DATA STRUCTURE AND ALGORITHM

Dr. Khine Thin Zar
Professor
Computer Engineering and Information Technology Dept.
Yangon Technological University

Lecture 10

Indexing

Outlines of Class (Lecture 10)

- Introduction
- Indexing
- Linear Indexing
- Indexed Sequential Access Method (ISAM)
- Tree-based Indexing
  - ✓ 2-3 Trees
  - ✓ B-Trees

Introduction

- Large database of records with multiple search keys requiring the ability to insert, delete, and search for records
- Hashing
  - ✓ Finding the record with key value K
- Many applications
  - ✓ Finding range query, visiting all records in order of their key value, finding the record with the greatest key value

Introduction (Cont.)

- Large database of records with multiple search keys requiring the ability to insert, delete, and search for records
- Hashing
  - ✓ Finding the record with key value K
- Many applications
  - ✓ Finding range query, visiting all records in order of their key value, finding the record with the greatest key value

Introduction (Cont.)

- Large database of records with multiple search keys requiring the ability to insert, delete, and search for records
- Hashing
  - ✓ Finding the record with key value K
- Many applications
  - ✓ Finding range query, visiting all records in order of their key value, finding the record with the greatest key value

Introduction (Cont.)

- File structure used to organize a large collection of records stored on disk
  - ✓ Efficient insertion, deletion, and search operations: for exact-match queries, range queries, and largest/smallest queries
- An entry-sequenced file
  - ✓ Stores records in the order that they were added to the file.
- Primary key
  - ✓ Each record of a database has a unique identifier (eg. Social Secuity Number or ID number).
- Secondary key
  - ✓ A key field might be duplicated in multiple records (eg. Salary).

Introduction (Cont.)

- Indexing
  - ✓ The process of associating a key with the location of a corresponding data record.

- Index File
  - ✓ An index file is created whose records consist of **key/pointer** pairs
  - ✓ Each key is associated with a pointer to a complete record in the main database file.
  - ✓ The full database might be searched directly for the record with that primary key, or there might be a **primary key index** (or primary index) that relates each primary key value with a pointer to the actual record on disk.
  - ✓ Only the primary index provides the location of the actual record on disk, while the secondary indices refer to the primary index.

✓

Linear Indexing

- A Linear index
  - ✓An index file organized as a sequence of key/pointer pairs
  - ✓Keys are sorted in order
- Pointers
  - ✓point to the position of the complete record on disk,
  - ✓point to the position of the primary key in the primary index, or
  - ✓are actually the value of the primary key.
- Depending on its size, a linear index might be stored in main memory or on disk.
- Advantages
  - ✓It provides convenient access to variable-length database records, because each entry in the index file contains a fixed-length key field and a fixed-length pointer to the beginning of a (variable-length) record.

Linear Indexing (Cont.)

- A linear index also allows for efficient search and random access to database records.

