Graphs

Chapter 10

Copyright © McGraw-Hill Education. All rights reserved. No reproduction or distribution without the prior written consent of McGraw-Hill Education.

Chapter Summary

- Graphs and Graph Models
- Graph Terminology and Special Types of Graphs
- Representing Graphs and Graph Isomorphism
- Connectivity (next week)
- Euler and Hamiltonian Graphs (next week)

Graphs and Graph Models

Section 10.1

Section Summary

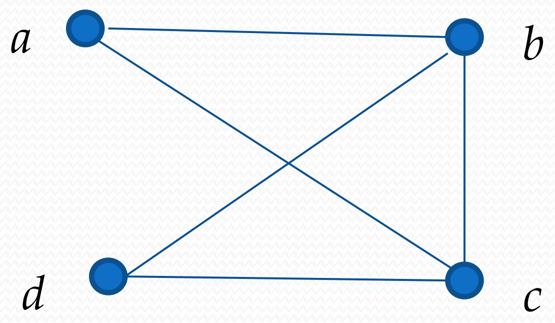
- Introduction to Graphs
- Graph Taxonomy
- Graph Models

Graphs

Definition: A *graph* $G = (V, E)$ consists of a nonempty set V of *vertices* (or *nodes*) and a set E of *edges*. Each edge has either one or two vertices associated with it, called its *endpoints*. An edge is said to *connect* its endpoints.

Example:

This is a graph with four vertices and five edges.



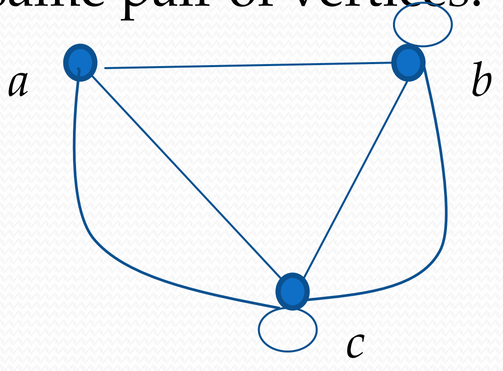
Remarks:

- The graphs we study here are unrelated to graphs of functions studied in Chapter 2.
- We have a lot of freedom when we draw a picture of a graph. All that matters is the connections made by the edges, not the particular geometry depicted. For example, the lengths of edges, whether edges cross, how vertices are depicted, and so on, do not matter
- A graph with an infinite vertex set is called an *infinite graph.* A graph with a finite vertex set is called a *finite graph*. We (following the text) restrict our attention to finite graphs.

Some Terminology

- In a *simple graph* each edge connects two different vertices and no two edges connect the same pair of vertices.
- *Multigraphs* may have multiple edges connecting the same two vertices. When m different edges connect the vertices u and v, we say that $\{u,v\}$ is an edge of *multiplicity* m.
- An edge that connects a vertex to itself is called a *loop*.
- A *pseudograph* may include loops, as well as multiple edges connecting the same pair of vertices.

Example:
This pseudograph has both multiple edges and a loop.



Remark: There is no standard terminology for graph theory. So, it is crucial that you understand the terminology being used whenever you read material about graphs.

Directed Graphs

Definition: A *directed graph* (or *digraph*) $G = (V, E)$ consists of a nonempty set V of *vertices* (or *nodes*) and a set E of *directed edges* (or *arcs*). Each edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u,v) is said to *start at u* and *end at v*.

Remark:

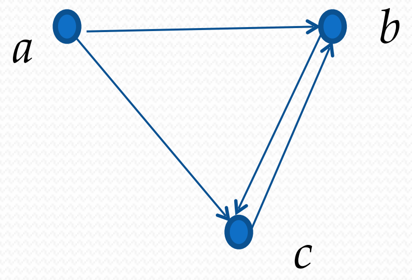
- Graphs where the end points of an edge are not ordered are said to be *undirected graphs*.

Some Terminology (*continued*)

- A *simple directed graph* has no loops and no multiple edges.

Example:

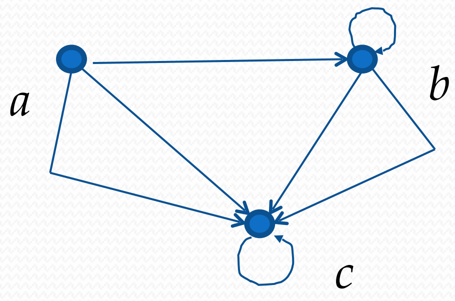
This is a directed graph with three vertices and four edges.



- A *directed multigraph* may have multiple directed edges. When there are m directed edges from the vertex u to the vertex v, we say that (u,v) is an edge of *multiplicity* m.

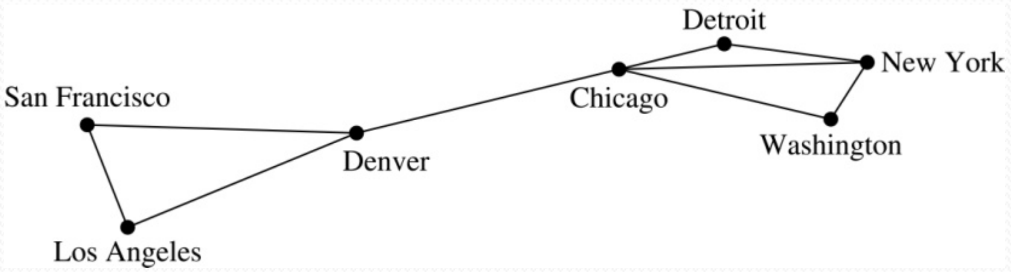
Example:

In this directed multigraph the multiplicity of (a,b) is 1 and the multiplicity of (b,c) is 2.



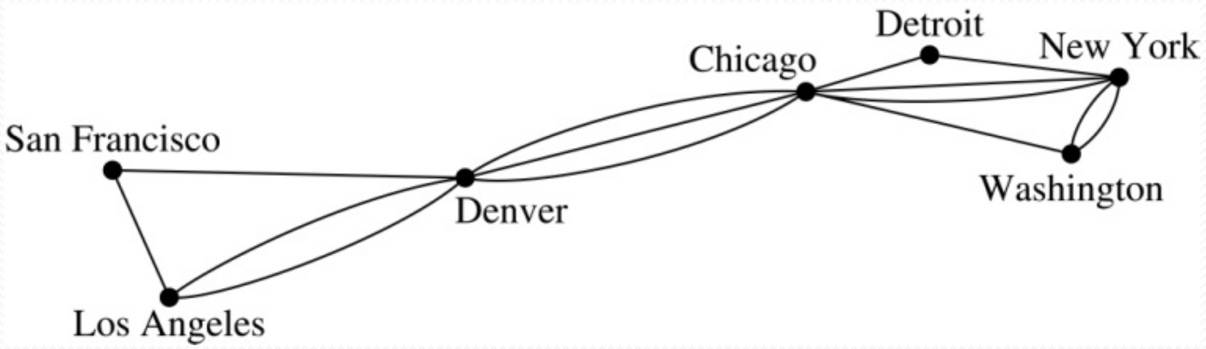
Graph Models: Computer Networks

- When we build a graph model, we use the appropriate type of graph to capture the important features of the application.
- We illustrate this process using graph models of different types of computer networks. In all these graph models, the vertices represent data centers and the edges represent communication links.
- Assume we want to model whether two data centers are connected by a communications link, we use a simple graph.
  - This is the appropriate type of graph when we only care whether two data centers are directly linked (and not how many links there may be) and all communications links work in both directions.



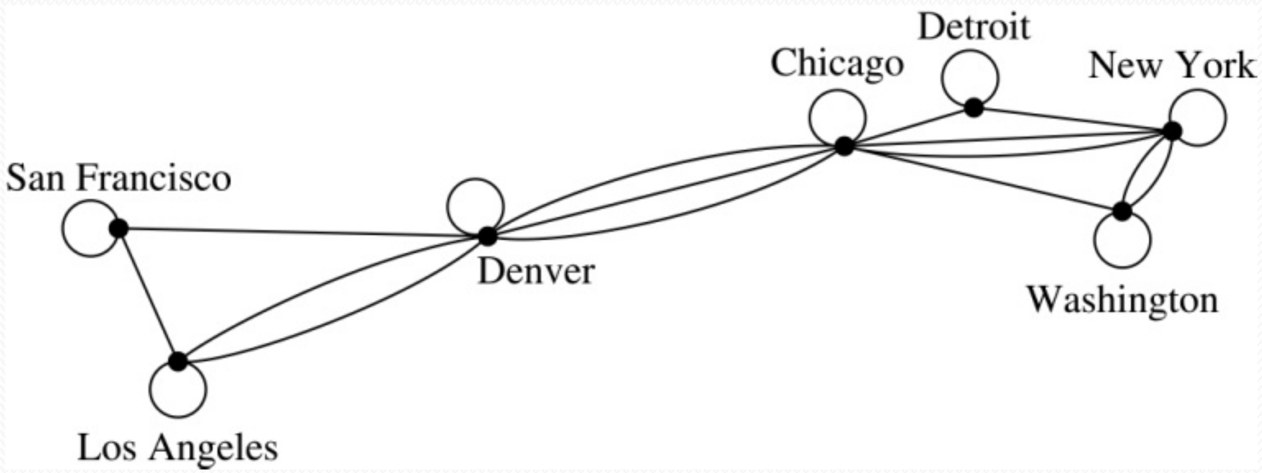
Graph Models: Computer Networks (*continued*)

- To model a computer network where we care about the number of links between data centers, we use a multigraph.



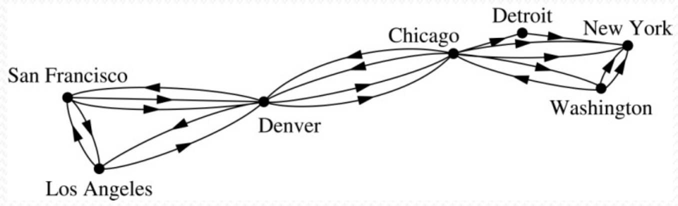
Graph Models: Computer Networks (*continued*)

- To model a computer network with diagnostic links at data centers, we use a pseudograph, as loops are needed.



