Mathematical Foundations of Computer Science

This Lecture: Graph Theory: Trees

Department of Mathematical and Computational Sciences
National Institute of Technology Karnataka, Surathkal

MA714 (Odd Semester [2025-26])

Definition

A **tree** is a connected, undirected graph with no cycles.

A **forest** is a disjoint union of trees (acyclic but possibly disconnected).

A **rooted tree** has one vertex designated as the root.

A **spanning tree** is a subgraph that includes all vertices and is a tree.

Definition

A **tree** is a connected, undirected graph with no cycles.

A **forest** is a disjoint union of trees (acyclic but possibly disconnected).

A **rooted tree** has one vertex designated as the root.

A **spanning tree** is a subgraph that includes all vertices and is a tree.

Illustration 1: A Simple Tree

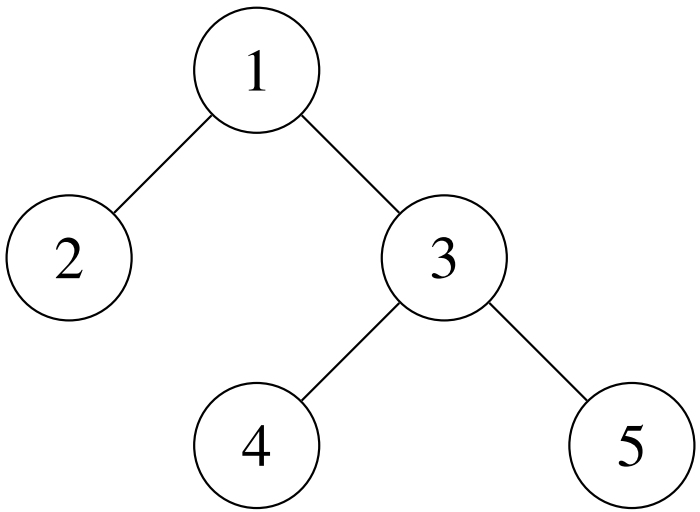


Illustration 2: A Forest

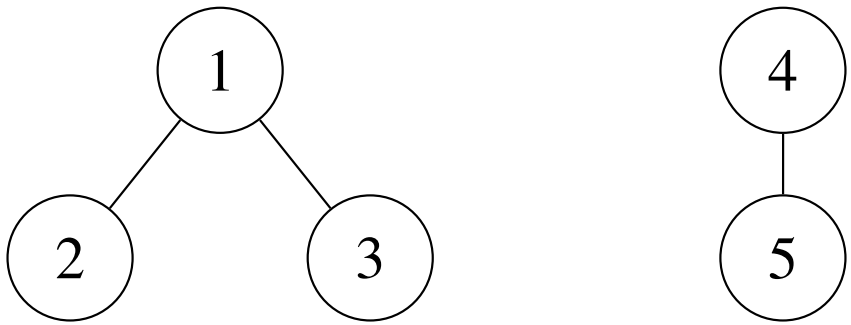


Illustration 3: Rooted Tree

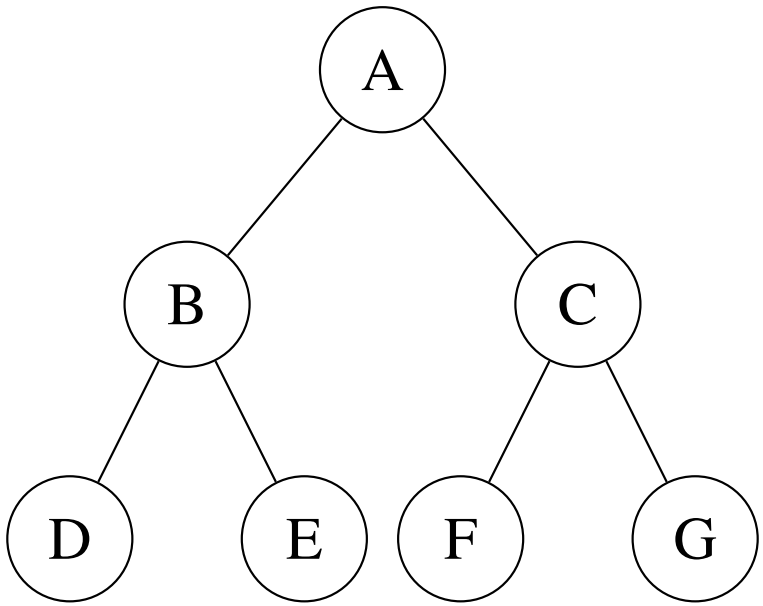


Illustration 4: Spanning Tree

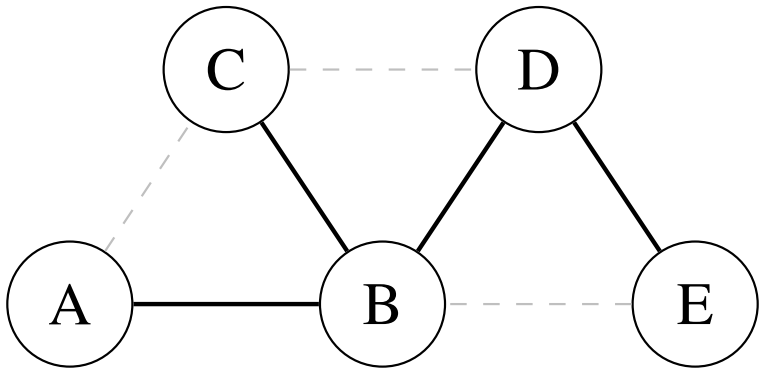
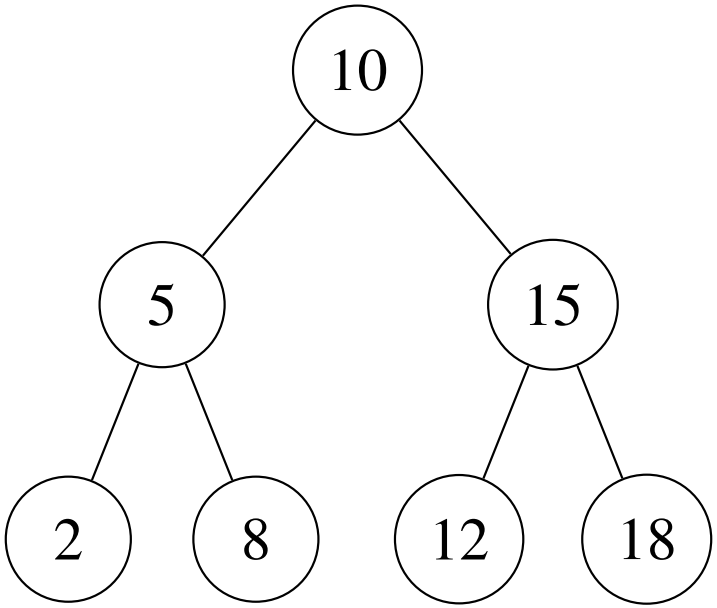


Illustration 5: Binary Tree



Property 1: A Tree with n Vertices has $n - 1$ Edges

Proof (Induction):

Base case ($n = 1$): 1 vertex, 0 edges — true.

Inductive step: Assume true for k vertices.

For tree with $k + 1$ vertices, remove a leaf and its edge.

Remaining graph has k vertices and $k - 1$ edges by induction.

Thus total edges = k, i.e., $n - 1$.

Property 2: Connected Graph with n Vertices and $n-1$ Edges is a Tree

Proof:

Suppose such a graph has a cycle.

Removing one edge from the cycle keeps it connected.

Now it has $n-2$ edges — contradiction, as connected graph needs at least $n-1$ edges.

Hence, no cycles — i.e., it's a tree.

Property 3: Acyclic Graph with n Vertices and $n-1$ Edges is Connected

Proof:

Let G be acyclic with $n-1$ edges and k components.

Each component is a tree with n_i vertices and $n_i - 1$ edges.

Total edges $= n - k$.

Given edges $= n - 1 \Rightarrow k = 1$.

Hence G is connected — a tree.

Property 4: Trees are Minimally Connected

Proof:

A tree is connected.

Between any two vertices, there exists a unique path.

Removing any edge breaks that path $\Rightarrow$ disconnected.

Hence, tree is minimally connected.

