COS 425:
Database and Information
Management Systems

Indexing files

Last time

- File = a collection of pages (blocks) of records
- Read/write in units of page
  - Put in main memory buffer
- File organizations:
  - Heap: linked list (or directory) of pages
    - no order
  - Sorted sequential pages
    - Designated sort field
    - can binary search: get i^{th} page in one disk read
  - Hashing:
    - Designated hash field
    - Bucket is (primary) page for hash function value

Focus on key elements of cost

Improvements only for field of sort or hash
Improve access using other fields? => index

Avg. time	Heap	Sorted	Hashed
Search = (unique)	.5BD	$D\log_2 B$	D
Search range	BD	**$D(\log_2 B$ + # extra matching pages)**	1.25 BD
Insert	2D	**Search + D + BD**	2D
Delete (have record location)	2D	2D+BD	2D

B data pages in file **D** avg time to R/W page **R** records per page

Index

- Auxillary information on location of a record or page to facilitate retrieval
- Index field: field (i.e. attribute, column) used as look-up value for index
  - R&G: use term "index key" if field is a candidate key
  - Others: use "search key" instead of "index field"
    - "Search key" need not be candidate key
  - Could actually be combination of fields
    - E.g. LastName, FirstName
- Basic index is a file containing mappings:

  Index field value ⟶pointer(s) to page(s) containing records with given index field value

4

Index Types

1. Index works *with* file organization
   - Index and file work off same field
   - Example: Hashing file organization
     - Use index to get pointer to page serving as primary bucket for given field value
   - Clustered index
   - Some refer to as *primary index* (not R&G)

5

Index Types

2. Index works independent of file organization
   - File not organized on index field
   - Index must provide

     index field value ⟶ list of pointers to *all* file pages that contain records with that field value
   - Example hash index:
     - bucket contains list of page pointers
     - pages may be scattered throughout the file
     - overflow if too many pointers for one bucket
   - Some refer to as *secondary index* (not R&G)

6

A Sorted Index

- Consider sorted but not sequential file
  - Each page sorted
  - Each page linked to next page in sorted order
  - ***Cannot*** binary search
- Index:

sorting field value	pointer to first page containing

↑ Sorted order ↓

- One entry per field value in data file => dense index
- Can binary search index entries if can keep in memory or in sequential disk pages

7

Indexing sorted files - notes

- If index on sorted file using same field, index need not be dense (so sparse)
- Insert/delete for sorted file with sorted index costs to maintain sorted order in both
- Index may be sorted on different field(s) than file, but clustered as file is
  - Example: file sorted on (last_name, first_name)
    index sorted on last_name

8

Alternative sparse index for sorted file

again: index field same as sort field for file

file page number	page location	first value of index field on page

↑ Sorted order ↓

One entry *per file page*

Again, binary search if keep in memory or sequentially on disk

9

Compare costs:

dense sorted index **versus**
sparse sorted index with one value per data file page

- Use our crude estimates with
  B data pages in file **D** avg time to R/W page
  R records per page
- Suppose index record 1/10 size of data record
- Suppose index field (= sort field) is candidate key
- Cost search for unique value using dense index?
- Cost search for unique value using sparse index?

10

Cost example dense sorted index

- Use our crude estimates with
  B data pages in file **D** avg time to R/W page
  R records per page
- Suppose index record 1/10 size of data record
- Suppose index field (= sort field) is candidate key
- Cost search for unique value using dense index:
  B/10 pages in index file (file page size is fixed for all files)
  Binary search cost = $D\log_2(B/10)$

  Total cost = $D\log_2(B/10) + D$ includes data page access

11

Cost example sparse sorted index

- Use our crude estimates with
  B data pages in file **D** avg time to R/W page
  R records per page
- Suppose index record 1/10 size of data record
- Suppose index field (= sort field) is candidate key
- Cost search for unique value using sparse index:
  B pages in data file => B entries in index file
  10R index records per file page => B/(10R) index pages
  Binary search cost = $D\log_2(B/(10R))$

  Total cost = $D\log_2(B/(10R)) + D$ includes data page access

12

Compare costs:

- Use our crude estimates with
  - **B** data pages in file
  - **D** avg time to R/W page
  - **R** records per page

- Suppose index record 1/10 size of data record
- Suppose index field (= sort field) is candidate key

- Cost search for unique value using dense index?

$$D\log_2(B/10) + D$$

- Cost search for unique value using sparse index?