Graph Models: Computer Networks (*continued*)

- To model a network with multiple one-way links, we use a directed multigraph.
  - Note that we could use a *directed graph without multiple edges* if we only care whether there is at least one link from a data center to another data center.



Graph Terminology: Summary

- To understand the structure of a graph and to build a graph model, we ask these questions:
  - Are the edges of the graph undirected or directed (or both)?
  - If the edges are undirected, are multiple edges present that connect the same pair of vertices?
  - If the edges are directed, are multiple directed edges present?
  - Are loops present?

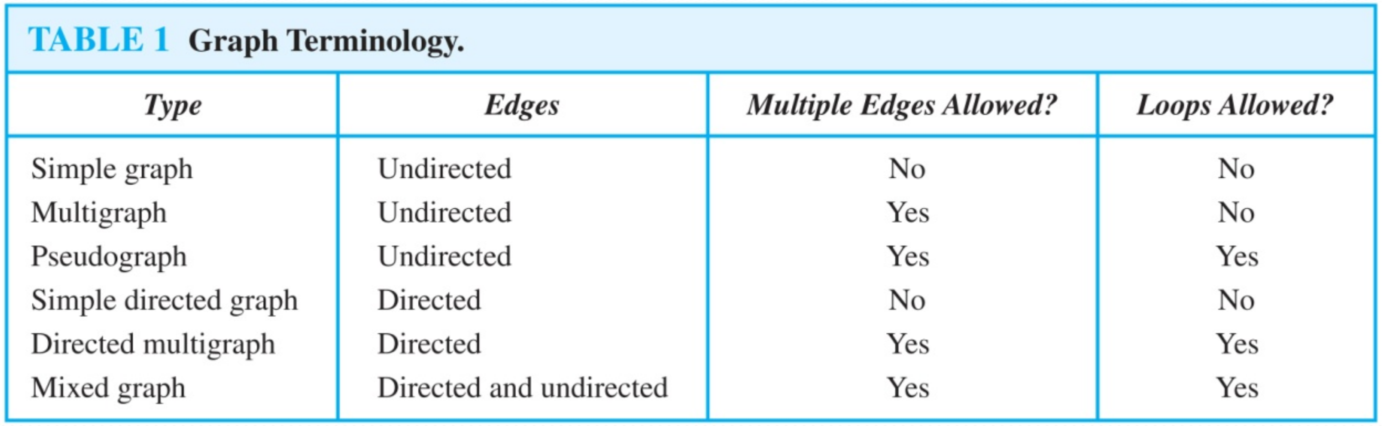


TABLE 1 Graph Terminology.

Type	*Edges*	*Multiple Edges Allowed?*	*Loops Allowed?*
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed graph	Directed and undirected	Yes	Yes

Other Applications of Graphs

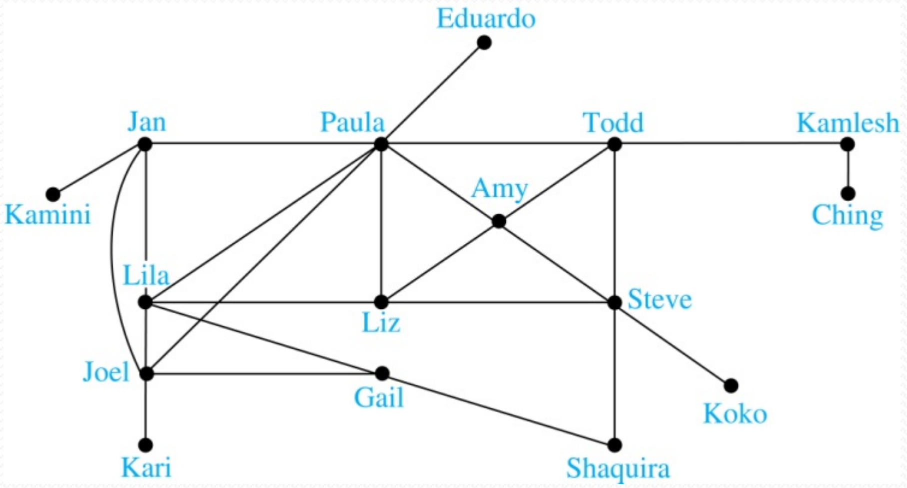
- We will illustrate how graph theory can be used in models of:
  - Social networks
  - Communications networks
  - Information networks
  - Software design
  - Transportation networks
  - Biological networks
  - Graph neural networks
- It's a challenge to find a subject to which graph theory has not yet been applied. Can you find an area without applications of graph theory?

Graph Models: Social Networks

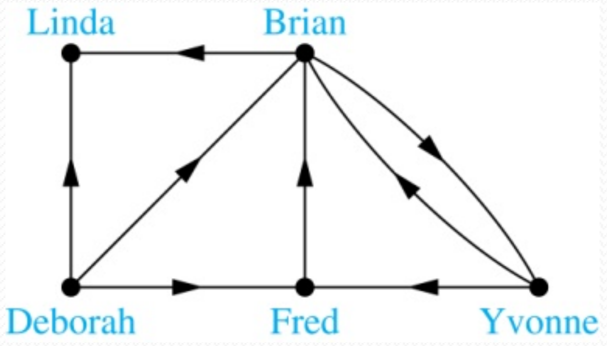
- Graphs can be used to model social structures based on different kinds of relationships between people or groups.
- In a *social network*, vertices represent individuals or organizations and edges represent relationships between them.
- Useful graph models of social networks include:
  - *friendship graphs* - undirected graphs where two people are connected if they are friends (in the real world, on Facebook, or in a particular virtual world, and so on.)
  - *collaboration graphs* - undirected graphs where two people are connected if they collaborate in a specific way
  - *influence graphs* - directed graphs where there is an edge from one person to another if the first person can influence the second person

Graph Models: Social Networks (*continued*)

Example: A friendship graph where two people are connected if they are Facebook friends.



Example: An influence graph



Next Slide: Collaboration Graphs

Examples of Collaboration Graphs

- The *Hollywood graph* models the collaboration of actors in films.
  - We represent actors by vertices and we connect two vertices if the actors they represent have appeared in the same movie.

- An *academic collaboration graph* models the collaboration of researchers who have jointly written a paper in a particular subject.
  - We represent researchers in a particular academic discipline using vertices.
  - We connect the vertices representing two researchers in this discipline if they are coauthors of a paper.

Applications to Information Networks

- Graphs can be used to model different types of networks that link different types of information.
- In a *web graph*, web pages are represented by vertices and links are represented by directed edges.
    - A web graph models the web at a particular time.
- In a *citation network*:
    - Research papers in a particular discipline are represented by vertices.
    - When a paper cites a second paper as a reference, there is an edge from the vertex representing this paper to the vertex representing the second paper.

Transportation Graphs

- Graph models are extensively used in the study of transportation networks.
- Airline networks can be modeled using directed multigraphs where
  - airports are represented by vertices
  - each flight is represented by a directed edge from the vertex representing the departure airport to the vertex representing the destination airport
- Road networks can be modeled using graphs where
  - vertices represent intersections and edges represent roads.
  - undirected edges represent two-way roads and directed edges represent one-way roads.

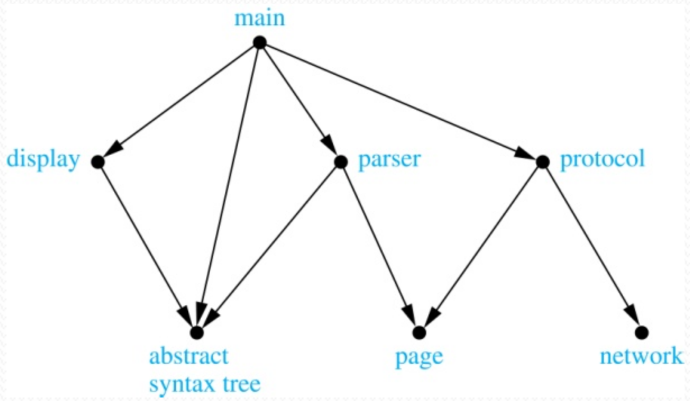
Software Design Applications

- Graph models are extensively used in software design. We will introduce two such models here;
  - one representing the dependency between the modules of a software application,
  - the other representing restrictions in the execution of statements in computer programs.

Software Design Applications

- Designing software top-down:
  - the system is divided into modules; each performing a specific task.
- We use a *module dependency graph;*
  - to represent the dependency between these modules.
  - In a module dependency graph vertices represent software modules and there is an edge from one module to another if the second module depends on the first.

Example: The dependencies between the seven modules in the design of a web browser are represented by this module dependency graph.

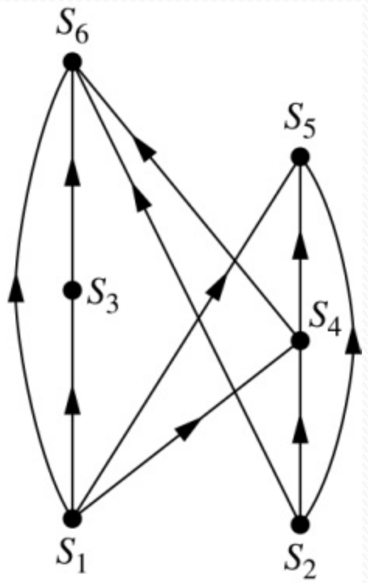


Software Design Applications (*continued*)

- We can use a directed graph called a *precedence graph* to represent which statements must have already been executed before we execute each statement.
  - Vertices represent statements in a computer program
  - There is a directed edge from a vertex to a second vertex if the second vertex cannot be executed before the first

Example: This precedence graph shows which statements must be executed before we can execute each of the six statements in the program.

