings.
  
  – Locate Caesar in the Dictionary;
    • Retrieve its postings.
  
  – “Merge” the two postings:

  Brutus
  2 → 4 → 8 → 16 → 32 → 64 → 128
  1 → 2 → 3 → 5 → 8 → 13 → 21 → 34

  Caesar

www.studentsfocus.com
The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries.

If the list lengths are \(x\) and \(y\), the merge takes \(O(x+y)\) operations.

**Crucial**: postings sorted by docID.
Intersecting two postings lists
(a “merge” algorithm)

\textbf{INTERSECT}(p_1, p_2)

1 \hspace{1em} \textit{answer} \gets \langle \rangle

2 \hspace{1em} \textbf{while} \ p_1 \neq \text{NIL} \ \text{and} \ p_2 \neq \text{NIL}

3 \hspace{1em} \textbf{do if} \ \text{docID}(p_1) = \text{docID}(p_2)

4 \hspace{2em} \textbf{then} \ \textbf{ADD}(\textit{answer, docID}(p_1))

5 \hspace{1em} p_1 \gets \text{next}(p_1)

6 \hspace{1em} p_2 \gets \text{next}(p_2)

7 \hspace{1em} \textbf{else if} \ \text{docID}(p_1) < \text{docID}(p_2)

8 \hspace{2em} \textbf{then} \ p_1 \gets \text{next}(p_1)

9 \hspace{2em} \textbf{else} \ p_2 \gets \text{next}(p_2)

10 \hspace{1em} \textbf{return} \ \textit{answer}
Inverted Index: TF.IDF

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham

<table>
<thead>
<tr>
<th>tf</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

www.studentsfocus.com
Positional Indexes

- Store term position in postings
- Supports richer queries (e.g., proximity)
- Naturally, leads to larger indexes…
Retrieval in a Nutshell

- Look up postings lists corresponding to query terms
- Traverse postings for each query term
- Store partial query-document scores in accumulators
- Select top $k$ results to return
Retrieval: Document-at-a-Time

- Evaluate documents one at a time (score all query terms)

<table>
<thead>
<tr>
<th>blue</th>
<th>9 2 21 1</th>
<th>35 1</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish</td>
<td>1 2 9 1</td>
<td>21 3</td>
<td>34 1</td>
</tr>
</tbody>
</table>

Accumulators (e.g. priority queue)

Document score in top k?
- Yes: Insert document score, extract-min if queue too large
- No: Do nothing

- Tradeoffs
  - Small memory footprint (good)
  - Must read through all postings (bad), but skipping possible
  - More disk seeks (bad), but blocking possible

www.studentsfocus.com
Evaluation of documents one query term at a time

- Usually, starting from the most rare term (often with tf-sorted postings)

Tradeoffs

- Early termination heuristics (good)
- Large memory footprint (bad), but filtering heuristics possible

www.studentsfocus.com
MapReduce it?

- The indexing problem
  - Scalability is critical
  - Must be relatively fast, but need not be real time
  - Fundamentally a batch operation
  - Incremental updates may or may not be important
  - For the web, crawling is a challenge in itself

- The retrieval problem
  - Must have sub-second response time
  - For the web, only need relatively few results

www.studentsfocus.com
Indexing: Performance Analysis

- Fundamentally, a large sorting problem
  - Terms usually fit in memory
  - Postings usually don’t

- How is it done on a single machine?

- How can it be done with MapReduce?

- First, let’s characterize the problem size:
  - Size of vocabulary
  - Size of postings
Vocabulary Size: Heaps’ Law

\[ M = kT^b \]

- \( M \) is vocabulary size
- \( T \) is collection size (number of documents)
- \( k \) and \( b \) are constants

Typically, \( k \) is between 30 and 100, \( b \) is between 0.4 and 0.6

- Heaps’ Law: linear in log-log space
- Vocabulary size grows unbounded!
Heaps’ Law for RCV1

k = 44
b = 0.49

First 1,000,020 terms:
Predicted = 38,323
Actual = 38,365

Postings Size: Zipf’s Law

\[ cf_i = \frac{c}{i} \]

- Zipf’s Law: (also) linear in log-log space
  - Specific case of Power Law distributions
- In other words:
  - A few elements occur very frequently
  - Many elements occur very infrequently
Zipf’s Law for RCV1

Fit isn’t that good… but good enough!


www.studentsfocus.com

Manning, Raghavan, Schütze, Introduction to Information Retrieval (2008)
Power Laws are everywhere!

