Index Compression

Alta Vista

PageRank

Administrivia

- No class Tues 10/26
  - Instead go to today’s colloquium
  - Group Meetings

- Never-Ending Language Learning
  - Today 3:30pm EEB 105

Class Overview

- Information Extraction
- Data Advertising
- Security
- Cloud Computing
- Revisiting

Query processing
  - Indexing
  - IR - Ranking
  - Content Analysis
  - Crawling
  - Network Layer

Other Cool Stuff

Retrieval

Document-term matrix

\[
\begin{array}{cccccccccc}
\text{dt} & t_1 & t_2 & \ldots & t_i & \ldots & t_m & \text{nf} \\
\text{d_1} & w_{11} & w_{12} & \ldots & w_{1i} & \ldots & w_{1m} & l/d_1 \\
\text{d_2} & w_{21} & w_{22} & \ldots & w_{2i} & \ldots & w_{2m} & l/d_2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\
\text{d_i} & w_{i1} & w_{i2} & \ldots & w_{ii} & \ldots & w_{im} & l/d_i \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\
\text{d_m} & w_{m1} & w_{m2} & \ldots & w_{mi} & \ldots & w_{mm} & l/d_m \\
\end{array}
\]

\[w_{ij}\] is the weight of term \(t_i\) in document \(d_j\).
Most \(w_{ij}\)'s will be zero.

Review

- Vector Space Representation
  - Dot Product as Similarity Metric

- TF-IDF for Computing Weights
  - \(\text{wij} = f(i,j) \times \log(N/n_i)\)
  - Where \(q = \ldots \text{word} \ldots\)
  - \(N = |\text{docs}|\)
  - \(n_i = |\text{docs with word}_i|\)

- But How Process Efficiently?

Inverted Files for Multiple Documents

<table>
<thead>
<tr>
<th>LEXICON</th>
<th>OCCURRENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{WORD}</td>
<td>OCCUR</td>
</tr>
<tr>
<td>journalism</td>
<td>20</td>
</tr>
<tr>
<td>intense</td>
<td>2</td>
</tr>
<tr>
<td>index</td>
<td>1</td>
</tr>
<tr>
<td>internet</td>
<td>1</td>
</tr>
<tr>
<td>journalist</td>
<td>1</td>
</tr>
<tr>
<td>journal</td>
<td>10</td>
</tr>
<tr>
<td>journalists</td>
<td>1</td>
</tr>
<tr>
<td>journalism</td>
<td>1</td>
</tr>
</tbody>
</table>

"occurred" occurs 4 times in document 34, 3 times in document 44, 4 times in document 56...
Many Variations Possible
• Address space (flat, hierarchical)
  – Alta Vista uses flat approach
• Record term-position information
• Precalculate TF-IDF info
• Stored header, font & tag info
• Compression strategies

Compression
• What Should We Compress?
  – Repository
  – Lexicon
  – Inv Index
• What properties do we want?
  – Compression ratio
  – Compression speed
  – Decompression speed
  – Memory requirements
  – Pattern matching on compressed text
  – Random access

Inverted File Compression
Each inverted list has the form \( <f_1, d_1, f_2, d_2, \ldots, d_k > \)
A naïve representation results in a storage overhead of \( (f + n) \cdot \lceil \log N \rceil \)
This can also be stored as \( <f_1, d_1, d_2, \ldots, d_k > \)
Each difference is called a d-gap. Since \( \sum (d - gaps) \leq N \),
each pointer requires fewer than \( \lceil \log N \rceil \) bits.

A naïve representation results in a storage overhead of \( (f + n) \cdot \lceil \log N \rceil \)

Classifying d-gap Compression Methods:
• Global: each list compressed using same model
  – non-parameterized: probability distribution for d-gap sizes is predetermined.
  – parameterized: probability distribution is adjusted according to certain parameters of the collection.
• Local: model is adjusted according to some parameter, like the frequency of the term
• By definition, local methods are parameterized.

Conclusion
• Local methods best
  • Parameterized global models ~ non-parameterized
    – Pointers not scattered randomly in file
  • In practice, best index compression algorithm is:
    – Local Bernoulli method (using Golomb coding)
  • Compressed inverted indices usually faster smaller than
    – Signature files
    – Bitmaps

Local < Parameterized Global < Non-parameterized Global
Not by much
CSE 454 - Case Studies

Design of Alta Vista

Based on a talk by Mike Burrows

AltaVista: Inverted Files

- Map each word to list of locations where it occurs
- Words = null-terminated byte strings
- Locations = 64 bit unsigned ints
  - Layer above gives interpretation for location
    - URL
    - Index into text specifying word number
- Slides adapted from talk by Mike Burrows