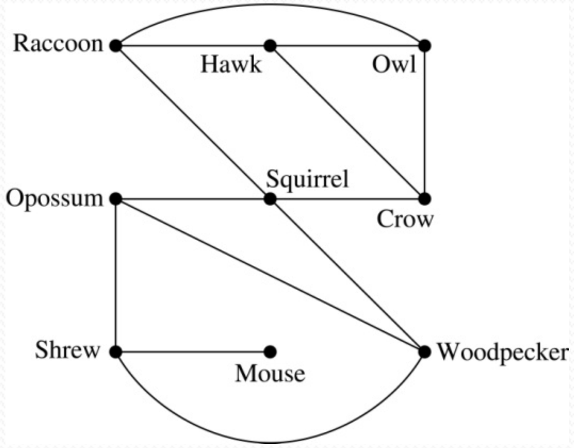
S_1	$a := 0$
S_2	$b := 1$
S_3	$c := a + 1$
S_4	$d := b + a$
S_5	$e := d + 1$
S_6	$e := c + d$



Biological Applications

- Graph models are used extensively in many areas of the biological science. We will describe two such models, one to ecology and the other to molecular biology.
- *Niche overlap graphs* model competition between species in an ecosystem
  - Vertices represent species and an edge connects two vertices when they represent species who compete for food resources.

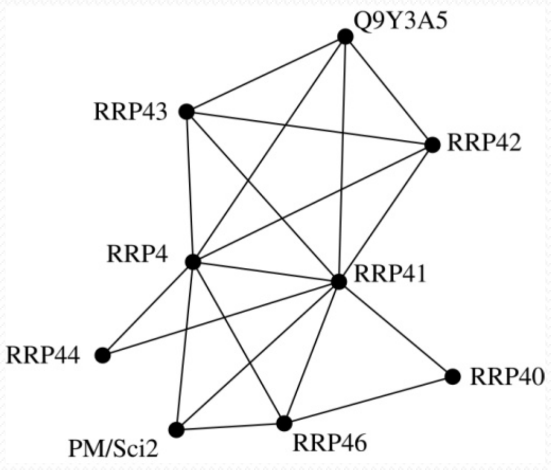
Example: This is the niche overlap graph for a forest ecosystem with nine species.



Biological Applications (*continued*)

- We can model the interaction of proteins in a cell using a *protein interaction network.*
- In a *protein interaction graph*, vertices represent proteins and vertices are connected by an edge if the proteins they represent interact.
- Protein interaction graphs can be huge and can contain more than 100,000 vertices, each representing a different protein, and more than 1,000,000 edges, each representing an interaction between proteins
- Protein interaction graphs are often split into smaller graphs, called *modules*, which represent the interactions between proteins involved in a particular function.

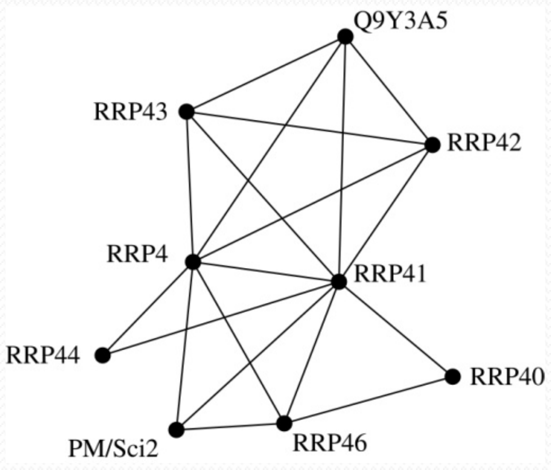
Example: This is a module of the protein interaction graph of proteins that degrade RNA in a human cell.



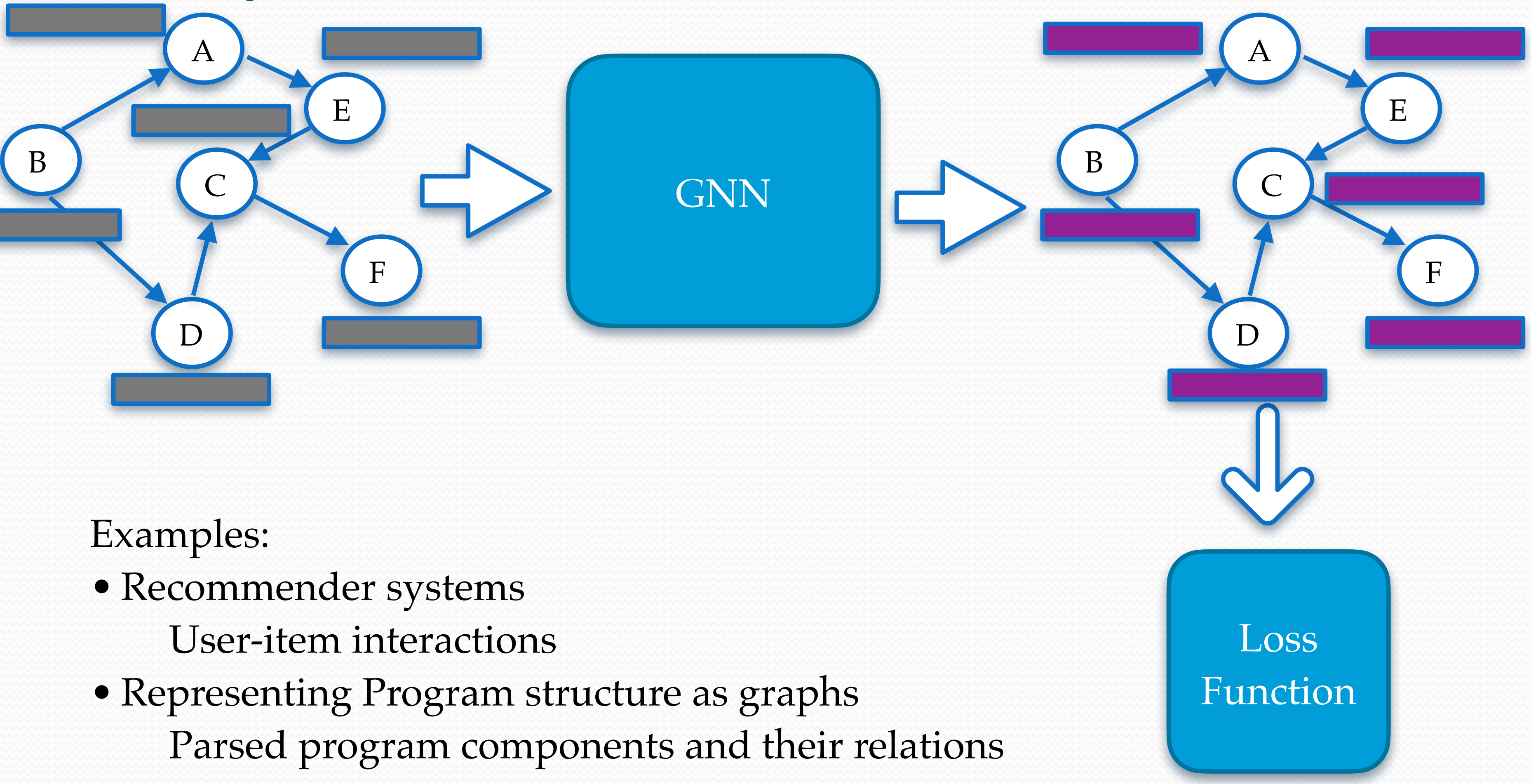
Biological Applications (*continued*)

- We can model the interaction of proteins in a cell using a *protein interaction network.*
- In a *protein interaction graph*, vertices represent proteins and vertices are connected by an edge if the proteins they represent interact.
- Protein interaction graphs can be huge and can contain more than 100,000 vertices, each representing a different protein, and more than 1,000,000 edges, each representing an interaction between proteins
- Protein interaction graphs are often split into smaller graphs, called *modules*, which represent the interactions between proteins involved in a particular function.

Example: This is a module of the protein interaction graph of proteins that degrade RNA in a human cell.



