

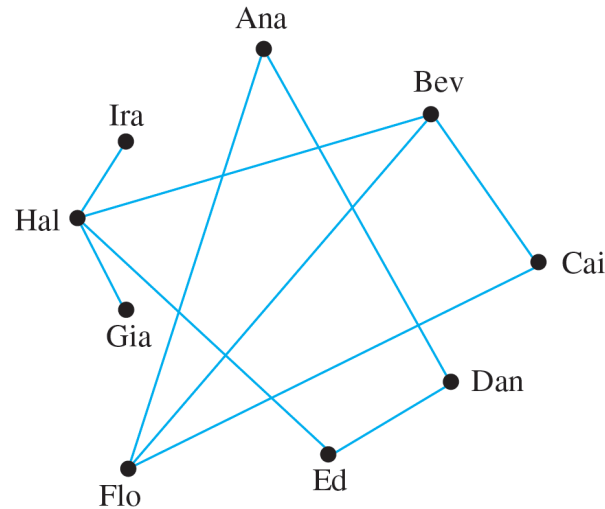
LECTURE 8

# Graphs

# The Language of Graphs (1/8)

- ▶ Imagine an organization that wants to set up teams of three to work on some projects.
- ▶ In order to maximize the number of people on each team who had previous experience working together successfully, the director asked the members to provide names of their previous partners.

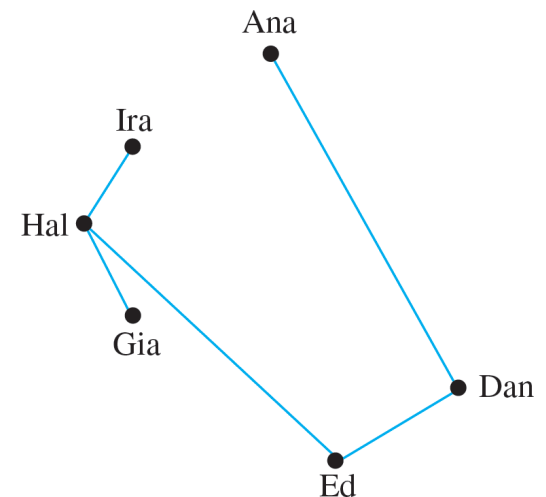
| Name | Previous Partners |
|------|-------------------|
| Ana  | Dan, Flo          |
| Bev  | Cai, Flo, Hal     |
| Cai  | Bev, Flo          |
| Dan  | Ana, Ed           |
| Ed   | Dan, Hal          |
| Flo  | Cai, Bev, Ana     |
| Gia  | Hal               |
| Hal  | Gia, Ed, Bev, Ira |
| Ira  | Hal               |



# The Language of Graphs (2/8)

# The Language of Graphs (3/8)

► From the diagram, it is easy to see that Bev, Cai, and Flo are a group of three previous partners, and so it would be reasonable for them to form one of these teams. The drawing below shows the result when these three names are removed from the diagram.



# The Language of Graphs

## (4/8)

- ▶ This drawing shows that placing Hal on the same team as Ed would leave Gia and Ira on a team where they would not have a previous partner.
- ▶ However, if Hal is placed on a team with Gia and Ira, then the remaining team would consist of Ana, Dan, and Ed, and everyone on both teams would be working with a previous partner.



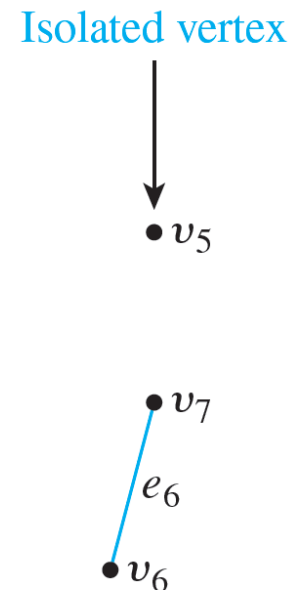
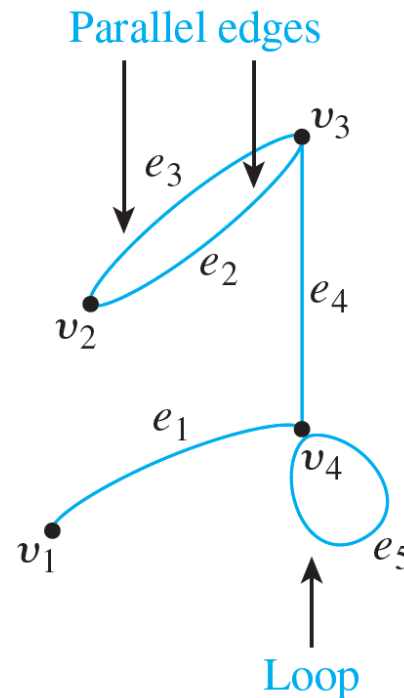
# The Language of Graphs (5/8)

► Drawings such as these are illustrations of a structure known as a graph. The dots are called vertices (plural of vertex) and the line segments joining vertices are called edges.

