Part I: Review of Basic Concepts

Introduction

- **Graph theory** is a powerful mathematical framework used to model and analyze complex relationships in various fields, including computer science, telecommunications, and information networks.
- In this course, we will explore the fundamental concepts and algorithms of graph theory with a focus on their practical **applications in network science**.

Applications in information networks

- **Topology Modeling**: Graphs represent network structures, nodes are routers or servers, and edges are connections.
  - *Ex:* A data center network modeled as a **Fat-tree topology** ensures efficient data flow.
- **Shortest Path Algorithms**: Used to find the most efficient route between nodes in a network.
  - *Ex: Dijkstra's algorithm helps routers compute the fastest path for data packets.*
- **Multicast & Broadcast Routing:** Ensures efficient data transmission to multiple recipients using spanning trees.
  - *Ex: IP multicast delivers a live video stream to multiple users without duplicate transmissions.*
- **Flow Algorithms:** Optimize data flow through a network by modeling capacity constraints.
  - *Ex: Ford-Fulkerson algorithm helps balance traffic across network links to avoid congestion.*

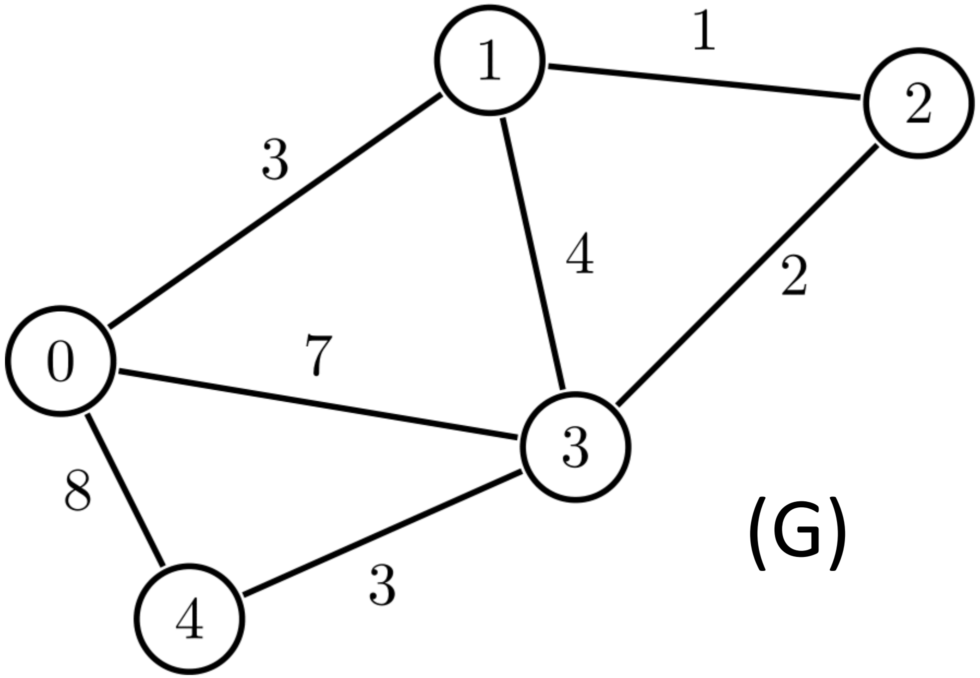
Applications in information networks

- **Connectivity & Vulnerability:** Analyzes how network failures affect connectivity and identifies weak points.
  - Ex: Finding articulation points in a network shows which nodes' failure would break connectivity.
- **Influence & Information Spread**: Models how information, trends, or viruses propagate through networks.
  - Ex: Identifying key influencers in a social media network using centrality measures.
- **Anomaly Detection**: Detects unusual patterns in network activity by monitoring graph structure changes.
  - Ex: A sudden change in node degree could indicate an attack on a server.

