Data Management and File Organization

Indexing

Topics

- Motivation
- Linear Indexing
- Tree Indexing
- B-Trees

Motivation

- Some file operations are very slow
- Example: Reading all records in order of an attribute may take several hours in a large file
- Sorting files can speed up file operations but still there are some problems

Problems with Pile Files

- Finding a record T_F, Finding next record in an order T_N, and Deleting a record T_D are very slow.

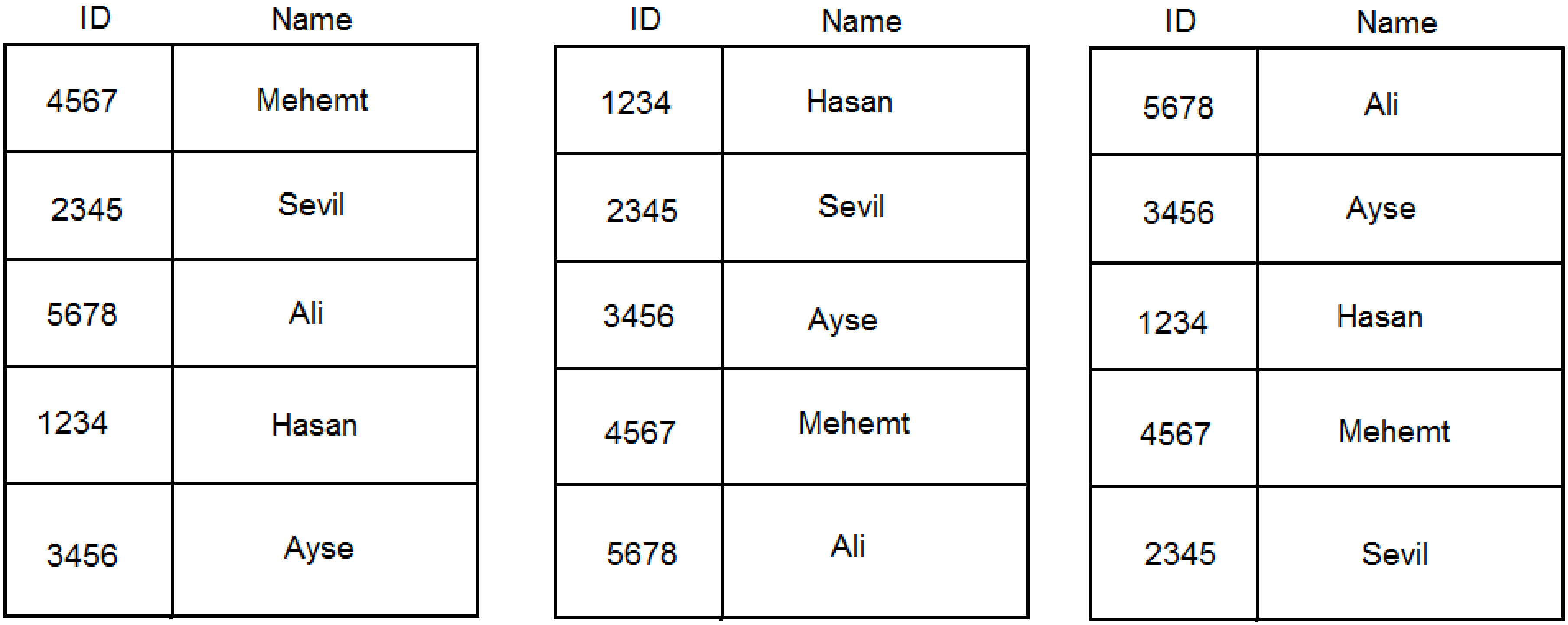
Problems with Sorted Sequential Files

- Sorting large files need external sorting which is slow compared to the internal sorting
- File will not remain sorted after new insertions
- Search using binary search needs $\text{Log}_2 n$ file access which is slow in large files.
    - Example: for a file with 16,000,000 records, 24 file access is needed

Problems with Sorted Sequential Files

- Files are sorted according to one attribute. Searches with other attributes need other copies of the file

Problems with Sorted Sequential Files



ID	Name
4567	Mehemt
2345	Sevil
5678	Ali
1234	Hasan
3456	Ayse

Pile File

ID	Name
1234	Hasan
2345	Sevil
3456	Ayse
4567	Mehemt
5678	Ali

Sorted by ID

ID	Name
5678	Ali
3456	Ayse
1234	Hasan
4567	Mehemt
2345	Sevil

Sorted by Name

Indexing

- Indexes are lookup tables for finding records quickly
- The simplest index is a list in order of the key values (linear indexing)

Case 1: Linear Indexing

- Linear indexing is a sorted list of keys and record locations
- Search is done in index list before going to the main file
- Linear indexing is suitable for small files

