CS522 Advanced Database Systems
Indexes

Chengyu Sun
California State University, Los Angeles

Outline
◆ Overview
◆ B-tree
◆ Hash indexes
◆ Multidimensional indexes

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

A Simple Index Example

Index Issues
◆ Search key
◆ Index pages
  ▪ What is stored in an entry?
  ▪ How are index entries organized?
◆ One entry per record??
  ▪ Dense or sparse
◆ Are the records sorted, hashed??
  ▪ Clustered or un-clustered
  ▪ Primary or secondary

Entries in an Index
◆ Actual data record
◆ <key, rid>
◆ <key, list of rid>
Organization of Index Entries

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

Clustered or Un-clustered

- Clustered index – records are sorted in the same order as the index entries
- Primary and secondary index
  - Stanford book: primary = clustered
  - Wisconsin book: index on primary key

Dense or Sparse

- Dense index – one entry per record
- Sparse – one entry per page
- Comparison??

Files and Indexes

- A primary index must be clustered??
- A secondary index cannot be clustered??
- We can build a clustered index on top of a hashed file??
- We can build a dense index on top of a sorted file??
- We can build a sparse index on top of a heap file??

Managing Indexes During Data Modification

- Assume clustered index

<table>
<thead>
<tr>
<th>Action</th>
<th>Dense</th>
<th>Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>create empty overflow page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete empty overflow page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>create empty sequential page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete empty sequential page</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete record</td>
<td></td>
<td></td>
</tr>
<tr>
<td>slide record</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Inverted Index

- Ever wondering how search engines work?
A Better Inverted Index

Outline
- Overview
- B-tree
- Hash indexes
- Multidimensional indexes

From BST to BBST to B
- Binary Search Tree
  - Worst case??
- Balanced Binary Search Tree
  - AVL, Red-Black, ...
- B-tree
  - Why not just use BBST in databases??

B+ Tree Example

B+ Tree Properties
- Order n
  - n keys, n+1 pointers
- Occupancy – at least half full
  - Non-leaf: \( \lceil (n+1)/2 \rceil \) pointers
  - Leaf: \( \lfloor (n+1)/2 \rfloor \) pointers
  - Except ROOT
- Balance
  - All leaf nodes on the same level

B+ Tree Insert
- Find the appropriate leaf
- Insert into the leaf
  - there's room → 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
- (c) non-leaf overflow
- (c) new root

(a) Insert key = 32

(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
    - borrow a \(<\text{key},\text{pointer}>\) from one sibling node, or
    - merge with a sibling node, and delete from a parent node
- Recursively apply previous step if necessary
  - When do we need a new ROOT (or decrease the height of the tree)?
B+ Tree Delete Examples

(a) simple case  
- no example
(b) leaf overflow
(c) non-leaf overflow
(d) new root

(b) Coalesce with sibling
- Delete 50

(c) Redistribute keys
- Delete 50

(d) Non-leaf coalesce
- Delete 37

B+tree deletions in practice
- Often, coalescing is not implemented
  - Too hard and not worth it!

B+ Tree Implementation
- Search
- Insert
- Delete
Implementation Issues

- Node size
  - How many entries do we want to put in a node?
- Buffering
  - Is LRU a good strategy for buffering B+ tree?
- Duplicates
  - What changes need to be made to the insert, delete, and search algorithms?

Calculate $n_{\text{opt}}$ – Assumptions

- Time to read from disk
  - $(70+0.05n)$ ms
- Once the page in memory, do binary search to locate the key
  - $(a+\log_2 n)$ ms
- Assume B+ tree is full, $\log_2 N$ nodes need to be examined.

Calculated $n_{\text{opt}}$

$f(n) = 0$
- n is a few hundreds

Outline

- Overview
- B-tree
- Hash indexes
- Multidimensional indexes

Static Hashing

- A commonly used hash function: $K\%B$
  - $K$ is the key value
  - $B$ is the number of buckets
    - prime, or
    - $2^n$
- String??

Hash Function
Dynamic Hashing

- Problem of static hashing??
- Dynamic hashing
  - Extensible hashing
  - Linear hashing

Extensible Hashing

- b-bit hash value, only i bits used, and i ≤ b
  - 2^i buckets
- When insert into a page that is already full
  - Split the page into two
  - Increment i
  - Double the number of buckets

Example: h(k) is 4 bits; 2 keys/bucket

Insert 1010

Example continued

Insert: 0111
0000

Efficiency of Extensible Hashing

- No overflow block
- Guaranteed single I/O access
- Problems??
Linear Hashing

- Start with $n$ buckets, $i = \lceil \log_2 n \rceil$
- Insert: let last $i$ bits of $H(k)$ be $m$
  - $m < n$, insert into bucket $m$
  - $m \geq n$, insert into bucket $m - 2^{-i}$
- Add one bucket when on average each page is more than 80% full

Example $b=4$ bits, $i=2$, 2 keys/bucket

- Insert 0101
- Can have overflow chains!

