CSCI 333
Indexing

Chapter 10
10 November 2003

Notes with the dark blue background were prepared by the textbook author

Clifford A. Shaffer
Department of Computer Science
Virginia Tech

Indexing

Goals:

- Store large files
- Support multiple search keys
- Support efficient insert, delete, and range queries

Terms

Entry sequenced file: Order records by time of insertion.

- Search with sequential search

Index file: Organized, stores pointers to actual records.

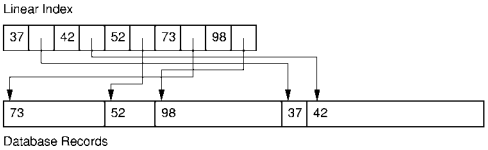
- Could be organized with a tree or other data structure.

Primary Key: A unique identifier for records. May be inconvenient for search.

Linear Indexing

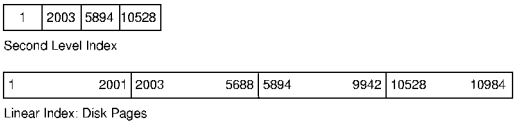
Linear index: Index file organized as a simple sequence of key/record pointer pairs with key values are in sorted order.

Linear indexing is good for searching variable-length records.



Linear Indexing (2)

If the index is too large to fit in main memory, a second-level index might be used.



Tree Indexing (1)

Linear index is poor for insertion/deletion.

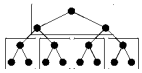
Tree index can efficiently support all desired operations:

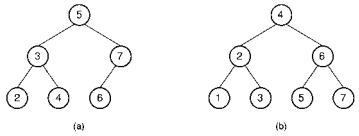
- Insert/delete
- Multiple search keys (multiple indices)
- Key range search

Tree Indexing (2)

Difficulties when storing tree index on disk:

- Tree must be balanced.
- Each path from root to leaf should cover few disk pages.





2-3 Tree (1)

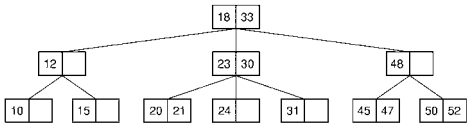
A 2-3 Tree has the following properties:

1. A node contains one or two keys
2. Every internal node has either two children (if it contains one key) or three children (if it contains two keys).
3. All leaves are at the same level in the tree, so the tree is always height balanced.

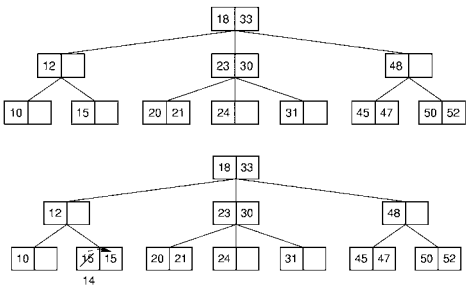
The 2-3 Tree has a search tree property analogous to the BST.

2-3 Tree (2)

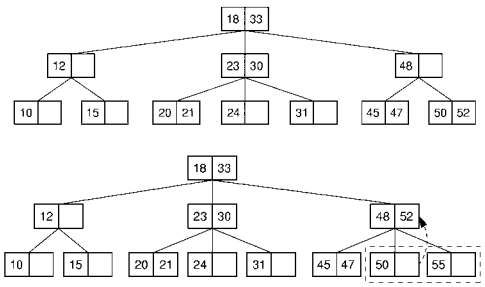
The advantage of the 2-3 Tree over the BST is that it can be updated at low cost.



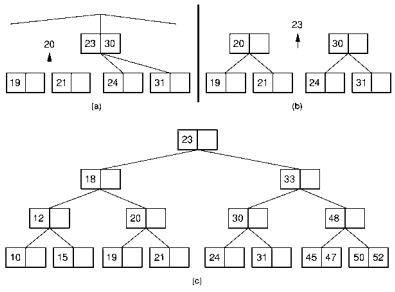
2-3 Tree Insertion (1)



2-3 Tree Insertion (2)



2-3 Tree Insertion (3)



B-Trees (1)

The B-Tree is an extension of the 2-3 Tree.

The B-Tree is now **the** standard file organization for applications requiring insertion, deletion, and key range searches.

B-Trees (2)

1. B-Trees are always balanced.
2. B-Trees keep similar-valued records together on a disk page, which takes advantage of locality of reference.
3. B-Trees guarantee that every node in the tree will be full at least to a certain minimum percentage. This improves space efficiency while reducing the typical number of disk fetches necessary during a search or update operation.