Graph Neural Networks

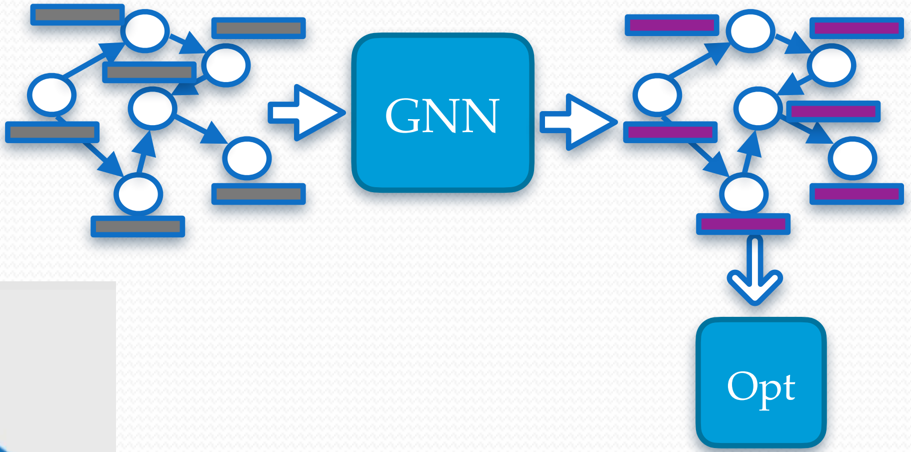


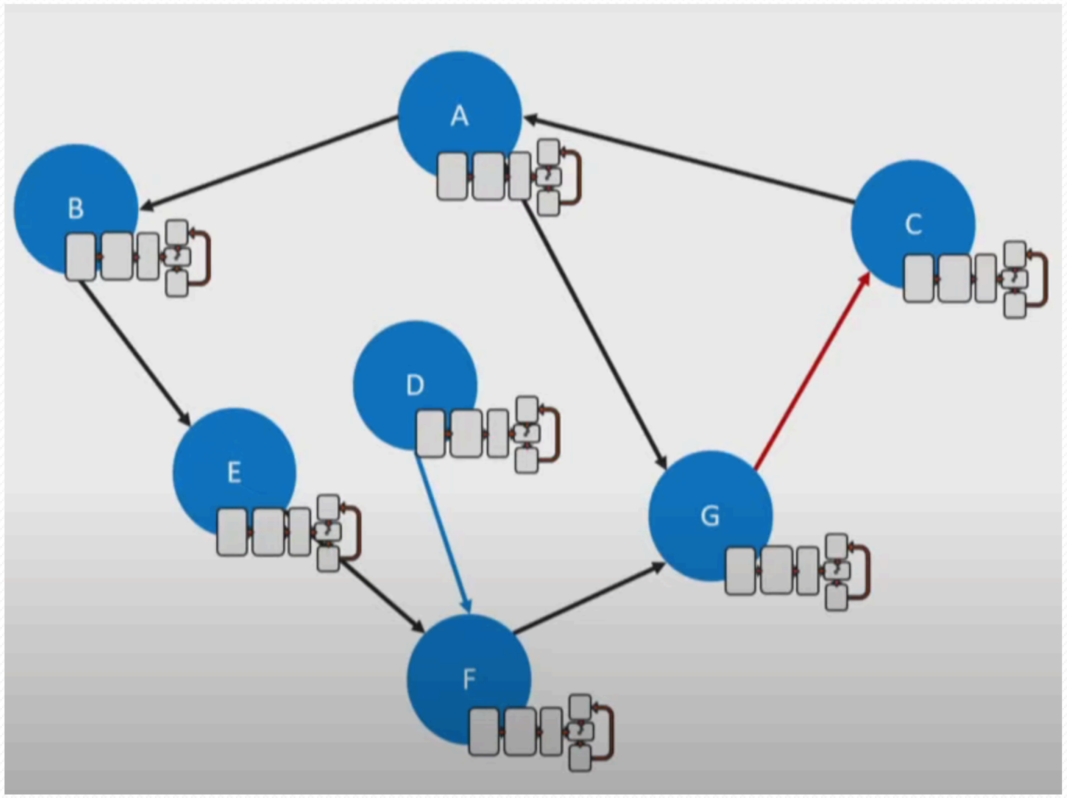
Examples:

- Recommender systems
    User-item interactions
- Representing Program structure as graphs
    Parsed program components and their relations

Graph Neural Networks

- Neural message passing from neighboring nodes
- State updates at each node at time t





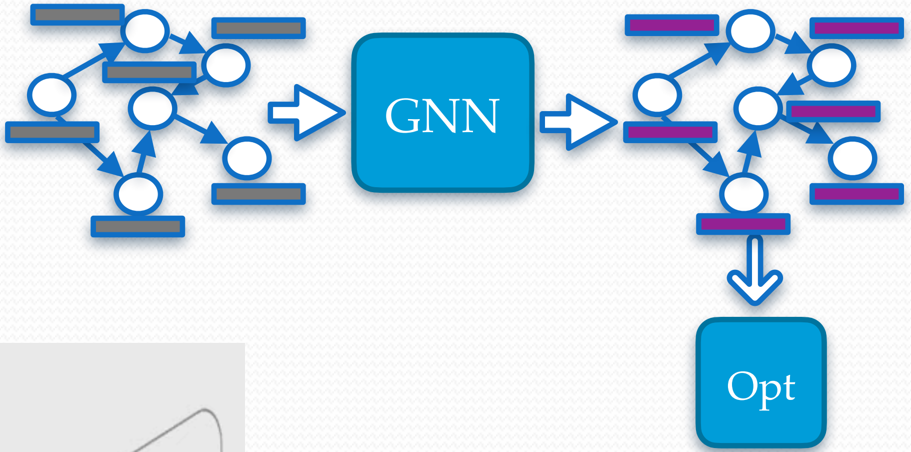
Network parameters are shared

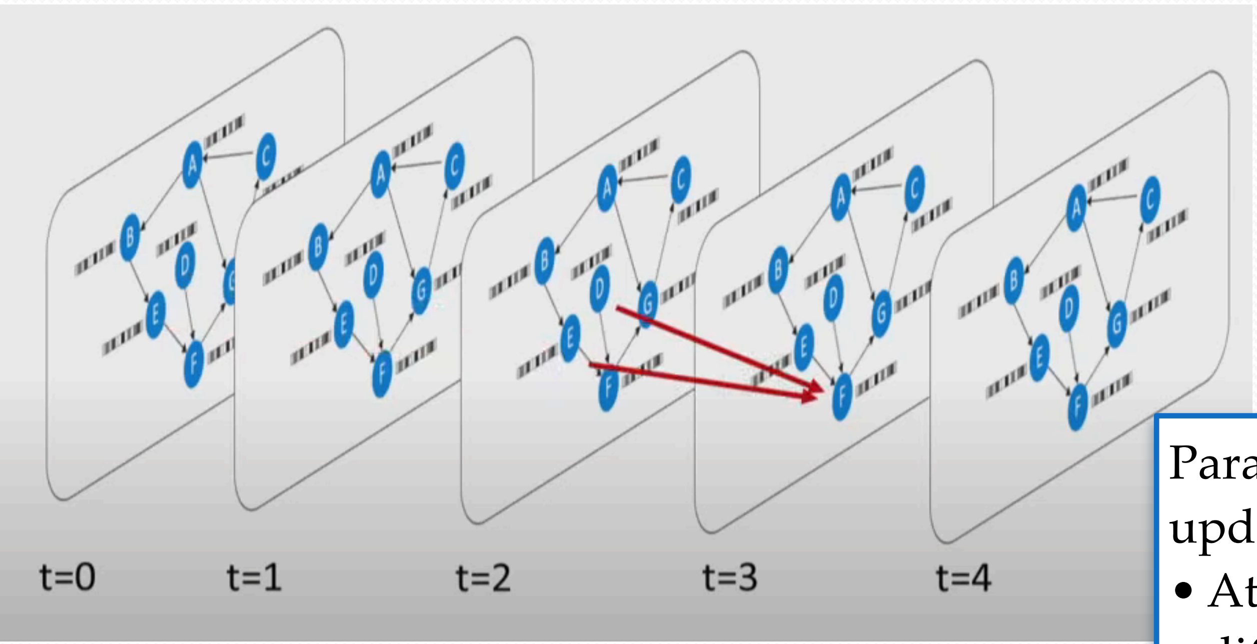
Image credit: An Introduction to Graph Neural Networks: Models and Applications

For more details: https://www.youtube.com/watch?v=zCEYiCxrL_0

Graph Neural Networks

- Neural message passing from neighboring nodes
- State updates at each node at time t





Parameters of each node are updated simultaneously at time t

- At each time step the graph is in a different state

Image credit: An Introduction to Graph Neural Networks: Models and Applications

For more details: https://www.youtube.com/watch?v=zCEYiCxrL_0

Graph Terminology and Special Types of Graphs

Section 10.2

Section Summary

- Basic Terminology
- Some Special Types of Graphs
- Bipartite Graphs
- Bipartite Graphs and Matchings (*not currently included in overheads*)
- Some Applications of Special Types of Graphs (*not currently included in overheads*)
- New Graphs from Old

Basic Terminology

Definition 1. Two vertices u, v in an undirected graph G are called *adjacent* (or *neighbors*) in G if there is an edge e between u and v.

- Such an edge e is called *incident with* the vertices u and v and
- e is said to *connect* u and v.

Basic Terminology

Definition 2. The set of all neighbors of a vertex v of $G = (V, E)$, denoted by $N(v)$, is called the *neighborhood of* v.

If A is a subset of V, we denote by $N(A)$ the set of all vertices in G that are adjacent to at least one vertex in A. So,

$$N(A) = \bigcup_{v \in A} N(v).$$

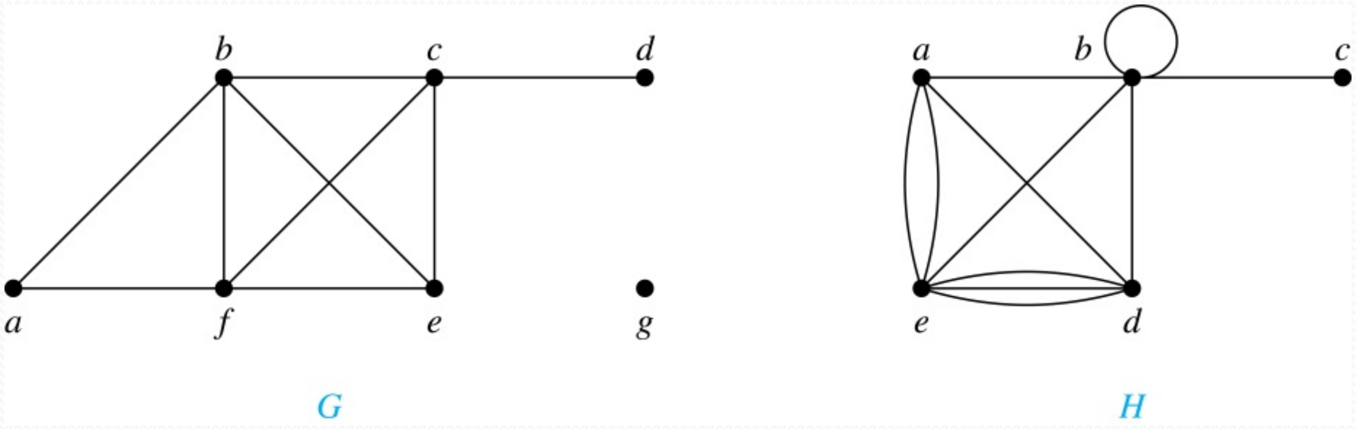
Basic Terminology

Definition 3. The *degree of a vertex in a undirected graph* is the number of edges incident with it,

- except that a loop at a vertex contributes two to the degree of that vertex.
- The degree of the vertex v is denoted by $deg(v)$.

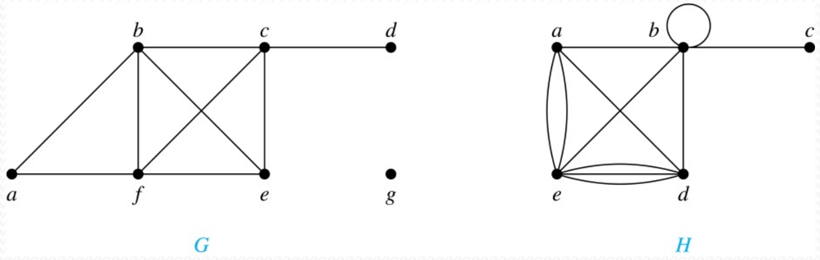
Degrees and Neighborhoods of Vertices

Example: What are the degrees and neighborhoods of the vertices in the graphs *G* and *H*?



