Unit IV

**AVL Trees, Definition, Heights of an AVL Tree, Operations-Insertion, Deletion and Searching. B-
Trees of order m, height of a B-Tree, insertion, deletion and searching, Comparison of Search Trees**

Graphs: Basic terminology, representations of graphs, graph search methods DFS, BFS. Text

**Processing: Pattern matching algorithms-Brute force, the Boyer Moore algorithm, the Knuth-Morris-
Pratt algorithm, Rabin Karp algorithm.**

AVL Trees

1. **Definition**:
   - **Self-Balancing Binary Search Trees**: Maintains a balance factor to ensure logarithmic time complexity for operations.
   - **Balanced Condition**: Heights of the left and right subtrees differ by at most one.
2. **Heights of an AVL Tree**:
   - **Balanced Property**: Ensuring the tree remains balanced.
   - **Balancing Operations**: Rotations to maintain balance during insertions and deletions.
3. **Operations**:
   - **Insertion**: Ensuring the balance factor remains within acceptable limits.
   - **Deletion**: Adjusting the tree while maintaining the balance.
   - **Searching**: Leveraging the tree's balanced nature for efficient searches.

B-Trees

1. **Definition of B-Trees of Order m**:
   - **Balanced Tree Structure**: A self-balancing tree designed for disk storage and databases.
   - **Properties**: Nodes can have more than two children, reducing disk access.
2. **Height of a B-Tree**:
   - **Optimal Height**: Ensuring an efficient tree structure for disk access.
3. **Operations**:
   - **Insertion**: Maintaining B-Tree properties while accommodating new elements.
   - **Deletion**: Adjusting the tree while maintaining the order and balance.

- **Searching**: Leveraging B-Tree properties for efficient searches in disk-based storage.

Comparison of Search Trees

1. **Comparative Analysis**:
    - **Efficiency Comparison**: Analyzing the performance of various search trees.
    - **Trade-offs**: Considering factors like insertion, deletion, search times, memory usage, etc.

Graphs

1. **Basic Terminology**:
    - **Vertices and Edges**: Elements and connections in a graph structure.
    - **Directed and Undirected Graphs**: Presence or absence of direction in edges.
2. **Representations of Graphs**:
    - **Adjacency Matrix**: Representing connections between vertices.
    - **Adjacency List**: Storing connections as linked lists.
3. **Graph Search Methods**:
    - **Depth-First Search (DFS)**: Traversing graph structures depth-wise.
    - **Breadth-First Search (BFS)**: Exploring nodes level by level.

Text Processing

1. **Pattern Matching Algorithms**:
    - **Brute Force**: Directly comparing a pattern with substrings.
    - **Boyer Moore Algorithm**: Utilizes heuristic to jump in the search space.
    - **Knuth-Morris-Pratt Algorithm**: Utilizes a pattern's own information to avoid unnecessary comparisons.
    - **Rabin Karp Algorithm**: Utilizes hashing for efficient pattern searching.

1. AVL Trees

Definition and Properties:

AVL Trees are self-balancing binary search trees where the height difference between the left and right subtrees (balance factor) of any node is at most 1.

Heights of an AVL Tree:

The height of an AVL tree is approximately log base 2 (n), where n is the number of nodes in the tree.

Operations: Insertion, Deletion, Searching

Insertion: Ensures the tree remains balanced by performing rotations to maintain AVL properties.

Deletion: Balances the tree by performing rotations after deletion.

Searching: Follows the standard BST search algorithm.

```
// Example: AVL Tree operations
class AVLTree {
public:
void insert(int value) { /* ... */ }
void remove(int value) { /* ... */ }
bool search(int value) { /* ... */ }
};
```

2. B-Trees of Order m

Definition and Properties:
B-Trees are balanced tree structures with a variable number of children per node, designed to work well with secondary storage.

Height of a B-Tree:
The height of a B-Tree is logarithmic and depends on the number of keys and the order of the tree.

Operations: Insertion, Deletion, Searching
Insertion: Maintains B-Tree properties by redistributing keys and splitting nodes if necessary.
Deletion: Ensures B-Tree properties are preserved by merging nodes or redistributing keys.
Searching: Uses a similar mechanism as in BSTs but traverses multiple levels due to multiple children.

3. Comparison of Search Trees

Comparison Factors:

Comparison of search trees involves analyzing factors like the average case and worst-case time complexity of operations (searching, insertion, deletion), memory usage, and structural properties.

4. Graphs

Basic Terminology and Representations:
Graphs consist of vertices (nodes) and edges (connections between nodes).

Graph Search Methods: DFS, BFS
DFS (Depth-First Search): Traverses as far as possible along each branch before backtracking.
BFS (Breadth-First Search): Explores all the vertices at the present depth before moving on to the vertices at the next depth level.

4. **Text Processing: Pattern Matching Algorithms**

Brute Force, Boyer Moore, Knuth-Morris-Pratt, Rabin-Karp
Brute Force: Compares the pattern to each substring of the text.
Boyer Moore: Skips comparisons based on a "bad character" heuristic.
Knuth-Morris-Pratt: Utilizes a "failure function" to skip unnecessary comparisons.
Rabin-Karp: Uses hashing to compare the pattern with substrings of the text.