► As you can see from the first drawing, it is possible for two edges to cross at a point that is not a vertex. Note also that the type of graph described here is quite different from the “graph of an equation” or the “graph of a function.”

# The Language of Graphs (6/8)

► In general, a graph consists of a set of vertices and a set of edges connecting various pairs of vertices. The edges may be straight or curved and should either connect one vertex to another or a vertex to itself, as shown below.



# The Language of Graphs

## (7/8)

- ▶ In this drawing, the vertices are labeled with  $v$ 's and the edges with  $e$ 's. When an edge connects a vertex to itself (as  $e_5$  does), it is called a loop.
- ▶ When two edges connect the same pair of vertices (as  $e_2$  and  $e_3$  do), they are said to be parallel. It is quite possible for a vertex to be unconnected by an edge to any other vertex in the graph (as  $v_5$  is), and in that case the vertex is said to be isolated.



## Definition

A **graph**  $G$  consists of two finite sets: a nonempty set  $V(G)$  of **vertices** and a set  $E(G)$  of **edges**, where each edge is associated with a set consisting of either one or two vertices called its **endpoints**. The correspondence from edges to endpoints is called the **edge-endpoint function**.

An edge with just one endpoint is called a **loop**, and two or more distinct edges with the same set of endpoints are said to be **parallel**. An edge is said to **connect** its endpoints; two vertices that are connected by an edge are called **adjacent**; and a vertex that is an endpoint of a loop is said to be **adjacent to itself**.

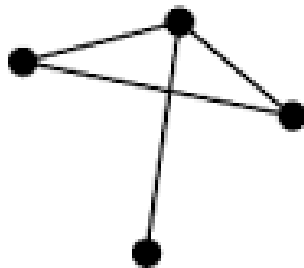
An edge is said to be **incident on** each of its endpoints, and two edges incident on the same endpoint are called **adjacent**. A vertex on which no edges are incident is called **isolated**.

# The Language of Graphs (8/8)

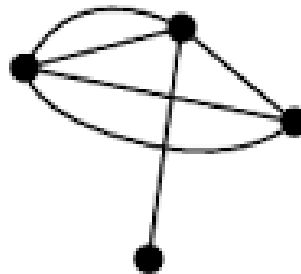
# Terminologies

10

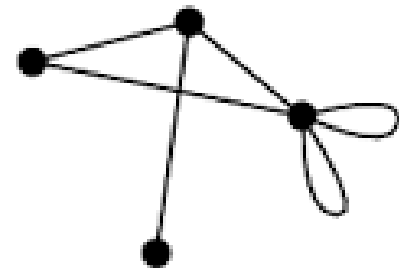
- **Simple Graph:** A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called simple graph.
- **Multigraphs:** A graph that has more than one(multiple) edge connecting the same pair of vertices
- **Loops:** A loops is an edge that connects a vertex to itself.
- **Pseudograph:** Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself, are sometimes called as Pseudographs.



*simple graph*



*nonsimple graph  
with multiple edges*



*nonsimple graph  
with loops*

# Terminologies

- **Disconnected Graph:** Any vertices/node that are not connected by edges are called disconnected graphs. No path exist for a node- Isolated node.
- **Connected Graph:** All the vertices/nodes are connected by at least one or more edges are called connected graphs. At lease one path should exist.
- **Fully connected Graph (strongly):** All the vertices/nodes connected to all other vertices/nodes, then it is called fully connected graphs. Mainly directed graphs.
- **Complete:** All vertices/nodes are connected to each other directly. Represented by  $K_n$

# Graph Terminology

12

| 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            |

# More Terminologies

- **Complete Bipartite Graph:** When all the elements of one group are connected to all the elements in the other group. But no elements are connected within the group. Graph is in two parts such that nodes in one part is not connected to each other while they are connected to all the nodes in other parts. Represented by  $K_{m,n}$
- **Mixed Graph:** Includes undirected and directed edges.
- **Neighbors:** Two vertices  $a$  and  $b$  in an undirected graph  $G$  are called neighbors in  $G$  if  $a$  and  $b$  are endpoints of an edge  $e$  of  $G$ .

