Partitioned Elias-Fano Indexes

Giuseppe Ottaviano
ISTI-CNR, Pisa

Rossano Venturini
Università di Pisa
Inverted indexes

- Core data structure of Information Retrieval
- We seek fast and space-efficient encoding of posting lists (index compression)

<table>
<thead>
<tr>
<th>DocId</th>
<th>Document</th>
<th>Posting list</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[it is what it is not]</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>[what is a]</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>3</td>
<td>[it is a banana]</td>
<td>1, 3</td>
</tr>
</tbody>
</table>

- a
- banana
- is
- it
- not
- what

```plaintext
Docid    Document                  Posting list
1        [it is what it is not]    3
2        [what is a]              1, 2, 3
3        [it is a banana]         1, 3
```
Sequences in posting lists

• Generally, a posting list is composed of
  – Sequence of docids: strictly monotonic
  – Sequence of frequencies/scores: strictly positive
  – Sequence of positions: concatenation of strictly monotonic lists
  – Additional occurrence data: ???

• We focus on docids and frequencies
Sequences

- Monotonic
- Non-negative
- Strictly monotonic
- Strictly non-negative (Positive)

Elias Fano!

Interpolative coding

Docids

Gap coding

$a_i + i$

$\sum_{i=0}^{i} a_i$

$a_i - a_{i-1}$

$a_i - 1$
Elias-Fano encoding

- Data structure from the ’70s, mostly in the succinct data structures niche
- Natural encoding of monotonically increasing sequences of integers
- Recently successfully applied to inverted indexes [Vigna, WSDM13]
  - Used by Facebook Graph Search!
Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

Count in unary the size of upper bits “buckets” including empty ones

Concatenate lower bits

Elias-Fano representation of the sequence
Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

\[
\begin{array}{c}
00010 \\
00011 \\
00101 \\
00111 \\
01011 \\
01101 \\
11000 \\
\end{array}
\]

\[11011010100010101101111110100\]

Elias-Fano representation of the sequence

n: sequence length
U: largest sequence value

Maximum bucket: \([U / 2^l]\)
Example: \([24 / 2^2] = 6 = 110\)

Upper bits: one 0 per bucket and one 1 per value

Space
\([U / 2^l] + n + n l\) bits
Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

Can show that
\[ \ell = \lceil \log(U/n) \rceil \]
is optimal

\[ \lceil U / 2^\ell \rceil + n + n\ell \text{ bits} \]

(2 + \log(U/n))n bits

U/n is “average gap”
Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

nextGEQ(6) = 7

\[ \frac{6}{2^2} = 1 = 001 \]

Find the first GEQ bucket

= find the 1-th 0 in upper bits

With additional data structures and broadword techniques -> O(1)

Linear scan inside the (small) bucket
Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

\[
\begin{array}{c}
00010 \\
00011 \\
00101 \\
00111 \\
01011 \\
01101 \\
11000 \\
\end{array}
\]

\[
\begin{array}{c}
11011010100010101101111110100 \\
\end{array}
\]

Elias-Fano representation of the sequence 

\[(2 + \log(U/n))n\text{-bits space independent of values distribution!}\]

... is this a good thing?
Term-document matrix

- Alternative interpretation of inverted index

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>banana</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>it</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>not</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>what</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

• Gaps are distances between the Xs
Gaps are *usually* small

- Assume that documents from the same domain have similar docids

<table>
<thead>
<tr>
<th>...</th>
<th>...</th>
<th>unipi.it/</th>
<th>unipi.it/ students</th>
<th>unipi.it/ research</th>
<th>unipi.it/.../ ottaviano</th>
<th>...</th>
<th>sigir.org/</th>
<th>sigir.org/ venue</th>
<th>sigir.org/ fullpapers</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>pisa</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sigir</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Clusters” of docids

Posting lists contain long runs of very close integers
- That is, long runs of very small gaps
Elias-Fano and clustering

• Consider the following two lists
  – 1, 2, 3, 4, ..., n – 1, U
  – n random values between 1 and U
• Both have n elements and largest value U
  – Elias-Fano compresses both to the exact same number of bits: \((2 + \log(U/n))n\)
• But first list is far more compressible: it is “sufficient” to store n and U: \(O(\log n + \log U)\)
• Elias-Fano doesn’t exploit clustering
Partitioned Elias-Fano

- Partition the sequence into *chunks*
- Add *pointers* to the beginning of each chunk
- Represent each chunk and the sequence of pointers with Elias-Fano
- If the chunks “approximate” the clusters, compression improves
Partition optimization

• We want to find, among all the possible partitions, the one that takes up less space

• Exhaustive search is exponential

• Dynamic programming can be done quadratic

• Our solution: (1 + \(\varepsilon\))-approximate solution in linear time \(O(n \log(1/\varepsilon)/\log(1 + \varepsilon))\)
  