Example: Linear Indexing

ID	Name
4567	Mehemt
2345	Sevil
5678	Ali
1234	Hasan
3456	Ayse

Data File

ID (key)	location
1234	3
2345	1
3456	4
4567	0
5678	2

Index

Searching in an Indexed File

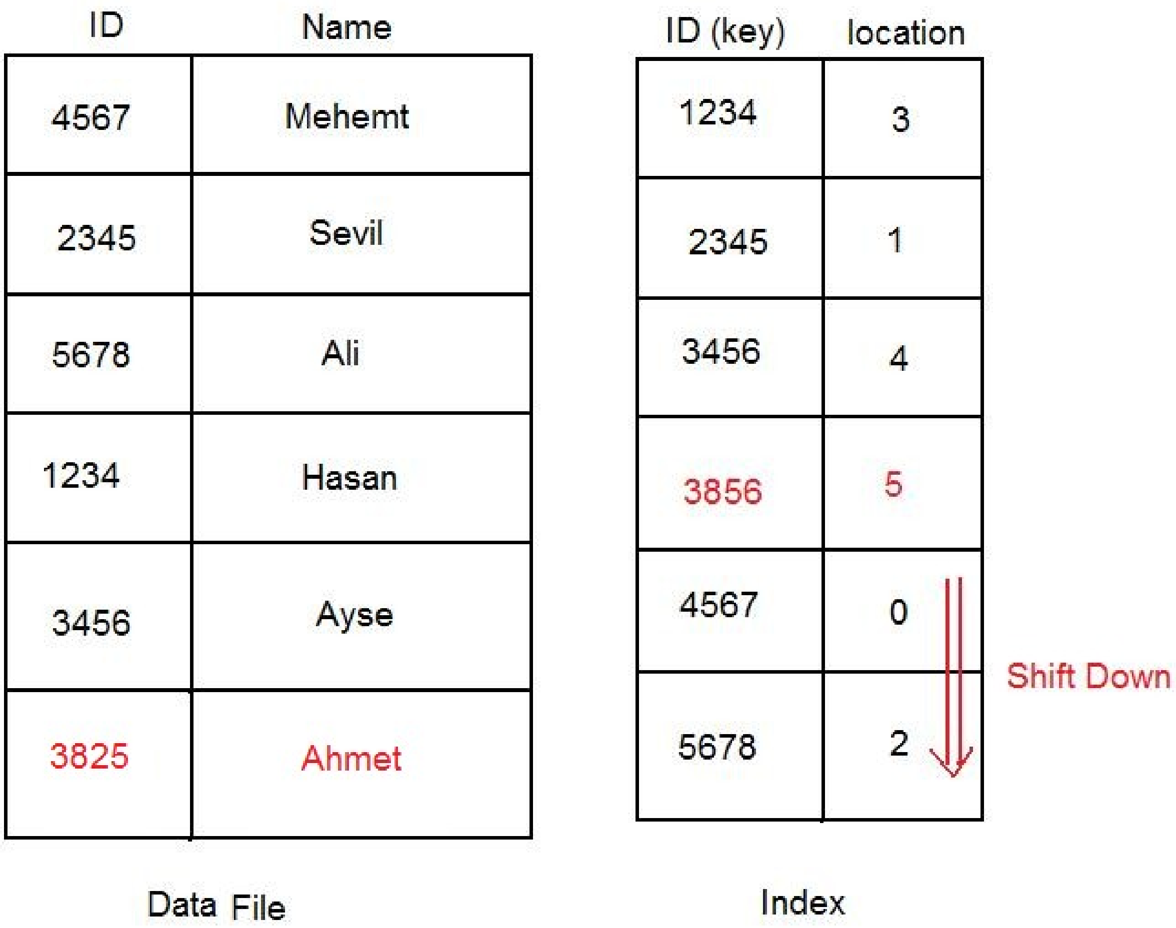
- Given a key value do:
  - Search the index list using binary search
  - Getting the location go to the block and read the record (s+r+btt)

- If the index list is in the memory, the search is fast

Insertion into an Indexed File

- Insertion is done at the end of the data file
- Add new key value to the index file. Then the index is updated to be sorted again