Neighbor ( $a$ ) =  $b$

Neighbor ( $b$ ) = ( $a, c, d$ )



# Graph Models



## Graph Models:

Social Networks  
Acquaintance and  
Friendship Graphs  
Influence Graphs  
Collaboration Graphs



## Computer Networks:

Call Graphs



## Information Networks:

The web graph  
Citation Graphs



## Software Design Applications:

Model Dependency Graphs  
Precedence Graphs and  
Concurrent Processing



## Transportation:

Airline Routes/Air-Traffic  
control  
Road networks



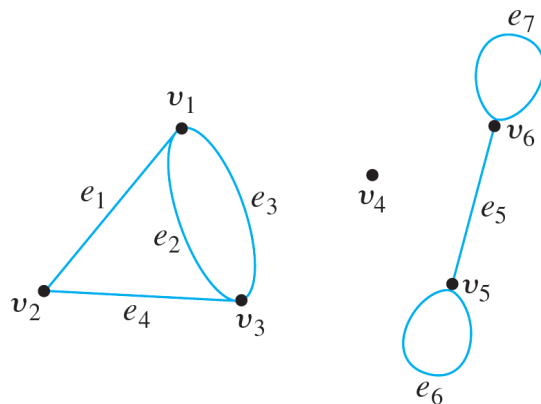
Biological Networks: Niche Overlap Graph in  
Ecology, Protein Interaction Graph



Tournaments: Round-robin and Single-  
Elimination Tournaments.

# Example 1.4.1 – Terminology (1/2)

Consider the following graph:



a. Write the vertex set and the edge set, and give a table showing the edge-endpoint function.

# Example 1.4.1 – Terminology (2/2)

- ▶ b. Find all edges that are incident on  $v_1$ , all vertices that are adjacent to  $v_1$ , all edges that are adjacent to  $e_1$ , all loops, all parallel edges, all vertices that are adjacent to themselves, and all isolated vertices.

# Example

## 1.4.1 –

### Solution (1/2)

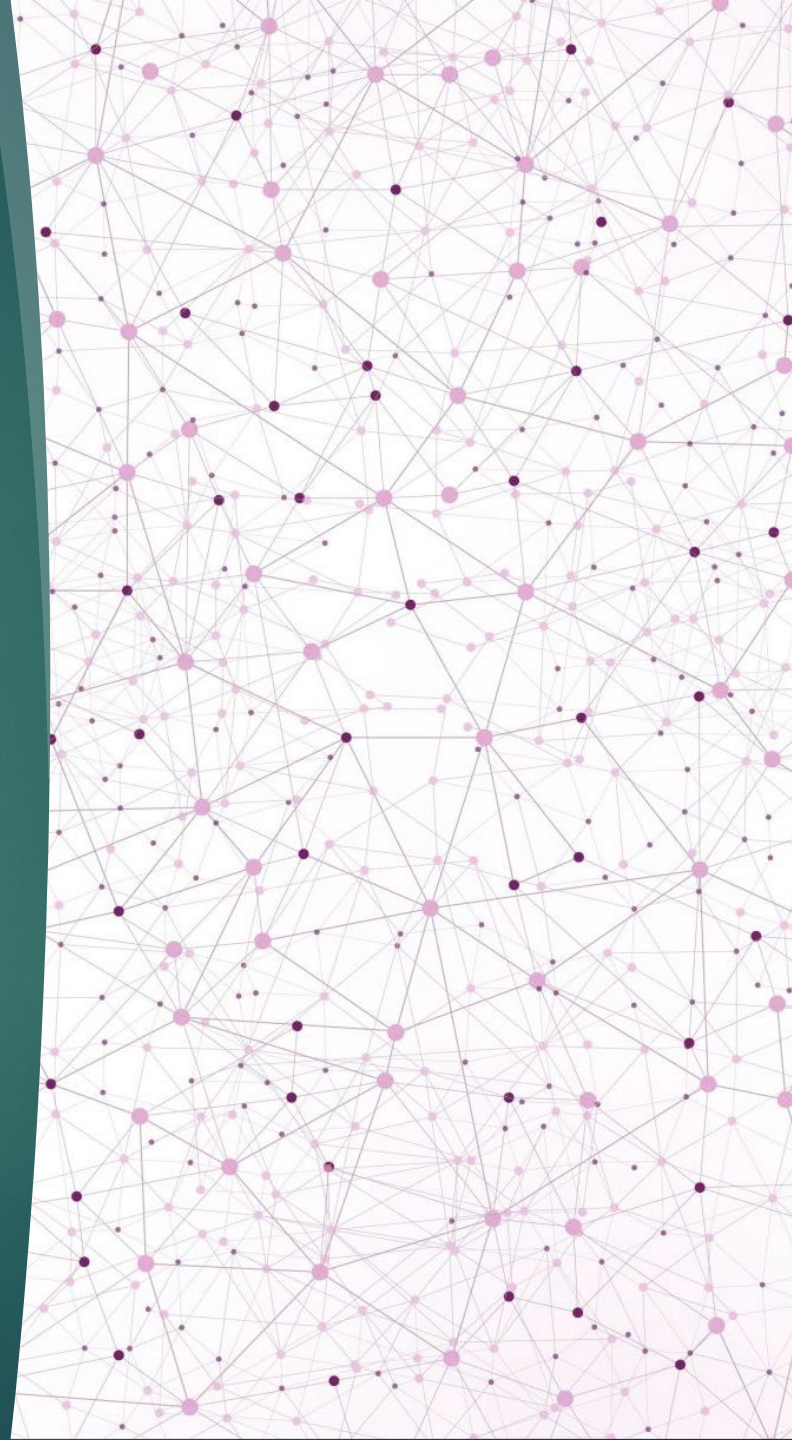
- ▶ a. vertex set =  $\{v_1, v_2, v_3, v_4, v_5, v_6\}$
- ▶ edge set =  $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- ▶ edge-endpoint function:

