EFFICIENT EXTERNAL TABLE REORGANIZATION

Walt Burkhard

Computer Science and Engineering Department
University of California, San Diego
Overview

- Related work.

- External hashing.

- Performance: analysis & simulation.

- Conclusions and future work.
Related work II

- Efficient Ordering of Hash Tables. 1977
- Uniform Hashing. 1983
- Hashing with Separators. 1987
- Double Hashing with Multiple Passbits. 2003
- Efficient Reordering of External Tables ...
Example table comprised of buckets ... with $b = 3$ slots per bucket.

<table>
<thead>
<tr>
<th>DOG</th>
<th>CAT</th>
</tr>
</thead>
</table>

| DOG | CAT | JAY |

Double hashing scheme: $h_1$ & $h_2$
Insertion of GNU probes this bucket –

gnu
next probe

dog
next probe

cat
next probe

jay
next probe

tree is explored but not explicitly built.
Efficient Reordering of External Tables III

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAY</td>
<td>YAK</td>
<td>FOX</td>
<td>PIG</td>
<td>EEL</td>
<td>CAT</td>
<td>HOG</td>
<td>DOG</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>start</th>
<th>step</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAT</td>
<td>2</td>
</tr>
<tr>
<td>DOG</td>
<td>4</td>
</tr>
<tr>
<td>APE</td>
<td>6</td>
</tr>
<tr>
<td>PIG</td>
<td>4</td>
</tr>
<tr>
<td>HOG</td>
<td>3</td>
</tr>
<tr>
<td>YAK</td>
<td>2</td>
</tr>
<tr>
<td>JAY</td>
<td>0</td>
</tr>
<tr>
<td>EEL</td>
<td>6</td>
</tr>
<tr>
<td>FOX</td>
<td>3</td>
</tr>
<tr>
<td>GNU</td>
<td>2</td>
</tr>
</tbody>
</table>

1st GNU: 2,1:1
2nd GNU: 3,1:2
3rd GNU: 4,1:5

next CAT: 6,3:6
next FOX: 4,1:7
next DOG: 6,2:8
next PIG: 3,6:9
next YAK: 3,4:11
next APE: 0,1:12

unfilled
Filled bucket probability

\[ \beta(\alpha) = q_b(\alpha). \]

\( q_i(\alpha) \) is the probability a bucket contains \( i \) records for a table with loading factor \( \alpha = m / (b\ n) \) for \( 0 \leq i \leq b \).

\[
\frac{dq_0}{d\alpha} = -b \times q_0(\alpha) / (1 - q_b(\alpha)) \\
\frac{dq_i}{d\alpha} = b \times (q_{i-1}(\alpha) - q_i(\alpha)) / (1 - q_b(\alpha)) \quad 0 < i < b \\
\frac{dq_b}{d\alpha} = b \times q_{b-1}(\alpha) / (1 - q_b(\alpha)).
\]
Efficient Reordering of External Tables

- **Successful search length** \( L_S \)

\[
E[L_S] = \frac{1}{m} \sum_{k=0}^{m-1} \sum_{j \geq 0} \beta\left(\frac{k}{bn}\right)^{(b+1)^j - 1}/b
\]

- Reasonably accurate for \( \alpha \) no greater than 0.8;
  7% error decreases with increasing \( b \).

- **Unsuccessful search length** \( L_U \)

\[
E[L_U] = \frac{1}{1 - \beta(\alpha)}
\]

- Very accurate.
Tree hashing, $b$ slots per bucket: loading factor

- unsuccessful search
  - expected values
  - experimental values

Buckets accessed vs. Tree hashing, $b$ slots per bucket: loading factor
Efficient Reordering of External Tables VII

- Improved **successful search length** via Brent and then Gonnet and Munro.

- Let $p_i(\beta)$ be the probability that a record at a particular location will find at least the next $i$ buckets probed to be filled; the sequence of $i$ buckets is referred to as a chain. The quantity $\beta p_i(\beta)$ is the probability of finding such a chain at any table location.
\[
\frac{d(\beta p_v)}{d\alpha} = \beta^v + \frac{1}{1-\beta} \sum_{i=0}^{v-1} \beta^{v-i} (p_i(\beta) - p_{i+1}(\beta)) + \sum_{i=0}^{v-1} \beta^{v-i-1} Q_i(\beta)
\]

\[p_v(0) = 0 \quad \text{for} \quad v = 1, 2, 3, \ldots \quad \& \quad p_0(\beta) = 1.\]
$Q_i(\beta)$ designates the sum of probabilities of all trees for which the tree traversal ends at a chain of length at least $i$.

\[
Q_0 = \beta^2 (1-p_1) \times \\
(1 + p_1 + \beta (p_1^2 + p_1^3 + p_1 p_2 + p_1 p_1 p_2 + p_1 p_1^2 p_2 + p_1^2 p_1 p_2) + \beta^2 (p_1^6 p_2^2 + \ldots) 
\]

\textit{second level} \hspace{2cm} \textit{third level} \hspace{2cm} \textit{fourth level}

\[
Q_1 = \beta^3 (p_1 - p_2) \times (p_1^3 + p_1^4 p_2 \ldots)
\]
IC(\( \alpha \)) is \( 1 + \beta + \beta^2 p_1^2 + \beta^3 p_1^6 p_2^2 + \beta^4 p_1^{18} p_2^6 p_3^2 + \cdots \)

\[ E[L_S] = \frac{1}{\alpha} \int_0^\alpha IC(t) \, dt \]
Tree hashing, $b$ slots per bucket: loading factor

**successful search**
- approximate values
- expected values
- experimental values

- $b = 1$
- $b = 2$
- $b = 4$
- $b = 8$
Conclusion and future work.

Tree hashing provides excellent expected access and insertion runtimes; without additional space.

Inclusion of passbits: extra space to reduce unsuccessful search lengths.

What is the best possible expected access runtime; it is known for capacity 1 buckets.