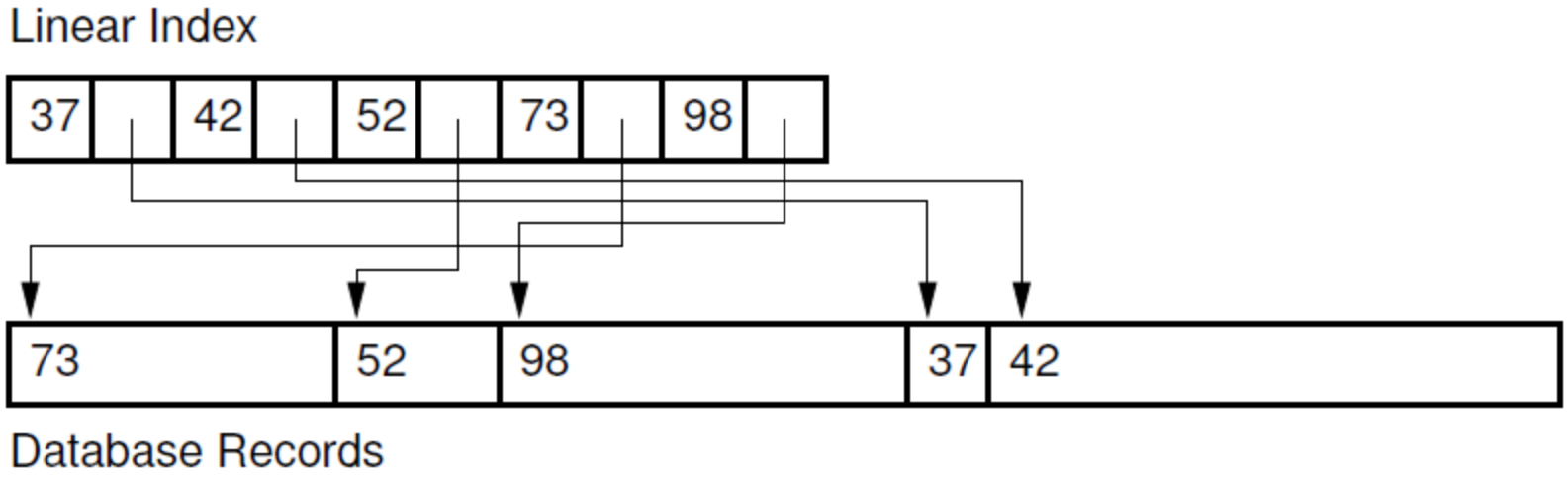

Figure: Linear indexing for variable-length records. Each record in the index file is of fixed length and contains a pointer to the beginning of the corresponding record in the database file.

Linear Indexing (Cont.)

- If the database contains enough records, the linear index might be too large to store in main memory.
- One solution to this problem
  - ✓ to store a second-level linear index in main memory that indicates which disk block in the index file stores a desired key.
  - ✓ For example, the linear index on disk might reside in a series of 1024-byte blocks. If each key/pointer pair in the linear index requires 8 bytes (a 4-byte key and a 4-byte pointer), then 128 key/pointer pairs are stored per block.

Linear Indexing (Cont.)

- The second-level index, stored in main memory, consists of a simple table storing the value of the key in the first position of each block in the linear index file.

- If the linear index requires 1024 disk blocks (1MB), the second-level index contains only 1024 entries, one per disk block.

- To find which disk block contains a desired search key value, first search through the 1024-entry table to find the greatest value less than or equal to the search key.

Linear Indexing (Cont.)

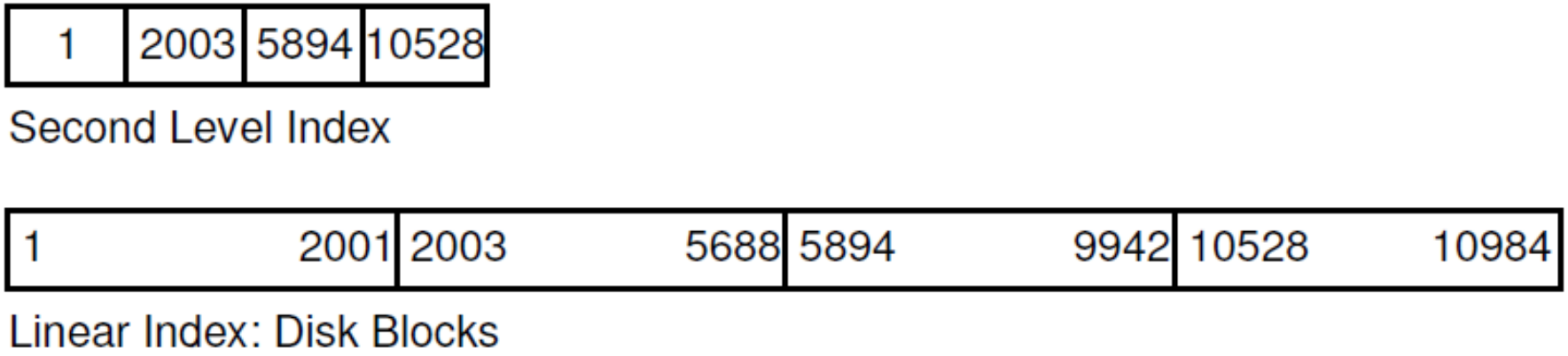

Figure: A simple two-level linear index. The linear index is stored on disk. The smaller, second-level index is stored in main memory. Each element in the second-level index stores the first key value in the corresponding disk block of the index file. In this example, the first disk block of the linear index stores keys in the range 1 to 2001, and the second disk block stores keys in the range 2003 to 5688. Thus, the first entry of the second-level index is key value 1 (the first key in the first block of the linear index), while the second entry of the second-level index is key value 2003.

Linear Indexing (Cont.)

- Second-level index is stored in main memory, accessing a record by this method requires two disk reads: one from the index file and one from the database file for the actual record.
- Every time a record is inserted to or deleted from the database, all associated secondary indices must be updated.
- Updates to a linear index are expensive, because the entire contents of the array might be shifted.
- Another problem is that multiple records with the same secondary key each duplicate that key value within the index.