| Edge  | Endpoints      |
|-------|----------------|
| $e_1$ | $\{v_1, v_2\}$ |
| $e_2$ | $\{v_1, v_3\}$ |
| $e_3$ | $\{v_1, v_3\}$ |
| $e_4$ | $\{v_2, v_3\}$ |
| $e_5$ | $\{v_5, v_6\}$ |
| $e_6$ | $\{v_5\}$      |
| $e_7$ | $\{v_6\}$      |



# Example 1.4.1 – Solution (2/2)

- ▶ b.  $e_1, e_2,$  and  $e_3$  are incident on  $v_1$ .
  - ▶  $v_2$  and  $v_3$  are adjacent to  $v_1$ .
  - ▶  $e_2, e_3,$  and  $e_4$  are adjacent to  $e_1$ .
  - ▶  $e_6$  and  $e_7$  are loops.
  - ▶  $e_2$  and  $e_3$  are parallel.
  - ▶  $v_5$  and  $v_6$  are adjacent to themselves.
  - ▶  $v_4$  is an isolated vertex.

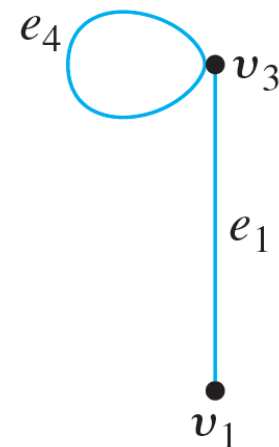
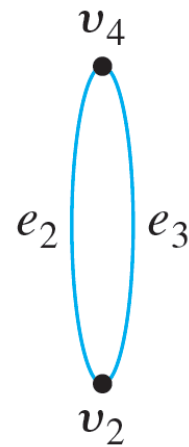
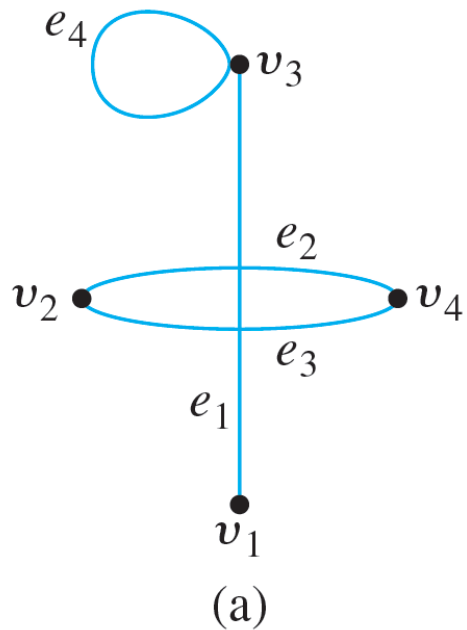




## Example 1.4.2 – Drawing More Than One Picture for a Graph (1/2)

- ▶ Consider the graph specified as follows:
- ▶ vertex set =  $\{v_1, v_2, v_3, v_4\}$
- ▶ edge set =  $\{e_1, e_2, e_3, e_4\}$
- ▶ edge-endpoint function:

| Edge  | Endpoints      |
|-------|----------------|
| $e_1$ | $\{v_1, v_3\}$ |
| $e_2$ | $\{v_2, v_4\}$ |
| $e_3$ | $\{v_2, v_4\}$ |
| $e_4$ | $\{v_3\}$      |



(b)

## Example 1.4.2 – Drawing More Than One Picture for a Graph (2/2)

BOTH DRAWINGS (A) AND (B) SHOWN BELOW ARE PICTORIAL REPRESENTATIONS OF THIS GRAPH.

