ABHYASMITRA.IN

COMPREHENSIVE UNIVERSITY STUDY NOTES

DATA STRUCTURES

Unit IV — Non-Linear Data Structures: Trees, BST, TBST, OBST, AVL & Heaps

---

Topics Covered in this Unit

Tree Terminology & Representations: Root, Depth, Height, Degree, Left-Child Right-Sibling (LCRS) & Linked Representations

Binary Tree Foundations: Types (Full, Complete, Perfect, Skewed) & Mathematical Theorems (Leaves $L = I_2 + 1$ Proof)

Traversals: Preorder, Inorder, Postorder (Recursive & Stack Iterative), Level-Order & Unique Tree Reconstruction

Binary Search Tree (BST): Definition, Operations (Search, Insert, Delete 3 Cases), Time Complexities

Threaded Binary Search Tree (TBST): Concept, Null Pointer Utilization, Single & Double Threading, Stack-less Traversals

Optimal Binary Search Tree (OBST): Probability Models (p_i, q_i), Cost Formula, Dynamic Programming Recurrence & Construction

Height Balanced AVL Trees: Balance Factor, Single Rotations (LL, RR), Double Rotations (LR, RL), Insertion & Deletion

Heap Trees & Heap Sort: Min/Max Heaps, Array Mapping, O(n) Floyd's Build-Heap, Heapify-up/down, In-Place Heap Sort

High-Yield University Exam Question Bank with Detailed Model Answers & Unit Summary

Compiled in a structured, easy-to-understand, textbook style format
Specially curated for B.Tech / B.E. / BCA / MCA / B.Sc Computer Science & Engineering Students

COURSE SYLLABUS & MAPPING

[UNIT IV SYLLABUS — PRESCRIBED UNIVERSITY CURRICULUM]

This section contains the official syllabus for Unit IV. Verify with your university curriculum mapping.

Unit IV Syllabus Breakdown:

IV Non-linear Data Structure: Tree: Tree: Introduction of tree, Representations, Traversals, Binary tree, Binary search tree, Threaded Binary search tree- concepts, threading, insertion and deletion of nodes, Optimal Binary Search Tree (OBST), Height Balanced TreeAVL tree, Heap Tree

• Course Code: _____ • Semester / Year: _____

TABLE OF CONTENTS

4.1 Tree Terminology & Representations

Definition: Tree (Non-Linear Hierarchical Structure)

A Tree is a non-linear, hierarchical data structure consisting of a finite set T of one or more nodes such that:

1. There is a specially designated node called the ROOT.

2. The remaining nodes are partitioned into $n \geq 0$ disjoint subsets $T_1, T_2, ..., T_n$, each of which is itself a tree, called a Subtree of the root.

Key Tree Terminology

- **1. Root:** The topmost ancestor node in a tree with in-degree 0 (no incoming edges).
- **2. Edge:** The directed connecting link from a parent node to its child. A tree with n nodes always has exactly n - 1 edges.
- **3. Leaf / Terminal Node:** A node with degree 0 (no children).
- **4. Degree of a Node:** The total number of children belonging to that node.
- **5. Degree of a Tree:** The maximum degree among all nodes in the tree.
- **6. Level of a Node:** Distance from the root. Root is at Level 0 (or 1), children of root at Level 1, and so on.
- **7. Height / Depth of a Tree:** The maximum level of any node in the tree (length of the longest path from root to leaf).
- **8. Forest:** A set of zero or more disjoint trees obtained by removing the root node of a tree.

Tree Representations in Memory

- **1. Left-Child Right-Sibling (LCRS) Representation:** Standard way to represent an arbitrary general multi-way tree as a binary tree. Each node has two pointers: leftChild (points to first/eldest child) and rightSibling (points to immediate right sibling).
- **2. Linked Representation:** Each node stores data and an array/list of pointers to its child nodes.

4.2 Binary Trees: Types, Properties & Theorems

Definition: Binary Tree

A Binary Tree is a finite set of nodes that is either empty (NULL) or consists of a root node and two disjoint binary trees called the Left Subtree and the Right Subtree. Maximum degree of any node is at most 2.

