UNIT II · LESSON 03

Trees, Binary Trees & BST

terminology · properties · representations · traversals · expression trees · BST operations · AVL · heap · B-tree

20
questions in Unit II
Lists · Trees · Sorting · Hashing · Graphs

TREE WORDS TERMINOLOGY

Term	Meaning
Root	node with no parent
Leaf / external	node with no children
Internal	node with ≥ 1 child
Degree of node	number of children
Degree of tree	maximum node degree
Level	root = 0 (some books: 1)
Depth of node	edges from root to node
Height of node	edges on longest path to a leaf
Height of tree	height of root (single node = 0)
Siblings	same parent
Ancestor / descendant	on the path above / below

- Tree with n nodes has **n − 1 edges**; no cycles; connected

ARRAY & LINKED REPRESENTATIONS

- **Array** (1-based): children of i at **2i, 2i+1**; parent ⌊i/2⌋
- 0-based: children **2i+1, 2i+2**; parent ⌊(i−1)/2⌋

FORMULAS BINARY TREE PROPERTIES

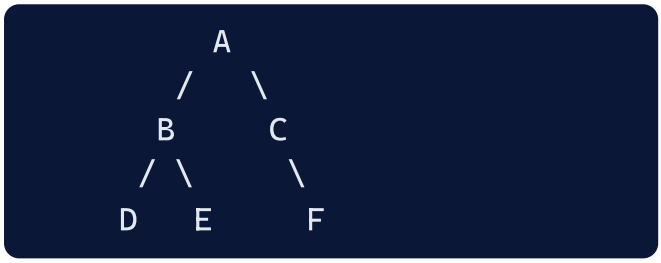
Property	Value
Max nodes at level i (root 0)	2^i
Max nodes, height h	$2^{h+1} - 1$
Min height with n nodes	$\lceil \log_2(n+1) \rceil - 1$
Leaves vs degree-2 nodes	$\mathbf{n_0 = n_2 + 1}$
NULL links with n nodes	n + 1
Distinct binary trees, n unlabelled nodes	Catalan $C_n = (2n)! / ((n+1)!\, n!)$
Labelled binary trees	$n! \cdot C_n$

- $C_{1..6}$ = 1, 2, 5, 14, 42, 132
- **Full/strict**: every node 0 or 2 children; internal i → nodes = 2i + 1, leaves = i + 1
- **Complete**: all levels full except last, filled **left to right**
- **Perfect**: all internal nodes have 2 children, all leaves same level
- **Skewed**: every node has one child (height n − 1)

ARRAY & LINKED REPRESENTATIONS (CONTD.)

- Array wastes space for skewed trees (needs $2^n - 1$ cells)
- **Linked**: `struct tnode { int d; struct tnode *l, *r; };`
- **Threaded** binary tree: NULL links point to inorder predecessor/successor

WORKED TRAVERSAL EXAMPLE



Traversal	Result
Preorder	A B D E C F
Inorder	D B E A C F
Postorder	D E B F C A
Level	A B C D E F

OPERATORS INSIDE EXPRESSION TREE

- Leaves = operands, internal nodes = operators
- Build from postfix with a **stack** of tree pointers
- `(a + b) * (c - d)`: preorder `* + a b - c d`, postorder `a b + c d - *`
- Inorder gives infix (add brackets)

DFS & BFS TRAVERSALS

Order	Visit sequence	Use
Preorder	**N** L R	copy tree, prefix expression
Inorder	L **N** R	BST → sorted order, infix
Postorder	L R **N**	delete tree, postfix
Level order	level by level (queue)	BFS, shortest path in tree

```
void inorder(struct tnode *t) {
    if (t) { inorder(t->l);
             printf("%d ", t->d);
             inorder(t->r); }
}
```

- Unique tree from **inorder + preorder** or **inorder + postorder**
- Preorder + postorder alone → unique only for **full** binary trees
- All traversals O(n) time; recursion stack O(h)

COMPLETE BINARY TREE HEAP

- **Max-heap**: parent ≥ children; **min-heap**: parent ≤ children
- Stored in an array; height $\lfloor \log_2 n \rfloor$

Operation	Time
Find max (max-heap)	O(1)
Insert (sift-up)	O(log n)
Delete max (sift-down)	O(log n)
Build heap (bottom-up)	**O(n)**

- Leaves are at indices $\lfloor n/2 \rfloor + 1$ … n (1-based)

BST BINARY SEARCH TREE

- Left subtree keys < node < right subtree keys; **inorder = sorted**

Operation	Average	Worst (skewed)
Search	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)
Find min / max	leftmost / rightmost	O(n)

```
struct tnode *ins(struct tnode *t, int x) {
  if (!t) {
    t = calloc(1, sizeof *t); t->d = x;
  }
  else if (x < t->d) t->l = ins(t->l, x);
  else if (x > t->d) t->r = ins(t->r, x);
  return t;
}
```

- **Delete**: leaf → remove; one child → link child to parent; two children → replace with **inorder successor** (min of right subtree) or predecessor, then delete that
- Inserting sorted data gives a **skewed** tree

BALANCED BST AVL TREE

- Balance factor = h(left) − h(right) ∈ **{−1, 0, +1}**

Imbalance	Fix
LL (left child's left)	single **right** rotation
RR	single **left** rotation
LR	left rotate child, then right rotate
RL	right rotate child, then left rotate

- Height ≤ 1.44 $\log_2$ n; all operations **O(log n)**
- Min nodes in AVL of height h: N(h) = N(h−1) + N(h−2) + 1 → 1, 2, 4, 7, 12, 20

SURVEY OTHER TREES

Tree	Key idea
Red–black	colours, height ≤ 2 $\log_2$(n+1)
Splay	recently used node moved to root
B-tree (order m)	multiway, ≤ m children, all leaves same level; disks
B+ tree	data only in leaves, leaves linked; DB indexes
Trie	keys by characters; prefix search
Huffman tree	optimal prefix codes (greedy)

- B-tree of order m: non-root nodes have ⌈**m/2**⌉ **… m** children

Q → A RAPID RECALL

Leaves in binary tree with 10 degree-2 nodes	11
Max nodes in binary tree of height 4	31
Binary trees with 3 nodes (shapes)	5
Traversal giving sorted BST keys	inorder
Traversal used to delete a tree	postorder
Data structure for level order	queue
Array index of left child of i (1-based)	2i
BST worst-case search	O(n)
Rotation for RR case	single left rotation
Build-heap time	O(n)

TRAP EXAM TRAPS

- Height convention: single node = 0 (edges) or 1 (nodes) — read the question
- Preorder + postorder do **not** fix a general binary tree
- Complete ≠ full: complete fills the last level left-to-right
- BST delete with two children uses the inorder **successor** (min of right), not the right child
- Heap is **not** sorted; inorder of a heap is meaningless
- Number of BSTs with n distinct keys is also C_n