Lecture 6: Corpus and index compression

Matthias Petri

Fri 20/3/15
What we’ll learn today

Efficient storage and access of

- Vocabulary
- Posting lists
- Documents
Index compression

Benefits of index compression:

▶ Reduce storage requirements
▶ Keep larger parts of the index in memory
▶ Faster query processing

Example: A state-of-the-art inverted index of 25 million websites (420GB) requires only 5GB (1.2%) and can answer queries in $\approx 10$ milliseconds.
Compressibility is bounded by the information content of a data set

Information content of a text $T$ is characterized by its Entropy $H$:

$$H(T) = - \sum_{s \in \Sigma} \frac{f_s}{n} \log_2 \frac{f_s}{n}$$

where $f_s$ is the frequency of symbol $s$ in $T$ and $n$ is the length of $T$.

For example, $H(\text{abracadabra}) = 2.040373$ bits with $n = 11$, $f_a = 5$, $f_b = 2$, $f_c = 1$, $f_d = 1$, $f_r = 2$.

Intuition: Spend less bits on items that occur often.
What we’ll learn today

Efficient storage and access of

- Vocabulary
- Posting lists
- Documents
Vocabulary Storage

Vocabulary Operations

- Lookup word
- Retrieve postings list and other metadata.

Standard Data Structures

- Hashtable
- Binary Search Tree
- B-Tree
Vocabulary Compression

Support vocabulary operations while reducing space usage requirements.

Simple scheme

- Store metadata and pointers in an array
- Concatenate all tokens into one string
- Store pointers into the string and perform binary search

Optimizations

- Utilize common prefixes of adjacent tokens (Frontcoding)
- Compress string using entropy encoders such as Huffman
Vocabulary Compression - Example

$T = \ldots$automateautomotiveautopilotautopsyautosautum$\ldots$
Vocabulary Compression - Example

\[
T = \ldots \text{automate} \text{automotive} \text{autopilot} \text{autopsy} \text{autos} \text{autum} \ldots
\]

- \( f = 8, \text{pl} \rightarrow s = 123 \)
- \( f = 4, \text{pl} \rightarrow s = 140 \)
- \( f = 12, \text{pl} \rightarrow s = 149 \)
- \( f = 12, \text{pl} \rightarrow s = 149 \)
Vocabulary Compression - Example

\[ f = 8, \text{pl} \rightarrow s = 123 \]

\[ f = 4, \text{pl} \rightarrow \]

\[ f = 12, \text{pl} \rightarrow s = 140 \]

\[ f = 12, \text{pl} \rightarrow \]

\[ f = 12, \text{pl} \rightarrow s = 149 \]

\[ f = 12, \text{pl} \rightarrow \]

\[ T = \ldots \text{autom}\$ate\#otiveautop\$ilot\#syaut\$os\#um\ldots \]
Vocabulary Compression - Effectiveness

- Simple schemes can reduce vocabulary size by more than 50%
- Can retrieve entries in microseconds
- Cost of accessing the vocabulary negligible compared to list processing costs
- Today vocabulary always in-memory. Previously stored partially on disk
What we’ll learn today

Efficient storage and access of

- Vocabulary
- Posting lists
- Documents
Posting list Compression - Requirements

- Minimize storage costs
- Fast sequential access
- Support \( \text{GEQ}(x) \) operation: Return the smallest item in the list that is greater or equal to \( x \)

Sample collection

| Documents | 24,622,347 |
| Terms     | 35,636,425 |
| Postings  | 5,742,630,292 |
| Uncompressed Storage Cost\(^1\) | 16.71 GB |

\(^1\)Only document identifier, no frequencies or positions
Posting list Compression - Concepts

- Postings list corresponds to an increasing sequence of integers

- Each integer can be in $[1, N]$ requiring $\log_2(N)$ bits

- Idea: Gaps between two adjacent integers can be much smaller

<table>
<thead>
<tr>
<th></th>
<th>ids:</th>
<th>25</th>
<th>26</th>
<th>29</th>
<th>...</th>
<th>12345</th>
<th>12347</th>
<th>12349</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>gaps:</td>
<td>25</td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>ids:</td>
<td>5123</td>
<td>5234</td>
<td>5454</td>
<td>5591</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gaps:</td>
<td>5123</td>
<td>1</td>
<td>220</td>
<td>137</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>house</td>
<td>ids:</td>
<td>251235</td>
<td>251239</td>
<td>251239</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gaps:</td>
<td>251235</td>
<td>4</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Variable Byte Compression

Idea
Use variable number of bytes to represent integers. Each byte contains 7 bits “payload” and one continuation bit.

