Indexing

Compression and QP
Posting list Layout

Data blocks, say of size 64KB, as basic unit for list caching

- Posting list storage is implemented as
  - list of document identifiers (did)
  - list of scores, positions
- Many chunks are skipped over, but very few blocks are
- Also, may prefetch the next, say 2MB of index data from disk

List chunks, say of 128 postings, as basic unit of decompression
Posting list Compression

- In a collection of $2^{12}$ documents
- Minimum number of bits required?
- What is the uncompressed size for this list?
- Variable byte encoding:
  - **7-bit encoding**: Use 7 bits for every byte and a continuation bit

1220 = 00000000 00000100 11000100

= 00001001 11000100

Doc. id list

<table>
<thead>
<tr>
<th>Doc. id list</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>1220</td>
</tr>
<tr>
<td>1221</td>
</tr>
<tr>
<td>1229</td>
</tr>
<tr>
<td>1235</td>
</tr>
<tr>
<td>1237</td>
</tr>
<tr>
<td>10000001</td>
</tr>
<tr>
<td>10000003</td>
</tr>
</tbody>
</table>
## d-gap Encoding

- Store the first doc. id and gaps or relative differences — d-gaps
- Lists are read sequentially
- Compress them using variable byte encoding
- Compression and decompression is fast

<table>
<thead>
<tr>
<th>Doc. id list</th>
<th>Doc. id list (gaps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1 \rightarrow 3-2</td>
</tr>
<tr>
<td>8</td>
<td>5 \rightarrow 8-3</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>1220</td>
<td>1200</td>
</tr>
<tr>
<td>1221</td>
<td>1</td>
</tr>
<tr>
<td>1229</td>
<td>8</td>
</tr>
<tr>
<td>1235</td>
<td>5</td>
</tr>
<tr>
<td>1237</td>
<td>2</td>
</tr>
<tr>
<td>10000001</td>
<td>998764</td>
</tr>
<tr>
<td>10000003</td>
<td>2</td>
</tr>
</tbody>
</table>

13 x 3 bytes
d-gap Encoding

- Store the first doc. id and gaps or relative differences — d-gaps
- Lists are read sequentially
- Compress them using variable byte encoding
- Compression and decompression is fast

<table>
<thead>
<tr>
<th>Doc. id list</th>
<th>Doc. id list (gaps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1 → 3-2</td>
</tr>
<tr>
<td>8</td>
<td>5 → 8-3</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>1220</td>
<td>1200</td>
</tr>
<tr>
<td>1221</td>
<td>1</td>
</tr>
<tr>
<td>1229</td>
<td>8</td>
</tr>
<tr>
<td>1235</td>
<td>5</td>
</tr>
<tr>
<td>1237</td>
<td>2</td>
</tr>
<tr>
<td>10000001</td>
<td>998764</td>
</tr>
<tr>
<td>10000003</td>
<td>2</td>
</tr>
</tbody>
</table>

13 x 3 bytes (11 x 1 + 2 + 3) bytes
Rice Encoding

For a given \( b \), each number \( x \) in the list we do the following:

\[
\begin{align*}
    x-1 / b & \quad \text{in unary} \\
    x-1 \mod b & \quad \text{in binary}
\end{align*}
\]

For a given \( b \), each number \( x \) in the list we do the following:

- \( b \) is nearest (smaller) power of two of the average of the numbers \( g = (34+144+113+162) / 4 = 113.33 \)
- There are no zeros to encode
- simple to implement (bitwise operations) and better compression than var-byte, but slightly slower
For a given $b$, each number $x$ in the list we do the following:

- $x-1 / b$ in unary
- $x-1 \mod b$ in binary

**Doc. id list (gaps)**

<table>
<thead>
<tr>
<th>Number</th>
<th>Calculation</th>
<th>Golomb code</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>$33 / 78$</td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>$010001$</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td></td>
<td></td>
</tr>
<tr>
<td>162</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- $b$ is 0.69 of the average of the numbers $[g = (34+144+113+162) / 4 = 113.33 \times 0.69 \sim 78]$
- Need fixed encoding of number 0 to 77 using 6 or 7 bits
- Optimal for random gaps (dart board, random page ordering)
Each number $x$ in the list, $b = 2^{\lfloor \log(x) \rfloor}$

- $\log(b)$ in unary
- $x - 1 \mod (b)$ in binary ($b$ bits)

```
Doc. id list (gaps)
34  log(33) = 5
144 33 % 32
113 11110 00001
162  gamma code
```

- Good compression for small values, e.g., frequencies, bad for large numbers, and fairly slow
- **Delta coding**: Gamma code; then gamma the unary part
Variable sizes for each number is bad for decompression due to branching.