Linear Indexing (Cont.)

- One improvement on the simple sorted array is a two-dimensional array where each row corresponds to a secondary key value.
- A row contains the primary keys whose records have the indicated secondary key value.
- The cost of insertion and deletion is reduced, because only one row of the table need be adjusted.
- A new row is added to the array when a new secondary key value is added.

Linear Indexing (Cont.)

Jones	AA10	AB12	AB39	FF37
Smith	AX33	AX35	ZX45	
Zukowski	ZQ99			

Figure: A two-dimensional linear index. Each row lists the primary keys associated with a particular secondary key value. In this example, the secondary key is a name. The primary key is a unique four-character code.

- A drawback to this approach is that the array must be of fixed size, which imposes an upper limit on the number of primary keys that might be associated with a particular secondary key.
- Furthermore, those secondary keys with fewer records than the width of the array will waste the remainder of their row.

Linear Indexing (Cont.)

- A better approach is to have a one-dimensional array of secondary key values, where each secondary key is associated with a linked list.
- This works well if the index is stored in main memory, but not so well when it is stored on disk because the linked list for a given key might be scattered across several disk blocks.
- Inverted List (Inverted File)
  - ✓ If the primary key is the employee's ID number and the secondary key is the employee's name, then each record in the name index associates a name with one or more ID numbers.
  - ✓ The ID number index in turn associates an ID number with a unique pointer to the full record on disk.
  - ✓ The secondary key index in such an organization is also known as an **inverted list** or **inverted file.**

Linear Indexing (Cont.)

- It is inverted in that searches work backwards from the secondary key to the primary key to the actual data record.

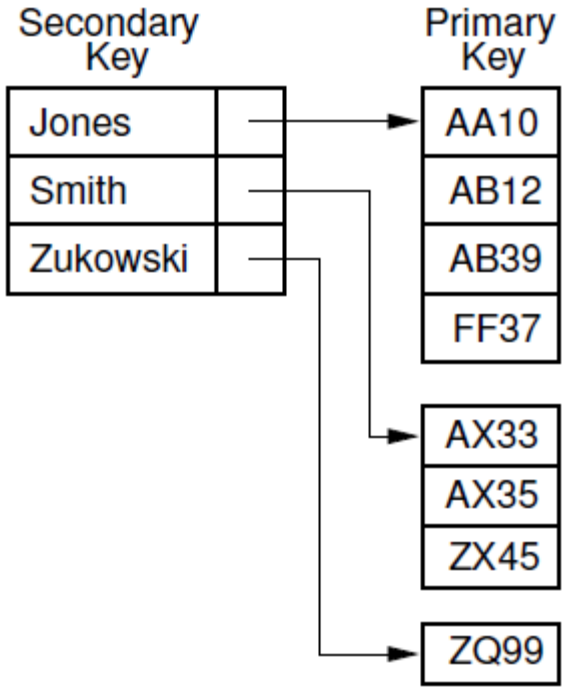

Figure: Illustration of an inverted list. Each secondary key value is stored in the secondary key list. Each secondary key value on the list has a pointer to a list of the primary keys whose associated records have that secondary key value.

Linear Indexing (Cont.)

A better approach to storing inverted lists:

- An array of secondary key values is shown as before.
- Associated with each secondary key is a pointer to an array of primary keys.
- The primary key array uses a linked-list implementation.
- This approach combines the storage for all of the secondary key lists into a single array, probably saving space.
- Each record in this array consists of a primary key value and a pointer to the next element on the list.

Linear Indexing (Cont.)