$0000 \rightarrow 0101$

Rule: If $h(k)[/] \leq m$, then look at bucket $h(k)[/]$
else, look at bucket $h(k)/ \cdot 2^{-i}$

Example Continued: How to grow beyond this?

$i=3$

About Linear Hashing

- Advantages??
- Overflow pages??
- Why 80%??

Outline

- Overview
- B-tree
- Hash indexes
- Multidimensional indexes
Concept of Dimension

- Dimensions in real life
  - 3D
  - 4D if you're a physicist
- Dimensions in databases
  - Each attribute is considered as a dimension
  - Number of dimensions (dimensionality) could be very high

GIS Data

- Attributes correspond to physical dimensions
  - e.g. longitude and latitude

Multimedia Data

- Dimensions in a mathematical sense

Commercial Data

- "Cubing" for aggregation queries
  - e.g. total sales in January, or total sales of San Gabriel stores in months from January to May

Queries on Multidimensional Data

- Data
  - points or regions in a multidimensional space
- Queries
  - Partial match queries
  - Range queries
  - Nearest-neighbor queries
  - Point-in-shape queries

Example 1: Assumptions

- 1,000,000 points, uniform distribution
- $x$-coordinate range: [0,1000]
- $y$-coordinate range: [0,1000]
- 100 entries per B-tree leaf node

Query: find points where $450 \leq x \leq 550$ and $450 \leq y \leq 550$
Example 1: Two Indexes
- Search x-index: 1000 leaf node accesses
- Search y-index: 1000 leaf node accesses
- To find 1% of the data, we need to access 10% of leaf nodes, twice
- Plus an intersection of two sets of pointers, which may or may not fit in memory

Example 1: One Multi-attribute Index
- Would it help??
- Problems??

Example 2
- Nearest-neighbor query: can we do it with a B-tree?

Partitioned Hashing
- Given n attributes \( A_1, A_2, \ldots, A_n \)
- Use n hash functions \( H_1, H_2, \ldots, H_n \)
- Apply \( H_i \) to \( A_i \) and concatenate the result

\[
H(K) = H_1(A_1)H_2(A_2)\ldots H_i(A_i)\ldots H_n(A_n)
\]

Grid File – Static
- Good for what type of queries??
- Problems??

Grid File – Dynamic
- Split – which dimension?? which position??
- Problems??
kd-Tree

- A binary tree
- At each node, the space is partitioned into two along a certain dimension
- Alternate dimensions between levels

kd-Tree Issues

- What if there's a “un-splittable” dimension??
- Unbalance??
- From main memory to secondary storage??

Quad-Tree Example

R-tree Insert

- Which subtree to choose??
- How to split a full node??

R-tree

- A B-tree for multidimensional data
- Works naturally for points, segments, and regions
Minimum Bounding Rectangle (MBR)

- MBR of a single object
- MBR of a set of objects

B-tree Entry vs. R-tree Entry

- B-tree internal node entry
- R-tree internal node entry
- Child node where key values are in \([K_r, K_{r+1})\)
- Child node where MBRs are in MBR_i

MBR overlapping is allowed.

R-tree Example

R-tree Example

R-tree Example

R-tree Point Search
R-tree Point Search

R-tree Range Search

R-tree Example

Some Observations So Far

- Arrange entries by spatial locality
- MBR overlapping may lead to searching multiple subtrees
  - Queue vs. Stack

R-tree Insert

- Which subtree to choose??
- How to split a full node??

R-tree Distance Query Optimization

- Minmax distance
- Maxmin distance
Bitmap Index

- Assume n records, m unique key values
- A bitmap index consists of m bit-vectors of length n such that
  - Each bit-vector corresponds to a unique key value
  - Each bit corresponds to a record
  - A bit is 1 if the record has the key value, and 0 otherwise

Bitmap Index Example

Data:
- 6 records
- 3 distinct values
  - ('john', 'A')
  - ('smith', 'B')
  - ('joe', 'A')
  - ('sue', 'C')
  - ('lisa', 'A')
  - ('jen', 'B')

Index:
- 'A': 1 0 1 0 1 0
- 'B': 0 1 0 0 0 1
- 'C': 0 0 0 1 0 0

Compress Bitmaps

- Run-length encoding
  - Run length i = 5
  - a unary representation of i's binary length
  - 1 in binary
  - 1 is implied

Bitmap Compression Examples

- Encode
  - 10000001000
- Decode
  - 1101101001011

Other Bitmap Index Issues

- Given value or value range, find bit vectors??
- Retrieve k-th record efficiently??
- Insert and delete??

Why Bitmap Index

- Bitmap performs well for what types of queries??
Readings

- Stanford book: Chapter 13, 14
- [Wisconsin book: Chapter 9, 10]
- [Kleinberg paper]
- [VA-file paper]