Example: Insertion into Indexed Files



ID	Name
4567	Mehemt
2345	Sevil
5678	Ali
1234	Hasan
3456	Ayse
3825	Ahmet

Data File

ID (key)	location
1234	3
2345	1
3456	4
3856	5
4567	0
5678	2

Shift Down

Index

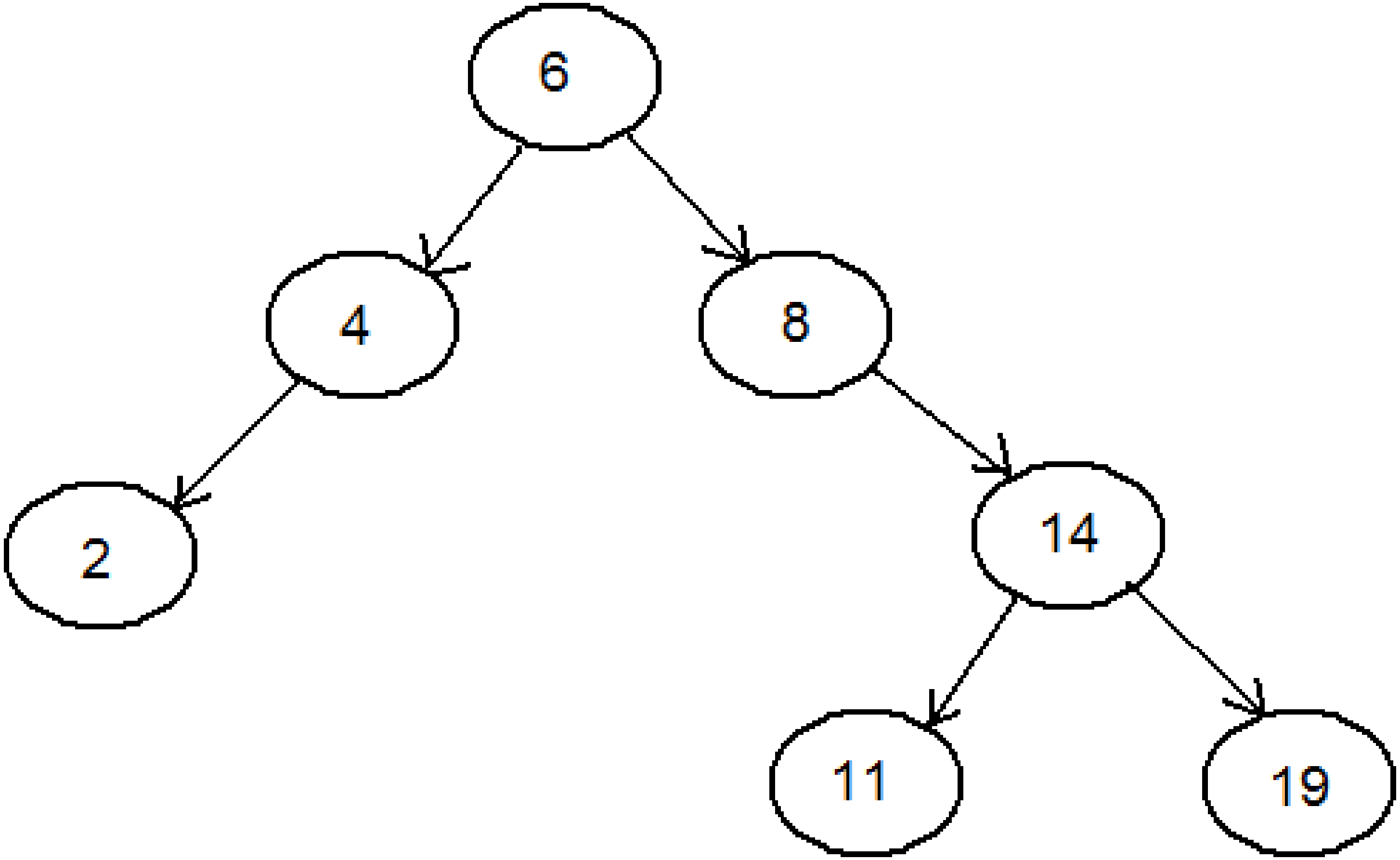
Case 2: Tree Indexing

- If the index list is larger than the memory, the search should be done in the file
- Searching the index file will be slow if it is a binary search ($\mathrm{Log}_2 n$)
- Tree indexes are used for faster search

Review: Binary Search Trees

- A Binary Search Tree (BST) is a
  - Binary Tree (at most two children at each node)
  - The value of the left child is less than the current node
  - The value of the right child is greater than the current node