Degrees and Neighborhoods of Vertices

Example: What are the degrees and neighborhoods of the vertices in the graphs G and H?



Solution:

G: $\deg(a) = 2$, $\deg(b) = \deg(c) = \deg(f) = 4$, $\deg(d) = 1$,
$\deg(e) = 3$, $\deg(g) = 0$.
$N(a) = \{b, f\}$, $N(b) = \{a, c, e, f\}$, $N(c) = \{b, d, e, f\}$, $N(d) = \{c\}$,
$N(e) = \{b, c, f\}$, $N(f) = \{a, b, c, e\}$, $N(g) = \varnothing$.

H: $\deg(a) = 4$, $\deg(b) = \deg(e) = 6$, $\deg(c) = 1$, $\deg(d) = 5$.
$N(a) = \{b, d, e\}$, $N(b) = \{a, b, c, d, e\}$, $N(c) = \{b\}$,
$N(d) = \{a, b, e\}$, $N(e) = \{a, b, d\}$.

Degrees of Vertices

Theorem 1 (*Handshaking Theorem*): If $G = (V,E)$ is an undirected graph with m edges, then

$$2m = \sum_{v \in V} \deg(v)$$

Proof:

Each edge contributes twice to the degree count of all vertices. Hence, both the left-hand and right-hand sides of this equation equal twice the number of edges.

◀

Think about the graph where vertices represent the people at a party and an edge connects two people who have shaken hands.

Handshaking Theorem

We now give two examples illustrating the usefulness of the handshaking theorem.

Example: How many edges are there in a graph with 10 vertices of degree six?

Solution: Because the sum of the degrees of the vertices is $6 \cdot 10 = 60$, the handshaking theorem tells us that $2m = 60$. So the number of edges $m = 30$.

Handshaking Theorem

We now give two examples illustrating the usefulness of the handshaking theorem.

Example: If a graph has 5 vertices, can each vertex have degree 3?

Solution: This is not possible by the handshaking theorem, because the sum of the degrees of the vertices $3 \cdot 5 = 15$ is odd.

Degree of Vertices (*continued*)

Theorem 2: An undirected graph has an even number of vertices of odd degree.

Proof: Let V_1 be the vertices of even degree and V_2 be the vertices of odd degree in an undirected graph $G = (V, E)$ with m edges. Then

$$\text{even} \longrightarrow 2m = \sum_{v\in V} \deg(v) = \sum_{v\in V_1} \deg(v) + \sum_{v\in V_2} \deg(v).$$

must be even since $\deg(v)$ is even for each $v \in V_1$

This sum must be even because $2m$ is even and the sum of the degrees of the vertices of even degrees is also even. Because this is the sum of the degrees of all vertices of odd degree in the graph, there must be an even number of such vertices.

Directed Graphs

Recall the definition of a directed graph.

Definition: A *directed graph* $G = (V, E)$ consists of V, a nonempty set of *vertices* (or *nodes*), and E, a set of *directed edges* or *arcs*. Each edge is an ordered pair of vertices. The directed edge (u,v) is said to start at u and end at v.

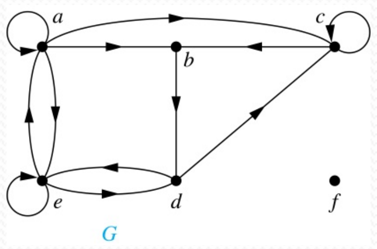
Definition: Let (u,v) be an edge in G. Then u is the *initial vertex* of this edge and is *adjacent to* v and v is the *terminal* (or *end*) *vertex* of this edge and is *adjacent from* u. The initial and terminal vertices of a loop are the same.

Directed Graphs (*continued*)

Definition: The *in-degree of a vertex v*, denoted *deg*$^-$(*v*), is the number of edges which terminate at *v*. The *out-degree of v*, denoted *deg*$^+$(*v*), is the number of edges with *v* as their initial vertex.

- Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: In the graph *G* we have



$\deg^-(a) = 2$, $\deg^-(b) = 2$, $\deg^-(c) = 3$, $\deg^-(d) = 2$, $\deg^-(e) = 3$, $\deg^-(f) = 0$.

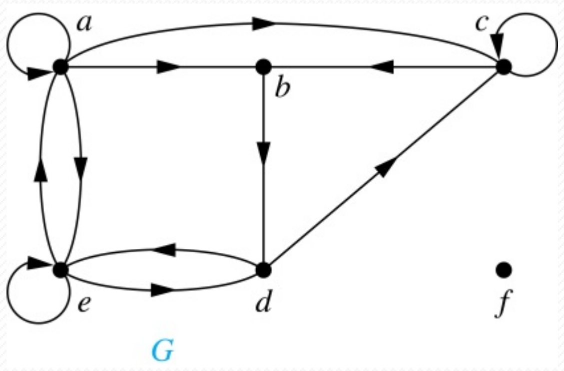
$\deg^+(a) = 4$, $\deg^+(b) = 1$, $\deg^+(c) = 2$, $\deg^+(d) = 2$, $\deg^+(e) = 3$, $\deg^+(f) = 0$.

Directed Graphs (*continued*)

Definition: The *in-degree of a vertex v*, denoted *$\deg^-(v)$*, is the number of edges which terminate at v. The *out-degree of v*, denoted *$\deg^+(v)$*, is the number of edges with v as their initial vertex.

- Note that a loop at a vertex contributes 1 to both the in-degree and the out-degree of the vertex.

Example: In the graph G we have



$\deg^-(a) = 2$, $\deg^-(b) = 2$, $\deg^-(c) = 3$,
$\deg^-(d) = 2$, $\deg^-(e) = 3$, $\deg^-(f) = 0$.

$\deg^+(a) = 4$, $\deg^+(b) = 1$, $\deg^+(c) = 2$,
$\deg^+(d) = 2$, $\deg^+(e) = 3$, $\deg^+(f) = 0$.

Directed Graphs (*continued*)

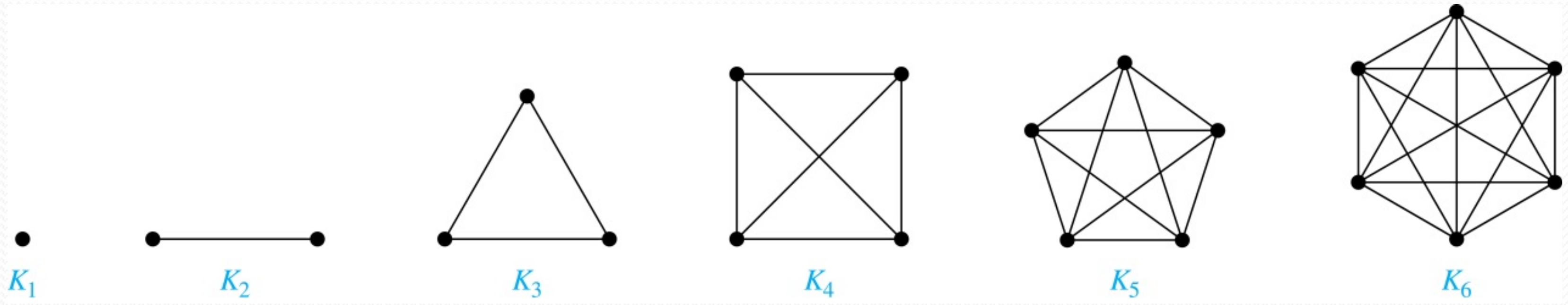
Theorem 3: Let $G = (V, E)$ be a graph with directed edges. Then:

$$|E| = \sum_{v \in V} deg^-(v) = \sum_{v \in V} deg^+(v).$$

Proof: The first sum counts the number of outgoing edges over all vertices and the second sum counts the number of incoming edges over all vertices. It follows that both sums equal the number of edges in the graph. ◀

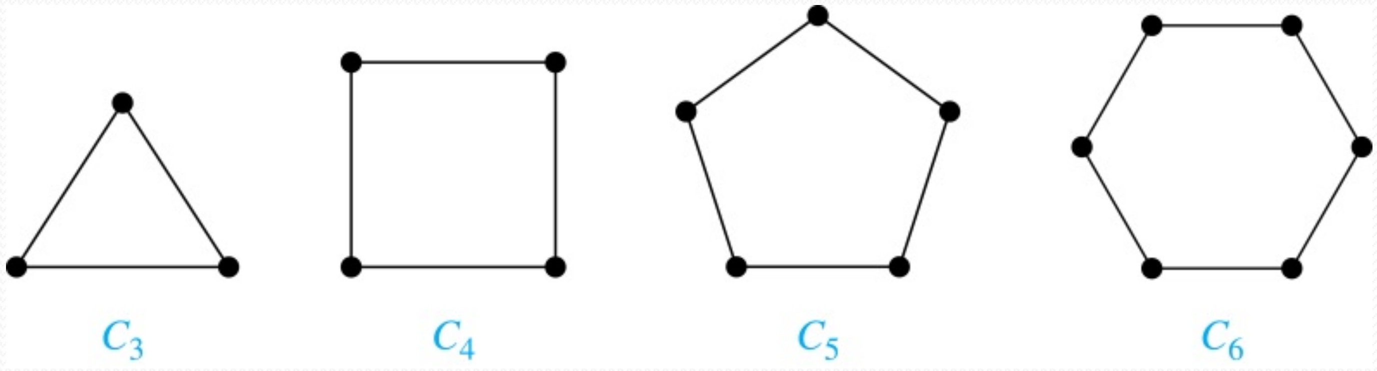
Special Types of Simple Graphs: Complete Graphs

A *complete graph on n vertices*, denoted by K_n, is the simple graph that contains exactly one edge between each pair of distinct vertices.



