CS460: Intro to Database Systems

Class 17: Hash Indexing

Instructor: Manos Athanassoulis

https://midas.bu.edu/classes/CS460/
Hash Indexing

Static Hashing

Extendible Hashing

Linear Hashing
Introduction

1. Actual data record (with key value \( k \))
2. \(<k, \text{rid of matching data record}>\)
3. \(<k, \text{list of rids of matching data records}>\)

Choice is **orthogonal to the indexing technique**

*Hash-based* indexes \( \rightarrow \) *equality selections*

**Cannot** support range searches

*Static* and *dynamic* hashing techniques exist
Static Hashing

#primary bucket pages fixed, allocated sequentially, never de-allocated; overflow pages if needed

\[ h(k) \mod M = \text{bucket to insert data entry with key } k \ (M: \#\text{buckets}) \]
Static Hashing (Contd.)

Buckets contain data entries

Hash function on search key field of record $r$

Must distribute values over range 0 ... M-1

What is a good hash function?

$h(key) = (a * key + b)$ usually works well

$a$ and $b$ are constants; lots known about how to tune $h$
Static Hashing (Problems!)

Long overflow chains can develop and degrade performance

Ways to solve?

– Reorganization is expensive and may block queries
– *Extendible* and *Linear Hashing*: Dynamic techniques to fix this problem
Hash Indexing

Static Hashing

Extendible Hashing

Linear Hashing
Extendible Hashing

Why not double the number of buckets?
Note that reading and writing all pages is expensive!

Idea:
Use directory of pointers to buckets
On overflow, double the directory (not the # of buckets)

Why does this help?
- Directory is much smaller than the entire index file
- Only one page of data entries is split

No overflow page! (caveat: duplicates w.r.t. the hash function)

Trick lies in how the hash function is adjusted!
Extendible Hashing

Directory: an array

Search for k:
- Apply hash function $h(k)$
- Take last global depth # bits of $h(k)$

Insert:
- If the bucket has space, insert, done
- If the bucket if full, split it, re-distribute – If necessary, double the directory
what is the hash function?
Example: Insert 6

$6^* = 0110$

global depth: $h$

directory

data pages

$00$

$01$

$10$

$11$

$4^* 12^*$

$1^* 13^*$

$10^*$

$15^* 7^*$
Example: Insert 6

6* = 0110

global depth: 2

directory

data pages

h

00

01

10

11

2 4* 12*

2 1* 13*

2 10*

2 15* 7*
Example: Insert 6

6* = 0110

h

global depth:

directory

data pages
Example 2: Insert 9

9* = 1001

h

directory
data pages
Example 2: Insert 9

9* = 1001

directory

data pages

now what??
Example 2: Insert 9

9* = 1001

(1) double the directory
Example 2: Insert 9

9*=1001

(1) double the directory
(2) re-distribute the split bucket
Example 2: Insert 9

9* = 1001

(1) double the directory
(2) re-distribute the split bucket
(3) connect corresponding buckets
Example 2: Insert 9

9* = 1001

h
Example 2: Insert 9

$9^* = 1001$

do we have to re-distribute all?
Example 3: Insert 5

5* = 0101

h
Example 3: Insert 5

5* = 0101

h

what happens if we want to insert 17?
do we have to re-distribute all?

[17 → 10001] so, double the dir again!
Example 3: Insert 5

5* = 0101

h

do we have to double the directory every time we split a bucket?
Example 3: Insert 14

$14^* = 1110$

$4^* 12^*$
$1^* 9^*$
$10^* 6^*$
$15^* 7^*$
$13^* 5^*$
Example 3: Insert 14

$14^* = 1110$

Diagram showing the binary search tree with the node $14^*$ inserted.
Example 3: Insert 14

14*=1110

h
Example 3: Insert 14

$14^* = 1110$

Diagram showing a binary search tree with nodes labeled with values and arrows indicating the search path. The tree root is labeled $h$ and the value 14 is inserted into the tree at the position indicated by the arrows.
Notes on Extendible Hashing

How many disk accesses for equality search?
- One if directory fits in memory, else two

Directory grows in spurts, and, if the distribution of hash values is skewed, can grow large
Notes on Extendible Hashing

Do we ever need overflow pages?

– Multiple entries with same hash value cause problems!

