7. Hash-based Access Paths

Theo Härder
www.haerder.de

Goals and Restrictions
- Use of key transformation as design principle for access paths to the records of a table, for which a search criterion is supported
- Principles to make hashing dynamic
- Limitation to key access, no range search, etc.

Main reference:

Hash-based Access Paths

- Faster key access requires hashing methods
  - Hashing methods on external storage
    - Static methods
    - Dynamic Hashing
  - (Only) direct access
  - Ideally a single page access

- Extendible Hashing
  - Combination of concepts concerning digital trees and B-trees
  - Extendible Hashing supports strongly growing data volumes (≤ 2 page accesses needed)

- External Hashing without overflow areas

- Linear Hashing

Important parameters:
- \( n \) = #records of a record type
- \( b \) = #records/bucket (capacity)
- \( N \) = #buckets
- \( \beta \) = occupancy factor
Scattered Storage Structures (Hashing Methods)

**Direct computation** of record address via key (key transformation)

**Hashing function**

\[ h: \mathcal{S} \rightarrow \{0, 1, \ldots, N-1\} \]

\( \mathcal{S} = \) key space

\( N = \) size of the static hashing area in pages (buckets)

**Ideal case:** \( h \) is injective (no collisions)

- Application only in exceptional cases possible (‘dense’ key set required)
- Each record can be located with a single page access

Scattered Storage Structures (2)

**Static hashing areas with collision handling**

- Existing set of keys \( K (K \subseteq \mathcal{S}) \) should be distributed as uniform as possible onto \( N \) buckets

![Distribution on storage](image)

- Handling of synonyms
  - Insertion in the same bucket if possible
  - Allocation and chaining of overflow pages, if necessary
- Typical access factor: 1.1 to 1.4

**Multiplicity of hashing functions applicable**

- e.g. division-remainder method, folding, coding method, ...
Realization of DBS

Static Hashing
Hashing - overview
Static hashing
Dynamic hashing methods
Extendible hashing
External hashing
without overflow
Linear hashing

Static Hashing Method with Overflow Areas: Example

- Address computation for key K02:

\[
\begin{align*}
1101\ 0010 & \oplus \ 1111\ 0000 \\
& = \ 1111\ 0010 \\
& = 208_{10}
\end{align*}
\]

208 mod 5 = 3

K36 +
K41 +
K55 +
K67 +
K78 +
K88 +
K86 +
K36 +
K41 +
K55 +
K67 +
K78 +
K88 +
K86 +
K88 +
K86 +
K02 +
K25 +
K35 +
K43 +
K03 +
K26 +
K47 +
K51 +

Occupancy of Hash Areas - Measurement (1)

Primary area: 2000 buckets
Occupancy factor b: 5
Number of keys: 20,000
Data type of the key: Integer
Ideal hashing function: \( h(i) = i \mod N \)

\[ 1 \leq i \leq 20,000 \]
### Analysis of the Hashing Function

- **Collision if**
  
  \[ K_i \mod N = K_j \mod N = (K_i + l \cdot N) \mod N; \quad l = 1, 2, 3, \ldots \]

  key allocation is assumed to be \( K_i = K_j + j \cdot \Delta k; \quad j = 1, 2, 3, \ldots \)

  \[ \Rightarrow \text{critical relationship: } j \cdot \Delta k = l \cdot N \]

- **Which distance** \( j \cdot \Delta k \) **causes a collision?**

  Example: \( N = 576, \Delta k = 256 \)
  
  \[ j = (l \cdot N) / \Delta k \]
Dynamic Hashing Methods

- **Growth problem for static methods**
  - Static allocation of storage areas: storage occupancy?
  - In case of address space expansion: Rehashing
- **Cost, availability, addressability**

All records obtain a new address

- **Design goals**
  - Dynamic structure enables growth and shrinkage of the hash area (file)
  - No overflow techniques
  - Access factor $\leq 2$ for direct search