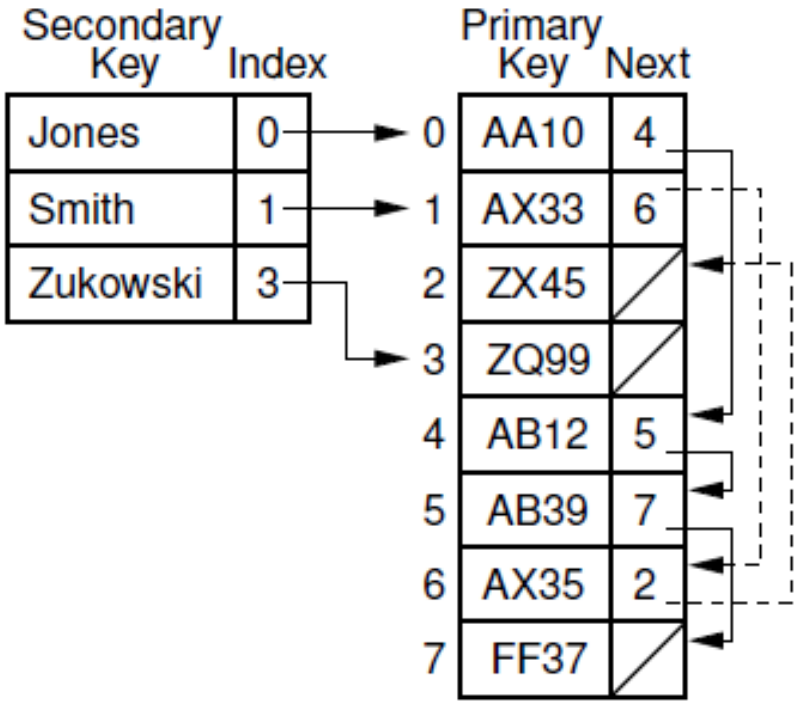

Figure: An inverted list implemented as an array of secondary keys and combined lists of primary keys. Each record in the secondary key array contains a pointer to a record in the primary key array. The **next** field of the primary key array indicates the next record with that secondary key value.

Indexed Sequential Access Method (ISAM)

- ISAM
  - ✓ An early attempt to solve the problem of large databases requiring frequent update.
  - ✓ ISAM is based on a modified form of the linear index.
  - ✓ Records are stored in sorted order by primary key.
  - ✓ The disk file is divided among a number of cylinders on disk.
  - ✓ Each cylinder holds a section of the list in sorted order.
  - ✓ Initially, each cylinder is not filled to capacity, and the extra space is set aside in the cylinder overflow.

Indexed Sequential Access Method (ISAM) (Cont.)

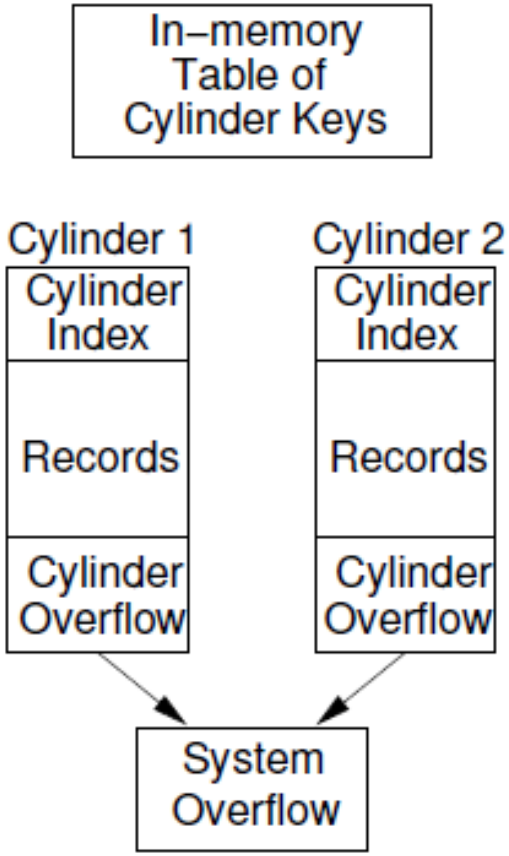

Figure: Illustration of the ISAM indexing system.

Indexed Sequential Access Method (ISAM) (Cont.)

- In memory is a table listing the lowest key value stored in each cylinder of the file.
- Each cylinder contains a table listing the lowest key value for each block in that cylinder, called the cylinder index.
- When new records are inserted, they are placed in the correct cylinder's overflow area.
- If a cylinder's overflow area fills completely, then **a system-wide overflow area is used.**
- **Searching Process**
  - ✓ Search proceeds by determining the proper cylinder from the system-wide table kept in main memory.
  - ✓ The cylinder's block table is brought in from disk and consulted to determine the correct block.
  - ✓ If the record is found in that block, then the search is complete.
  - ✓ Otherwise, the cylinder's overflow area is searched.
  - ✓ If that is full, and the record is not found, then the system-wide overflow is searched.

Tree-based Indexing

- Linear indexing is efficient when the database is static, that is, when records are inserted and deleted rarely or never.
- ISAM is adequate for a limited number of updates, but not for frequent changes.
- In their most general form, database applications have the following characteristics:
  - ✓ Large sets of records that are frequently updated.
  - ✓ Search is by one or a combination of several keys.
  - ✓ Key range queries or min/max queries are used.

Tree-based Indexing (Cont.)

