CSE 454

Index Compression
Alta Vista
PageRank
Review

- **Vector Space Representation**
  - Dot Product as Similarity Metric

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

- **But How Process Efficiently?**
### Retrieval

#### Document-term matrix

<table>
<thead>
<tr>
<th></th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>...</th>
<th>$t_j$</th>
<th>...</th>
<th>$t_m$</th>
<th>nf</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>$w_{11}$</td>
<td>$w_{12}$</td>
<td>...</td>
<td>$w_{1j}$</td>
<td>...</td>
<td>$w_{1m}$</td>
<td>$1/</td>
</tr>
<tr>
<td>$d_2$</td>
<td>$w_{21}$</td>
<td>$w_{22}$</td>
<td>...</td>
<td>$w_{2j}$</td>
<td>...</td>
<td>$w_{2m}$</td>
<td>$1/</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$d_i$</td>
<td>$w_{i1}$</td>
<td>$w_{i2}$</td>
<td>...</td>
<td>$w_{ij}$</td>
<td>...</td>
<td>$w_{im}$</td>
<td>$1/</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$d_n$</td>
<td>$w_{n1}$</td>
<td>$w_{n2}$</td>
<td>...</td>
<td>$w_{nj}$</td>
<td>...</td>
<td>$w_{nm}$</td>
<td>$1/</td>
</tr>
</tbody>
</table>

$w_{ij}$ is the weight of term $t_j$ in document $d_i$

Most $w_{ij}$’s will be zero.
Inverted Files for Multiple Documents

<table>
<thead>
<tr>
<th>WORD</th>
<th>NDOCS</th>
<th>PTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>jezebel</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>jezer</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>jezerit</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jeziah</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jeziel</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jeziah</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jezoar</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jezrahiah</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>jezreel</td>
<td>39</td>
<td></td>
</tr>
</tbody>
</table>

**LEXICON**

```
<table>
<thead>
<tr>
<th>DOCID</th>
<th>OCCUR</th>
<th>POS 1</th>
<th>POS 2</th>
<th>. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>6</td>
<td>1</td>
<td>118</td>
<td>2087</td>
</tr>
<tr>
<td>44</td>
<td>3</td>
<td>215</td>
<td>2291</td>
<td>3010</td>
</tr>
<tr>
<td>56</td>
<td>4</td>
<td>5</td>
<td>22</td>
<td>134</td>
</tr>
<tr>
<td>566</td>
<td>3</td>
<td>203</td>
<td>245</td>
<td>287</td>
</tr>
<tr>
<td>67</td>
<td>1</td>
<td>132</td>
<td></td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>4</td>
<td>322</td>
<td>354</td>
<td>381</td>
</tr>
<tr>
<td>232</td>
<td>6</td>
<td>15</td>
<td>195</td>
<td>248</td>
</tr>
<tr>
<td>677</td>
<td>1</td>
<td>481</td>
<td></td>
<td></td>
</tr>
<tr>
<td>713</td>
<td>3</td>
<td>42</td>
<td>312</td>
<td>802</td>
</tr>
</tbody>
</table>
```

“jezebel” occurs 6 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 \(< f_t ; d_1 , d_2 , d_3 , ... , d_{f_t} >\)

A naïve representation results in a storage overhead of \((f + n) * [\log N]\)

This can also be stored as \(< f_t ; d_1 , d_2 - d_1 , ... , d_{f_t} - d_{f_t-1} >\)

Each difference is called a d-gap. Since \(\sum (d - gaps) \leq N\),

each pointer requires fewer than \([\log N]\) bits.

**Trick is encoding …. since worst case ….**

*Assume d-gap representation for the rest of the talk, unless stated otherwise*

Slides adapted from Tapas Kanungo and David Mount, Univ Maryland
Text Compression

Two classes of text compression methods

• Symbolwise (or statistical) methods
  – Estimate probabilities of symbols - modeling step
  – Code one symbol at a time - coding step
  – Use shorter code for the most likely symbol
  – Usually based on either arithmetic or Huffman coding

• Dictionary methods
  – Replace fragments of text with a single code word
  – Typically an index to an entry in the dictionary.
    • eg: Ziv-Lempel coding: replaces strings of characters with a pointer to
      a previous occurrence of the string.
  – No probability estimates needed

Symbolwise methods are more suited for coding d-gaps
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 instructs, 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)

![Diagram showing combining ISRs with examples]
What About File Boundaries?

- fixx
- enddoc
- mp3
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`
  - `prev(E) ≤ loc(A)`
  - `loc(A) ≤ loc(E)`
  - `prev(E) ≤ loc(B)`
  - `loc(B) ≤ loc(E)`
Advanced Search

• **Query:**  a in Title of page
• **Let BT, ET be ISRP of words** begintitle, endtitle
• **Constraints:**
  – $\text{loc}(BT) \leq \text{loc}(A)$
  – $\text{loc}(A) \leq \text{loc}(ET)$
Advanced Search

- **Query:** a in Title of page
- **Let BT, ET be ISRP of words** begintitle, endtitle
- **Constraints:**
  - $\text{loc}(BT) \leq \text{loc}(A)$
  - $\text{loc}(A) \leq \text{loc}(ET)$
  - prev(ET) $\leq$ loc(BT) \( X \)
Advanced Search

- **Field query:** a in Title of page
- **Let BT, ET be ISRP of words** begintitle, endtitle
- **Constraints:**
  - $\text{loc}(\text{BT}) \leq \text{loc}(\text{A})$
  - $\text{loc}(\text{A}) \leq \text{loc}(\text{ET})$
  - $\text{prev}(\text{ET}) \leq \text{loc}(\text{BT})$
Neat, Huh?

• Are we done?
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

while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())

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

while (unsatisfied_constraints)
satisfy_constraint(choose_unsat_constraint())

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

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

while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())

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

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

• To satisfy: \( \text{loc}(A) \leq \text{prev}(B) + K \)
Solver Algorithm

while (unsatisfied_constraints)
satisfy_constraint(choose_unsat_constraint())

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

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

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

```
  b a b a a b a b
  \text{P}_B \text{B} \text{A}
  \uparrow K \uparrow \uparrow K
```
Solver Algorithm

while (unsatisfied_constraints)
    satisfy_constraint(choose_unsat_constraint())

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

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

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

\[\begin{array}{cccccccc}
  & & & & & & & \\
  & b & a & b & a & & & \\
  & b & a & b & a & & & \\
  & & & & & & & \\
  \text{P}_B & \uparrow K & \uparrow & \uparrow \\
  \text{A} & \text{B} & \\
\end{array}\]
Solver Algorithm

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)

• **To satisfy:** $\text{loc}(A) \leq \text{prev}(B) + K$
  – Execute: seek(B, loc(A) - K),
  – next(B)

• **To satisfy:** $\text{prev}(A) \leq \text{prev}(B) + K$
  – Execute: seek(B, prev(A) - K)
  – next(B)
Solver Algorithm

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)

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

• To satisfy: \( \text{prev}(A) \leq \text{prev}(B) + K \)
  – Execute: seek(B, prev(A) - K)
  – next(B)

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

Hash bucket(s) for word a
Hash bucket(s) for word b
Hash bucket(s) for word zebra
What if Word Removed from Doc?

- **Delete documents by adding** *deleted* word

- **Expunge deletions when merging tier 1**

![](diagram.png)
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 $\frac{1}{N}$th of index