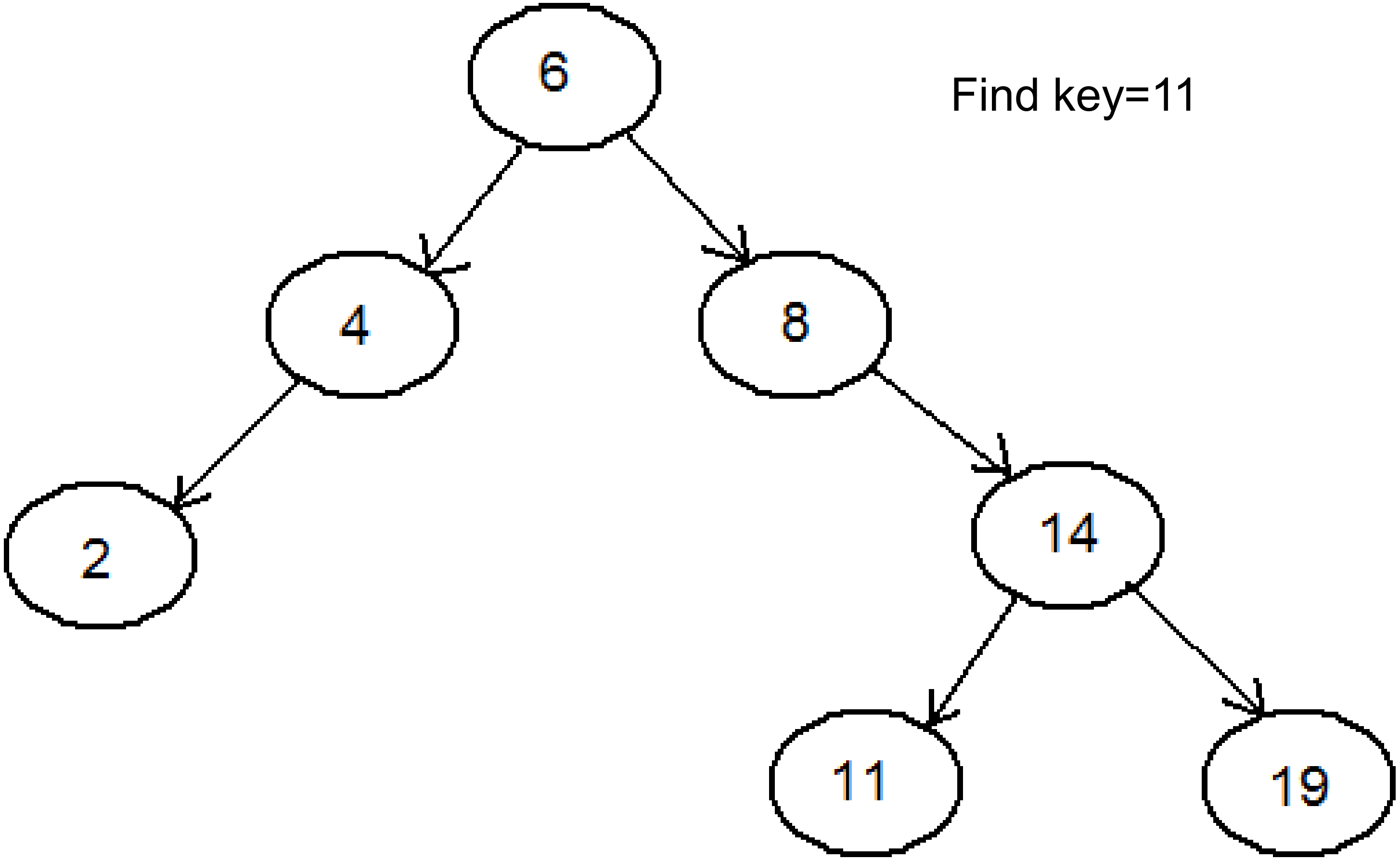
Example: BST



Search in BST

- Algorithm:
  - If key = value at the node
    - Return node
  - Else if key < value at the node
    - Search at the left sub tree (recursive call)
  - Else
    - Search at the right sub tree

Example: Search in a BST

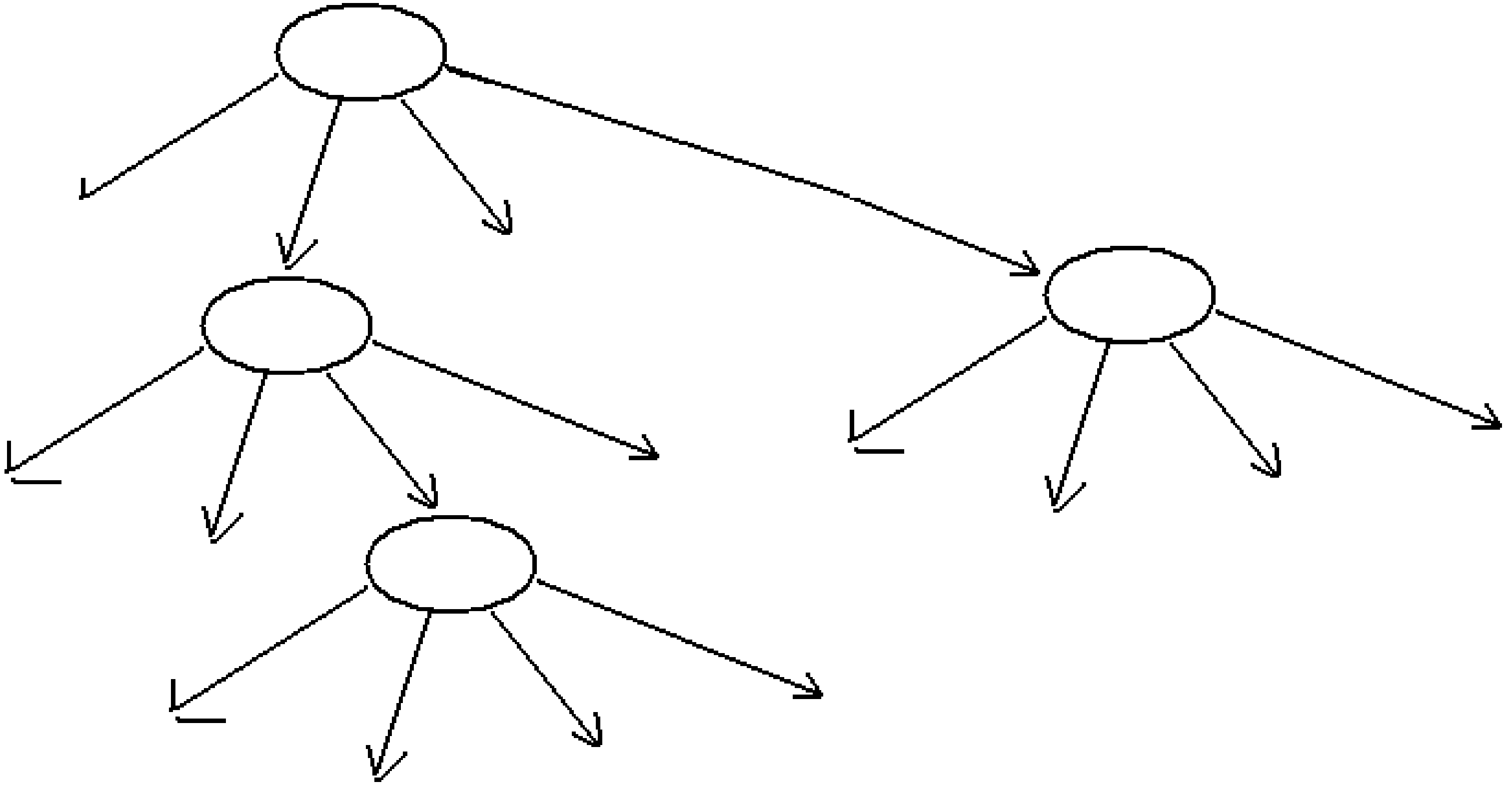


Find key=11

Trees with Higher Order

- If the height of the tree is large then the search will be slow(more I/O)
- For faster search we may have more children at each node. Ex: 8, 16 or 64 children at each node