MapReduce: Index Construction

- Map over all documents
  - Emit term as key, (docno, tf) as value
  - Emit other information as necessary (e.g., term position)
- Sort/shuffle: group postings by term
- Reduce
  - Gather and sort the postings (e.g., by docno or tf)
  - Write postings to disk
- MapReduce does all the heavy lifting!

www.studentsfocus.com
## Inverted Indexing with MapReduce

### Map

<table>
<thead>
<tr>
<th></th>
<th>Doc 1 (one fish, two fish)</th>
<th>Doc 2 (red fish, blue fish)</th>
<th>Doc 3 (cat in the hat)</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>1 1</td>
<td>2 1</td>
<td>3 1</td>
</tr>
<tr>
<td>two</td>
<td>1 1</td>
<td>2 1</td>
<td>3 1</td>
</tr>
<tr>
<td>fish</td>
<td>1 2</td>
<td>2 2</td>
<td></td>
</tr>
</tbody>
</table>

### Shuffle and Sort: aggregate values by keys

### Reduce

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cat</td>
<td>3 1</td>
<td></td>
</tr>
<tr>
<td>fish</td>
<td>1 2</td>
<td>2 2</td>
</tr>
<tr>
<td>one</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>red</td>
<td>2 1</td>
<td></td>
</tr>
<tr>
<td>blue</td>
<td></td>
<td>2 1</td>
</tr>
<tr>
<td>hat</td>
<td></td>
<td>3 1</td>
</tr>
<tr>
<td>two</td>
<td></td>
<td>1 1</td>
</tr>
</tbody>
</table>

www.studentsfocus.com
Inverted Indexing: Pseudo-Code

1: class Mapper
2:     procedure MAP(docid n, doc d)
3:         H ← new AssociativeArray
4:         for all term t ∈ doc d do
5:             H{t} ← H{t} + 1
6:         for all term t ∈ H do
7:             EMIT(term t, posting ⟨n, H{t}⟩)

1: class Reducer
2:     procedure REDUCE(term t, postings [⟨a₁, f₁⟩, ⟨a₂, f₂⟩ . . . ])
3:         P ← new List
4:         for all posting ⟨a, f⟩ ∈ postings [⟨a₁, f₁⟩, ⟨a₂, f₂⟩ . . . ] do
5:             APPEND(P, ⟨a, f⟩)
6:         SORT(P)
7:         EMIT(term t, postings P)
Positional Indexes

Map

Doc 1
one fish, two fish

- one: 1 1 [1]
- two: 1 1 [3]
- fish: 1 2 [2,4]

Doc 2
red fish, blue fish

- red: 2 1 [1]
- blue: 2 1 [3]
- fish: 2 2 [2,4]

Doc 3
cat in the hat

- cat: 3 1 [1]
- hat: 3 1 [2]

Shuffle and Sort: aggregate values by keys

Reduce

- cat: 3 1 [1]
- fish: 1 2 [2,4] 2 2 [2,4]
- one: 1 1 [1]
- red: 2 1 [1]
- blue: 2 1 [3]
- hat: 3 1 [2]
- two: 1 1 [3]

www.studentsfocus.com
Inverted Indexing: Pseudo-Code

1: class Mapper
2:   procedure Map(docid n, doc d)
3:     H ← new AssociativeArray
4:     for all term t ∈ doc d do
5:         H{t} ← H{t} + 1
6:     for all term t ∈ H do
7:         Emit(term t, posting ⟨n, H{t}⟩)

1: class Reducer
2:   procedure Reduce(term t, postings [⟨a₁, f₁⟩, ⟨a₂, f₂⟩ . . . ])
3:     P ← new List
4:     for all posting ⟨a, f⟩ ∈ postings [⟨a₁, f₁⟩, ⟨a₂, f₂⟩ . . . ] do
5:         Append(P, ⟨a, f⟩)
6:     Sort(P)
7:     Emit(term t, postings P)

What’s the problem?
Scalability Bottleneck

- Initial implementation: terms as keys, postings as values
  - Reducers must buffer all postings associated with key (to sort)
  - What if we run out of memory to buffer postings?
- Uh oh!
### How is this different?