Graph vs Network

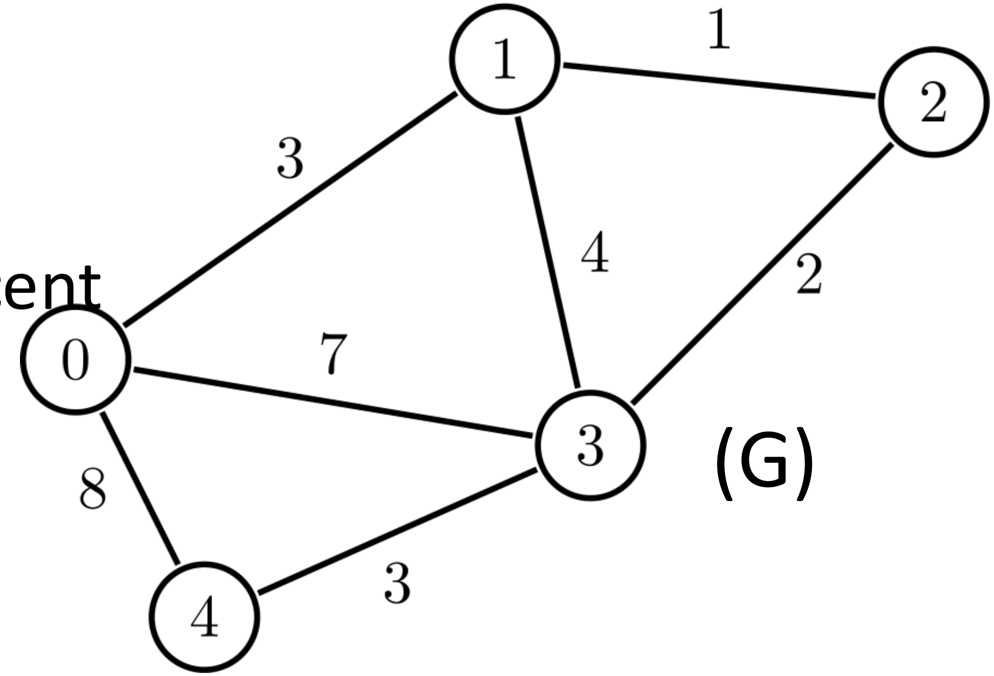
- Graph: A set of vertices V and edges E connecting them: G=(V,E).
  - *V={v1, v2, …, vn}, vi is a* ***vertex***
  - *E={e1, e2, …, em}, ej is an* ***edge****, e={u,v} is simply noted e=uv*
  - ***Adjacent*** *nodes have a common edge*
  - ***Adjacent*** *edges have a common node*
  - *If e=uv, the edge e and the vertices u and v are* ***incidents***
- Networks: Real structures that can be represented by graphs
  - *Big size (nodes & links)*
  - *Complex structure with attributed nodes and relations*

Basic metrics



- **Order**: number of vertices, Order(G)=|V|
  - Ex: order(G)=5
- **Size**: number of edges, Size(G)=|E|
  - Ex:size(G)=7
- **Degree**: number of edges connected to a vertex, deg(v)=number of edges connected to v, deg(G)=max degree of all vertices
  - $\sum_{v \in V}$ degree(v) = 2 × size(G)
  - $\sum$: 3+2+4+3+2=14=2x7
- **Weight**: A value assigned to an edge
  - Ex: weight(0,4)=8, weight(1,3)=4