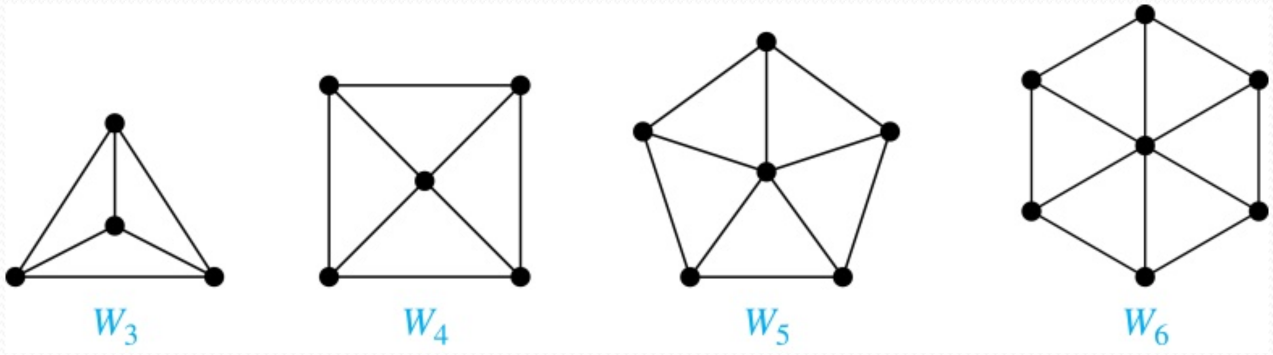
Special Types of Simple Graphs: Cycles and Wheels

A *cycle* C_n for $n \geq 3$ consists of n vertices $v_1, v_2, \cdots, v_n$, and edges $\{v_1, v_2\}, \{v_2, v_3\}, \cdots, \{v_{n-1}, v_n\}, \{v_n, v_1\}$.



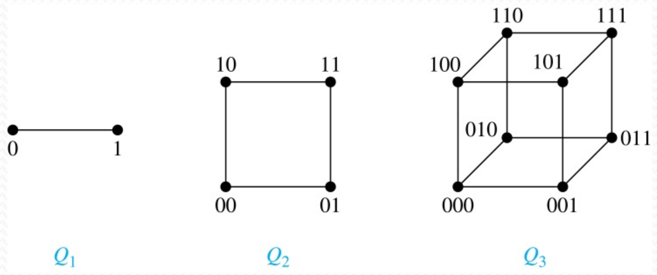
Special Types of Simple Graphs: Cycles and Wheels

A *wheel* W_n is obtained by adding an additional vertex to a cycle C_n for $n \geq 3$ and connecting this new vertex to each of the n vertices in C_n by new edges.



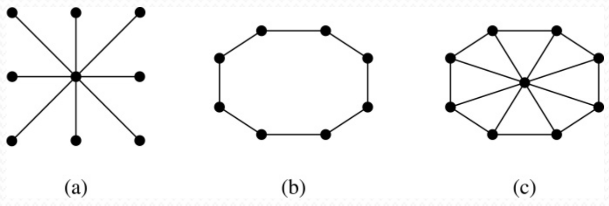
Special Types of Simple Graphs: *n*-Cubes

An *n-dimensional hypercube*, or *n-cube*, Q_n, is a graph with 2^n vertices representing all bit strings of length n, where there is an edge between two vertices that differ in exactly one bit position.



Special Types of Graphs and Computer Network Architecture

Various special graphs play an important role in the design of computer networks.

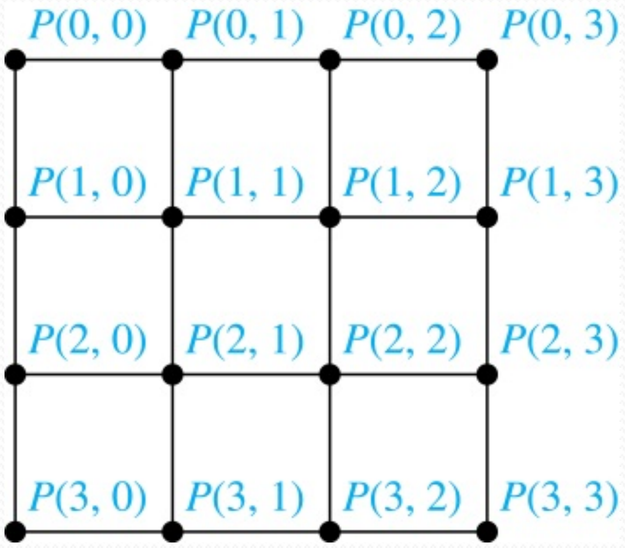


- Some local area networks use a *star topology*, which is a complete bipartite graph $K_{1,n}$,as shown in (a). All devices are connected to a central control device.
- Other local networks are based on a *ring topology*, where each device is connected to exactly two others using C_n ,as illustrated in (b). Messages may be sent around the ring.
- Others, as illustrated in (c), use a W_n – based topology, combining the features of a star topology and a ring topology.
-

Special Types of Graphs and Computer Network Architecture

Various special graphs also play a role in parallel processing where processors need to be interconnected as one processor may need the output generated by another.

- The *n-dimensional hypercube*, or *n-cube*, Q_n, is a common way to connect processors in parallel, e.g., Intel Hypercube.
- Another common method is the *mesh* network, illustrated here for 16 processors.



Bipartite Graphs

Definition: A simple graph G is bipartite if V can be partitioned into two disjoint subsets V_1 and V_2 such that every edge connects a vertex in V_1 and a vertex in V_2.

- In other words, there are no edges which connect two vertices in V_1 or in V_2.

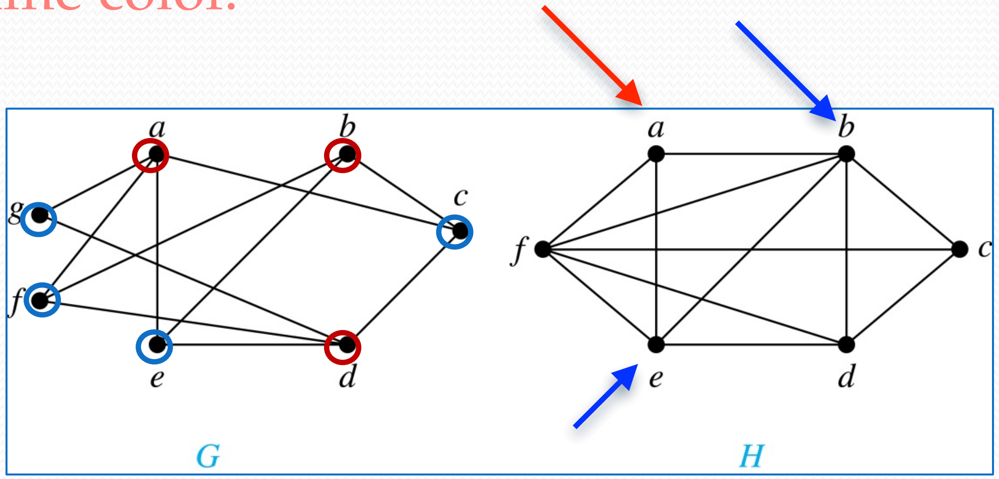
Bipartite Graphs

Definition: A simple graph G is bipartite if V can be partitioned into two disjoint subsets V_1 and V_2 such that every edge connects a vertex in V_1 and a vertex in V_2.

- In other words, there are no edges which connect two vertices in V_1 or in V_2.

Equivalent definition of a bipartite graph: if it is possible to color the vertices to red or blue so that no two adjacent vertices are the same color.

G is bipartite

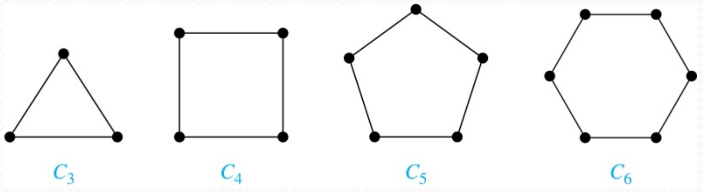


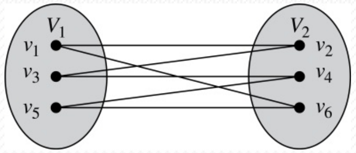
H is not bipartite since if we color a red, then the adjacent vertices f and b must both be blue.

Bipartite Graphs (*continued*)

Example: Show that C_6 is bipartite.

Solution: We can partition the vertex set into $V_1 = \{v_1, v_3, v_5\}$ and $V_2 = \{v_2, v_4, v_6\}$ so that every edge of C_6 connects a vertex in V_1 and V_2 .





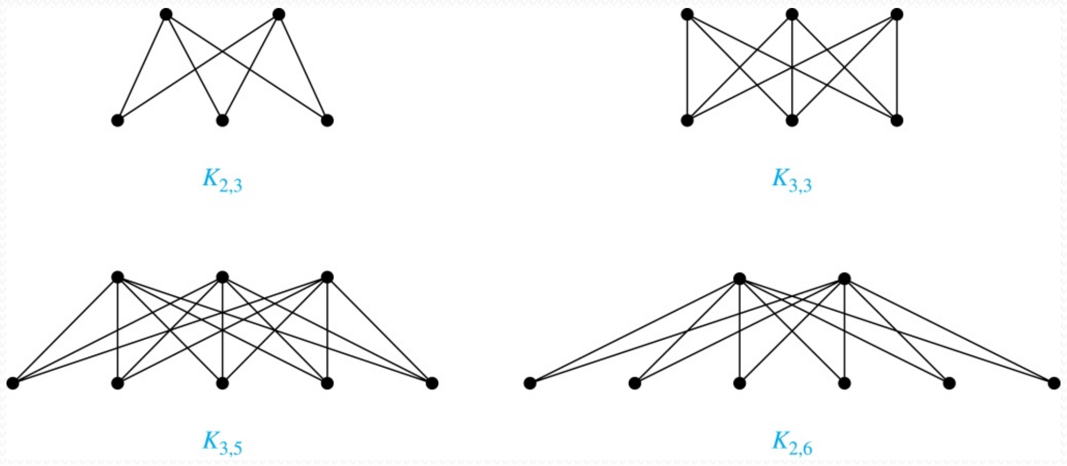
Example: Show that C_3 is not bipartite.

