# Trees

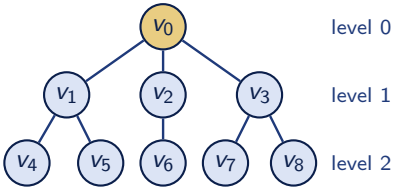
Fundamentals of Discrete Mathematical Structures

by Prof K R Chowdhary

$|V| = |E| + 1$, preorder: root, left, right, $\min \sum_{e \in T} wt(e)$

# Trees: Definition and Properties

- A **tree** is a connected undirected graph with no simple circuit. A rooted tree has a root $v_0$ with a unique path to every other node.
- **Recursively:** a root plus disjoint subtrees $T_1, \ldots, T_m$.
- **Degree** of a node = number of subtrees below it; degree 0 means a *leaf*. Other nodes are *internal*.
- Every non-root node has one *parent*; children of a parent are *siblings*.
- **Binary tree:** at most two children. **Forest:** a set of disjoint trees.

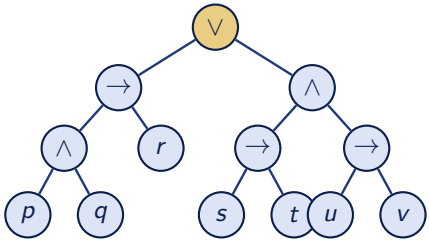



**Theorems:** $G$ is a tree iff it is connected and deleting any edge disconnects it; and $|V| = |E| + 1$.

# Parse Trees of Well Formed Formulae

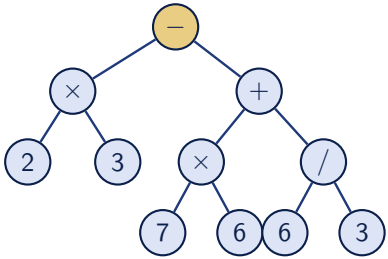
- Each bracketed sub-expression of a wff becomes a node. Scanning left to right: "(" creates a new child and descends; ")" backtracks to the parent.
- Internal nodes carry the operator, leaves carry atoms, so the tree also gives the evaluation order.

For $((p \wedge q) \rightarrow r) \vee ((s \rightarrow t) \wedge (u \rightarrow v))$ compute $t_1 = p \wedge q$, then $t_2 = t_1 \rightarrow r$, then the right subtree $t_3$, and finally $t_2 \vee t_3$ at the root.



# Expression Trees

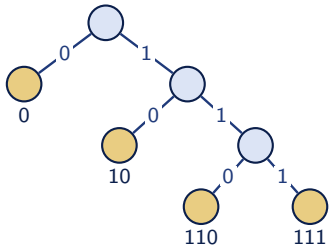
- Leaves hold values; each internal node holds the operator applied to its two children.
- Evaluating collapses a subtree to a single node: $5 + 6$ reduces to the single node 11.
- Conversely, a leaf can be replaced by a subtree. Put $5 = 2 + 3$ and $6 = 2 \times 3$ to get a tree for $(2 + 3) + (2 \times 3)$.
- Brackets fix the order of computation, for example $(2 \times 3) - \big((7 \times 6) + (6/3)\big)$; the tree is read bottom-up.




Value: $6 - (42 + 2) = -38$

# Prefix Codes

- ASCII is fixed length (7 bits). Variable-length codes save space, giving short strings to frequent letters, but can be ambiguous: is 1011 one letter or 10 followed by 11?
- A set of strings is a **prefix code** (Huffman code) if no string is a prefix of another, e.g. $\{01, 000, 11, 10, 001\}$. Decoding is then unambiguous.
- **Tree from a code:** label the two edges at each node 0 and 1; each codeword is the edge sequence from the root to a leaf.
- **Code to tree:** build a full binary tree of height $n$ (the longest word), mark each word's end vertex, delete edges not leading to a word.