Summary of Key Properties

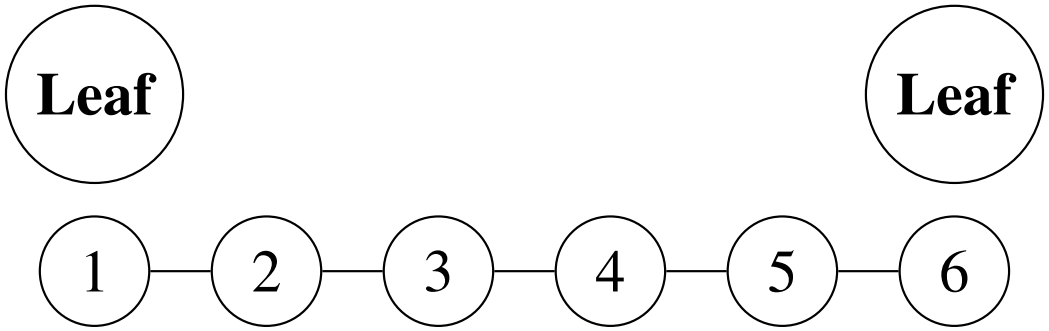
Property	**Proof Idea**
n vertices $\Rightarrow$ $n-1$ edges	Induction
Connected + $n-1$ edges $\Rightarrow$ Tree	Contradiction (cycle removal)
Acyclic + $n-1$ edges $\Rightarrow$ Connected	Counting components
Minimally connected	Unique path argument

Property 5: Existence of Leaves

Statement: Every tree with at least two vertices has at least two vertices of degree one.

Proof: Let $P = v_1, v_2, \ldots, v_k$ be a longest path in the tree. If v_1 had $\deg(v_1) > 1$, there exists another vertex connected to v_1 not on P, extending the path — contradiction. Hence, both v_1 and v_k are leaves. □