Important concepts

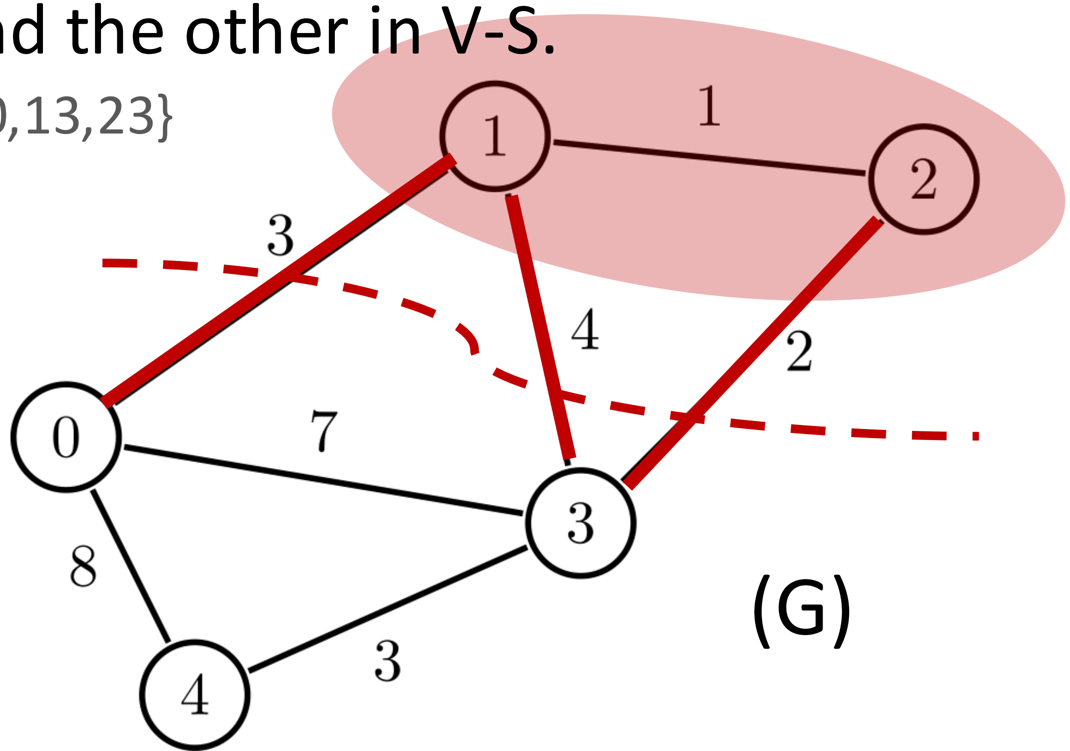


(G)

- **Walk**: A sequence of vertices pairwise adjacent
  - Ex: 0,1,3,0,1,2,3,4
  - Ex: 4,1,2,1,0 (not a walk)
- **Trail**: A walk with no repeated edges
  - Ex: 0,3,1,0,4
- **Path**: A trail with no repeated vertices
  - Ex: 4,3,1,2
- **Cycle**: A path that starts and ends at the same vertex
  - Ex: 2,1,0,3,2
- **Distance** between 2 vertices is the shortest path length between them
  - Ex: distance(1,4)=1+2+3=6

Important concepts

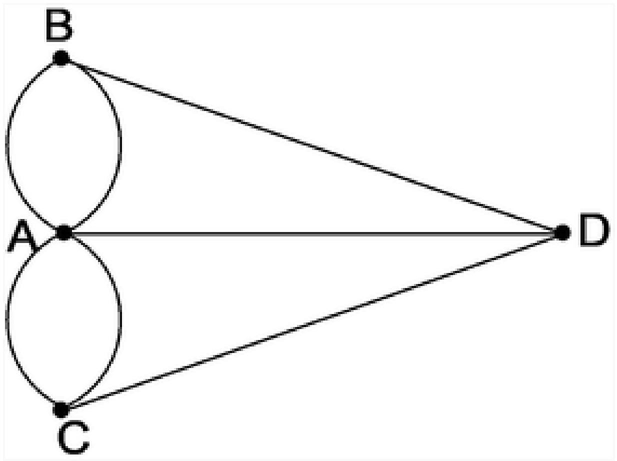
- **Co-cycle(cutset)**: set of edges whose removal increases the number of connected components in the graph.
  It is often associated with a partition of the vertex set V into two disjoint subsets S and V-S. The cocycle consists of all edges that have one endpoint in S and the other in V-S.
  - Ex: co-cycle({1,2})={10,13,23}