Example Tree



Tree with 4 children at each node

B-Trees

- A tree with
  - Several children at each node
  - All leaves are at the same level

Nodes of a B-Tree

1. Internal Nodes

   Have 2N key values and 2N+1 pointers (order N)

2. Leaf nodes

   Keys and record locations

All nodes except the root should be at least half full

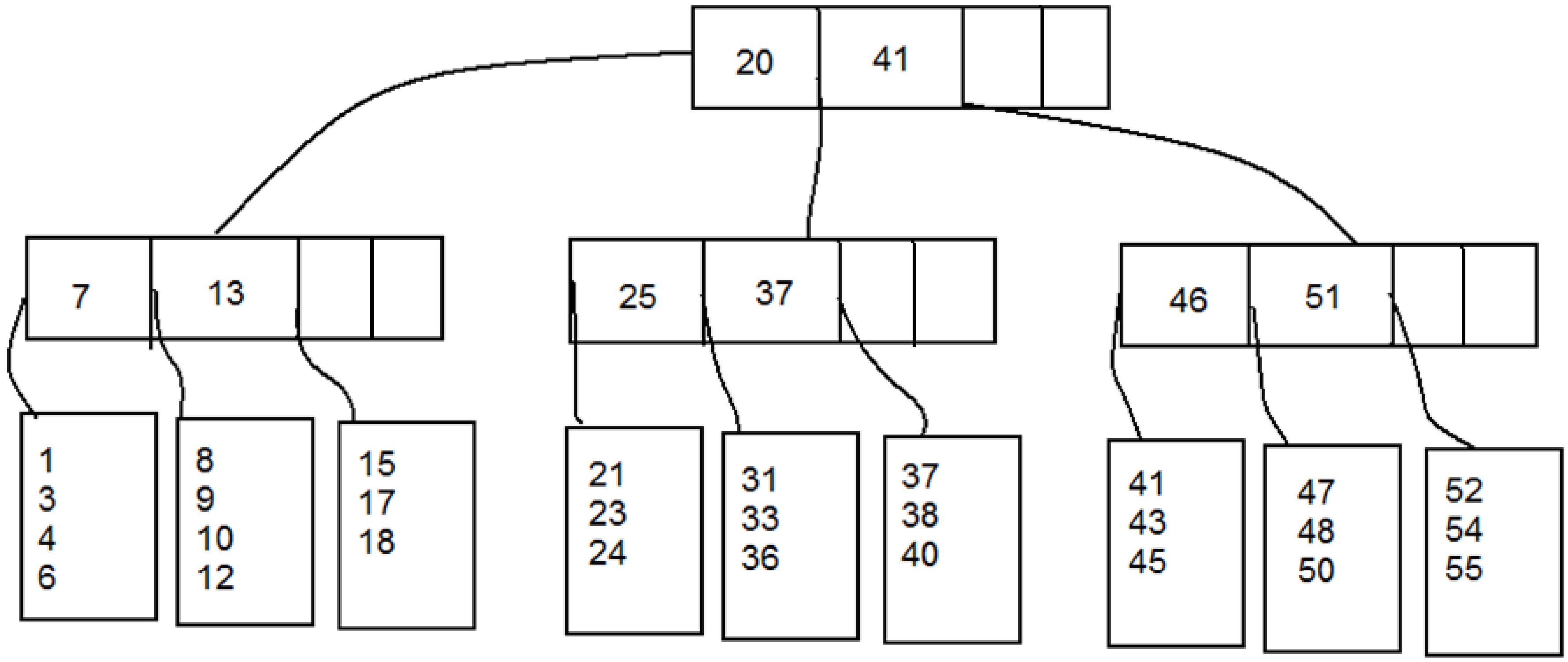
Example Internal Node

pointer1	Key1	pointer2	Key2	pointer3	Key3	pointer4	Key4	pointer5

Structure of a B-Tree

- If the attribute value is less than a key in internal node, it is stored at its left side leaf node
- Otherwise the attribute is compared with the next key in the internal node.