Solution: If we divide the vertex set of C_3 into two nonempty sets, one of the two must contain two vertices. But in C_3 every vertex is connected to every other vertex. Therefore, the two vertices in the same partition are connected. Hence, C_3 is not bipartite.

Complete Bipartite Graphs

Definition: A *complete bipartite graph, $K_{m,n}$*, is a graph that has its vertex set partitioned into two subsets V_1 of size m and V_2 of size n such that *there is an edge from every vertex in V_1 to every vertex in V_2*.

Example: We display four complete bipartite graphs here.

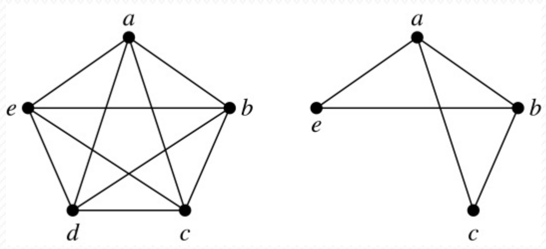


New Graphs from Old

Definition: A *subgraph of a graph* $G = (V, E)$ is a graph $H = (W, F)$, where $W \subset V$ and $F \subset E$.

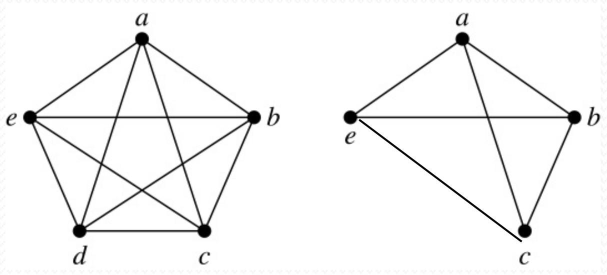
- A subgraph H of G is a *proper subgraph of* G *if* $H \neq G$.

Example: Here we show K_5 and one of its subgraphs.



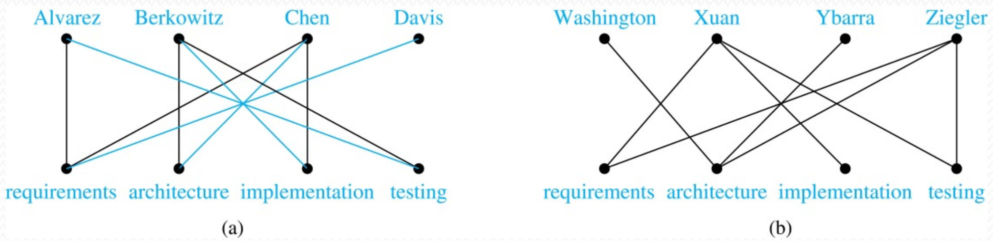
Definition: Let $G = (V, E)$ be a simple graph. The *subgraph induced* by a subset W of the vertex set V is the graph (W, F), where *the edge set* F *contains an edge in* E *if and only if both endpoints are in* W.

Example: Here we show K_5 and the *subgraph induced by* $W = \{a,b,c,e\}$.



Bipartite Graphs and Matchings

- Bipartite graphs are used to model applications that involve matching the elements of one set to elements in another, for example:
- *Job assignments* - vertices represent the jobs and the employees, edges link employees with those jobs they have been trained to do. A common goal is to match jobs to employees so that the most jobs are done.



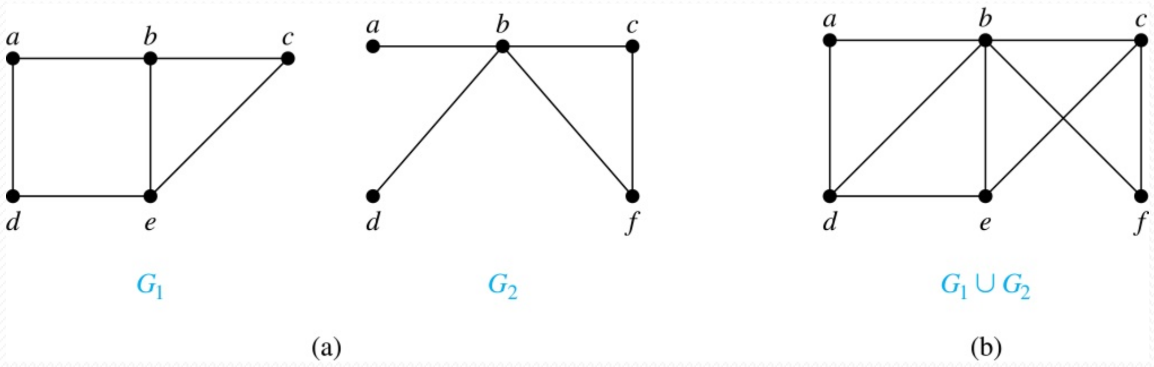
- *Marriage* - vertices represent the men and the women and edges link a a man and a woman if they are an acceptable spouse. We may wish to find the largest number of possible marriages.

See the text for more about matchings in bipartite graphs.

New Graphs from Old (*continued*)

Definition: The *union* of two simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the simple graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. The union of G_1 and G_2 is denoted by $G_1 \cup G_2$.

Example:



Representing Graphs and Graph Isomorphism

Section 10.3

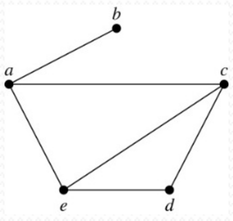
Section Summary

- Adjacency Lists
- Adjacency Matrices
- Incidence Matrices
- Isomorphism of Graphs

Representing Graphs: Adjacency Lists

Definition: An *adjacency list* can be used to represent a graph with no multiple edges by specifying the vertices that are adjacent to each vertex of the graph.

Example:



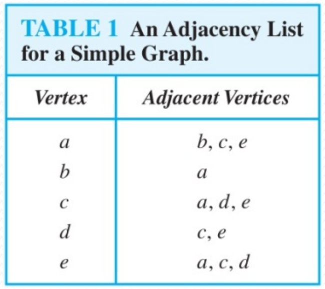
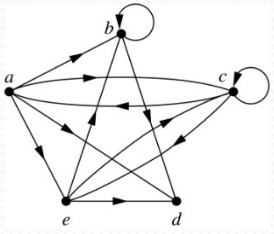


TABLE 1 An Adjacency List for a Simple Graph.

Vertex	*Adjacent Vertices*
a	*b*, *c*, *e*
b	*a*
c	*a*, *d*, *e*
d	*c*, *e*
e	*a*, *c*, *d*

Example:



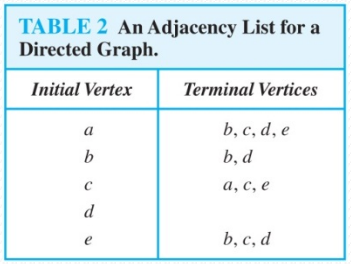


TABLE 2 An Adjacency List for a Directed Graph.

Initial Vertex	*Terminal Vertices*
a	*b*, *c*, *d*, *e*
b	*b*, *d*
c	*a*, *c*, *e*
d	
e	*b*, *c*, *d*

Representation of Graphs: Adjacency Matrices

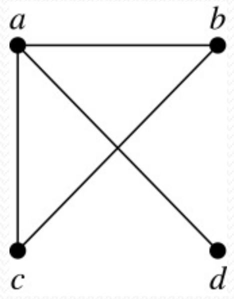
Definition: Suppose that $G = (V, E)$ is a simple graph where $|V| = n$.

- Arbitrarily list the vertices of G as $v_1, v_2, \ldots, v_n$.
- The *adjacency matrix* $\mathbf{A}_G$ of G, with respect to the listing of vertices, is the $n \times n$ zero-one matrix; formally:
- $\mathbf{A}_G = [a_{ij}]$, then $$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

Adjacency Matri

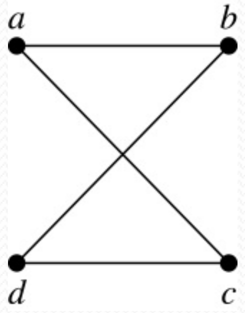
When a graph is sparse, that is, it has few edges relatively to the total number of possible edges, it is much more efficient to represent the graph using an adjacency list than an adjacency matrix.

Example:



$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

The ordering of vertices is a, b, c, d.



$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

The ordering of vertices is a, b, c, d.

But for a dense graph, which includes a high percentage of possible edges, an adjacency matrix is preferable.

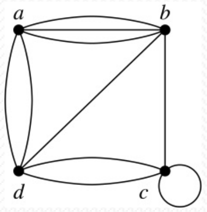
Note: The adjacency matrix of a simple graph is symmetric, i.e., $a_{ij} = a_{ji}$

Also, since there are no loops, each diagonal entry a_{ij} for $i = 1, 2, 3, \ldots, n$, is 0.

Adjacency Matrices (*continued*)

- Adjacency matrices can also be used to represent graphs with loops and multiple edges.
- A loop at the vertex v_i is represented by a 2 at the (i, i)th position of the matrix.
- When multiple edges connect the same pair of vertices v_i and v_j, (or if multiple loops are present at the same vertex), the (i, j)th entry equals the number of edges connecting the pair of vertices.

Example: We give the adjacency matrix of the pseudograph shown here using the ordering of vertices *a*, *b*, *c*, *d*.



$$\begin{bmatrix} 0 & 3 & 0 & 2 \\ 3 & 0 & 1 & 1 \\ 0 & 1 & \boxed{1} & 2 \\ 2 & 1 & 2 & 0 \end{bmatrix}$$

each loop adds 2, not 1! [Ref]
Notice this correction above also

Ref: https://en.wikipedia.org/wiki/Adjacency_matrix

Adjacency Matrices (*continued*)

- Adjacency matrices can also be used to represent directed graphs. In other words, if the graphs adjacency matrix is $\mathbf{A}_G = [a_{ij}]$, then:

$$a_{ij} = \begin{cases} 1 & \text{if } \{v_i, v_j\} \text{ is an edge of } G, \\ 0 & \text{otherwise.} \end{cases}$$

  - The adjacency matrix for a directed graph does not have to be symmetric, why?
  - To represent directed multigraphs, the value of a_{ij} is the number of edges connecting v_i to v_j.
  - When there is a loop, corresponding a_{ii} term contains 1 this time:

The in-degrees of a vertex can be computed by summing the entries of the corresponding column and the out-degree of vertex by summing the entries of the corresponding row.