Constant size for each number is bad for compression since valuable space is wasted.

90% of numbers are small and 10% are large.

Trade-off — Use fixed size for most small integers and treat others as exceptions.

11 /13 can be encoded in 3 bits and 3 bytes for remaining 2.
PforDelta Compression

OPT-PFD:

- keep the last $b$ bits of exceptions in the original list and
- the higher-order bits and the offsets of the exceptions are separately encoded using another compression method.
- Instead of choosing a $b$ that guarantees less than 10% exceptions in the block, select the $b$ that results in the smallest compressed size for the block.
Redundancy in Temporal Collections

• Documents with multiple versions
  • each modification results in a new version
  • Wikipedia, Web Archives

• Most of the edits in versioned collections are minor edits

• Major edits occur in bursts

• Adjacent versions have considerable redundant content

• How do we exploit this for compressing inverted indexes?
Temporal Coalescing


- Recollect time-travel queries: *interstellar* @ 2013-2014
- Indexing each version results in index blowup
- Term frequencies do not change much hence have small impact on the final score
- Coalesce multiple versions of the same doc. into fewer postings
Temporal Coalescing

- Document with multiple versions (hence multiple postings) all having similar scores
- Not much difference in scores, hence no impact on overall scoring
- Coalesce postings for adjacent versions of same doc. with almost identical scores
Temporal Coalescing

- Document with multiple versions (hence multiple postings) all having similar scores
- Not much difference in scores, hence no impact on overall scoring
- Coalesce postings for adjacent versions of same doc. with almost identical scores
Temporal Coalescing

- Problem Statement: Given sequence $I$ of postings for term $v$ in document $d$, determine minimal-length output sequence that keeps relative approximation error below threshold $\epsilon$

$$\forall i : \left| p_i - p' \right| / |p_i| \leq \epsilon$$

- Approach:
  - Keep a moving average of already seen values
  - Once the average is outside the error bounds of a posting finalize the coalesced posting
Two-Level Index

- Construct a two-level index
- For each posting list for term \( t \)
  - First level: maintains doc. ids if any of its versions mentions the term
  - Second level: contains a bitmap of the versions which contain \( t \)
- First level considers a doc. as the union of all terms in its versions
- Use PforDelta compression as in standard inverted indexes
- Second level is bit-level representation
  - Since terms are bursty, spans consisting bursts can be effectively compressed
Process one list at a time

Maintain accumulators for partial results and update them

Best for unions
Query Processing: Term-At-A-Time

hannover  12  23  48  71  93  96  101

messe  18  23  71  77  112  189

Term-at-a-time

• Process one list at a time

• Maintain accumulators for partial results and update them

• Best for unions

Accumulators in memory

12  71
23  93  96
48  101
Process one list at a time

Maintain accumulators for partial results and update them

Best for unions
Query Processing: Document-At-A-Time

Document-at-a-time
- Open cursors to all lists
- Systematically move cursors to satisfy boolean expression
- Best for intersections

Conjunctive query semantics
- In each iteration find the max did M
- Move other cursors to greater or equal to M
- If all cursors point to M, move all one step further
Query Processing: Document-At-A-Time

**Document-at-a-time**
- Open cursors to all lists
- Systematically move cursors to satisfy boolean expression
- Best for intersections

**Conjunctive query semantics**
- In each iteration find the max did M
- Move other cursors to greater or equal to M
- If all cursors point to M, move all one step further
Query Processing: Document-At-A-Time

### Document-at-a-time
- Open cursors to all lists
- Systematically move cursors to satisfy boolean expression
- Best for intersections

### Conjunctive query semantics
- In each iteration find the max did $M$
- Move other cursors to greater or equal to $M$
- If all cursors point to $M$, move all one step further
Skip Lists

What if we are allowed to store extra information for each list?
Skip Lists

What if we are allowed to store extra information for each lists?

- Skip list allow fast intersections acting as a secondary index over posting lists
- Typically skip over fixed number of postings and are square root of the length of the postings list
Skip Lists

What if we are allowed to store extra information for each lists?