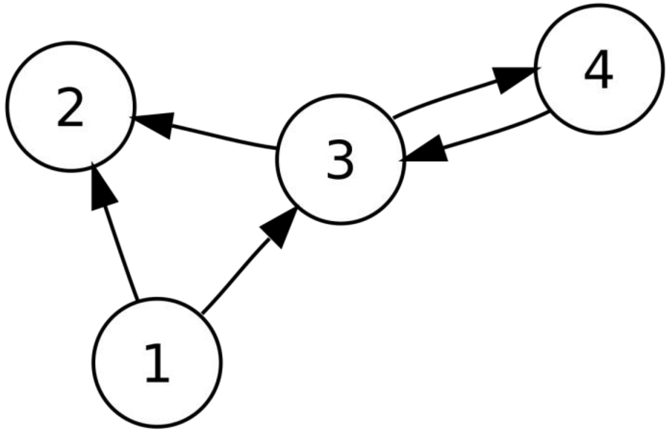
(G)

Types of graphs

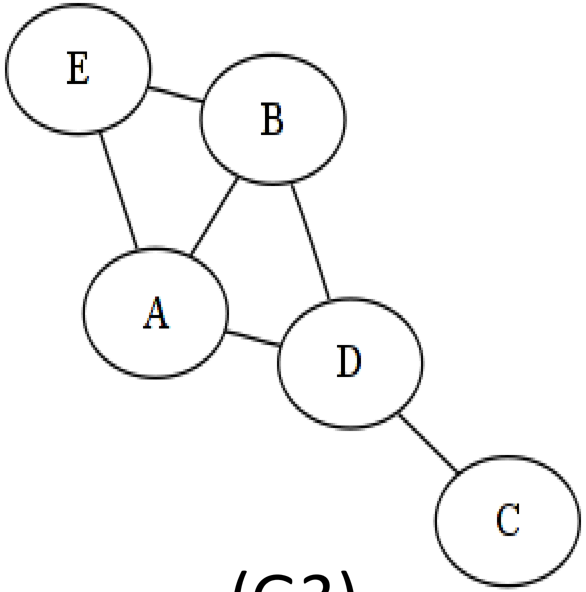
- **Indirected**: Edges have no direction (G1)
- **Directed(DiGraph)**: Edges have direction (G2)
- **Multi-Graph**: Allows multiple edges between two vertices (G3)



(G1)



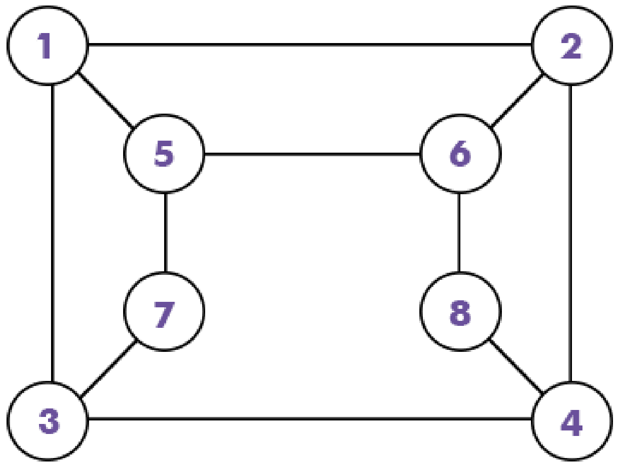
(G2)



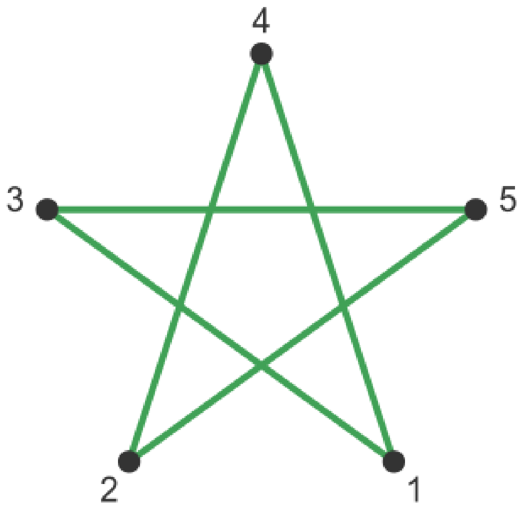
(G3)