Special Types of Binary Trees

Binary Tree Type	Formal Structural Definition	Key Mathematical Characteristics
Full / Proper Binary Tree	Every internal node has exactly TWO children (0 or 2 children).	Number of leaf nodes L = Number of internal nodes I + 1.
Complete Binary Tree	All levels are completely filled except possibly the last, which is filled from left to right.	Height h = floor($\log_2$ n). Ideal for array-based Binary Heaps.
Perfect Binary Tree	All internal nodes have 2 children and all leaf nodes are at the same level.	Total nodes n = 2^(h+1) - 1, Total leaves L = 2^h.
Skewed Binary Tree	Every node has only one child (either all left or all right).	Degenerates into a Linked List with worst-case height h = n - 1.
Balanced Binary Tree	Height of left and right subtrees of every node differs by at most 1.	Guarantees O(log n) height (e.g., AVL tree, Red-Black tree).

Theorem & Proof: Relation Between Leaf Nodes and Degree-2 Nodes

Theorem: In any non-empty binary tree T, if n_0 is the number of leaf nodes (degree 0) and n_2 is the number of nodes of degree 2, then: $n_0 = n_2 + 1$

Proof:

Let n be total nodes, e be total edges, and n_1 be nodes of degree 1.

1. Total nodes $n = n_0 + n_1 + n_2$ --- (Equation 1)

2. In any tree, total edges e = n - 1. Also, each node of degree 1 produces 1 edge and each node of degree 2 produces 2 edges:

$e = 1 \cdot n_1 + 2 \cdot n_2 \Rightarrow n - 1 = n_1 + 2n_2 \Rightarrow n = n_1 + 2n_2 + 1$ --- (Equation 2)

3. Equating (1) and (2):

$n_0 + n_1 + n_2 = n_1 + 2n_2 + 1$

$n_0 = n_2 + 1$ [Q.E.D.]

4.3 Binary Tree Traversals & Tree Reconstruction

1. Depth-First Traversals

- **a. Preorder Traversal (DLR):** Visit Root -> Traverse Left Subtree -> Traverse Right Subtree. Used for prefix expressions, cloning trees.
- **b. Inorder Traversal (LDR):** Traverse Left Subtree -> Visit Root -> Traverse Right Subtree. In BST, yields keys in strictly ascending sorted order.
- **c. Postorder Traversal (LRD):** Traverse Left Subtree -> Traverse Right Subtree -> Visit Root. Used for deleting tree bottom-up, postfix expressions.

2. Breadth-First Traversal (Level-Order)

Visits nodes level by level from top to bottom (Level 0, Level 1, ...) and left to right within each level. Implemented iteratively using a FIFO Queue.

3. Reconstruction of Unique Binary Tree

Reconstruction Rules

• *To construct a UNIQUE Binary Tree, INORDER traversal MUST be provided along with either PREORDER or POSTORDER traversal.*

• *Preorder/Postorder identifies the ROOT node (Preorder root is at start; Postorder root is at end).*

• *Inorder splits elements into Left Subtree (elements before root) and Right Subtree (elements after root).*

TREE RECONSTRUCTION TRACE

```
  Given: Inorder  = [ D, B, E, A, F, C ]
         Preorder = [ A, B, D, E, C, F ]

  Step 1: Preorder root is 'A'.
  Step 2: In Inorder, 'A' splits: Left subtree = [D, B, E], Right subtree = [F, C].
  Step 3: In Left subtree, Preorder gives 'B' as root; Inorder gives 'D' (left) and 'E'
(right).
  Step 4: In Right subtree, Preorder gives 'C' as root; Inorder gives 'F' (left).

  Constructed Tree:
          A
        /   \
       B     C
      / \   /
     D   E F
```

Figure: Step-by-step reconstruction of unique binary tree from Inorder and Preorder traversals

4.4 Binary Search Tree (BST): Operations & Analysis

Definition: Binary Search Tree (BST)

