INDEX COMPRESSION
INVERTED INDEX

Inverted index consists of two principal components
- Dictionary
- Posting lists
**Uncompressed Inverted Index**

Uncompressed inverted index for a given document can be very large.
COMPRESSED INVERTED INDEX

Advantages of compressed inverted index

- Less storage
- Fast query retrieval time
- can index large collections
GENERAL PURPOSE DATA COMPRESSION

Compression algorithm takes the data and converts into another data which requires fewer bits to store and transfer.

- Encoder: converts original data A to B
  \[ \text{sizeof}(A) > \text{sizeof}(B) \]
- Decoder: takes B and converts into C
  - lossy: C can be approximation of A (JPEG, MP3)
  - lossless: C is exact copy of A
**Symbolwise Data Compression**

- Data compression techniques can treat the information (M) as a sequence of symbols.
- Not all symbols in M appear with the same frequency.
- Symbols can depend on the previous symbols, e.g., ‘q’ and ‘u’.
MODELING AND CODING

Symbolwise compression methods work in two phases

- **Modeling**: a probability distribution M is computed that maps symbols to their probability of occurrence
- **Coding**: symbols in the message M are reencoded according to a code C

Ex: Huffman algorithm calculates probability of the frequency of characters and using that to find the code for each character
**BITWISE CODING**

- Prefix property: no code word is an initial substring of any other code word
- a-0, b-11, c-100, d-101
HUFFMAN CODING ALGORITHM

- Generate probability of occurrences of each character
- Create each individual node with the probability
- Find two minimum nodes and combined them into one node with sum of their probabilities
- Two minimum nodes become left and right nodes.
- Repeat it until ends with a single node
HUFFMAN CODING ALGORITHM

Chapter 6  Index Compression

\[
\begin{align*}
\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} & : 1.00 \\
\{\sigma_1, \sigma_3\} & : 0.49 \\
\{\sigma_2, \sigma_4, \sigma_5\} & : 0.51 \\
\{\sigma_1\} & : 0.18 \\
\{\sigma_3\} & : 0.31 \\
\{\sigma_4\} & : 0.34 \\
\{\sigma_2, \sigma_5\} & : 0.17 \\
\{\sigma_2\} & : 0.11 \\
\{\sigma_5\} & : 0.06
\end{align*}
\]
ARITHMETIC CODING

- improve upon the Huffman code for single symbols by taking pairs of symbols and making the Huffman code for them
- Ex: “aa”, “ab”, “ba”, “bb”,…

![Diagram showing arithmetic coding with examples of codes for different lengths of messages:](image-url)
CONTD…

- Find the sub intervals of the sequences of symbols and then find its binary representation and encode the message using those.
- Ex: aaa => [0,0.512) => 0, aab => [0.512, 0.64) => 0.10011, aba => [0.64,0.75) => 0.11
- Decodes as soon as it sees 0 to “aaa” .10 to “aab” and so on…
Posting Lists

- Majority of data in an inverted index are postings data.
- A posting list consists of a sequence of integers giving the doc id's of the document that contained a particular word.
  \[ L = (3, 7, 11, 23, 29, 37, 41, \ldots) \]
- List can be very large and each element occurs single time
- Standard compression methods like Huffman coding is not feasible
COMPRESSING POSTING LISTS: \( \Delta \)-VALUES

Transformed into an equivalent sequence of difference between consecutive elements(\( \Delta \)-values)
\( \Delta (L) = (3, 4, 4, 12, 6, 8, 4, \ldots) \)

- Elements are smaller and can be encoded using fewer bits.
- Elements can occur multiple times
- Nonparametric Gap Compression: does not consider the actual \( \Delta \)-gap distribution (\( \gamma \) Codes)
- Parametric Gap Compression: conducts an analysis of some statistical properties of the list to be compressed (Golomb/Rice codes)
γ Codes

- Offsets in a posting list: (100000, 100005, 100011, ...)
- Gap compression: (1: 5, 6...)
- To compress these small numbers: We write (number of bits -1) we want in unary with 0's, followed by a 1, followed by the number in binary.
- Unary representation for 1=>0, 2=>00, 3=>000....

<table>
<thead>
<tr>
<th>k</th>
<th>selector(k)</th>
<th>body(k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>7</td>
<td>001</td>
<td>111</td>
</tr>
<tr>
<td>16</td>
<td>00001</td>
<td>10000</td>
</tr>
</tbody>
</table>
High-order bit of the binary code for the number is redundant given that we known the length of the number, so we can drop this bit to get the actual encoding.

<table>
<thead>
<tr>
<th>$k$</th>
<th>selector($k$)</th>
<th>body($k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>5</td>
<td>01 1</td>
<td>01</td>
</tr>
<tr>
<td>7</td>
<td>01 1</td>
<td>11</td>
</tr>
<tr>
<td>16</td>
<td>001 01</td>
<td>0000</td>
</tr>
</tbody>
</table>
GOLOMB/RICE CODES

- Compress a list whose $\Delta$-values follow a geometric distribution $\Pr[\Delta=k]=(1-p)^{k-1}p$.
- Arbitrary Modulus $M$ (Golomb)
- $M$ is a power of 2 (Rice)
Determine an appropriate modulus

- Split each value into two components:
  - quotient q(k)
  - remainder r(k)

Where $q(k) = \lfloor k - 1/M \rfloor$, $r(k) = (k - 1) \mod M$

<table>
<thead>
<tr>
<th>Integer</th>
<th>Golomb Codes</th>
<th>Rice Codes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M = 3$</td>
<td>$M = 6$</td>
</tr>
<tr>
<td>1</td>
<td>1 0</td>
<td>1 00</td>
</tr>
<tr>
<td>2</td>
<td>1 10</td>
<td>1 01</td>
</tr>
<tr>
<td>3</td>
<td>1 11</td>
<td>1 100</td>
</tr>
<tr>
<td>4</td>
<td>01 0</td>
<td>1 011</td>
</tr>
<tr>
<td>5</td>
<td>01 10</td>
<td>1 110</td>
</tr>
<tr>
<td>6</td>
<td>01 11</td>
<td>1 111</td>
</tr>
<tr>
<td>7</td>
<td>001 0</td>
<td>01 00</td>
</tr>
<tr>
<td>8</td>
<td>001 10</td>
<td>01 01</td>
</tr>
<tr>
<td>9</td>
<td>001 11</td>
<td>01 100</td>
</tr>
<tr>
<td>31</td>
<td>0000000000001 0</td>
<td>000001 00</td>
</tr>
</tbody>
</table>
BYTE-ALIGNED CODES

- vByte(variable-byte coding): Splits the binary representation of each Δvalue into 7-bit chunk + 1 bit continuation flag
  
  \[ L = (1624, 1650, 1876, 1972, \ldots) \]
  \[ \Delta(L) = (1624, 26, 226, 96, 384, \ldots) \]

0 1011000 0 0001100 1 1100010 0 0000001 0 1100000 1 0000000 0 0000011…

- 0 at the beginning of the chunk indicates the end of the current code word. (88+12 * 2^7 =1624)
WORD-ALIGNED CODES(SIMPLE-9)

- Inspects $\Delta$ values in a postings sequence and insert as many as possible into a 32-bit.
- Reserve 4 bits for selector

0001 000111001011000 00000000011001 0010 0111000001 0010111111 1011111111 U

Table 6.6: Word-aligned postings compression with Simple-9. After reserving 4 out of 32 bits for the selector value, there are 9 possible ways of dividing the remaining 28 bits into equal-size chunks.
REFERENCES