Types of graphs

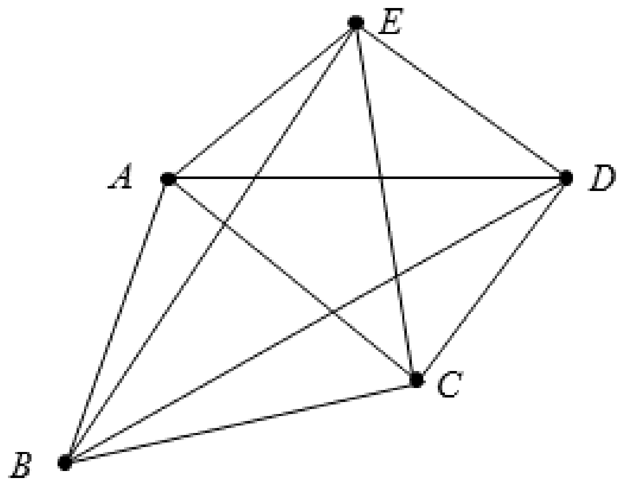
- **Complete (k_n)**: Every pair of vertices is connected (G1)
- **Regular(n-rugular)**: All vertices have the same degree (G2)
- **Bipartite**: Vertices can be divided into two disjoint sets, edges link only vertices from one set to the other set (G3)



(G3)



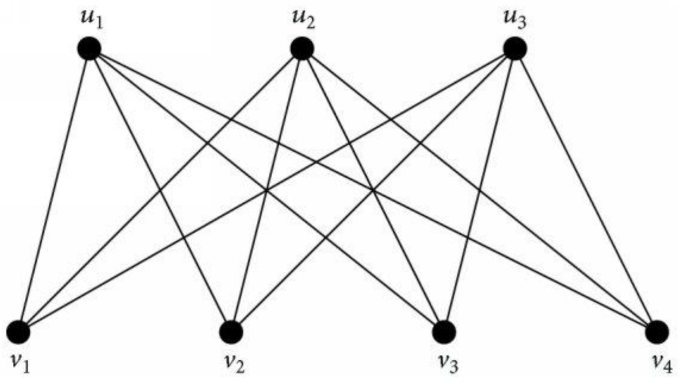
(G2)



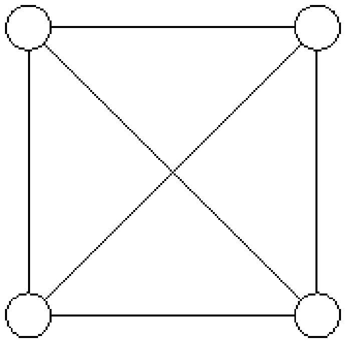
(G1)

Types of graphs

- **Complete bipartite**: A bipartite graph with all possible edges (G1)
- **Planar**: Can be drawn without edge crossings (G2). Delimited areas are named **Faces**, the graph of faces is named **Dual graph**
  - ∑ degree(face) = 2 × size(G)
  - Order(G) – size(G) + # of face = 2
- **Simple**: No loops or multiple edges (G3 is not a simple graph)

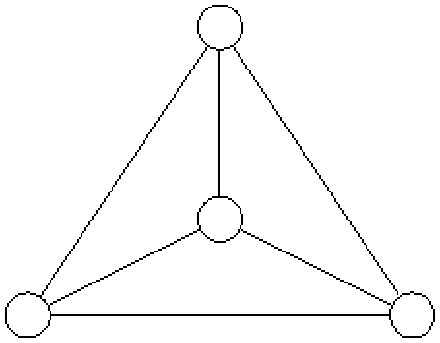


(G1)

