Indexes

- Auxiliary structures that speed up operations that are not supported efficiently by the basic file organization

A Simple Index Example

<table>
<thead>
<tr>
<th>Index blocks</th>
<th>Data blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>70</td>
<td>70</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
</tbody>
</table>

Entries in an Index

- `<key, rid>`
- `<key, list of rid>`
- Data records

Organization of Index Entries

- Tree-structured
  - B-tree, R-tree, Quad-tree, kd-tree, ...
- Hash-based
  - Static, dynamic
- Other
  - Bitmap, VA-file, ...

From BST to BBST to B

- Binary Search Tree
  - Worst case??
- Balance Binary Search Tree
  - E.g. AVL, Red-Black
- B-tree
  - Why not use BBST in databases??
B-tree (B+-tree) Example

B-tree Properties
- Each node occupies one block
- Order $n$
  - $n$ keys, $n+1$ pointers
- Nodes (except root) must be at least half full
  - Internal node: $\lceil (n+1)/2 \rceil$ pointers
  - Leaf node: $\lceil (n+1)/2 \rceil$ pointers
- All leaf nodes are on the same level

B-tree Operations
- Search
- Insert
- Delete

B-tree Insert
- Find the appropriate leaf
- Insert into the leaf
  - there’s room $\Rightarrow$ we’re done
  - no room
    - split leaf node into two
      - insert a new <key,pointer> pair into leaf’s parent node
- Recursively apply previous step if necessary
  - A split of current ROOT leads to a new ROOT

B-tree Insert Examples
- (a) simple case
  - space available in leaf
- (b) leaf overflow
  - there’s room $\Rightarrow$ we’re done
  - no room
    - split leaf node into two
      - insert a new <key,pointer> pair into leaf’s parent node
  - Recursively apply previous step if necessary
  - A split of current ROOT leads to a new ROOT
(b) Insert key = 7

(c) Insert key = 160

(d) New root, insert 45

B-tree Delete

- Find the appropriate leaf
- Delete from the leaf
  - still at least half full \(\rightarrow\) we're done
  - below half full – coalescing
    - borrow a <key,pointer> from one sibling node, or
    - merge with a sibling node, and delete from a parent node
- Recursively apply previous step if necessary

B-tree Delete in Practice

- Coalescing is usually not implemented because it's too hard and not worth it

Static Hash Index
Hash Function

- A commonly used hash function: \( K \mod B \)
  - \( K \) is the key value
  - \( B \) is the number of buckets

Dynamic Hashing

- Problem of static hashing??
- Dynamic hashing
  - Extendable Hash Index

Extendable Hash Index ...

- \( 2^M \) buckets
  - \( M \) is maximum depth of index
- Multiple buckets can share the same block
  - Empty buckets do not take up space
  - Buckets are indexed by a bucket directory

... Extendable Hash Index

- Each block has a local depth \( L \), which means that the hash values of the records in the block have the same rightmost \( L \) bit
- The bucket directory keeps a global depth \( d \), which is the highest local depth

Extendable Hash Index Example

- \( M = 4 \)
- Hash function: \( K \mod 2^4 \)
- 2 index entries per block

Extendable Hashing (I)

- Bucket directory
- Bucket blocks
- Insertion process for keys 1000, 1011, 0100
Extendable Hashing (II)

Bucket directory

\[ d+1 \]

\[ \begin{array}{c}
0 \\
1
\end{array} \]

Bucket blocks

\[ L=1 \]

\[ 1000 \]

\[ 0100 \]

\[ 1011 \]

\[ L=1 \]

insert 1110

Extendable Hashing (III)

Bucket directory

\[ d+2 \]

\[ \begin{array}{c}
00 \\
10 \\
01 \\
11
\end{array} \]

Bucket blocks

\[ L=2 \]

\[ 1000 \]

\[ 0100 \]

\[ 1110 \]

\[ L=2 \]

\[ 1011 \]

\[ L=1 \]

Readings

- Textbook Chapter 21.1 – 21.4