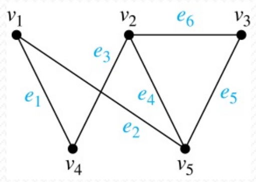
Representation of Graphs: Incidence Matrices

Definition: Let $G = (V, E)$ be an undirected graph with vertices where $v_1, v_2, \ldots v_n$ and edges $e_1, e_2, \ldots e_m$. The incidence matrix with respect to the ordering of V and E is the $n \times m$ matrix $\mathbf{M} = [m_{ij}]$, where

$$m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

Incidence Matrices (*continued*)

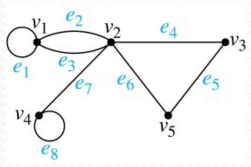
Example: Simple Graph and Incidence Matrix



$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

The rows going from top to bottom represent v_1 through v_5 and the columns going from left to right represent e_1 through e_6.

Example: Pseudograph and Incidence Matrix



$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

The rows going from top to bottom represent v_1 through v_5 and the columns going from left to right represent e_1 through e_8.

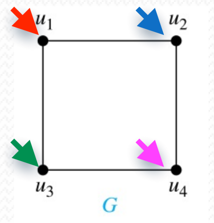
Isomorphism of Graphs

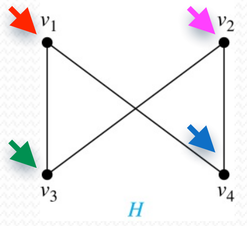
Definition: The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there is a one-to-one and onto function f from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2 , for all a and b in V_1 .

- Such a function f is called an *isomorphism*. Two simple graphs that are not isomorphic are called *nonisomorphic*.

Isomorphism of Graphs (*cont.*)

Example: Show that the graphs $G = (V, E)$ and $H = (W, F)$ are isomorphic.





Solution: The function f with $f(u_1) = v_1$, $f(u_2) = v_4$, $f(u_3) = v_3$, and $f(u_4) = v_2$ is a one-to-one correspondence between V and W. Note that adjacent vertices in G are u_1 and u_2, u_1 and u_3, u_2 and u_4, and u_3 and u_4. Each of the pairs $f(u_1) = v_1$ and $f(u_2) = v_4$, $f(u_1) = v_1$ and $f(u_3) = v_3$, $f(u_2) = v_4$ and $f(u_4) = v_2$, and $f(u_3) = v_3$ and $f(u_4) = v_2$ consists of two adjacent vertices in H.

Isomorphism of Graphs (*cont.*)

- It is difficult to determine whether two simple graphs are isomorphic using brute force because there are *n*! possible one-to-one correspondences between the vertex sets of two simple graphs with *n* vertices.
- The best algorithms for determining weather two graphs are isomorphic have *quasi-polynomial* worst case complexity in terms of the number of vertices of the graphs.
- Sometimes it is not hard to show that two graphs are not isomorphic. We can do so by finding a property, preserved by isomorphism, that only one of the two graphs has. Such a property is called *graph invariant*.

Next slide..

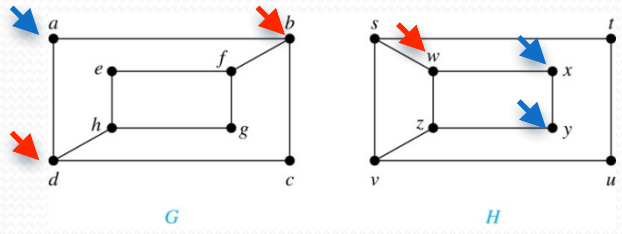
Isomorphism of Graphs (*cont.*)

- Sometimes it is not hard to show that two graphs are not isomorphic. We can do so by finding a property, preserved by isomorphism, that only one of the two graphs has. Such a property is called *graph invariant*.
- There are many different useful graph invariants that can be used to distinguish nonisomorphic graphs, such as:
  - the number of vertices,
  - the number of edges,
  - the degree sequence (list of the degrees of the vertices in nonincreasing order).

  - We will encounter others in later sections of this chapter.

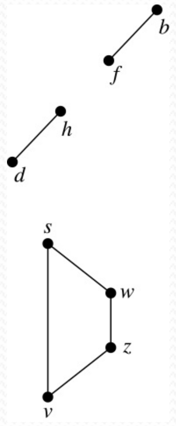
Isomorphism of Graphs (*cont.*)

Example: Determine whether these two graphs are isomorphic.



Solution: Both graphs have eight vertices and ten edges.
They also both have four vertices of degree two and four of degree three.

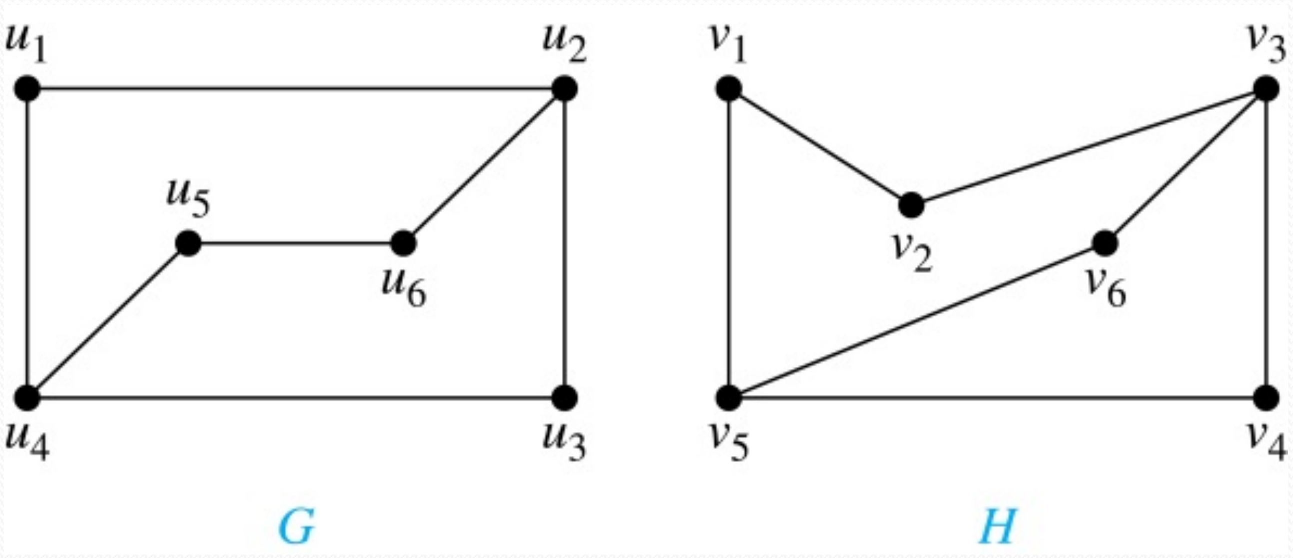
However, *G* and *H* are not isomorphic. Note that since *deg*(*a*) = 2 in *G*, *a* must correspond to *t*, *u*, *x*, or *y* in H, because these are the vertices of degree 2. But each of these vertices is adjacent to another vertex of degree two in *H*, which is not true for *a* in *G*.



Alternatively, note that the subgraphs of *G* and *H* made up of vertices of degree three and the edges connecting them must be isomorphic.
But the subgraphs, as shown at the right, are not isomorphic.

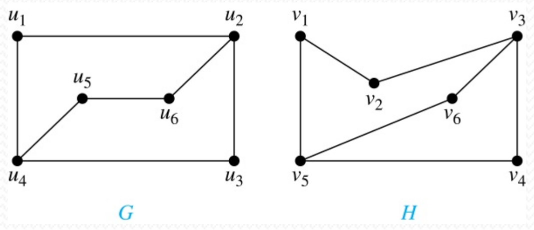
Isomorphism of Graphs (*cont.*)

Example: Determine whether these two graphs are isomorphic.



Isomorphism of Graphs (*cont.*)

Example: Determine whether these two graphs isomorphic.



Solution: Both graphs have six vertices and seven edges.
They also both have four vertices of degree two and two of degree three.
The subgraphs of *G* and *H* consisting of all the vertices of degree two and the edges connecting them are isomorphic. So, it is reasonable to try to find an isomorphism *f*.

We define an injection *f* from the vertices of *G* to the vertices of *H* that preserves the degree of vertices. We will determine whether it is an isomorphism.

The function *f* with $f(u_1) = v_6$, $f(u_2) = v_3$, $f(u_3) = v_4$, and $f(u_4) = v_5$, $f(u_5) = v_1$, and $f(u_6) = v_2$ is a one-to-one correspondence between *G* and *H*. Showing that this correspondence preserves edges is straightforward, so we will omit the details here. Because *f* is an isomorphism, it follows that *G* and *H* are isomorphic graphs.

See the text for an illustration of how adjacency matrices can be used for this verification.

Algorithms for Graph Isomorphism

- The best algorithms known for determining whether two graphs are isomorphic have *quasi-polynomial worst-case time complexity* (in the number of vertices of the graphs).
- However, there are algorithms with *linear average-case time complexity*.
- You can use a public domain program called NAUTY to determine in less than a second whether two graphs with as many as 100 vertices are isomorphic.
- Graph isomorphism is a problem of special interest because it is one of a few NP problems not known to be either tractable or NP-complete (see Section 3.3).

Applications of Graph Isomorphism

- Isomorphic graphs plays an important role in applications of graph theory. For example,
  - chemists use molecular graphs to model chemical compounds. Vertices represent atoms and edges represent chemical bonds. When a new compound is synthesized, a database of molecular graphs is checked to determine whether the graph representing the new compound is isomorphic to the graph of a compound that this already known.
  - Electronic circuits are modeled as graphs in which the vertices represent components and the edges represent connections between them. Graph isomorphism is the basis for
    - the verification that a particular layout of a circuit corresponds to the design's original schematics.
    - determining whether a chip from one vendor includes the intellectual property of another vendor.