## Example 1.4.3 – Labeling Drawings to Show They Represent the Same Graph

Consider the two drawings shown in Figure 1.4.1. Label vertices and edges in such a way that both drawings represent the same graph.

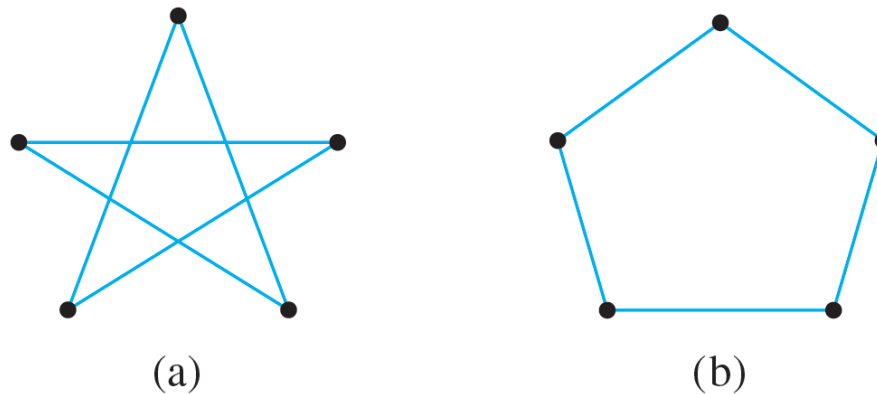
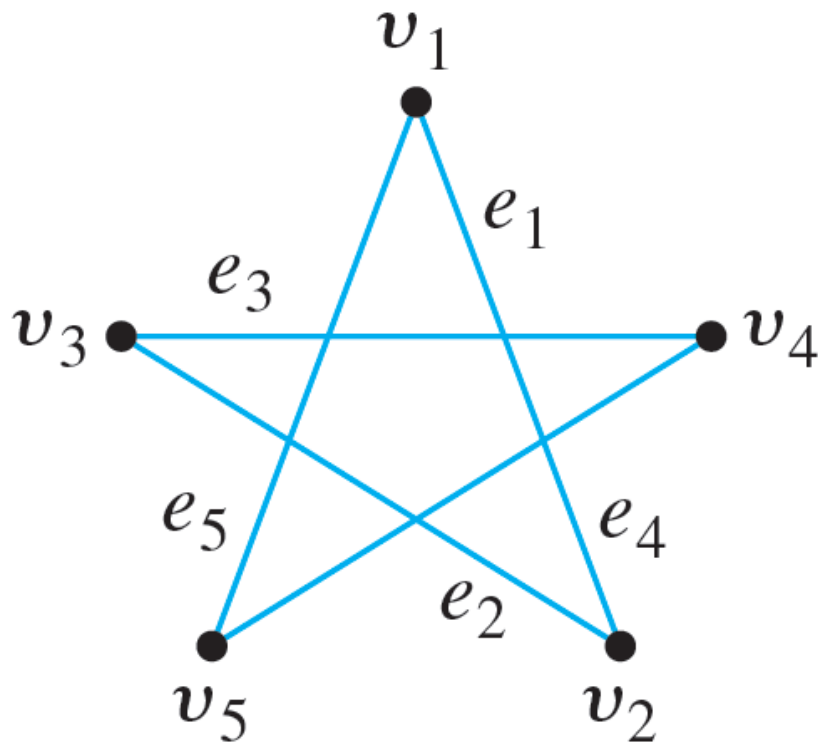