Illustration: Leaves of a Tree

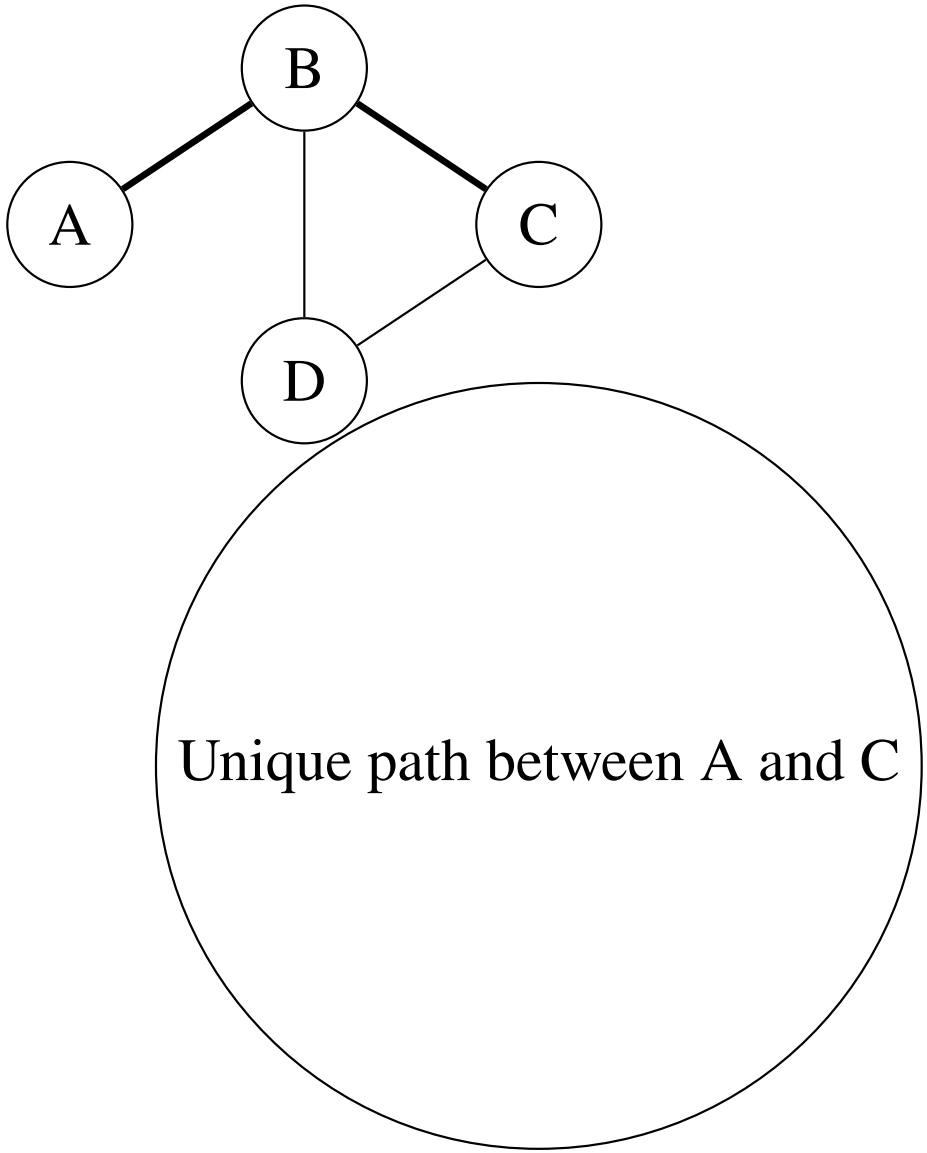


Property 6: Unique Path Property

Statement: Between any two vertices in a tree, there exists exactly one simple path.

Proof: A tree is connected $\Rightarrow$ at least one path exists. If two distinct paths existed between vertices u, v, their union would form a cycle — contradiction. Hence, exactly one path exists. $\square$

Illustration: Unique Path

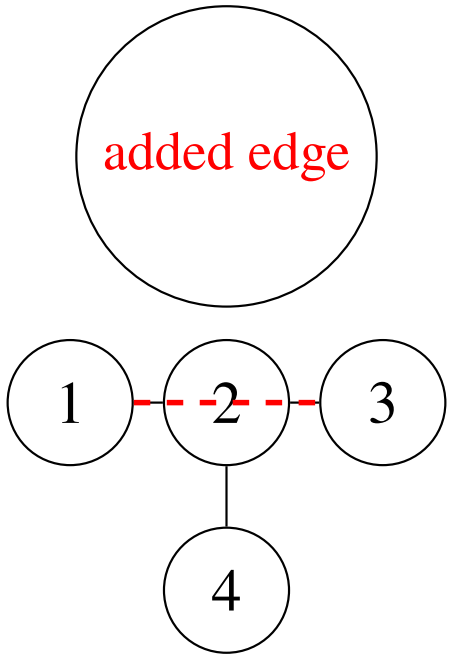


Property 7: Adding an Edge Creates a Cycle

Statement: Adding an edge between any two vertices of a tree creates exactly one cycle.

Proof: There already exists a unique path between those two vertices. Adding the new edge closes that path, forming a single cycle. □

Illustration: Adding an Edge



Property 8: Removing an Edge

Statement: Removing an edge from a tree increases the number of connected components by one.

Proof: A tree is minimally connected. Removing an edge breaks the only path between its incident vertices, splitting the tree into two components. □

Property 9: Center and Diameter

Definition:

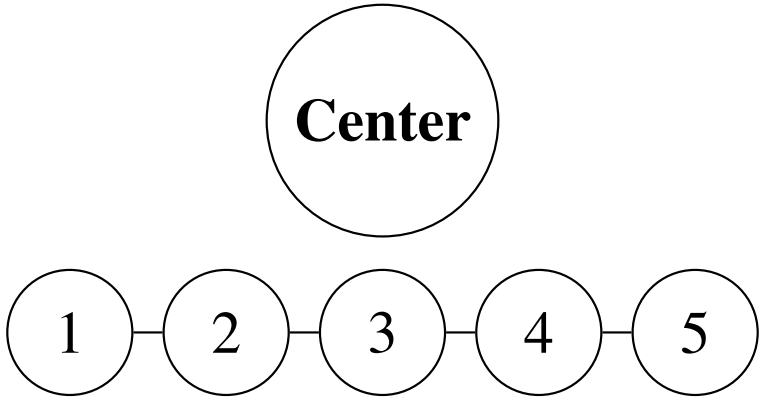
The **diameter** is the length of the longest path in a tree.