- Let the framework do the sorting
- Term frequency implicitly stored
- Directly write postings to disk!
Postings Encoding

Conceptually:

fish

In Practice:

- Don’t encode docnos, encode gaps (or $d$-gaps)
- But it’s not obvious that this save space...

www.studentsfocus.com
Overview of Index Compression

- Byte-aligned vs. bit-aligned
  - VarInt
  - Group VarInt
  - Simple-9

- Non-parameterized bit-aligned
  - Unary codes
  - $\gamma$ codes
  - $\delta$ codes

- Parameterized bit-aligned
  - Golomb codes (local Bernoulli model)

Want more detail? Read *Managing Gigabytes* by Witten, Moffat, and Bell!

www.studentsfocus.com
Unary Codes

- $x \geq 1$ is coded as $x-1$ one bits, followed by 1 zero bit
  - $3 = 110$
  - $4 = 1110$
- Great for small numbers… horrible for large numbers
  - Overly-biased for very small gaps

Watch out! Slightly different definitions in different textbooks

www.studentsfocus.com
γ codes

- $x \geq 1$ is coded in two parts: length and offset
  - Start with binary encoded, remove highest-order bit = offset
  - Length is number of binary digits, encoded in unary code
  - Concatenate length + offset codes

- Example: 9 in binary is 1001
  - Offset = 001
  - Length = 4, in unary code = 1110
  - γ code = 1110:001

- Analysis
  - Offset = ⌊log x⌋
  - Length = ⌊log x⌋ + 1
  - Total = 2 ⌊log x⌋ + 1

www.studentsfocus.com
δ codes

- Similar to γ codes, except that length is encoded in γ code
- Example: 9 in binary is 1001
  - Offset = 001
  - Length = 4, in γ code = 11000
  - δ code = 11000:001
- γ codes = more compact for smaller numbers
- δ codes = more compact for larger numbers
Golomb Codes

- \( x \geq 1 \), parameter \( b \):
  - \( q + 1 \) in unary, where \( q = \lfloor (x - 1) / b \rfloor \)
  - \( r \) in binary, where \( r = x - qb - 1 \), in \( \lfloor \log b \rfloor \) or \( \lceil \log b \rceil \) bits

- Example:
  - \( b = 3 \), \( r = 0, 1, 2 \) (0, 10, 11)
  - \( b = 6 \), \( r = 0, 1, 2, 3, 4, 5 \) (00, 01, 100, 101, 110, 111)
  - \( x = 9 \), \( b = 3 \): \( q = 2 \), \( r = 2 \), code = 110:11
  - \( x = 9 \), \( b = 6 \): \( q = 1 \), \( r = 2 \), code = 10:100

- Optimal \( b \approx 0.69 \) (N/df)
  - Different \( b \) for every term!

www.studentsfocus.com
## Comparison of Coding Schemes

<table>
<thead>
<tr>
<th>Unary</th>
<th>$\gamma$</th>
<th>$\delta$</th>
<th>Golomb $b=3$</th>
<th>Golomb $b=6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0:0</td>
<td>0:00</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>10:0</td>
<td>100:0</td>
<td>0:10</td>
</tr>
<tr>
<td>3</td>
<td>110</td>
<td>10:1</td>
<td>100:1</td>
<td>0:11</td>
</tr>
<tr>
<td>4</td>
<td>1110</td>
<td>110:00</td>
<td>101:00</td>
<td>10:0</td>
</tr>
<tr>
<td>5</td>
<td>11110</td>
<td>110:01</td>
<td>101:01</td>
<td>10:10</td>
</tr>
<tr>
<td>6</td>
<td>111110</td>
<td>110:10</td>
<td>101:10</td>
<td>10:11</td>
</tr>
<tr>
<td>7</td>
<td>1111110</td>
<td>110:11</td>
<td>101:11</td>
<td>110:0</td>
</tr>
<tr>
<td>8</td>
<td>11111110</td>
<td>1110:000</td>
<td>11000:000</td>
<td>110:10</td>
</tr>
<tr>
<td>9</td>
<td>111111110</td>
<td>1110:001</td>
<td>11000:001</td>
<td>110:11</td>
</tr>
<tr>
<td>10</td>
<td>1111111110</td>
<td>1110:010</td>
<td>11000:010</td>
<td>1110:0</td>
</tr>
</tbody>
</table>