Tree for $P = \{0, 10, 110, 111\}$

# Spanning Trees

- A **spanning tree** of a graph is a subgraph that contains all vertices and is a tree. A graph can have many; the triangle $K_3$ has three.
- **Weight** of a spanning tree = sum of its edge weights. The **minimum-cost spanning tree** (MST) has the least weight.
- **Theorem:** an undirected graph is connected iff it has a spanning tree. Proof by induction on cycles: removing an edge of a cycle keeps the graph connected.
- **Applications:** laying cable to join all cities or computers at least cost; approximating the travelling salesman tour.

**Why not try all spanning trees?**

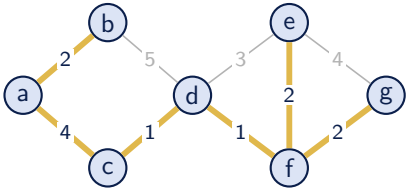
That works but is inefficient. Instead, build the MST one edge at a time. Both **Kruskal's** and **Prim's** algorithms are *greedy*: they take the cheapest safe choice at every step, as Dijkstra's algorithm does, and this is optimal here.

# Kruskal's and Prim's Algorithms

**Kruskal:** start with $E' = \emptyset$. Repeatedly add the cheapest edge that creates no cycle, until $G'$ is connected. Picks, in order: $(c, d)$ 1, $(d, f)$ 1, $(a, b)$ 2, $(f, e)$ 2, $(f, g)$ 2; $(d, e)$ 3 is skipped (cycle); then $(a, c)$ 4. Total weight 12.

**Prim:** start from one vertex in $T$. Repeatedly add the cheapest edge joining $T$ to a vertex outside $T$, until all vertices are in $T$.

Example: from $b$ it picks $ba(5)$, $ae(7)$, $ec(7)$, $cd(5)$, weight 24.



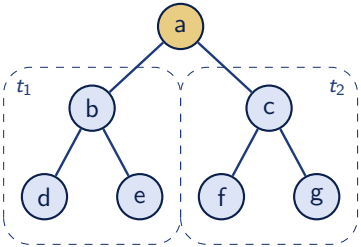

Gold edges form the minimum spanning tree

# Traversing Binary Trees

- **Preorder:** root, left subtree, right subtree.
- **Inorder:** left subtree, root, right subtree.
- **Postorder:** left subtree, right subtree, root.

**For the tree shown**

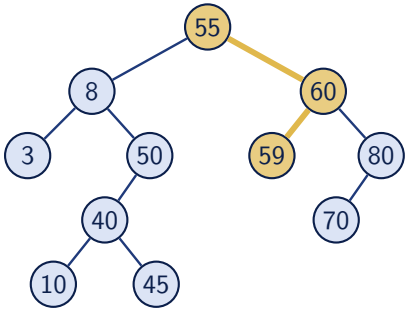
Preorder: *a b d e c f g*
Inorder: *d b e a f c g*
Postorder: *d e b f g c a*




- **BFS** goes level by level $(a, b, c, d, e, f, g)$; **DFS** goes deep before sideways. Both are exhaustive, *uninformed* searches.

# Binary Search Trees

- **Build:** insert numbers in order. Smaller than the node: go left; larger: go right; store at the first empty position.
- Input: $55, 8, 60, 3, 50, 40, 10, 45, 80, 70, 59$.
- **Search** follows the same path. Finding 59: $55 \to 60 \to 59$, only three comparisons.
- Worst case = height of the tree. A balanced tree with $n$ nodes needs about $\log_2 n$ comparisons (4 for 15 nodes), versus $n$ for a plain list.




Gold: search path for 59

# Reference

## Source of this presentation

Prof. K.R. Chowdhary, *Fundamentals of Discrete Mathematical Structures*, PHI Learning, Chapter 12: Trees.
ISBN 978-81-203-5074-8

- Kirchhoff's laws for electrical networks (1845) gave rise to trees and graphs; Cayley developed the theory of trees.
- Kruskal's and Prim's algorithms are greedy, like Dijkstra's shortest path algorithm.

Book link

https://www.phindia.com/Books/BookDetail/9788120350748/FUNDAMENTALS-OF-DISCRETE-MATHEMATICAL-STRUCTURES-CHOWDHARY