Announcements (February 10)

- Reading assignments
  - Query processing survey (due next Monday)
- Homework #2 will be assigned this Thursday
- Recitation session this Friday
- Midterm and course project proposal in 3½ weeks

Static hashing

Does it make sense to use a hash-based index as a sparse index on a sorted table?
Performance of static hashing

- Depends on the quality of the hash function!
  - Best (hopefully average) case: one I/O!
  - Worst case: all keys hashed into one bucket!
  - See Knuth vol. 3 for good hash functions
- Rule of thumb: keep utilization at 50%-80%
- How do we cope with growth?
  - Extensible hashing
  - Linear hashing

Extensible hashing (TODS 1979)

- Idea 1: use $i$ bits of output by hash function and dynamically increase $i$ as needed
  
  $b(k)$: 010110111

- Problem: $++i = \text{double the number of buckets!}$
- Idea 2: use a directory
  - Just double the directory size
  - Many directory entries can point to the same bucket
  - Only split overflowed buckets
  - “One more level of indirection solves everything!”

Extensible hashing example (slide 1)

- Insert $k$ with $b(k) = 0101$

- Bucket too full?
  - $++$ local depth, split bucket, and $++$ global depth (double the directory size) if necessary
  - Allowing some overflow is fine too
Extensible hashing example (slide 2)

- Insert 1110, 0000

- Split again
  - No directory doubling this time

Extensible hashing example (slide 3)

- Insert 0001

Extensible hashing example (slide 4)

Delete is just the reverse:
If bucket is too empty, merge with sibling bucket, -- local depth; if possible, -- global depth and half the directory
Summary of extensible hashing

- **Pros**
  - Handles growing files
  - No full reorganization

- **Cons**

Linear hashing (*VLDB 1980*)

- Grow only when utilization exceeds a given threshold
- No extra indirection
  - Some extra math to figure out the right bucket

Insert 0101

Threshold exceeded; grow!

\[ i = 1 \quad \text{Number of bits in use} = \left \lfloor \log_2 n \right \rfloor \]

\[ u = 2 \quad \text{Number of primary buckets} \]

Linear hashing example (slide 2)

- Grows linearly (hence the name)
- Always split the \((n - 2^{(\log_2 u)})\)-th bucket (0-based index)
  - Intuitively, the first bucket with the lowest depth
  - Not necessarily the bucket being inserted into!

Insert 0001  Insert 1100

Threshold exceeded; grow!

\[ i = 2 \]
\[ u = 3 \]
Linear hashing example (slide 3)

Insert 1110
Threshold exceeded; grow!

\[ i = 2 \]
\[ n = 4 \]

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0000</td>
<td>0001</td>
<td>1010</td>
<td>1111</td>
</tr>
<tr>
<td>110</td>
<td>0101</td>
<td>1110</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>

Linear hashing example (slide 4)

- Look up 1110
  - Bucket 110 (6-th bucket) is not here
  - Then look in the \((6 - 2^{\log_2 n})\)-th bucket (= 2nd)

\[ i = 3 \]
\[ n = 5 \]

<table>
<thead>
<tr>
<th></th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0001</td>
<td>1010</td>
<td>1111</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>1110</td>
<td>1111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Summary of linear hashing

- **Pros**
  - Handles growing files
  - No full reorganization
  - No extra level of indirection

- **Cons**
Hashing versus B-trees