Extendible Hashing

- **Principal approach**
  - The single bits of a keys govern the path through the digital tree used for addressing
  - $K_i = (b_0, b_1, b_2, \ldots)$. In principle, it is possible to directly use the bit sequence of $K_i$ for addressing. Non-uniform key distribution produces an unbalanced digital tree
  - Because digital trees have no balancing mechanism for the height, balance must be enforced from “outside”
  - $h(K_i) = (b_0, b_1, b_2, \ldots)$. The use of $h(K_i)$ as so-called pseudo key (PK) should guarantee better uniform distribution

- **Uniform distribution of PKs**
  - implies minimal height of the digital tree

non-uniform distribution of keys $K$
$h(K_i) \rightarrow$ PK

uniform distribution of PKs
PKs are mapped onto directory
**Extendible Hashing**

**Extendible Hashing (2)**

- **Principal mapping of the pseudo keys**
  
  $h(K_i) = 01100101 \ldots$

  Dynamic border line

  - $d$ bits are required for bucket addressing which results in a dynamic border line of varying depth, in general
  - Digital tree addressing stops as soon as a bucket can accommodate the entire subtree

  - Balanced digital tree guarantees minimal $d_{\text{max}}$

**Extendible Hashing (3)**

- **Method does not need overflow areas, but access occurs via directory** (index)
  
  - Binary digital tree of height $d$ is implemented by a $(2^d)$ digital tree of height 1 (Trie of height 1 with $2^d$ entries)
  - $d$ is determined by the longest path in the binary digital tree
  - In a bucket, only records are stored whose PK match in the first $d'$ bits ($d'$ = local depth)
  - $d = \text{MAX (d')}$: $d$ bits of PK are used for addressing ($d = \text{global depth}$)
  - Directory contains $2^d$ entries

- **Storage structure**
  
  The Trie can be considered as directory or addressing table. The $d$ bits of $h(K_i)$ refer in the directory to an entry containing the address of the bucket which carries key $K_i$. If $d' < d$, (adjacent) entries can refer to the same bucket.

- **Cost of direct search**: max. 2 page accesses
**Extendible Hashing: Splitting of Buckets (1)**

- **Case 1:** overflow of a bucket whose local depth is smaller than the global depth $d$

  - Allocation of a new bucket (Split) with
    - Local redistribution of data
    - Increase of local depth
    - Local adjustment of the references in the directory

![Diagram of Extendible Hashing: Splitting of Buckets (1)](image1)

**Extendible Hashing: Splitting of Buckets (2)**

- **Case 2:** overflow of a bucket whose local depth is equal to the global depth

  - Allocation of a new bucket (Split) with
    - Local redistribution of data (increase of local depth)
    - Doubling of the directory (increase of global depth)
    - Global adjustment/redistribution of the references in the directory

![Diagram of Extendible Hashing: Splitting of Buckets (2)](image2)
Extendible Hashing: Splitting of Buckets (3)

Extendible Hashing (4)

- Dynamic growth and shrinkage of the hashing area
  - Buckets are only allocated on demand
  - Nodes in differing depth refer to a bucket
  - High bucket occupancy possible

- Prefix addressing

- Suffix addressing
**External Hashing without Overflow Areas**

- **Goal**
  - Each record can be located with **exactly one I/O access**
    - Chained overflow areas cannot be used

- **Static Hashing**
  - n records, N buckets with capacity b
  - Occupancy factor $\beta = \frac{n}{N \cdot b}$

- **Overflow handling**
  - Open Addressing (without chain or pointer)
  - Best known schemata: Linear Probing and Double Hashing
  - Probing sequence for a record with key k:
    - $H(k) = (h_1(k), h_2(k), ..., h_N(k))$
    - determines probing sequence of buckets (pages) for insertions and searches
    - is determined by k and a permutation of the set of bucket addresses {0, 1, ..., N - 1}

**First attempt**
- Search or insertion of $k = xy$

<table>
<thead>
<tr>
<th>ab</th>
<th>uv</th>
</tr>
</thead>
<tbody>
<tr>
<td>ij</td>
<td>cd</td>
</tr>
<tr>
<td>gh</td>
<td>no</td>
</tr>
</tbody>
</table>

- Probing sequence is assumed to be $H(xy) = (8, 27, 99, ...)$
- Many I/O accesses
- How do insertions occur?