A Binary Search Tree (BST) is a binary tree where each node key satisfies the BST Property:

1. All keys in the Left Subtree are strictly LESS THAN (<) the root key.

2. All keys in the Right Subtree are strictly GREATER THAN (>) the root key.

3. Both Left and Right subtrees are themselves Binary Search Trees.

BST Deletion: Three Fundamental Cases

- **Case 1: Deleting a Leaf Node (Degree 0):** Simply set parent pointer pointing to this node to NULL and free memory.
- **Case 2: Deleting a Node with ONE Child (Degree 1):** Bypass the node by pointing parent directly to the single child node, then free target node.
- **Case 3: Deleting a Node with TWO Children (Degree 2):** Find Inorder Successor (minimum key in right subtree) or Inorder Predecessor (maximum key in left subtree). Copy its value into the target node, then delete that successor/predecessor node (which falls into Case 1 or 2).

4.5 Threaded Binary Search Tree (TBST)

Concept & Motivation of TBST

In a binary tree with n nodes, there are 2n pointer fields, out of which exactly n + 1 are NULL (wasted space!).

A Threaded Binary Search Tree replaces NULL pointers with THREADS (pointers to Inorder Predecessors and Inorder Successors), enabling stack-less and recursion-free tree traversals without extra memory overhead.

Node Structure & Threading Types

- • **Inorder Right Threaded:** NULL right pointers store pointer to Inorder Successor.
- • **Inorder Left Threaded:** NULL left pointers store pointer to Inorder Predecessor.
- • **Fully (Double) Threaded Binary Tree:** Both left and right NULL pointers are threaded.
- • **Tag Bits:** lbit = 0 (lchild is thread) vs. 1 (lchild is actual left child); rbit = 0 (rchild is thread) vs. 1 (rchild is actual right child).

DOUBLE THREADED BINARY SEARCH TREE (TBST)

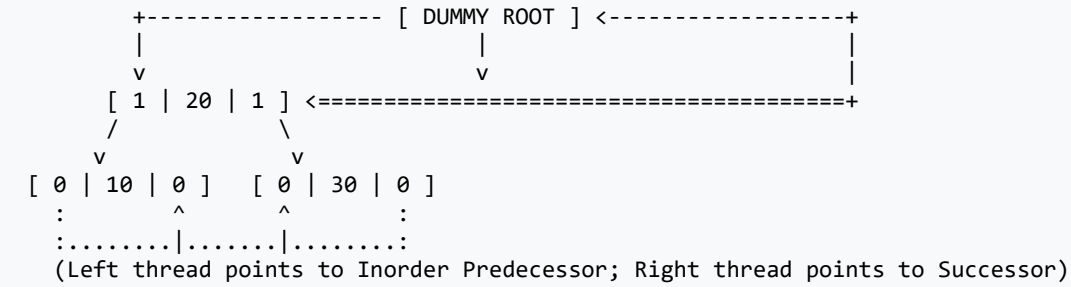

Figure: Threaded Binary Search Tree with boolean tag flags and bidirectional inorder threads

4.6 Optimal Binary Search Tree (OBST)

Definition: Optimal Binary Search Tree (OBST)

An Optimal Binary Search Tree (OBST) is a BST constructed for a given set of sorted keys $K = \{k_1, k_2, ..., k_n\}$ with known access probabilities $P = \{p_1, p_2, ..., p_n\}$ and dummy unsearched ranges $D = \{d_0, d_1, ..., d_n\}$ with probabilities $Q = \{q_0, q_1, ..., q_n\}$, such that the Total Expected Search Cost E(T) is MINIMIZED.

Expected Cost Formula: $E(T) = \sum [depth(k_i) + 1] \cdot p_i + \sum [depth(d_i)] \cdot q_i$

Dynamic Programming Recurrence Relations

Using Bellman's Principle of Optimality, the cost matrix c(i, j) and weight matrix w(i, j) for subtrees spanning keys k_{i+1} to k_j are computed as:

- **1. Weight Recurrence:** $w(i, j) = p_j + q_j + w(i, j - 1)$, with base case $w(i, i) = q_i$.
- **2. Cost Recurrence:** $c(i, j) = \min_{i < r \le j} [c(i, r - 1) + c(r, j)] + w(i, j)$, with base case $c(i, i) = 0$.
- **3. Root Matrix r(i, j):** Stores the optimal root index r that minimizes c(i, j). Time Complexity = $O(n^3)$ (or $O(n^2)$ using Knuth's optimization).

