Tree-Encoded Bitmaps

Harald Lang
Alexander Beischl
Viktor Leis (Friedrich-Schiller Universität)
Peter Boncz (CWI)
Thomas Neumann
Alfons Kemper
Intro & Motivation

- Tree-Encoded Bitmaps: **Compression technique for bitmaps**
- Bitmap compression plays an important role in bitmap indexes.
  - Bitmap indexes allow for efficient evaluation of multi-dimensional predicates.
  - Bitmap indexes can become quite large
  - Thus, bitmap compression is commonly used to reduce space consumption
  - In fact, the term bitmap index typically refers to a compressed bitmap index.
Many bitmap compression techniques have been proposed

- **Word-Aligned Hybrid (WAH)**, and variants PLWAH, Concise, VAL-WAH, EWAH, SBH, etc.
  - based on run-length encoding
  - no efficient random access, all preceding bits need to be decompressed
- **Roaring Bitmap**
  - uses integer arrays
  - efficient random access (binary search) => efficient evaluation of conjunctive predicates

---

**Tree-Encoded Bitmaps (TEBs):**

- up to 50% space savings compared to Roaring
- without giving up efficient random access
TEB Ingredients

- Tree-based bitmap compression
- Tree encoding
- Additional space optimizations
Tree-Based Compression

Plain Bitmap: 1 1 0 1 0 0 0 0
Tree-Based Compression

Plain Bitmap:

```
1 1 0 1 0 0 0 0
```
Tree-Based Compression

Plain Bitmap:

1 1 0 1 0 0 0 0
Tree-Based Compression

Plain Bitmap:

1 1 0 1 0 0 0 0
Tree-Based Compression

Plain Bitmap:

```
1 1 0 1 0 0 0 0
```
Tree-Based Compression

Plain Bitmap:

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Tree-Based Compression

Plain Bitmap:

```
1 1 0 1 0 0 0 0
```
Tree-Based Compression

Plain Bitmap:

```
1 1 0 1 0 0 0 0
```
Tree-Based Compression

Plain Bitmap:

```
1 1 0 1
```

```
0 0 0 0
```
Tree-Based Compression

Plain Bitmap:

```
1 1 0 1 0 0 0 0
```
Tree Encoding

Level-order binary marked representation (G. Jacobson, 1989):

Tree Structure: | Labels:
Tree Encoding

Tree Structure:

Labels:
Tree Encoding

Tree Structure:

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Labels:
Tree Encoding

Tree Structure:

Labels:

Harald Lang | Tree-Encoded Bitmaps
Tree Encoding

Tree Structure:

```
1 1 0 0
0 1 2 3
```

Labels:

```
0 1
2 3
```
Tree Encoding

Tree Structure:

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Labels:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
Tree Encoding

Tree Structure:

```
1 | 1 | 0 | 0 | 1 | 0
0 | 1 | 2 | 3 | 4 | 5
```

Labels:

```
0 | 1 | 0
2 | 3 | 5
```
Tree Encoding

Tree Structure:

```
1 1 0 0 1 0 0
```

Labels:

```
0 1 0 1
```
Tree Encoding

Level-order binary marked representation (G. Jacobson, 1989):

Tree Structure: 1 1 0 0 1 0 0

Labels: 0 1 0 1

0 1 2 3 4 5 6

2 3 5 6
Tree-Encoding

- Beside the encoded tree structure and the labels, we maintain an additional data structure: a rank lookup table
- Additional space requirements: 0.0625 * size of the tree structure
- Enables efficient navigation within the encoded tree
  - Determining the left/right child of a tree node in O(1)
Space Optimizations

- So far: space requirements of a TEB could be \( \sim 3x \) of the original plain bitmap
- Thus, additional optimizations are required
- Worst case \textit{w/ optimizations enabled}: plain bitmap size + small header
Experimental Results

- **Space consumption:**
  - TEB has the lowest space consumption in 7 (out of 8) real-world data sets
    - Up to 34% **space savings** compared to Roaring
    - Up to 63% **space savings** compared to WAH

- **Performance (bitwise-AND, in-memory setting, bitmap size 2^20):**
  - Plain bitmap shows the best performance
  - Roaring takes ~1.8x the time of plain bitmap intersection.
  - **TEB takes ~1.6x the time of Roaring**
  - WAH up to an **order of magnitude slower**, depending on bitmap characteristics
What else can be found in the paper

- More detailed evaluation
- Implementation details for modern hardware
- and a link to the source code
Thank you, for watching.