Example B-Tree



Operations on a B-Tree

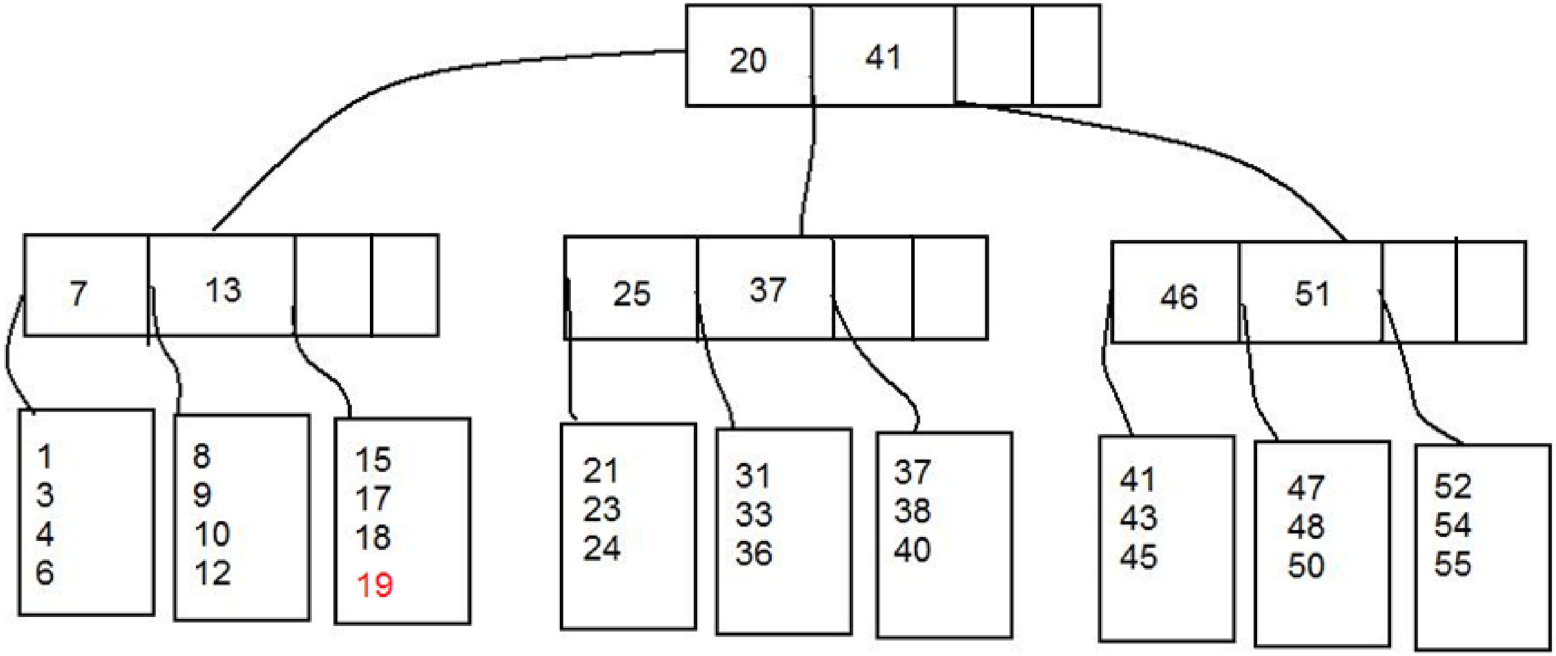
- Insertion
  - Insertion is done from the leaf nodes and the tree is updated. Nodes may split
- Deletion
  - Deletion is done from leaf nodes and nodes may merge

Insertion

- Algorithm
  - Using the key value of the data item, search the tree to a leaf node.
  - Insert the new data if the leaf node has enough space
  - Split the leaf node if there is no place to insert new data
  - Update tree

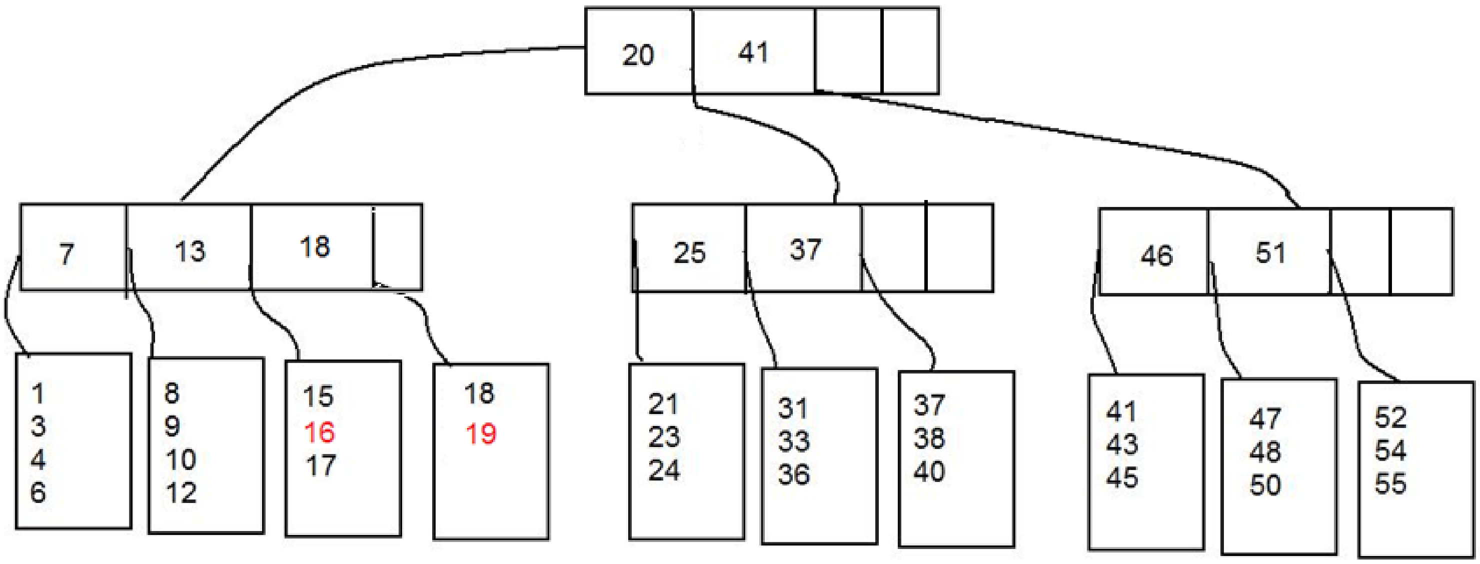
  Note: All nodes except the root should be at least half full