- Binary Search Tree (BST)
  - ✓ Is used to store primary and secondary key indices.
  - ✓ BSTs can store duplicate key values, they provide efficient insertion and deletion as well as efficient search, and they can perform efficient range queries.
  - ✓ When there is enough main memory, the BST is a viable option for implementing both primary and secondary key indices.
  - ✓ BST is unbalanced.

Tree-based Indexing (Cont.)

- Two significant issues on BST
  - ✓ The first is how to keep the tree balanced.
  - ✓ The second is how to arrange the nodes on blocks so as to keep the number of blocks encountered on any path from the root to the leaves at a minimum.
- Balancing the BST
  - ✓ A scheme for balancing the BST and allocating BST nodes to blocks in a way that minimizes disk I/O.
  - ✓ Maintaining such a scheme in the face of insertions and deletions is difficult.

Tree-based Indexing (Cont.)

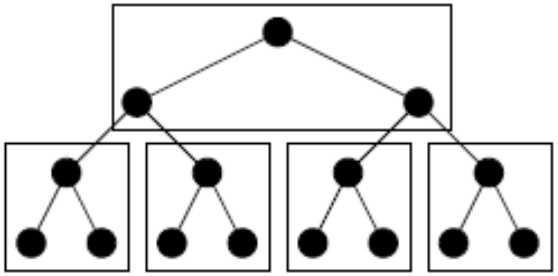

Figure: Breaking the BST into blocks. The BST is divided among disk blocks, each with space for three nodes. The path from the root to any leaf is contained on two blocks.

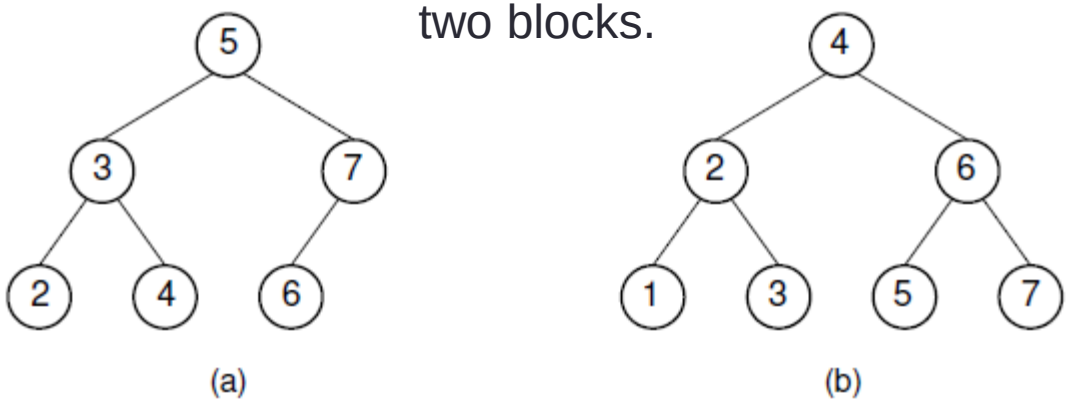

Figure: An attempt to re-balance a BST after insertion can be expensive. (a) A BST with six nodes in the shape of a complete binary tree. (b) A node with value 1 is inserted into the BST of (a). To maintain both the complete binary tree shape and the BST property, a major reorganization of the tree is required.

2-3 Tree

- 2-3 tree
  - ✓ A node contains one or two keys.
  - ✓ Every internal node has either two children (if it contains one key) or three children (if it contains two keys).
  - ✓ All leaves are at the same level in the tree (height balanced)
- Search tree property
  - ✓ For every node, the values of all descendants in the left subtree are less than the value of the first key, while values in the center subtree are greater than or equal to the value of the first key.
  - ✓ If there is a right subtree (equivalently, if the node stores two keys), then the values of all descendants in the center subtree are less than the value of the second key, while values in the right subtree are greater than or equal to the value of the second key.

2-3 Tree (Cont.)