Documents

- A document is a region of location space
  - Contiguous
  - No overlap
  - Densely allocated (first doc is location 1)
- All document structure encoded with words
  - enddoc at last location of document
  - begintitle, endtitle mark document title

Format of Inverted Files

- Words ordered lexicographically
- Each word followed by list of locations
- Common word prefixes are compressed
- Locations encoded as deltas
  - Stored in as few bytes as possible
  - 2 bytes is common
  - Sneaky assembly code for operations on inverted files
    - Pack deltas into aligned 64 bit word
    - First byte contains continuation bits
    - Table lookup on byte => no branch instructions, no mispredicts
    - 35 parallelized instructions/ 64 bit word = 10 cycles/word
- Index ~ 10% of text size

Index Stream Readers (ISRs)

- Interface for
  - Reading result of query
  - Return ascending sequence of locations
  - Implemented using lazy evaluation
- Methods
  - loc(ISR) return current location
  - next(ISR) advance to next location
  - seek(ISR, X) advance to next loc after X
  - prev(ISR) return previous location

Processing Simple Queries

- User searches for “mp3”
  - Open ISR on “mp3”
    - Uses hash table to avoid scanning entire file
  - Next(), next(), next() returns locations containing the word
Combining ISRs

- And: Compare locs on two streams
- Or: Merges two or more ISRs
- Not: Returns locations not in ISR (lazily)

What About File Boundaries?

ISR Constraint Solver

- Inputs:
  - Set of ISRs: A, B, ...
  - Set of Constraints
- Constraint Types
  - $\text{loc}(A) \leq \text{loc}(B) + K$
  - $\text{prev}(A) \leq \text{loc}(B) + K$
  - $\text{loc}(A) \leq \text{prev}(B) + K$
  - $\text{prev}(A) \leq \text{prev}(B) + K$

  For example, phrase “a b”
  - $\text{loc}(A) \leq \text{loc}(B)$, $\text{loc}(B) \leq \text{loc}(A) + 1$

Two words on one page

- Let $E$ be ISR for word enddoc
- Constraints for conjunction $a$ AND $b$
  - $\text{prev}(E) \leq \text{loc}(A)$
  - $\text{loc}(A) \leq \text{loc}(E)$
  - $\text{prev}(E) \leq \text{loc}(B)$
  - $\text{loc}(B) \leq \text{loc}(E)$

Advanced Search

- Field query: $a$ in Title of page
- Let BT, ET be ISRP of words begin{title}, end{title}
- Constraints:
  - $\text{loc}(BT) \leq \text{loc}(A)$
  - $\text{loc}(A) \leq \text{loc}(ET)$
  - $\text{prev}(ET) \leq \text{loc}(BT)$

Implementing the Solver

Constraint Types

- $\text{loc}(A) \leq \text{loc}(B) + K$
- $\text{prev}(A) \leq \text{loc}(B) + K$
- $\text{loc}(A) \leq \text{prev}(B) + K$
- $\text{prev}(A) \leq \text{prev}(B) + K$
**Remember: Index Stream Readers**

- **Methods**
  - loc(ISR) return current location
  - next(ISR) advance to next location
  - seek(ISR, X) advance to next loc after X
  - prev(ISR) return previous location

---

**Solver Algorithm**

```plaintext
while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())

- To satisfy: \( \text{loc}(A) \leq \text{loc}(B) + K \)
  - Execute: seek(B, loc(A) - K)

- To satisfy: \( \text{prev}(A) \leq \text{loc}(B) + K \)
  - Execute: seek(B, prev(A) - K)
```

---

**Heuristic:** Which choice advances a stream the furthest?
Update

- Can’t insert in the middle of an inverted file
- Must rewrite the entire file
  - Naïve approach: need space for two copies
  - Slow since file is huge
- Split data along two dimensions
  - Buckets solve disk space problem
  - Tiers alleviate small update problem

Buckets & Tiers

- Each word is hashed to a bucket
- Add new documents by adding a new tier
  - Periodically merge tiers, bucket by bucket

What if Word Removed from Doc?

- Delete documents by adding deleted word
- Expunge deletions when merging tier 1

Scaling

- How handle huge traffic?
  - AltaVista Search ranked #16
  - 10,674,000 unique visitors (Dec’99)
- Scale across N hosts
  1. Ubiquitous index. Query one host
  2. Split N ways. Query all, merge results
  3. Ubiquitous index. Host handles subrange of locations. Query all, merge results
  4. Hybrids

AltaVista Structure

- Front ends
  - Alpha workstations
- Back ends
  - 4-10 CPU Alpha servers
    - 8GB RAM, 150GB disk
  - Organized in groups of 4-10 machines
  - Each with 1/Nth of index