Figure 1.4.1

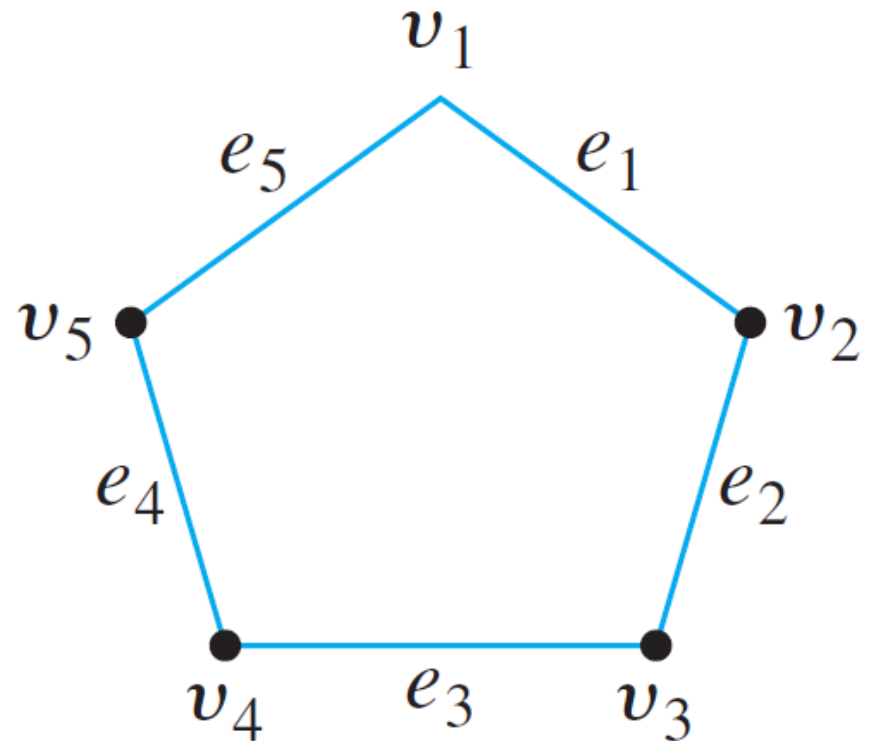
## Example 1.4.3 – Solution (1/3)



► Imagine putting one end of a piece of string at the top vertex of Figure 1.4.1 (a) (call this vertex  $v_1$ ), then laying the string to the next adjacent vertex on the lower right (call this vertex  $v_2$ ), then laying it to the next adjacent vertex on the upper left ( $v_3$ ), and so forth, returning finally to the top vertex  $v_1$ . Call the first edge  $e_1$ , the second  $e_2$ , and so forth, as shown below.

## Example 1.4.3 – Solution (2/3)

► Now imagine picking up the piece of string, together with its labels, and repositioning it as follows:







# Example 1.4.3 – Solution (3/3)

► This is the same as Figure 1.4.1 (b), so both drawings represent the graph with vertex set  $\{v_1, v_2, v_3, v_4, v_5\}$ , edge set  $\{e_1, e_2, e_3, e_4, e_5\}$ , and edge-endpoint function as follows:

| Edge  | Endpoints      |
|-------|----------------|
| $e_1$ | $\{v_1, v_2\}$ |
| $e_2$ | $\{v_2, v_3\}$ |
| $e_3$ | $\{v_3, v_4\}$ |
| $e_4$ | $\{v_4, v_5\}$ |
| $e_5$ | $\{v_5, v_1\}$ |