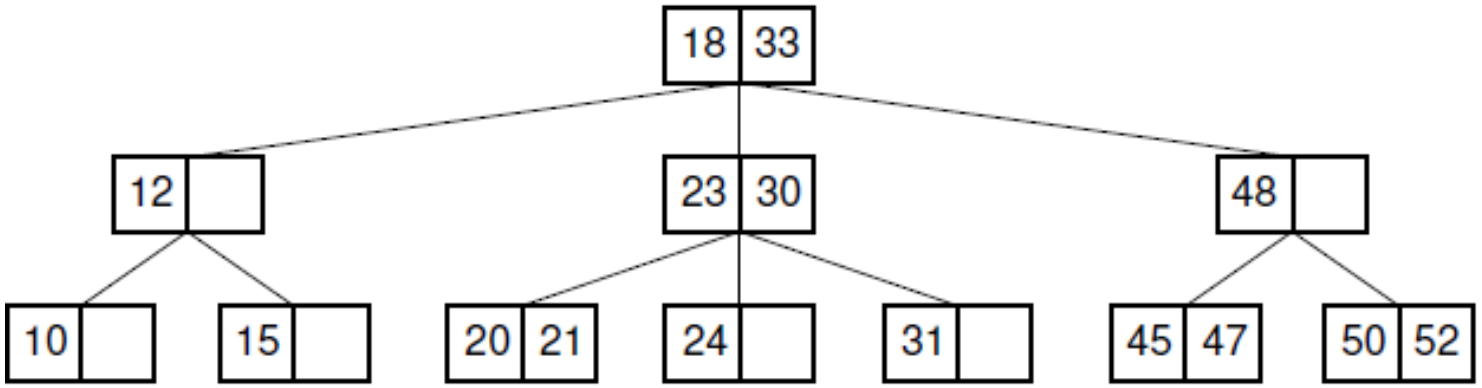

Figure: A 2-3 tree.

2-3 Tree (Cont.)

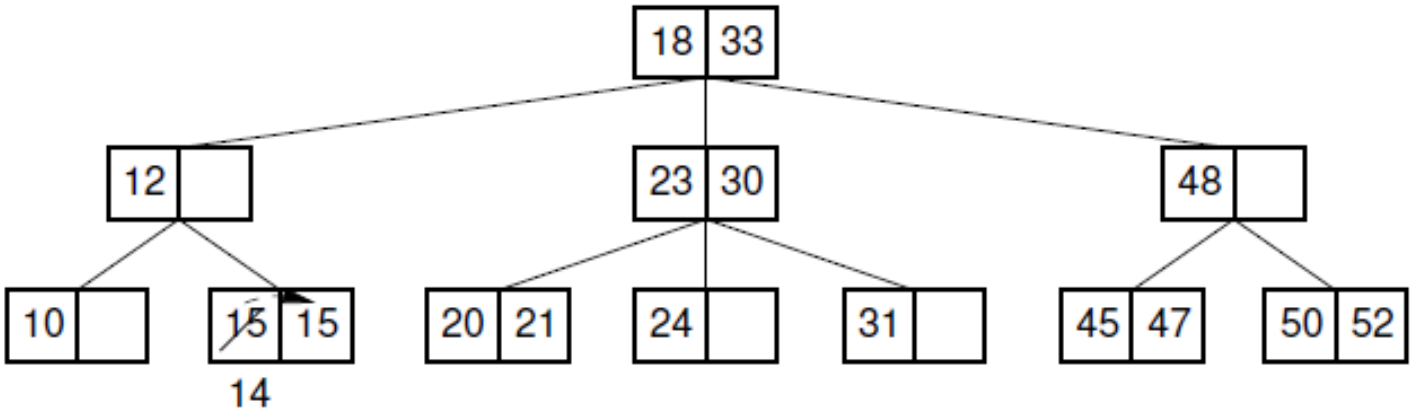

Figure: Simple insert into the 2-3 tree . The value 14 is inserted into the tree at the leaf node containing 15. Because there is room in the node for a second key, it is simply added to the left position with 15 moved to the right position.

2-3 Tree (Cont.)