$$D\log_2(B/(10R)) + D$$

13

Compare costs: insertion

- Use our crude estimates with
  - **B** data pages in file
  - **D** avg time to R/W page
  - **R** records per page
- Suppose index record 1/10 size of data record
- Suppose index field (= sort field) is candidate key

- Cost to insert = cost to insert in data file
  + cost to insert in index file

= Search cost
+ D + D*B write data file page and move records
+ D write index entry
+ { D*B/10 move records for dense index
  { D*B/(10R) move records for sparse index

14

Index independent of file organization

But look again,
if index field is a *candidate key*,
this index works for *any* file organization :

sorting field value	pointer to *unique* page containing

↑ Sorted order ↓

One entry per index field value - dense
Can binary search index as before if keep in memory or sequentially on disk

15

Sorted index for general case

- One value of index field found in many records
- Need list of pointers to pages containing these records
- Dense index still works
- Most common arrangement:

Index field	pointer to page containing list
↑ One entry per field value. Sorted order ↓	

16

Addressing costs

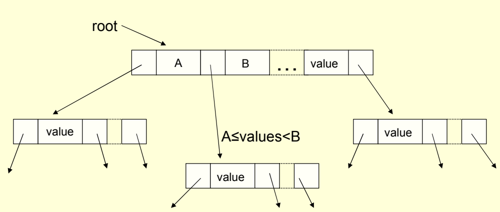
- Large sorted index costly in space and in time to insert/delete
  - When sorted index clustered, can use sparse index to avoid space
  - For general case, *must* have dense index
- Ideal: index to fit on one file page.
  - Keep in main memory
- Rarely achieve, so next best:
  - Index need not be stored sequentially on disk
  - Access cost is no worse than O($\log_2 B$)

=> **Search Tree!**

17

Tree index

•Each node of tree fits in one page
•Each node of tree contains index field values
and pointers to subtrees for ranges of values
•A leaf is
-For clustered index: a page of data file
-For general index: a page of pointers to records with given index values



18

Static Trees

- Build for file of records as balanced tree
- Not gracefully accommodate insert/delete
- ISAM: Indexed Sequential Access Method
  - See R&G text
- We focus on dynamic search trees

19

Dynamic Trees

- Tree will change to keep balance as file grows/shrinks
- Tree height: longest path root to leaf
- N data entries

  Data entry is page of data file if clustered index

  Data entry is page of (value, record pointer) pairs otherwise
- Want tree height proportional to logN always

20

B+ Trees

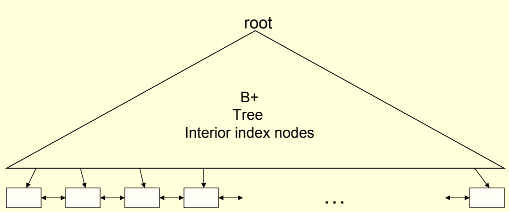
- Most widely used dynamic tree as index
- Most widely used index
- Properties
  - Data entries only in leaves
    - Compare B-trees
  - One page per tree node, including leaves
  - All leaves same distance from root => balanced
  - Leaves doubly linked
    - Gives sorted data entries
  - Call index field of tree “B+ key”

21

B+ trees continued

- To achieve equal distance all leaves to root cannot have fixed fanout
- To keep height low, need fanout high
  - Want interior nodes full
- Parameter d - order of the B+ tree
- Each interior node except root has m keys for $d \leq m \leq 2d$
  - m+1 children
- The root has m keys for $1 \leq m \leq 2d$
  - Tree height grows/shrinks by adding/removing root
- d chosen so each interior node fits in one page

22

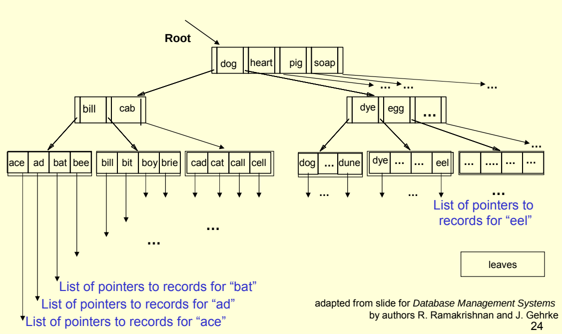


Leaves will be 1/2 full to full as well

23

Example B+ Tree

order = 2: 2 to 4 search keys per interior node



adapted from slide for *Database Management Systems* by authors R. Ramakrishnan and J. Gehrke

24

Board Examples

25