3.Tress

1. Basic Terminology of Trees:

Trees: Hierarchical data structures composed of nodes connected by edges, with a single root node at the top.

Nodes: Elements within a tree containing data and references to child nodes.

Root: The topmost node of a tree, serving as the starting point for traversals.

Parent, Child, Sibling: Relationships between nodes where a parent node has child nodes, and sibling nodes share the same parent.

2. Binary Trees:

Trees where each node has at most two children: left and right.

Efficient for organizing hierarchical data and searching algorithms.

3. Binary Tree Representation:

Array: Using arrays to represent binary trees, where each index corresponds to a node, and relationships are determined by indices.

Linked List: Nodes with left and right pointers, enabling dynamic allocation of memory for tree nodes.

4. Algebraic Expressions:

Binary trees can represent algebraic expressions efficiently, with operators as internal nodes and operands as leaves.

Useful for evaluating and simplifying mathematical expressions.

5. Complete Binary Tree:

A binary tree where every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

Efficient for storage and retrieval due to its predictable structure.

6. Extended Binary Trees:

Binary trees with additional links (threads) for efficient traversal without recursion.

Threads connect nodes based on specific rules, enhancing traversal speed.

7. Traversing Binary Trees:

Inorder: Traverse left subtree, visit node, traverse right subtree.

Preorder: Visit node, traverse left subtree, traverse right subtree.

Postorder: Traverse left subtree, traverse right subtree, visit node.

These traversal methods enable processing of nodes in different orders.

8. Threaded Binary Trees:

Extra pointers added to nodes for efficient traversal without recursion.

Threads link nodes based on their order of traversal, allowing for faster traversal operations.

9. Huffman Algorithm:

Lossless Data Compression technique for constructing variable-length codes based on character frequencies.

Utilizes binary trees to create efficient encoding schemes, reducing overall data size.

10. Binary Search Tree (BST):

Property: Left child < parent < right child for every node.

Insertion and Deletion: Maintain the BST property while adding or removing nodes.

Path Length: Length of the path from the root to a node, impacting search performance.

11. AVL Trees:

Self-balancing Binary Search Trees ensuring height balance for efficient search operations.

AVL trees automatically adjust their structure to maintain balance, optimizing performance.

12. B-trees:

Balanced Trees used for organizing data efficiently, especially in databases and filesystems.

B-trees maintain balance and minimize height, enhancing data retrieval and storage operations.