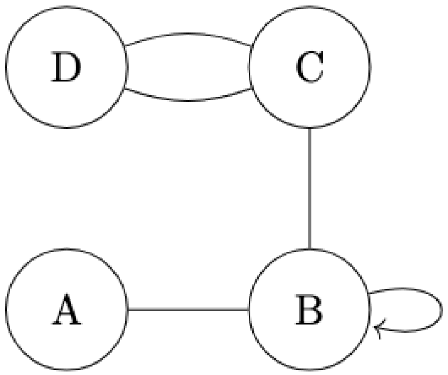


(a)

(G2)



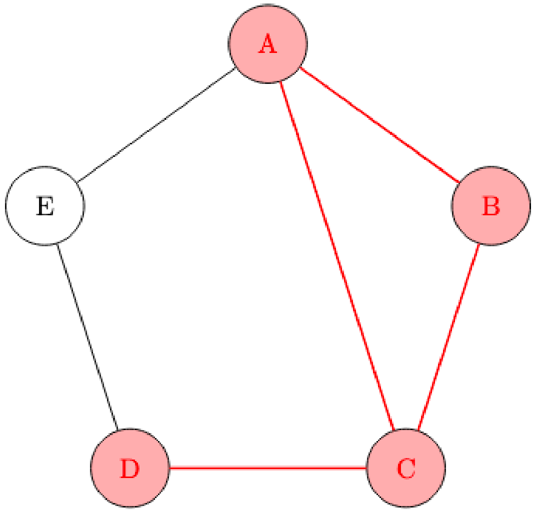
(b)



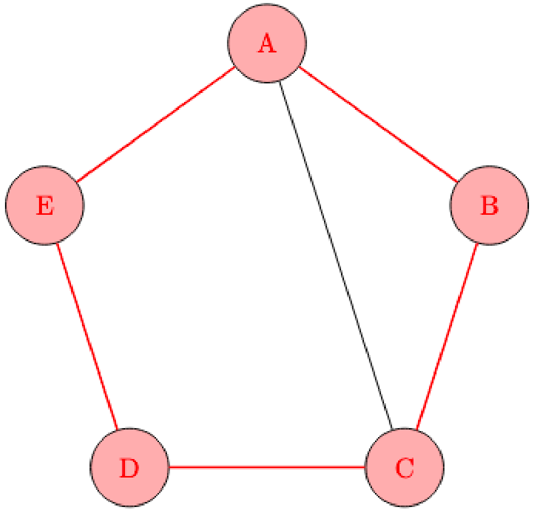
(G3)

Types of graphs

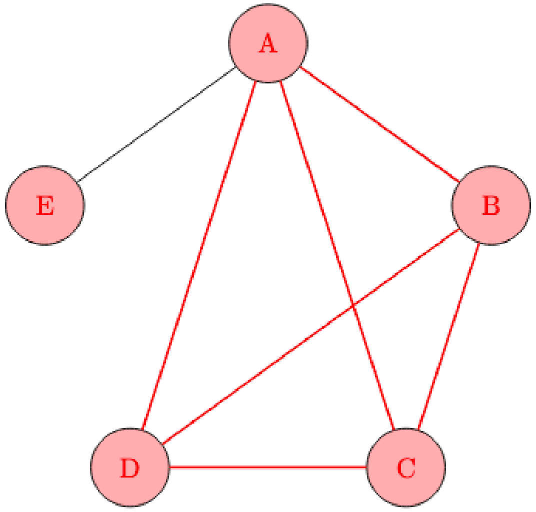
- **Subgraph**: A subset of a graph's vertices with incident edges (G1)
- **Partial**: A subgraph that retains all vertices but removes some edges (G2)
- **Clique**: A subgraph where every pair of vertices is connected (G3)



(G1)