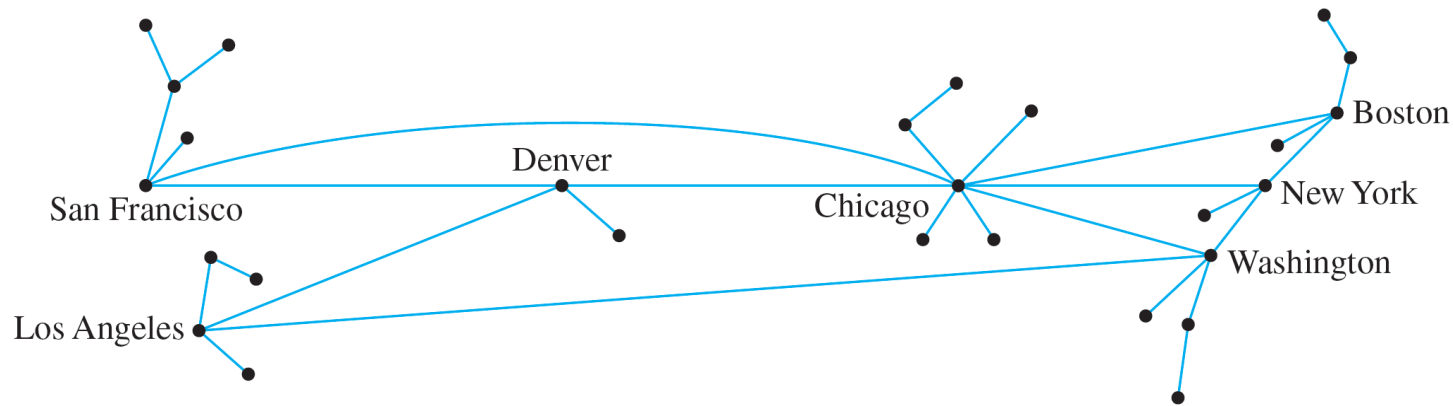


# Examples of Graphs

## Example 1.4.4 – *Using a Graph to Represent a Network* (1/2)

- ▶ Telephone, electric power, gas pipeline, and air transport systems can all be represented by graphs, as can computer networks—from small local area networks to the global Internet system that connects millions of computers worldwide.
- ▶ Questions that arise in the design of such systems involve choosing connecting edges to minimize cost, optimize a certain type of service, and so forth.





## Example 1.4.4 – Using a Graph to Represent a Network (2/2)

A TYPICAL NETWORK, CALLED A HUB-AND-SPOKE MODEL, IS SHOWN BELOW.

# Examples of Graphs (1/2)

- ▶ A directed graph is like an (undirected) graph except that each edge is associated with an ordered pair of vertices rather than a set of vertices. Thus each edge of a directed graph can be drawn as an arrow going from the first vertex to the second vertex of the ordered pair.

## Definition

A **directed graph**, or **digraph**, consists of two finite sets: a nonempty set  $V(G)$  of vertices and a set  $D(G)$  of directed edges, where each is associated with an ordered pair of vertices called its **endpoints**. If edge  $e$  is associated with the pair  $(v, w)$  of vertices, then  $e$  is said to be the **(directed) edge** from  $v$  to  $w$ .



## Example 1.4.6 – Using a Graph to Represent Knowledge (1/3)

► In many applications of artificial intelligence, a knowledge base of information is collected and represented inside a computer. Because of the way the knowledge is represented and because of the properties that govern the artificial intelligence program, the computer is not limited to retrieving data in the same form as it was entered; it can also derive new facts from the knowledge base by using certain built-in rules of inference. For example, from the knowledge that the Los Angeles Times is a big-city daily and that a big-city daily contains national news, an artificial intelligence program could infer that the Los Angeles Times contains national news.

# Example 1.4.6 – Using a Graph to Represent Knowledge (2/3)

The directed graph shown in Figure 1.4.2 is a pictorial representation for a simplified knowledge base about periodical publications.

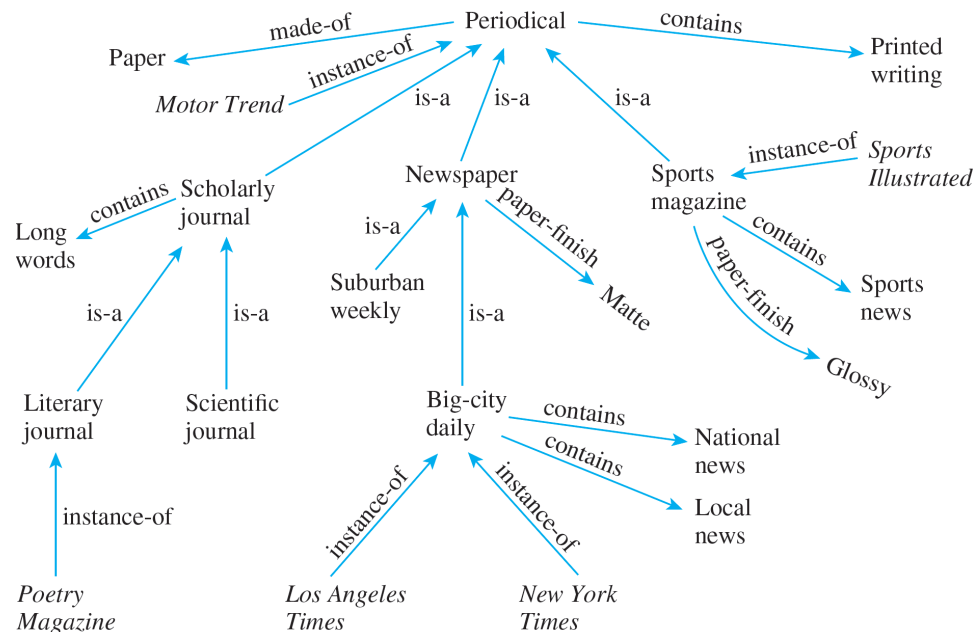


Figure 1.4.2



# Example 1.4.6 – Using a Graph to Represent Knowledge (3/3)

ACCORDING TO THIS KNOWLEDGE  
BASE, WHAT PAPER FINISH DOES THE  
NEW YORK TIMES USE?