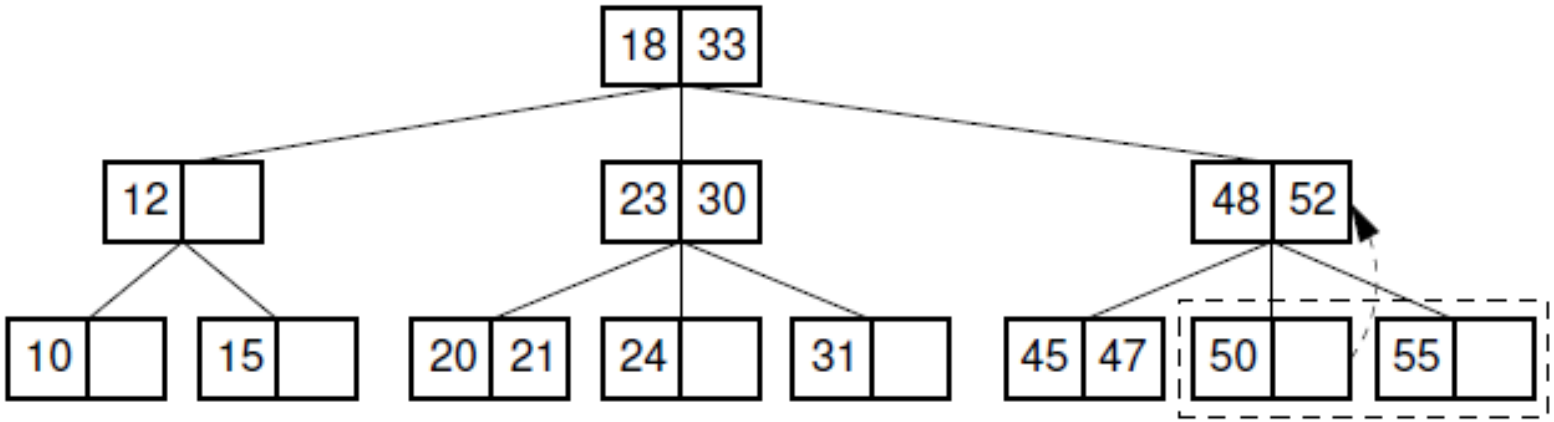

Figure: A simple node-splitting insert for a 2-3 tree. The value 55 is added to the 2-3 tree of Figure 10.9. This makes the node containing values 50 and 52 split, promoting value 52 to the parent node.

2-3 Tree (Cont.)

