Announcements

- Homework #2 due in one week (February 26)
- Recitation session this Friday (February 21)
  - Homework #1 sample solution and graded assignments
  - Homework #1 common problems
  - Homework #2 Q&A
- Reading assignment

Static hashing

![Diagram of static hashing](image)
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

\[
\begin{array}{c}
\text{hash function output} \\
\hline
0 & 1 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

- Problem: $++i$ = 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 $h(k) = 0101$
  - 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**
  - One more level of indirection
  - Directory size still doubles
  - Sometimes doubling is not enough!

Linear hashing *(VLDB 1980)*

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

<table>
<thead>
<tr>
<th>Insert 0101</th>
<th>Threshold exceeded; grow!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0111</td>
</tr>
<tr>
<td>1010</td>
<td>0101</td>
</tr>
</tbody>
</table>

- $i = 1$ Number of bits in use $= \lceil \log_2 n \rceil$
- $u = 2$ Number of primary buckets

Linear hashing example (slide 2)

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

<table>
<thead>
<tr>
<th>Insert 0001</th>
<th>Insert 1100</th>
<th>Threshold exceeded; grow!</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1111</td>
<td>1010</td>
</tr>
<tr>
<td>1100</td>
<td>0101</td>
<td></td>
</tr>
</tbody>
</table>

- $i = 2$
- $u = 3$
Linear hashing example (slide 3)

Insert 1110
Threshold exceeded; grow!

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

\begin{array}{c|c|c|c}
00 & 01 & 10 & 11 \\
0000 & 0001 & 1010 & 1111 \\
1100 & 0101 & 1110 & 1111 \\
\end{array}

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)

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

\begin{array}{c|c|c|c|c}
00 & 01 & 10 & 11 & 100 \\
0000 & 0001 & 1010 & 1111 & 1100 \\
0101 & 1110 & 1111 & 1100 & 1100 \\
\end{array}

Summary of linear hashing

- Pros
  - Handles growing files
  - No full reorganization
  - No extra level of indirection
- Cons
Hashing versus B-trees

- Hashing is faster on average, but the worst case can be really bad.
- B-trees provide performance guarantees, and they are not that tall in practice.