Delete: Reverse of inserts

– Can merge with split image
– Can shrink the directory by half. When?
  Each directory element points to same bucket as its split image
– Is shrinking/merging a good idea?
Hash Indexing

Static Hashing

Extendible Hashing

Linear Hashing
Linear Hashing

another dynamic hashing scheme

LH handles overflow chains without a directory

*Idea*: Use overflow pages, and split pages in a round-robin fashion
Example

this for information reasons!
it is not really kept.

<table>
<thead>
<tr>
<th>h_1</th>
<th>h_0</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
</tr>
<tr>
<td>011</td>
<td>11</td>
</tr>
</tbody>
</table>

what happens when we insert 5?

Next bucket to split
Example

this for information reasons!
it is not really kept.

\[ \begin{array}{c|c}
h_1 & h_0 \\
000 & 00 \\
001 & 01 \\
010 & 10 \\
011 & 11 \\
\end{array} \]

Next bucket to split

\[ \begin{array}{c|c|c}
4* & 8* \\
1* & 13* \\
10* \\
15* & 7* \\
\end{array} \]

what happens when we insert 5?

(1) 5 goes to an overflow page
Example

this for information reasons! it is not really kept.

<table>
<thead>
<tr>
<th>h₁</th>
<th>h₀</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next bucket to split

what happens when we insert 5?

(1) 5 goes to an overflow page
(2) we split the "next" page
Example

this for information reasons!
it is not really kept.

\[ h_1 \quad h_0 \]
\[
\begin{array}{c|c}
000 & 00 \\
001 & 01 \\
010 & 10 \\
011 & 11 \\
100 & \\
\end{array}
\]

Next bucket to split

what happens when we insert 5?

(1) 5 goes to an overflow page
(2) we split the “next” page
(3) we move the “next” pointer
Example: Insert 2

this for information reasons!
it is not really kept.

\[
\begin{array}{c|c}
\text{h}_1 & \text{h}_0 \\
000 & 00 \\
001 & 01 \\
010 & 10 \\
011 & 11 \\
100 & \\
\end{array}
\]

Next bucket to split
Example: Insert 2

this for information reasons!

it is not really kept.

...
Example: Insert 3

This is for information reasons!
It is not really kept.

<table>
<thead>
<tr>
<th>h1</th>
<th>h0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>00</td>
</tr>
<tr>
<td>0010</td>
<td>01</td>
</tr>
<tr>
<td>0100</td>
<td>10</td>
</tr>
<tr>
<td>0111</td>
<td>11</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
</tr>
</tbody>
</table>

Next bucket to split

What happens when we insert 3?
Example: Insert 3

This for information reasons! It is not really kept.

\[
\begin{array}{c|c}
  h_1 & h_0 \\
  000 & 00 \\
  001 & 01 \\
  010 & 10 \\
  011 & 11 \\
  100 & \\
\end{array}
\]

what happens when we insert 3?

(1) 3 goes to an overflow page

Next bucket to split
Example: Insert 3

this for information reasons!
it is not really kept.

<table>
<thead>
<tr>
<th>$h_1$</th>
<th>$h_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>00</td>
</tr>
<tr>
<td>001</td>
<td>01</td>
</tr>
<tr>
<td>010</td>
<td>10</td>
</tr>
<tr>
<td>011</td>
<td>11</td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>

what happens when we insert 3?

(1) 3 goes to an overflow page
(2) we split the “next” page
Example: Insert 3

this for information reasons!
it is not really kept.

Next bucket to split

what happens when we insert 3?

(1) 3 goes to an overflow page
(2) we split the "next" page
(3) we move the "next" pointer
Linear Hashing

$h_0, h_1, h_2 \ldots$ can be more general hash functions

when $h_0$ hits on a split buffer we employ $h_1$ and we have to look in both buffers

if the second is also split we use $h_2$ and so on

**Benefit:** buckets are split round-robin

$\rightarrow$ no long chains
Hash Indexing

Hash indexes: best for equality searches

*Static Hashing* can lead to long overflow chains

*Extendible Hashing*
- avoids overflow pages by splitting a bucket when full
- directory to keep track of buckets
- dir. can get too large (>memory) when data is skewed

*Linear Hashing*
- avoids directory by splitting buckets round-robin
- uses overflow pages
- overflow pages not likely to be long