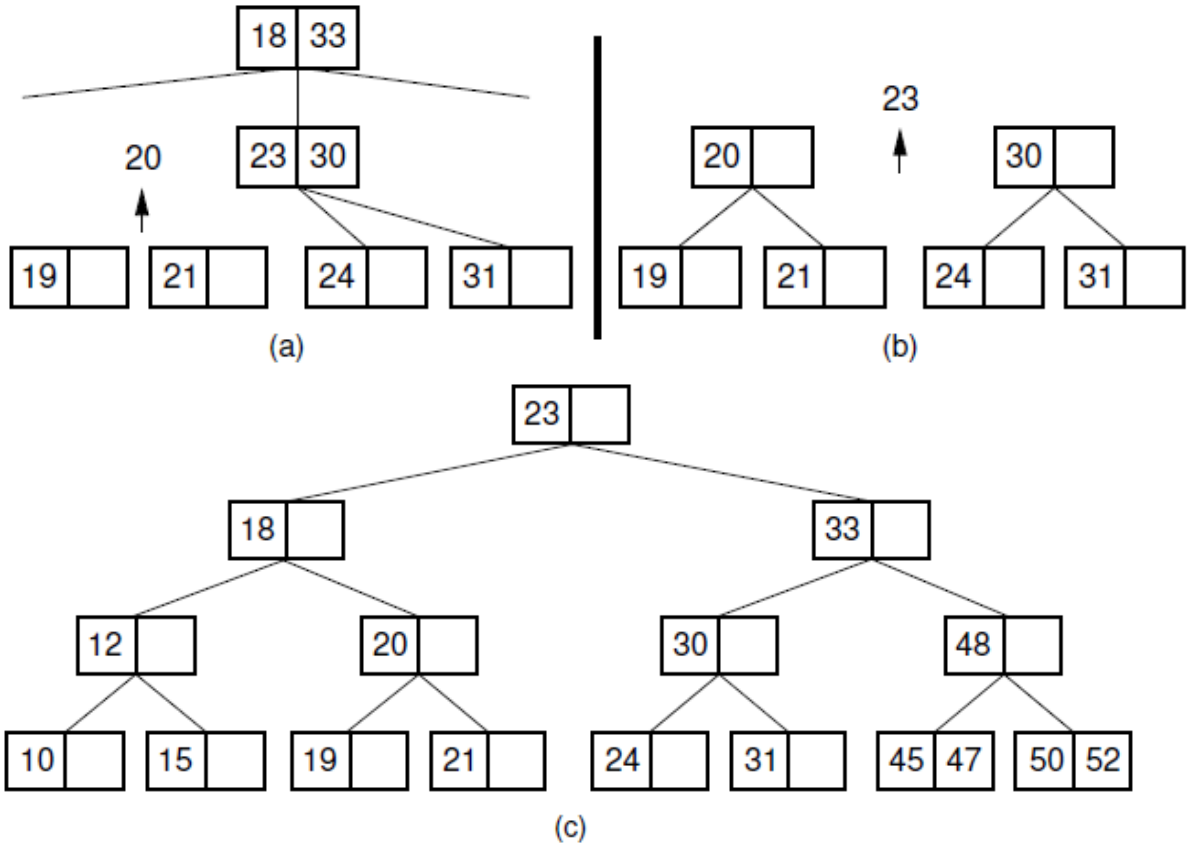

Figure: Example of inserting a record that causes the 2-3 tree root to split. (a) The value 19 is added to the 2-3 tree of Figure 10.9. This causes the node containing 20 and 21 to split, promoting 20. (b) This in turn causes the internal node containing 23 and 30 to split, promoting 23. (c) Finally, the root node splits, promoting 23 to become the left record in the new root. The result is that the tree becomes one level higher.

B-Tree

- Shape properties of B-tree of order m
  - ✓ The root is either a leaf or has at least two children.
  - ✓ Each internal node, except for the root, has between ⌈m/2⌉ and m children.
  - ✓ All leaves are at the same level in the tree, so the tree is always height balanced.

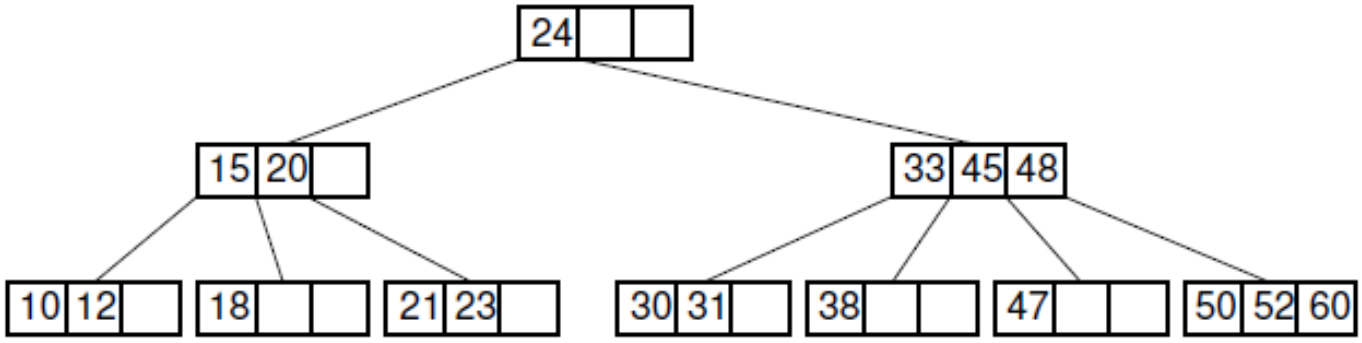

Figure: A B-tree of order four.

B+ Tree

- B+ Tree
  - ✓ Is a variant of the B-tree.
  - ✓ The B+-tree stores records only at the leaf nodes.
  - ✓ Internal nodes store key values.
  - ✓ Internal nodes are significantly different in structure from leaf nodes.

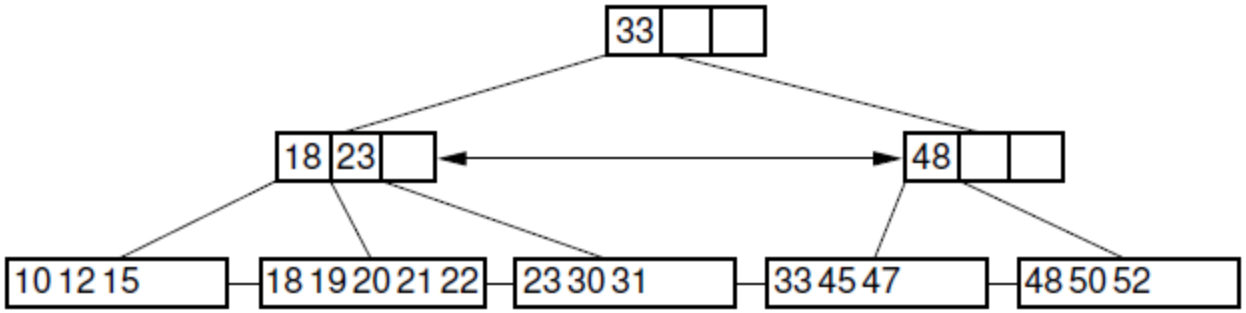

Figure: Example of a B+-tree of order four. Internal nodes must store between two and four children. For this example, the record size is assumed to be such that leaf nodes store between three and five records.

Next Week Lecture (Week 11)

Lecture 11: Graphs

Thank you!