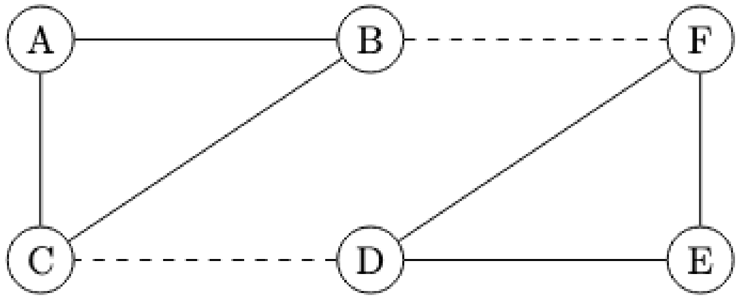
(G2)



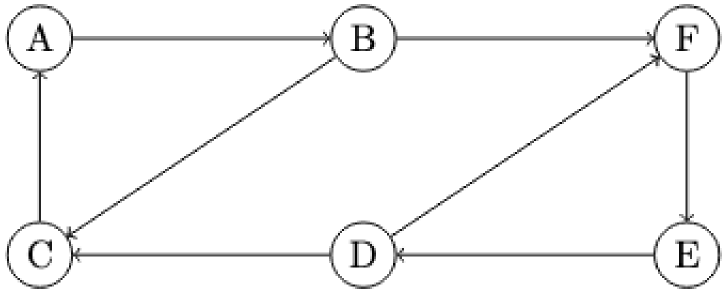
(G3)

Connectivity

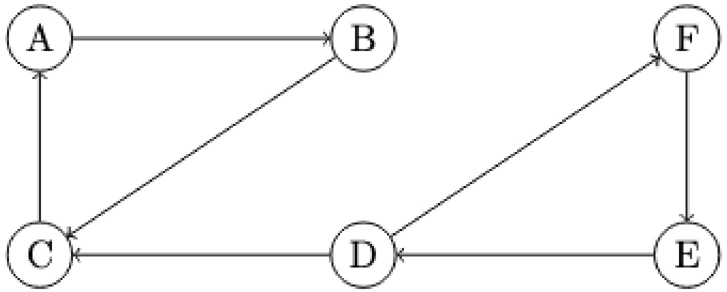
- Undirected graph: A graph is **connected** if there is a path between every pair of vertices and **disconnected** else (G1).
- Directed graph: A DiGraph can be disconnected, weakly or strongly connected.
  - A digraph is **strongly connected** if every vertex can reach every other vertex (G2).
  - A digraph is **weakly connected** if it will be connected after transforming their arcs to edges (G3).



(G1)



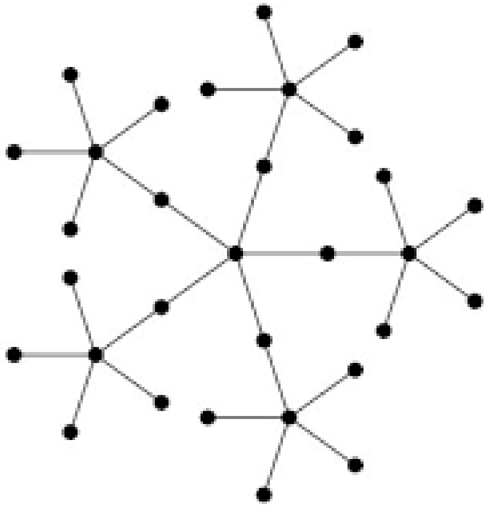
(G2)



(G3)