Witten, Moffat, Bell, Managing Gigabytes (1999)
Index Compression: Performance

Comparison of Index Size (bits per pointer)

<table>
<thead>
<tr>
<th></th>
<th>Bible</th>
<th>TREC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary</td>
<td>262</td>
<td>1918</td>
</tr>
<tr>
<td>Binary</td>
<td>15</td>
<td>20</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>6.51</td>
<td>6.63</td>
</tr>
<tr>
<td>$\delta$</td>
<td>6.23</td>
<td>6.38</td>
</tr>
<tr>
<td>Golomb</td>
<td>6.09</td>
<td>5.84</td>
</tr>
</tbody>
</table>

Recommend best practice

Bible: King James version of the Bible; 31,101 verses (4.3 MB)
TREC: TREC disks 1+2; 741,856 docs (2070 MB)

www.studentsfocus.com
Reachability Query and Transitive Closure Representation

**The problem**: Given two vertices $u$ and $v$ in a directed graph $G$, is there a path from $u$ to $v$?

Directed Graph $\rightarrow$ DAG (directed acyclic graph) by coalescing the strongly connected components
Chicken and Egg?

But wait! How do we set the Golomb parameter $b$?

Recall: optimal $b \approx 0.69 \ (N/df)$

We need the $df$ to set $b$...

But we don’t know the $df$ until we’ve seen all postings!

Write directly to disk

www.studentsfocus.com
Getting the df

- In the mapper:
  - Emit “special” key-value pairs to keep track of \( df \)

- In the reducer:
  - Make sure “special” key-value pairs come first: process them to determine \( df \)

- Remember: proper partitioning!
Getting the df: Modified Mapper

**Doc 1**

one fish, two fish

Input document...

<table>
<thead>
<tr>
<th>(key)</th>
<th>(value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fish</td>
<td>1</td>
</tr>
<tr>
<td>one</td>
<td>1</td>
</tr>
<tr>
<td>two</td>
<td>1</td>
</tr>
</tbody>
</table>

Emit normal key-value pairs...

fish ★ [1]
one ★ [1]
two ★ [1]

Emit “special” key-value pairs to keep track of df...

www.studentsfocus.com
Getting the df: Modified Reducer

First, compute the df by summing contributions from all “special” key-value pair...

Compute Golomb parameter $b$...

Important: properly define sort order to make sure “special” key-value pairs come first!

Write postings directly to disk

www.studentsfocus.com

Where have we seen this before?
MapReduce it?

- The indexing problem
  - Scalability is paramount
  - Must be relatively fast, but need not be real time
  - Fundamentally a batch operation
  - Incremental updates may or may not be important
  - For the web, crawling is a challenge in itself

- The retrieval problem
  - Must have sub-second response time
  - For the web, only need relatively few results
Retrieval with MapReduce?

- MapReduce is fundamentally batch-oriented
  - Optimized for throughput, not latency
  - Startup of mappers and reducers is expensive
- MapReduce is not suitable for real-time queries!
  - Use separate infrastructure for retrieval…
Important Ideas

- Partitioning (for scalability)
- Replication (for redundancy)
- Caching (for speed)
- Routing (for load balancing)

The rest is just details!

www.studentsfocus.com
Term vs. Document Partitioning

Term Partitioning

Document Partitioning

www.studentsfocus.com
Katta Architecture
(Distributed Lucene)

- Hadoop cluster or single server
- Create index and copy to shared filesystem
- HDFS, NAS or shared local filesystem
- Master
- Secondary Master
- Zookeeper
- Slave
- Shard replication (plug-able policy)
- Multicast query
- Distributed ranking plug-able selection policy (custom load balancing)
- Command line management
- Java API

http://katta.sourceforge.net/

www.studentsfocus.com