4.7 Height Balanced AVL Trees: Rotations & Rebalancing

Definition: AVL Tree & Balance Factor

An AVL Tree (Adelson-Velsky and Landis) is a self-balancing Binary Search Tree where for EVERY node, the Balance Factor (BF) satisfies:

Balance Factor BF(node) = Height(Left Subtree) - Height(Right Subtree) ∈{-1, 0, +1}

If an insertion or deletion causes BF(node) to become +2 or -2, rebalancing is performed using Rotations.

Four Rebalancing Rotations

Rotation Type	Imbalance Cause	Correction Mechanism	Resulting Subtree Height
LL Rotation (Single Right)	Node inserted into Left subtree of Left child (BF = +2, Left child BF = +1).	Perform single RIGHT rotation around unbalanced node.	Height restored to pre-insertion.
RR Rotation (Single Left)	Node inserted into Right subtree of Right child (BF = -2, Right child BF = -1).	Perform single LEFT rotation around unbalanced node.	Height restored to pre-insertion.
LR Rotation (Double Rotation)	Node inserted into Right subtree of Left child (BF = +2, Left child BF = -1).	Perform LEFT rotation on Left child, then RIGHT rotation on node.	Height restored to pre-insertion.
RL Rotation (Double Rotation)	Node inserted into Left subtree of Right child (BF = -2, Right child BF = +1).	Perform RIGHT rotation on Right child, then LEFT rotation on node.	Height restored to pre-insertion.

AVL ROTATION VISUALIZATION

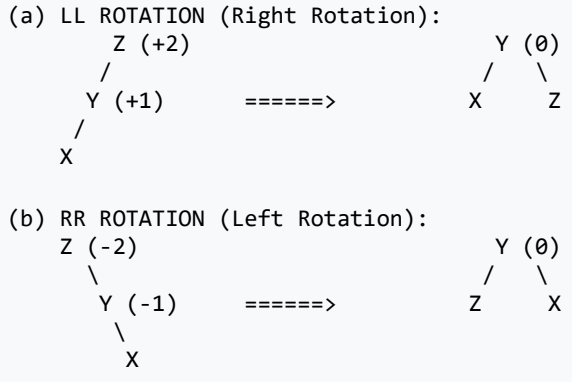

Figure: Single Right (LL) and Single Left (RR) rotations restoring AVL balance factor to 0

4.8 Heap Trees: Binary Heaps & Heap Sort

Definition: Binary Heap

A Binary Heap is a Complete Binary Tree stored in a contiguous array that satisfies the Heap Property:

- *Max Heap: Key(parent) ≥ Key(children) for all nodes (Root contains MAXIMUM key).*
- *Min Heap: Key(parent) ≤ Key(children) for all nodes (Root contains MINIMUM key).*

Array Representation of Binary Heap (0-Indexed)

- **• Parent of node at index i:** parent(i) = floor((i - 1) / 2)
- **• Left Child of node at index i:** left(i) = 2i + 1
- **• Right Child of node at index i:** right(i) = 2i + 2

Heap Operations & Heap Sort

- **1. Insertion (Heapify-Up):** Insert at end (next complete slot), compare with parent and swap upwards until heap property holds. Time: O(log n).
- **2. Delete-Max / Extract-Root (Heapify-Down):** Replace root with last element, decrease heap size, and push element downwards by swapping with larger child. Time: O(log n).
- **3. Floyd's Linear Build-Heap Algorithm:** Applies heapify-down from the last internal node (index n/2 - 1) down to index 0. Total time: O(n) (sum of heights $\sum h/2^h = 2$).
- **4. Heap Sort:** Build Max Heap in O(n) time. Repeatedly swap root (max) with last element and restore heap of size k - 1 via heapify-down. Time: O(n log n) in all cases, Space: O(1) in-place.