The **center** is the vertex (or two adjacent vertices) with minimum eccentricity.

Theorem: The center of a tree is either one vertex or two adjacent vertices.

Idea of Proof: Repeatedly remove all leaves. The remaining one or two vertices form the center. □

Illustration: Center of a Path Tree



Property 10: Degree Sum Formula

Statement: In any tree $T = (V, E)$ with n vertices:

$$\sum_{v \in V} \deg(v) = 2(n-1)$$

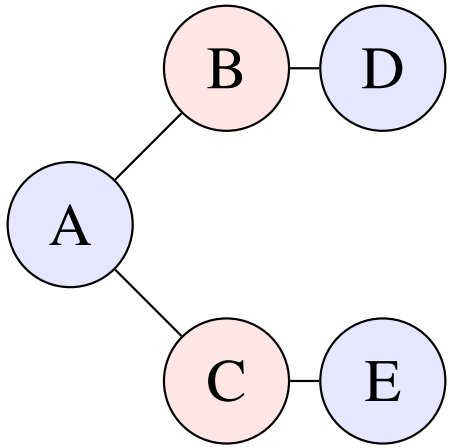
Proof: Each edge contributes 2 to the degree sum (once for each endpoint). Since there are $n - 1$ edges, total sum $= 2(n-1)$. □

Property 11: Bipartite Nature of Trees

Statement: Every tree is bipartite.

Proof: Acyclic graphs contain no odd cycles. Assign alternate colors to vertices along any path — no two adjacent vertices share a color. Thus, trees are bipartite. □

Illustration: Bipartite Coloring



Property 12: Forest Edge-Count Formula

Statement: If a forest has n vertices and k components, then it has $n - k$ edges.

Proof: Each component is a tree with n_i vertices and $n_i - 1$ edges.

$$|E| = \sum_i (n_i - 1) = n - k.$$

□

Statement: The number of labeled trees on n vertices is n^{n-2}.

Proof Idea: Using **Prufer codes**, there is a one-to-one correspondence between trees and sequences of length $n-2$ using $\{1, 2, \ldots, n\}$. Hence, total trees $= n^{n-2}$. □

Property 14: Spanning Trees

Statement: Every connected graph G has at least one spanning tree. If G itself is a tree, it has exactly one spanning tree (itself).

Proof: Remove edges from cycles one by one until no cycles remain but all vertices stay connected. If G is already acyclic, no edge can be removed — one spanning tree only. □

Summary Table

No.	Property	Description
1–4	Basic structure	$n-1$ edges, minimal, unique path
5	Existence of leaves	At least two degree-1 vertices
6	Unique path	Exactly one between any two vertices
7	Adding edge	Creates one cycle
8	Removing edge	Increases components by one
9	Center	One or two adjacent vertices
10	Degree sum	$2(n-1)$
11	Bipartite	No odd cycles
12	Forest formula	$\|E\| = n - k$
13	Cayley's formula	n^{n-2} labeled trees
14	Spanning trees	Always exist in connected graphs

Applications of Trees in Computer Science

1. **Hierarchical Data Representation**
   File systems, XML/JSON structures.
2. **Binary Search Trees (BST)**
   Efficient searching, insertion, deletion ($O(\log n)$ average).
3. **Expression Trees**
   Used in compilers to evaluate arithmetic expressions.
4. **Spanning Trees in Networks**
   Used in routing and broadcasting (Kruskal's, Prim's algorithms).
5. **Decision Trees & Machine Learning**
   Used in classification/regression models.

Syntax Trees — Abstract representation in compilers.

Huffman Coding Trees — For lossless data compression.

Game Trees — For AI and decision making.

Routing Trees — In communication networks.

Scene Graphs — In computer graphics hierarchy.

Trees are **connected acyclic graphs**.

They are **minimally connected** with $n - 1$ edges.

Play vital roles in:

- Data organization,
- Searching and sorting,
- Network design,
- Machine learning.

References


Bondy, J.A. and Murty, U.S.R. (1976). *Graph Theory with Applications*, Macmillan.


Harary, F. (1969). *Graph Theory*, Addison-Wesley.


Cormen, T.H. et al. (2009). *Introduction to Algorithms*, MIT Press.