Examples

<table>
<thead>
<tr>
<th>Number</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>824</td>
<td>00000110 10111000</td>
</tr>
<tr>
<td>5</td>
<td>10000101</td>
</tr>
</tbody>
</table>

Storage Cost

<table>
<thead>
<tr>
<th>Number Range</th>
<th>Number of Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 127</td>
<td>1</td>
</tr>
<tr>
<td>128 – 16383</td>
<td>2</td>
</tr>
<tr>
<td>16384 – 2097151</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Variable Byte Compression - Algorithm

Encoding

1: function $\text{Encode}(x)$  
2: \hspace{1em} while $x \geq 128$ do  
3: \hspace{2em} \text{WRITE}(x \mod 128)  
4: \hspace{2em} $x = x \div 128$  
5: \hspace{1em} end while  
6: \hspace{1em} \text{WRITE}(x + 128)  
7: end function  

Decoding

1: function $\text{Decode}(\text{bytes})$  
2: \hspace{1em} $x = 0$  
3: \hspace{2em} $y = \text{READBYTE}(\text{bytes})$  
4: \hspace{2em} while $y < 128$ do  
5: \hspace{3em} $x = 128 \times x + y$  
6: \hspace{3em} $y = \text{READBYTE}(\text{bytes})$  
7: \hspace{2em} end while  
8: \hspace{1em} $x = 128 \times x + (y - 128)$  
9: \hspace{1em} return $x$  
10: end function
OptPForDelta Compression

Idea

Group $k$ gaps and encode using fixed number of bits. Encode numbers $> 2^b$ separately as an exception. Pick $b$ “optimally” for each block so there are $\approx 10\%$ exceptions.

Example $k = 8$

$[1 \ 4 \ 7 \ 2 \ 4 \ 5 \ 123 \ 6]$  $[1 \ 4 \ 5 \ 232 \ 523 \ 7 \ 2 \ 3]$  $[3 \ 4 \ 755 \ 15 \ 12 \ 1 \ 8 \ 4]$  

$b = 3$  $b = 8$  $b = 4$

Encode $[1 \ 4 \ 7 \ 2 \ 4 \ 5 \ 123 \ 6]$ as:

$b = 3, \ #e = 1, \ epos = [6]$  

header  \  content  \  exceptions

16 bit  \  21 bit  \  8 bit
### Postings List Decompression Speeds/Space Usage

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Space [bits/int]</th>
<th>Speed [Million Integers/sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uncompressed</td>
<td>32</td>
<td>≈ 5400</td>
</tr>
<tr>
<td>Variable Byte</td>
<td>8.7</td>
<td>≈ 680</td>
</tr>
<tr>
<td>OptPForDelta</td>
<td>4.7</td>
<td>≈ 710</td>
</tr>
<tr>
<td>Simple-8b</td>
<td>4.8</td>
<td>≈ 780</td>
</tr>
<tr>
<td>SIMD-BP128</td>
<td>11</td>
<td>≈ 2300</td>
</tr>
</tbody>
</table>

...
Postings List Compression - Optimizations

- Commonly lists are split into blocks of 128 integers
- Choose optimal compression for each block
- Often, long lists (“the”) are represented more efficiently using bitvectors of size $N$
- State-of-the-art implementations use SIMD Instructions and bit-parallelism to increase decoding speed
Postings List Compression - Summary

- Compress increasing integer sequences
- Support iterating and searching the compressed sequence
- Store gaps between adjacent numbers
- Different compression schemes provide different time-space tradeoffs
What we’ll learn today

Efficient storage and access of

- Vocabulary
- Posting lists
- Documents
Document Compression

Storing documents is necessary to for example generate snippets of documents.

Idea
Sample text from all documents into a dictionary and compress all documents relative to the dictionary.

Approach

- Sample strings from all documents at random.
- Combine samples into a dictionary string.
- Compress documents by finding ("factorizing") substring matches in the dictionary.
- Achieves excellent compression ($\approx 10\%$) and fast document retrieval time.
Document Compression - Example

\[ D_1: \text{CDABCCBDABCC} \]
\[ D_2: \text{DADAXCBB} \]
\[ D_3: \text{DBCCDBABCC} \]

\[ \text{Collection: CDABCCBDABCCDADAXCBBDBCCDBABCC} \]

\[ \text{Dictionary: CDABCCBDB} \]

\[ D_1: \text{CDABCCBDABCC} \]
\[ D_2: \text{DADAXCBB} \]
\[ D_3: \text{DBCCDBABCC} \]

\[ F_1: \{(0,3) (3,1) (3,5) (2,4)\} \]
\[ F_2: \{(1,2) (1,2) (X',0) (5,2) (3,1)\} \]
\[ F_3: \{(1,1) (3,3) (7,2) (2,3)\} \]
Looking back and forward

Back

- Vocabulary compression
- Postings lists compression techniques
- Document storage and compression

Forward

- ???
Further Reading

- Manning, Christopher D; Raghavan, Prabhakar; Schtze, Hinrich; Introduction to information retrieval, Cambridge University Press 2008. (Chapter 5)