→ Solution: see External Hashing Using Separators*

**Linear Hashing**

- **Dynamic growth and shrinking of the (primary) hashing area (file)**
  - Minimal administration data
  - No large directories for hash files

- **But: overflow records cannot be completely avoided!**
  - A high rate of overflow records is considered as an indicator that the file is overloaded ($\beta$ is too high) and must therefore be extended
  - Buckets are split in a strictly specified sequence
  - Only information: next bucket to be split

- **Principal approach**
  - $N$: initial size of the file in buckets
  - Sequence of hashing functions
    - $h_0(k) = \{0, 1, \ldots, N - 1\}$
    - $h_{j+1}(k) = h_j(k)$ or $h_{j+1}(k) = h_j(k) + N \cdot 2^j$ holds for all $j \geq 0$ and all keys $k$
  - Uniform probability desired for both cases of $h_{j+1}$

- **Example**
  - $h_0(k) = k \cdot (\text{mod } N \cdot 2^j), j = 0, 1, \ldots$

---

**Linear Hashing – Example**

- **Principle: LH**
  - $h_0(k) = 0, 1, \ldots, N - 1 \rightarrow \{0, 1\}$ for $N = 2$
  - $h_0(k) = h_0(k)$ or $h_0(k) = h_0(k) + N \cdot 2^j$
  - in general:
  - $h_{j+1}(k) = h_j(k)$ or $h_{j+1}(k) = h_j(k) + N \cdot 2^j$ displaced by $N \cdot 2^j$

- **$h_j(k) = k \mod (2^j \cdot N)$**

- **Extension:**
  - if $\beta > \beta_s$, redistribution of bucket $p$: $p := p + 1$
  - Address computation: $h := h_0(k)$
    - if $h < p$ then $h := h_{j+1}(k)$

---

### Linear Hashing – Example (2)

- **$h_1(k) = k \mod (2^L \cdot N)$**
  
  **sequence: 12, 14, 15**
  
  \[ \beta = \frac{8}{9} = 0.88 \]

<table>
<thead>
<tr>
<th>p</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>3</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>$h_2$</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \beta_S = 0.8 \]

- **sequence: 19, 24**
  
  \[ \beta = \frac{10}{12} = 0.83 \]

<table>
<thead>
<tr>
<th>p</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>12</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>$h_2$</td>
<td>13</td>
<td>14</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Linear Hashing (2)

- **$h_1(k) = k \mod (2^L \cdot N)$**
  
  **sequence: 17, 21, 25**
  
  \[ \beta = \frac{13}{15} = 0.87 \]

<table>
<thead>
<tr>
<th>p</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_1$</td>
<td>24</td>
<td>5</td>
<td>10</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>$h_2$</td>
<td>13</td>
<td>14</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Description of the file state

- **L**: number of duplications already done
- **p**: points to next bucket to be split (0 ≤ p < N \cdot 2^L)
- **$\beta$**: occupancy factor \( \frac{n}{(N \cdot 2^L - p) \cdot b} \)
- **n**: number of records stored
- **b**: capacity of a bucket

---

© 2011 AG DBIS

7-21

7-22
**Linear Hashing (3)**

- Example: principle of linear hashing
  - \( h_0(k) = k \mod 5 \)
  - \( h_1(k) = k \mod 10, \ldots \)
  - \( b = 4, L = 0, N = 5 \)
  - Splitting as soon as \( \beta > \beta_S = 0.8 \)

<table>
<thead>
<tr>
<th>( p )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_0 )</td>
<td>105</td>
<td>111</td>
<td>512</td>
<td>413</td>
<td>144</td>
</tr>
<tr>
<td>( h_1 )</td>
<td>790</td>
<td>076</td>
<td>477</td>
<td>243</td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>335</td>
<td>837</td>
<td>888</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>995</td>
<td>002</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Overflow records: 055, 117

---

**Linear Hashing (4)**

- Insertion of 888 increases occupancy to \( \beta = 17/20 = 0.85 \) and causes splitting

<table>
<thead>
<tr>
<th>( p )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_1 )</td>
<td>790</td>
<td>111</td>
<td>512</td>
<td>413</td>
<td>144</td>
<td>105</td>
</tr>
<tr>
<td>( h_0 )</td>
<td>010</td>
<td>076</td>
<td>477</td>
<td>243</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>837</td>
<td>888</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>995</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_1 )</td>
<td>055</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Insertion of 244, 399, and 100. Insertion of 100 causes splitting