Insertion: Example



Insert 19

Insertion: Example



Insert 16

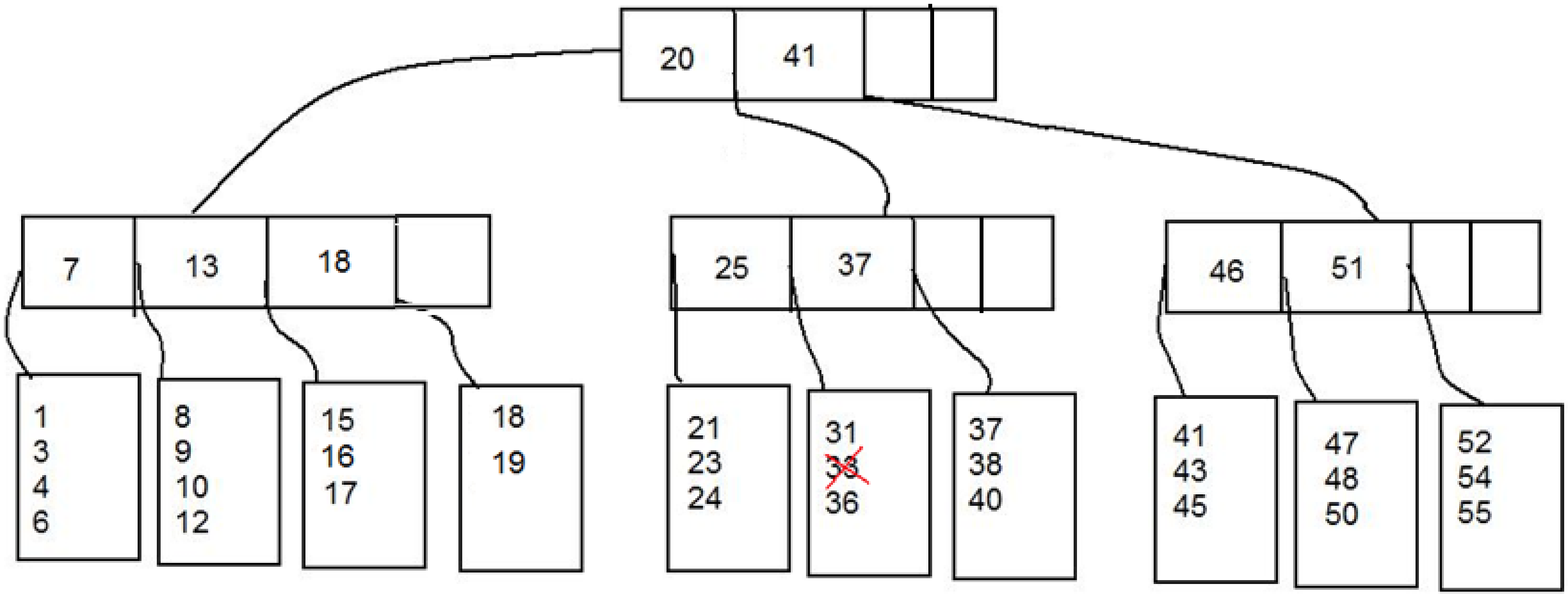
Deletion from a B-Tree

- Algorithm
  - Find the leaf node containing the data item
  - Remove the data item
  - If the leaf node is less than half full after deletion then
    - Merge with neighboring leaf nodes if they have space enough