4.9 High-Yield University Exam Question Bank

Part A: Short Answer Questions (2 to 5 Marks)

Q1. Prove that in any non-empty binary tree, number of leaf nodes $n_0 = n_2 + 1$. [4 Marks]

- **Answer:** Let total nodes $n = n_0 + n_1 + n_2$. Total edges $e = n - 1$. Since edges come from nodes of degree 1 (1 edge) and degree 2 (2 edges), $e = n_1 + 2n_2$. Substituting $n - 1 = n_1 + 2n_2$ gives $n = n_1 + 2n_2 + 1$. Equating the two expressions for n: $n_0 + n_1 + n_2 = n_1 + 2n_2 + 1 \Rightarrow n_0 = n_2 + 1$.

Q2. Why is Inorder traversal mandatory along with Preorder/Postorder for unique binary tree reconstruction? [3 Marks]

- **Answer:** Preorder/Postorder identifies the root node but cannot separate which remaining nodes belong to the Left Subtree versus the Right Subtree. Inorder traversal clearly splits the nodes: all

nodes occurring to the left of the root in inorder form the Left Subtree, and all nodes to the right form the Right Subtree.

Q3. What is an AVL Tree Balance Factor, and what rotations fix LR and RL imbalances? [3 Marks]

- **Answer:** Balance Factor BF = Height(Left) - Height(Right) ∈ {-1, 0, +1}. An LR imbalance (BF = +2, left child BF = -1) is resolved by an LR Double Rotation: Left rotation on left child followed by Right rotation on unbalanced node. An RL imbalance is resolved by a Right rotation on right child followed by Left rotation on unbalanced node.

Part B: Long Answer Questions (10 to 15 Marks Outline Guides)

- **Q4. Construct a unique Binary Tree from Inorder: [D, B, E, A, F, C, G] and Preorder: [A, B, D, E, C, F, G]. Write its Postorder traversal. [10 Marks]**
- **Q5. Insert keys {50, 25, 75, 15, 35, 60, 90, 30, 40} into an initially empty BST. Show tree after deleting: (a) Leaf node 15, (b) Node 75 with one child, (c) Root node 50 with two children. [12 Marks]**
- **Q6. Explain Threaded Binary Search Tree (TBST). Discuss structure of double threaded node with tag bits and write an algorithm for Inorder traversal of TBST without recursion or stack. [12 Marks]**
- **Q7. Construct an AVL Tree by inserting keys {14, 20, 11, 50, 40, 12, 10, 8, 15} step by step. State rotation type applied at each unbalanced step. [14 Marks]**
- **Q8. Explain Heap Sort algorithm. Build Max Heap for array [45, 30, 12, 60, 85, 20, 50, 70] using Floyd's build-heap and trace complete sorting passes. [12 Marks]**

4.10 Unit Summary & Key Takeaways

- Trees are non-linear hierarchical data structures; Binary trees restrict every node to at most two children.
- Key binary tree theorem: Leaf nodes always equal degree-2 nodes plus one ($n_0 = n_2 + 1$).
- Tree traversals include Preorder (DLR), Inorder (LDR), Postorder (LRD), and Level-order (BFS Queue). Unique tree reconstruction requires Inorder + Preorder/Postorder.
- Binary Search Trees (BST) organize data such that Left < Root < Right, supporting O(log n) average operations and 3-case deletion.
- Threaded Binary Search Trees (TBST) utilize null pointers for inorder threads, executing stack-less and recursion-free traversals.

- Optimal Binary Search Trees (OBST) minimize expected search costs for known access probabilities using Dynamic Programming.
- AVL Trees guarantee O(log n) height via self-balancing rotations (LL, RR, LR, RL) when Balance Factor deviates from {-1, 0, +1}.
- Binary Heaps are array-mapped complete trees enabling O(n) Floyd's heap construction and O(n log n) in-place Heap Sort.