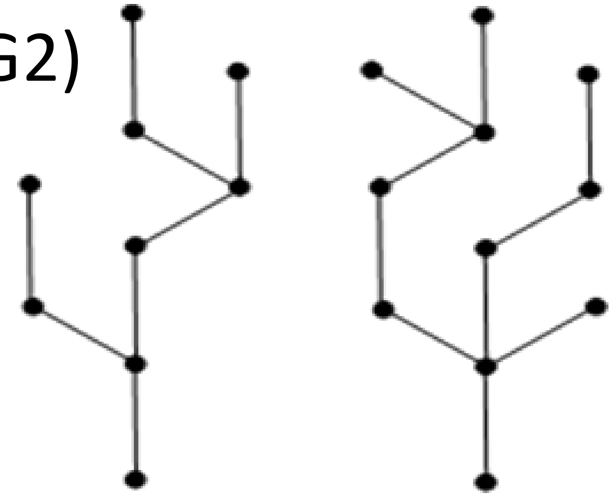
Trees

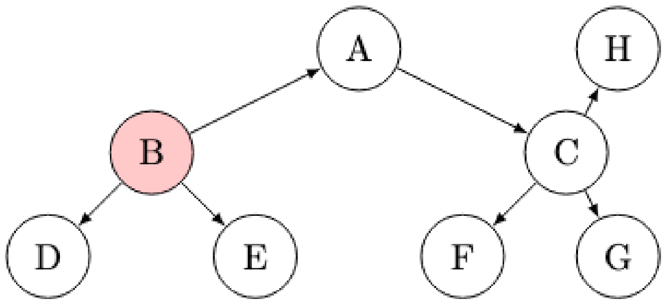
(G1)



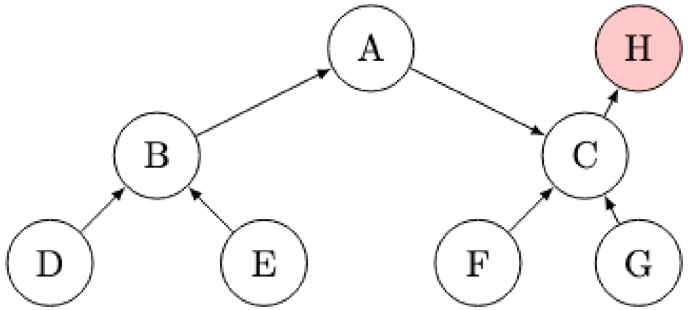
- **Tree**: A connected acyclic graph (G1)
  - Size=order-1
- **Forest**: A collection of disjoint trees (G2)
- **Arborescence/Anti-Arborescence**: A directed tree with a single root, from which all edges point outward/inward (G3)
- **Root/Anti-Root**: A vertex that allows reaching/reachable by all other vertices

(G2)



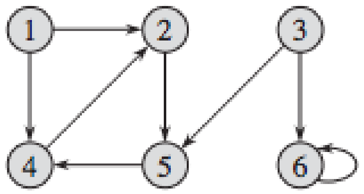


(G3)

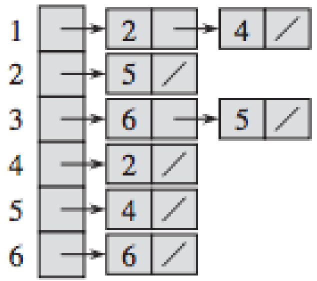


Computer representation of graphs

- **Adjacency matrix**: A |V|×|V| matrix where A[i][j]=1 if an edge exists between vi and vj.
- **Adjacency list**: A list where each vertex has a set of adjacent vertices.
- **Incidence matrix**: A |V|×|E| matrix where rows represent vertices and columns represent edges.
- **Adjacency matrix Power**: The matrix power A^k shows the number of paths of length k between vertices.



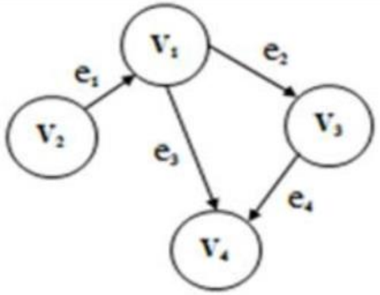
(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)



(i)

	e_1	e_2	e_3	e_4
V_1	-1	1	1	0
V_2	1	0	0	0
V_3	0	-1	0	1
V_4	0	0	-1	-1

(ii)