- Skip list allow fast intersections acting as a secondary index over posting lists

- Typically skip over fixed number of postings and are square root of the length of the postings list
Weak AND

- Generalized version of AND operation
  \[ \sum_{1 \leq i \leq k} x_i w_i \geq \theta, \]
  for a query of k words

- Weights and indicator variables specify the degree of AND
- Threshold helps avoid computation of non important documents
- An instance of DAAT scoring
• Assume a special iterator on the postings of the form “go to the first docID greater than X”
• Assume a special iterator on the postings of the form “go to the first docID greater than X”

• Typical state: we have a “cursor” at some docID in the postings of each query term
  • Each cursor moves only to the right, to larger docIDs
Assume a special iterator on the postings of the form “go to the first docID greater than X”

Typical state: we have a “cursor” at some docID in the postings of each query term
- Each cursor moves only to the right, to larger docIDs

Invariant – all docIDs lower than any cursor have already been processed, meaning
- These docIDs are either pruned away or
- Their scores have been computed
Weak AND

- At all times for each query term \( t \), we maintain an upper bound \( UB_t \) on the score contribution of any doc to the right of the cursor.
  - Max (over docs remaining in \( t \)'s postings) of \( w_t(doc) \)
  - As cursor moves right, the UB drops

\[
\begin{array}{c}
\text{hannover} \\
12 & 23 & 48 & 71 & 93 & 96 & 101
\end{array}
\]

\( UB(“hannover”) = 2.8 \)
• **Query:** *catcher in the rye*

• Let’s say the current cursor positions are as below

```
catcher  
  rye     
    in    
  the
```

```
    273
    304
    589
    762
```
• Query: *catcher in the rye*

• Let’s say the current cursor positions are as below

```
catcher  rye  in  the
273     304  589  762
```

\[
\begin{align*}
UB_{catcher} &= 2.3 \\
UB_{rye} &= 1.8 \\
UB_{in} &= 3.3 \\
UB_{the} &= 4.3
\end{align*}
\]

Threshold = 6.8
• Query: *catcher in the rye*
• Let’s say the current cursor positions are as below

<table>
<thead>
<tr>
<th>Term</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>catcher</td>
<td>273</td>
</tr>
<tr>
<td>rye</td>
<td>304</td>
</tr>
<tr>
<td>in</td>
<td>589</td>
</tr>
<tr>
<td>the</td>
<td>762</td>
</tr>
</tbody>
</table>

Threshold = 6.8

- $UB_{catcher} = 2.3$
- $UB_{rye} = 1.8$
- $UB_{in} = 3.3$
- $UB_{the} = 4.3$
- Query: *catcher in the rye*
- Let’s say the current cursor positions are as below

\[
\begin{align*}
\text{catcher} & : 273 \\
\text{rye} & : 304 \\
\text{in} & : 589 \\
\text{the} & : 762
\end{align*}
\]

Threshold = 6.8

\[
\begin{align*}
UB_{\text{catcher}} & = 2.3 \\
UB_{\text{rye}} & = 1.8 \\
UB_{\text{in}} & = 3.3 \\
UB_{\text{the}} & = 4.3
\end{align*}
\]
mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

Threshold = 6.8

\[ UB_{catcher} = 2.3 \]
\[ UB_{rye} = 1.8 \]
\[ UB_{in} = 3.3 \]
\[ UB_{the} = 4.3 \]
mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

\[
\begin{align*}
\text{catcher} & : 273, \text{Hopeless docs} \\
\text{rye} & : 304, \text{Hopeless docs} \\
\text{in} & : 589 \\
\text{the} & : 762
\end{align*}
\]

Threshold = 6.8

\[
\begin{align*}
UB_{\text{catcher}} & = 2.3 \\
UB_{\text{rye}} & = 1.8 \\
UB_{\text{in}} & = 3.3 \\
UB_{\text{the}} & = 4.3
\end{align*}
\]
mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

\[ UB_{catcher} = 2.3 \]
\[ UB_{rye} = 1.8 \]
\[ UB_{in} = 3.3 \]
\[ UB_{the} = 4.3 \]

Threshold = 6.8
mWAND Algorithm

- Terms sorted in order of cursor positions
- Move cursor to 589 or right

\begin{align*}
\text{catcher} & : 273 \quad \text{Hopeless docs} \\
\text{rye} & : 304 \quad \text{Hopeless docs} \\
\text{in} & : 589 \\
\text{the} & : 762 \\
\end{align*}

Threshold = 6.8

\begin{align*}
UB_{\text{catcher}} &= 2.3 \\
UB_{\text{rye}} &= 1.8 \\
UB_{\text{in}} &= 3.3 \\
UB_{\text{the}} &= 4.3 \\
\end{align*}

Update UB’s
• If 589 is present in enough postings, compute its full cosine score – else some cursors to right of 589

• Pivot again …

```plaintext
catcher
rye
in
the
```

- 589
- 589
- 589
- 762
• Disk access typically more expensive
• Can you trade-off sorting and pivoting?
  • Sorting cost (more CPU intensive) vs Pivoting cost (less disk access)
http://www.ir.uwaterloo.ca/book/

http://www.cis.upenn.edu/~jstoy/cis650/papers/WAND.pdf