<table>
<thead>
<tr>
<th>( p )</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>( h_1 )</td>
<td>790</td>
<td>111</td>
<td>512</td>
<td>413</td>
<td>144</td>
<td>105</td>
<td>076</td>
</tr>
<tr>
<td>( h_0 )</td>
<td>010</td>
<td>076</td>
<td>477</td>
<td>243</td>
<td>244</td>
<td>335</td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>100</td>
<td>837</td>
<td>888</td>
<td>399</td>
<td>995</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_0 )</td>
<td>002</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h_1 )</td>
<td>055</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- \( \beta_S = 0.8 \)

- \( \beta = 20/24 = 0.83 \)

<table>
<thead>
<tr>
<th>( h_1 )</th>
<th>117</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_0 )</td>
<td>( h_0 )</td>
</tr>
</tbody>
</table>
Realization of DBS

Static hashing

Hashing – overview

Dynamic hashing methods

Extendible hashing

External hashing

without overflow

Linear hashing

Realization of DBS

Realization

of DBS

Linear Hashing (5)

- Splitting
  - Trigger: β, position: p
  - File is increased by 1
  - p is increased by 1: p = (p + 1) mod (N ∙ 2^L)
  - If p is again set to Null (duplication of file completed), L is increased by 1

- Address computation
  - If h_0(k) ≥ p, then the requested address is h_0
  - If h_0(k) < p, then the bucket was already split. h_1(k) delivers the requested address
  - In general: h := h_L(k); if h < p then h := h_{L+1}(k);

- Split strategies
  - Uncontrolled splitting
    - Splitting as soon as a record enters the overflow area
      - β ~ 0.6, faster location
  - Controlled splitting
    - Splitting if a record enters the overflow area and β > β_s
      - β ~ β_s, longer overflow chains possible

Comparison of the Most Important Access Methods

<table>
<thead>
<tr>
<th>access method</th>
<th>storage structure</th>
<th>direct access</th>
<th>sequential processing</th>
<th>modification (without location)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential key comparison</td>
<td>sequential lists</td>
<td>O(n) = 5 ∙ 10^3</td>
<td>O(n) = 10^4</td>
<td>O(1) ≤ 2</td>
</tr>
<tr>
<td></td>
<td>chained lists</td>
<td>O(n) = 5 ∙ 10^3</td>
<td>O(n) = 10^4</td>
<td>O(1) ≤ 3</td>
</tr>
<tr>
<td>tree-based key comparison</td>
<td>balanced binary trees</td>
<td>O(log(n)) = 20</td>
<td>O(n) = 10^4</td>
<td>O(1) = 2</td>
</tr>
<tr>
<td></td>
<td>multi-way trees</td>
<td>O(log(n)) = 3 ∙ 4</td>
<td>O(n) = 10^4</td>
<td>O(1) = 2</td>
</tr>
<tr>
<td>constant key transformation</td>
<td>external hashing with</td>
<td>O(1) = 1.1 – 1.4</td>
<td>O(n log_2(n))^**</td>
<td>O(1) = 1.1</td>
</tr>
<tr>
<td>method</td>
<td>separate overflow area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>external hashing with</td>
<td>O(1) = 1</td>
<td>O(n log_2(n))^**</td>
<td>O(1) = 1 (+R)</td>
</tr>
<tr>
<td></td>
<td>separators</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable key</td>
<td>extendible hashing</td>
<td>O(1) = 2</td>
<td>O(n log_2(n))^**</td>
<td>O(1) = 1.1 (+R)</td>
</tr>
<tr>
<td>transformation method</td>
<td>linear hashing</td>
<td>O(1) = 1 + δ</td>
<td>O(n log_2(n))^**</td>
<td>O(1) &lt; 2</td>
</tr>
</tbody>
</table>

Example costs based on n = 10^6 (D = domino effect, R = reorganization cost)
- * In case of clustering up to a factor of 100 faster
- ** Physical sequential read, sorting and sequential processing of all records