  – Reduce to a shortest path in a sparsified DAG
Partition optimization

- Nodes correspond to sequence elements
- Edges to potential chunks
- Paths = Sequence partitions
Partition optimization

- Each edge weight is the cost of the chunk defined by the edge endpoints
- Shortest path = Minimum cost partition
- Edge costs can be computed in O(1)...
- ... but number of edges is quadratic!
Sparsification: idea n.1
Sparsification: idea n.1

- General DAG *sparsification* technique
- Quantize edge costs in *classes* of cost between $(1 + \varepsilon_1)^i$ and $(1 + \varepsilon_1)^{i+1}$
- For each node and each cost class, keep only one maximal edge
  - $O(\log n / \log (1 + \varepsilon_1))$ edges per node!
- Shortest path in sparsified DAG at most $(1 + \varepsilon_1)$ times more expensive than in original DAG
- Sparsified DAG can be computed *on the fly*
Sparsification: idea n.2

• If we split a chunk at an arbitrary position
  – New cost $\leq$ Old cost + 1 + cost of new pointer
• If chunk is “big enough”, loss is negligible
• We keep only edges with cost $O(1 / \varepsilon_2)$
• At most $O(\log (1 / \varepsilon_2) / \log (1 + \varepsilon_1))$ edges/node
Sparsification

- Sparsified DAG has $O(n \log (1 / \epsilon_2) / \log (1 + \epsilon_1))$ edges!
- Fixed $\epsilon_i$, it is $O(n)$ vs $O(n^2)$ in original DAG
- Overall approximation factor is $(1 + \epsilon_2) (1 + \epsilon_1)$
Dependency on $\varepsilon_1$

Notice the scale!

Bits per posting

Time in seconds
Dependency on $\varepsilon_2$

Here we go from $O(n \log n)$ to $O(n)$. 

![Graph showing the transition from $O(n \log n)$ to $O(n)$](image)
## Results on GOV2 and ClueWeb09

<table>
<thead>
<tr>
<th></th>
<th>Gov2</th>
<th></th>
<th></th>
<th>ClueWeb09</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>space GB</td>
<td>doc bpi</td>
<td>freq bpi</td>
<td>space GB</td>
<td>doc bpi</td>
<td>freq bpi</td>
</tr>
<tr>
<td><strong>EF single</strong></td>
<td>7.66 (64.7%)</td>
<td>7.53 (+83.4%)</td>
<td>3.14 (+32.4%)</td>
<td><strong>19.63 (+23.1%)</strong></td>
<td>7.46 (+27.7%)</td>
<td>2.44 (+11.0%)</td>
</tr>
<tr>
<td><strong>EF uniform</strong></td>
<td>5.17 (+11.2%)</td>
<td>4.63 (+12.9%)</td>
<td>2.58 (+8.4%)</td>
<td>17.78 (+11.5%)</td>
<td>6.58 (+12.6%)</td>
<td>2.39 (+8.8%)</td>
</tr>
<tr>
<td><strong>EF ϵ-optimal</strong></td>
<td>4.65</td>
<td>4.10</td>
<td>2.38</td>
<td>15.94</td>
<td>5.85</td>
<td>2.20</td>
</tr>
</tbody>
</table>

### Table 4: Times for AND queries

<table>
<thead>
<tr>
<th></th>
<th>Gov2</th>
<th></th>
<th></th>
<th>ClueWeb09</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TREC 05</td>
<td>TREC 06</td>
<td>TREC 05</td>
<td>TREC 06</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EF single</strong></td>
<td>80.7 (+8%)</td>
<td>175.0 (+10%)</td>
<td>261.0 (+0%)</td>
<td>444.0 (-2%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EF uniform</strong></td>
<td>72.1 (-3%)</td>
<td>154.0 (-3%)</td>
<td>254.0 (-3%)</td>
<td>435.0 (-4%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EF ϵ-optimal</strong></td>
<td>74.5</td>
<td>159.0</td>
<td>261.0</td>
<td>451.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 5: Times for AND top-10 BM25 queries

<table>
<thead>
<tr>
<th></th>
<th>Gov2</th>
<th></th>
<th></th>
<th>ClueWeb09</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TREC 05</td>
<td>TREC 06</td>
<td>TREC 05</td>
<td>TREC 06</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EF single</strong></td>
<td>2.1 (+10%)</td>
<td>4.7 (+1%)</td>
<td>13.6 (-5%)</td>
<td><strong>15.8 (-9%)</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EF uniform</strong></td>
<td>2.1 (+9%)</td>
<td>5.1 (+10%)</td>
<td>15.5 (+8%)</td>
<td>18.9 (+9%)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EF ϵ-optimal</strong></td>
<td>1.9</td>
<td>4.6</td>
<td>14.3</td>
<td>17.4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**OR queries**

**AND queries**

---

*Note: The table entries are highlighted to indicate significant differences.*
Thanks for your attention!

Questions?