Efficiency - Compression

(COSC 488)

Nazli Goharian

nazli@cs.georgetown.edu

Efficiency Techniques

- Indexing
  - Compression
    - Index Pruning (Top Doc)
    - Efficient Query Processing
      - Query thresholding
      - Partial result set processing
Facts

• Index (specially position index) may be similar size or generally larger than collection.
• Stop words removal eliminates about half the size of an inverted index. “the” occurs in 7 percent of English text.
• Half of terms occur only once (hapax legomena), so they only have one entry in their posting list
• Some terms have very long posting lists.

• Approaches:
  – Stop word removal, stemming, case folding (lossy)
  – Loss-less Compression

Sample Collection/Index Size before & after Compression
(from: Information Retrieval, Buttcher, Clarke, Cormack)

<table>
<thead>
<tr>
<th>Collection</th>
<th>Collection Uncompressed</th>
<th>Collection Compressed (gzip)</th>
<th>Index Uncompressed</th>
<th>Index Compressed (vByte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shakespeare TREC4-5</td>
<td>7.5 MB 1904.5 MB 425.8 GB</td>
<td>2.0 MB 582.9 MB 79.9 GB</td>
<td>10.5 MB 2331.1 MB 328.3 GB</td>
<td>2.7 MB 533.0 MB 62.1 GB</td>
</tr>
<tr>
<td>Gov2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compression: Goal

- Reducing the storage requirements
- Reducing I/O
- Storing more data in memory cache, and expedite query processing
- Efficiency of decompression algorithm is important!

Things to Compress

- **Lexicon**
  - Not compressed, if fits in memory
  - Compressed, if does not fit in the memory to support query throughput
- **Posting List**
  - Term Frequencies
  - Document Identifiers
  - Positions
Lexicon Compression

- Terms are in lexicographical order, sharing a common prefix. To prevent storing duplicates, use **Front coding** (*generally saves ~40%), as*

  \[(\text{prefix length}, \text{suffix length}, \text{suffix})\]

  \(<"\text{book"},(4,3,"\text{ing"}),(7,1,"\text{s"})><"\text{book"},(4,3,"\text{let"})…>\)

- Storing terms in lexicon as a string with pointers indicating end of a term and start of the next term.

- Hashing on terms

Delta (gaps) Encoding

- Change the numbers to smaller numbers, thus, fewer bits!
- More common terms -> larger PL-> smaller gaps-> smaller numbers
- Applied to posting lists
  - Term: \(<d_1,tf_1, \{\text{positions}\}, (d_2,tf_2), \{\text{positions}\}, ... (d_n,tf_n, \{\text{positions}\}>\)
- Documents are ordered, so each \(d_i\) is replaced by the interval difference (\(d\)-gaps), namely, \(d_i - d_{i-1}\)
- Smaller \(d\)-gaps for more common terms
- Index is reduced to ~15% of database size.
- Generally is applied first and then the gaps are further compressed.
Compression Techniques of Inverted Indices

• Fixed Length
  – Byte Aligned
  – ….

• Variable Length
  – Elias Encoding (γ), a family of universal codes
  – …

Byte-Aligned Compression

• Done within byte boundaries to improve Run-time at slight cost to compression ratio.

• Each number is represented by fixed number of bytes, from which 2 bits are length indicators.

• ~15-20% of uncompressed inverted index, when stop words are used.
Byte-Aligned Compression

• Algorithm:
  • Take doc id differences \((d\text{-}gaps)\)
  • Identify number of bytes needed for each \(d\text{-}gap\).
  • Write length indicator for each \(d\text{-}gap\) in preceding 2 bits.
  • Write the binary representation of \(d\text{-}gaps\).

Byte-Aligned Compression

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 63</td>
<td>00xxxxxx</td>
<td></td>
</tr>
<tr>
<td>64 - (16K-1)</td>
<td>01xxxxxx xxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>16K - (4M-1)</td>
<td>10xxxxxx xxxxxxxx xxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>4M - (1G-1)</td>
<td>11xxxxxx xxxxxxxx xxxxxxxx xxxxxxxx</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0000000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0000001</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>0011111</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>01000000 01000000</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>01000000 01000001</td>
<td></td>
</tr>
</tbody>
</table>

• The hope here is that the document distance between posting list nodes will be small.
Gamma (Elias) Encoding ($\gamma$)

<table>
<thead>
<tr>
<th>$X$</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>10 0</td>
</tr>
<tr>
<td>3</td>
<td>10 1</td>
</tr>
<tr>
<td>4</td>
<td>110 00</td>
</tr>
<tr>
<td>5</td>
<td>110 01</td>
</tr>
<tr>
<td>6</td>
<td>110 10</td>
</tr>
<tr>
<td>7</td>
<td>110 11</td>
</tr>
<tr>
<td>8</td>
<td>1110 00</td>
</tr>
<tr>
<td>63</td>
<td>111110 11111</td>
</tr>
</tbody>
</table>

To represent value $X$:
- $\lfloor \log_2 X \rfloor$ ones representing the highest power of 2 not exceeding $X$.
- A 0 marker.
- $\lfloor \log_2 X \rfloor$ bits representing the remainder $X - 2^\lfloor \log_2 X \rfloor$ in binary.
- Uses $2^\lfloor \log_2 X \rfloor + 1$ bits to represent value $X$. The smaller the integer, the fewer the bits used to represent the value. Most $\gamma$'s are relatively small.

Gamma (Elias) Encoding ($\gamma$) Example

$X = 22$

$\lfloor \log_2 22 \rfloor = 4 \quad 2^4 \leq x < 2^5$

4 is highest power of 2 not exceeding 22 => 4 bits unary: 1111

$x - 2^\lfloor \log_2 22 \rfloor = 22 - 2^4 = 6$

=> 4 bits binary to represent the remaining number 6: 0110

1111 0 0110

4 bits unary for 16 0 marker 4 bits binary for 6
- Decompression is in one pass.
Reordering Documents Prior to Indexing

- Reduce doc id gap for better compression
- Similar documents contain similar terms
- Thus, find similar documents and process in that order
  \( d_3, d_{50}, d_{200} \) will be \( d_1, d_2, d_3 \)
- Methods:
  - Clustering
  - URL info (from same Web server, same directory, …)
  - …..

Compression Summary

- Pro
  - Reducing the storage requirements of inverted index
  - Reducing I/O for querying the inverted index
  - Reducing disk seek time
  - Store more data in memory cache, and expedite processing

- Con
  - Takes longer to build the inverted index.
  - Software becomes much more complicated.
  - Uncompress required at query time – note that this time is usually offset by dramatic reduction in I/O.