    OR

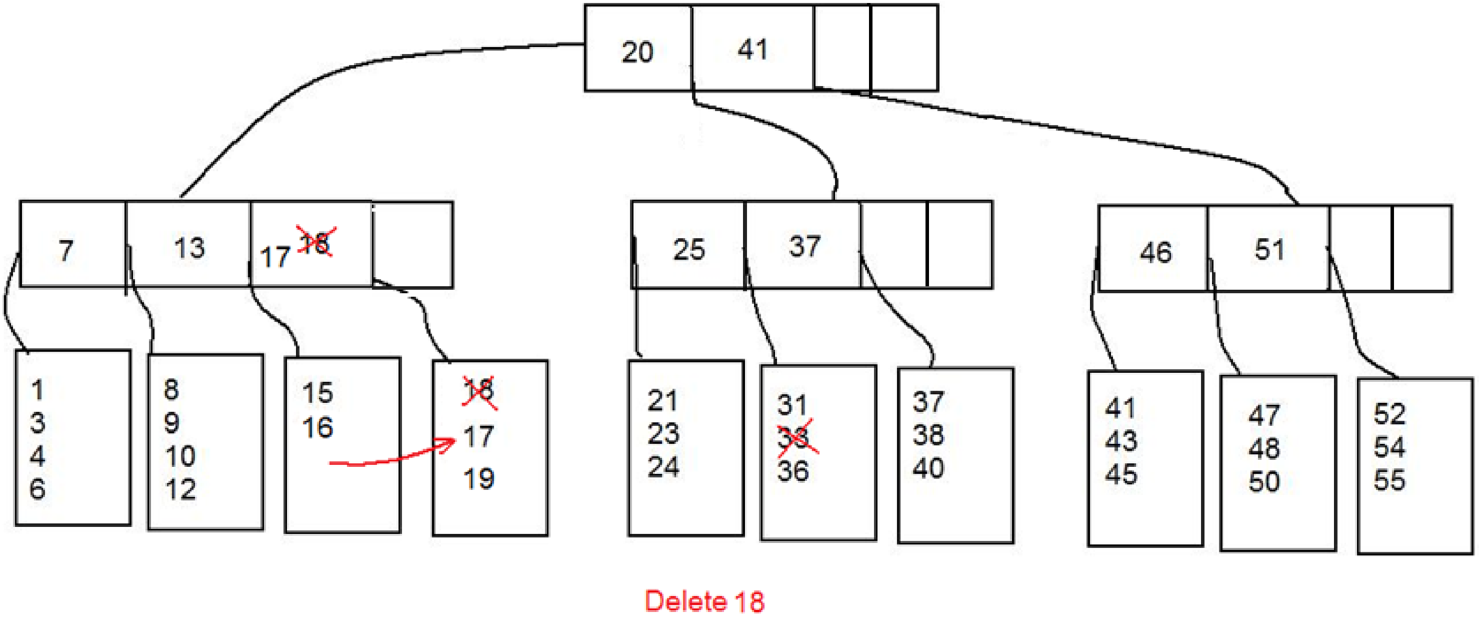
    - Re-distribute data by using data from the neighboring nodes
  - Update the tree

Example 1: Deletion

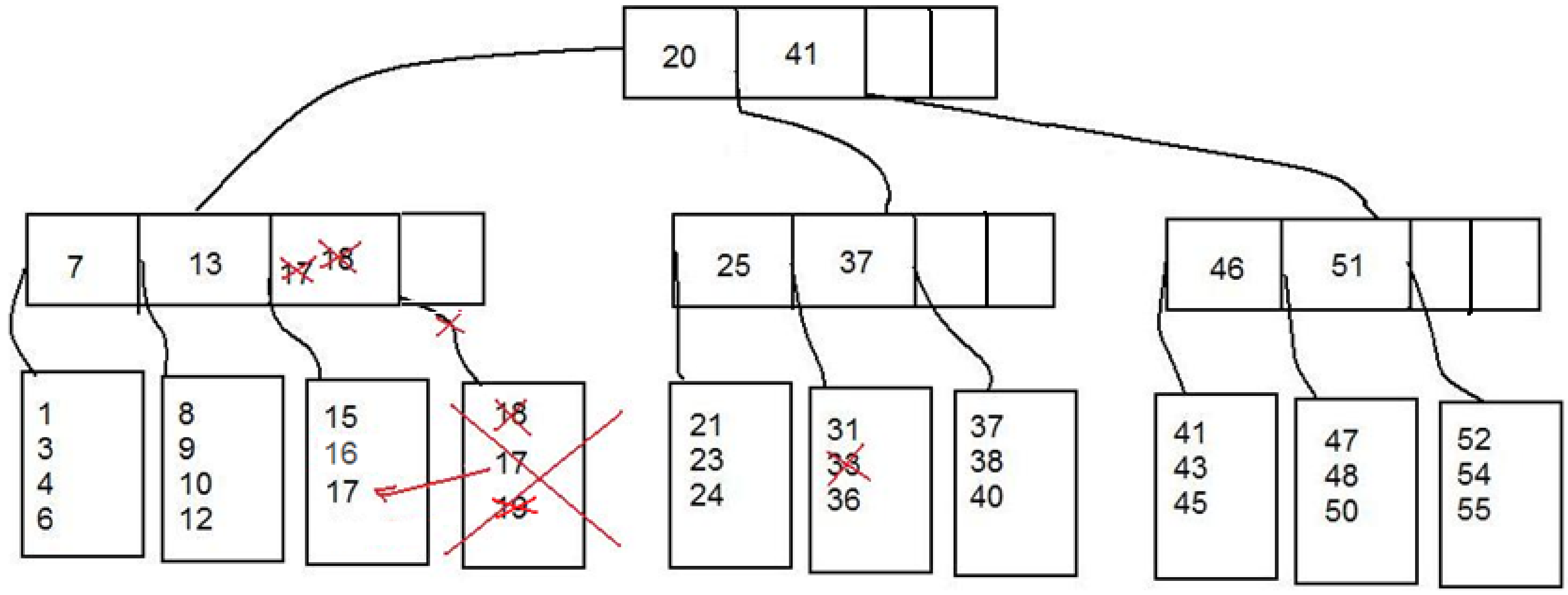


Delete 33

Example 2: Deletion (Re-distribute)



Example 3: Deletion (Merge)



Delete 19

Questions?