B-Tree Definition

A B-Tree of order m has these properties:

- The root is either a leaf or has two children.
- Each node, except for the root and the leaves, has between $\lceil m/2 \rceil$ and m children.
- All leaves are at the same level in the tree, so the tree is always height balanced.

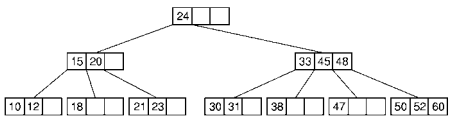
A B-Tree node is usually selected to match the size of a disk block.

- A B-Tree node could have hundreds of children.

B-Tree Search (1)

Search in a B-Tree is a generalization of search in a 2-3 Tree.

1. Do binary search on keys in current node. If search key is found, then return record. If current node is a leaf node and key is not found, then report an unsuccessful search.
2. Otherwise, follow the proper branch and repeat the process.



B+-Trees

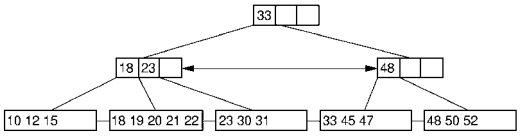
The most commonly implemented form of the B-Tree is the B+-Tree.

Internal nodes of the B+-Tree do not store record -- only key values to guild the search.

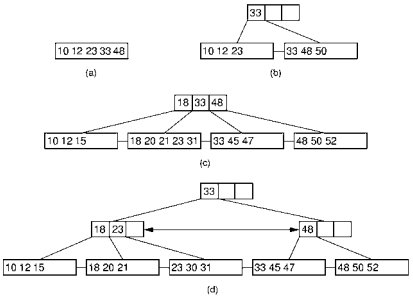
Leaf nodes store records or pointers to records.

A leaf node may store more or less records than an internal node stores keys.

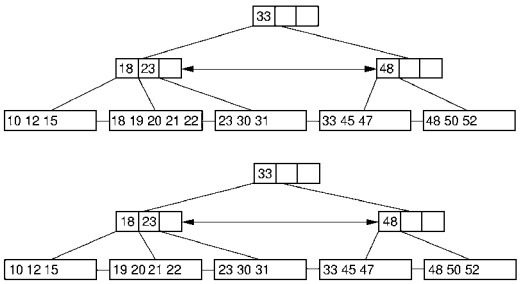
B+-Tree Example



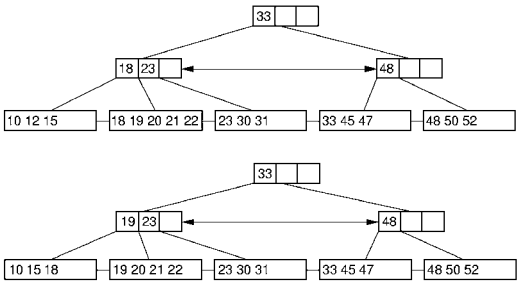
B+-Tree Insertion



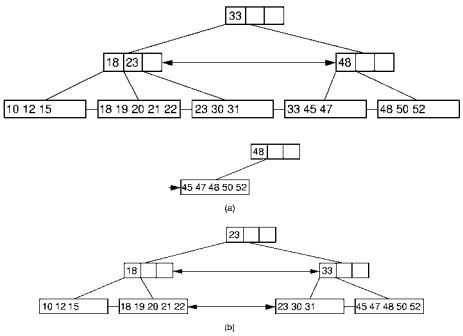
B+-Tree Deletion (1)



B+-Tree Deletion (2)



B+-Tree Deletion (3)



B-Tree Space Analysis (1)

B+-Trees nodes are always at least half full.

The B*-Tree splits two pages for three, and combines three pages into two. In this way, nodes are always 2/3 full.

Asymptotic cost of search, insertion, and deletion of nodes from B-Trees is $\Theta(\log n)$.

- Base of the log is the (average) branching factor of the tree.

B-Tree Space Analysis (2)

Example: Consider a B+-Tree of order 100 with leaf nodes containing 100 records.

1 level B+-tree:
2 level B+-tree:
3 level B+-tree:
4 level B+-tree:

Ways to reduce the number of disk fetches:

- Keep the upper levels in memory.
- Manage B+-Tree pages with a buffer pool.

B-tree ADT

- B-tree abstract data type definition
- B-tree on-disk format

B-tree operations

- Create operations
- Internal "helper" routines
- Disk I/O routines
- Update routine

Driver and test routines

- Create a B-tree
- Insert record into B-tree
- List a B-tree
- Create a monster B-tree
- Low-level B-tree dump