# Example 1.4.6 – Solution

- ▶ The arrow going from New York Times to big-city daily (labeled “instanceof”) shows that the New York Times is a big-city daily. The arrow going from big-city daily to newspaper (labeled “is-a”) shows that a big-city daily is a newspaper.
- ▶ The arrow going from newspaper to matte (labeled “paper-finish”) indicates that the paper finish on a newspaper is matte. Hence it can be inferred that the paper finish on the New York Times is matte.



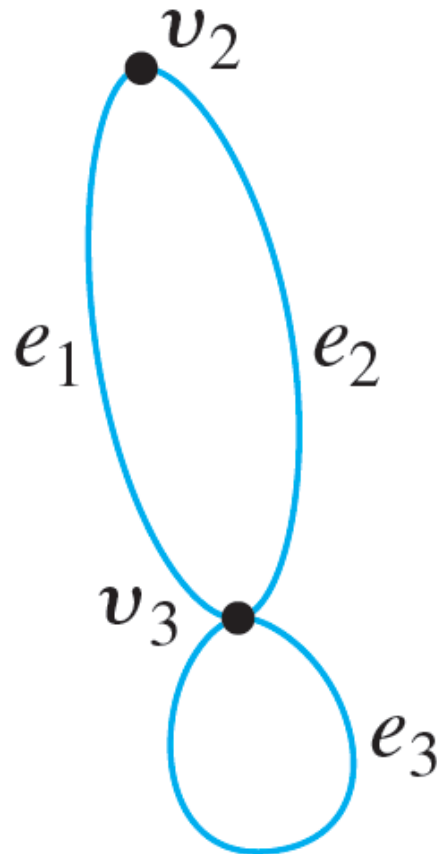
## Definition

Let  $G$  be a graph and  $v$  a vertex of  $G$ . The **degree of  $v$** , denoted  $\deg(v)$ , equals the number of edges that are incident on  $v$ , with an edge that is a loop counted twice.

# Examples of Graphs (2/2)



$v_1$



## Example 1.4.8 – Degree of a Vertex

FIND THE DEGREE OF  
EACH VERTEX OF THE  
GRAPH  $G$  SHOWN  
BELOW.

# Example 1.4.8 – Solution

$\deg(v_1) = 0$  since no edge is incident on  $v_1$  ( $v_1$  is isolated).

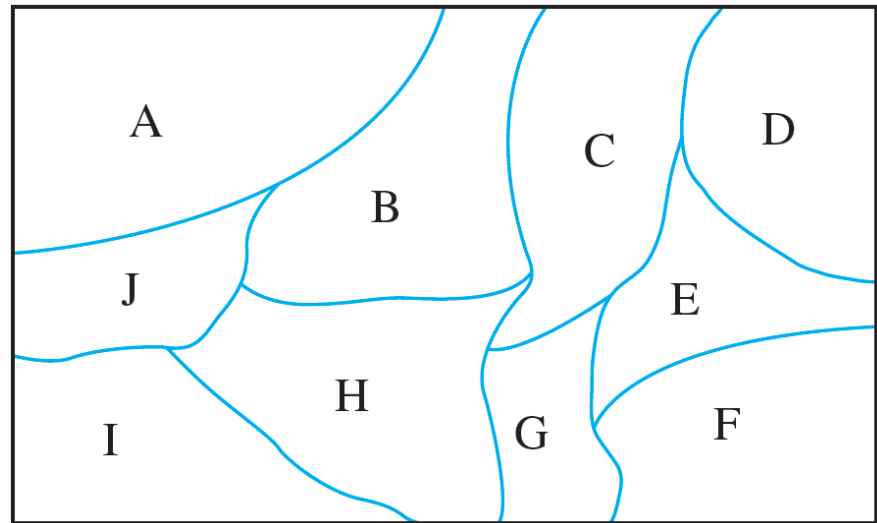
$\deg(v_2) = 2$  since both  $e_1$  and  $e_2$  are incident on  $v_2$ .

$\deg(v_3) = 4$  since  $e_1$  and  $e_2$  are incident on  $v_3$  and the loop  $e_3$  is also incident on  $v_3$  (and contributes 2 to the degree of  $v_3$ ).

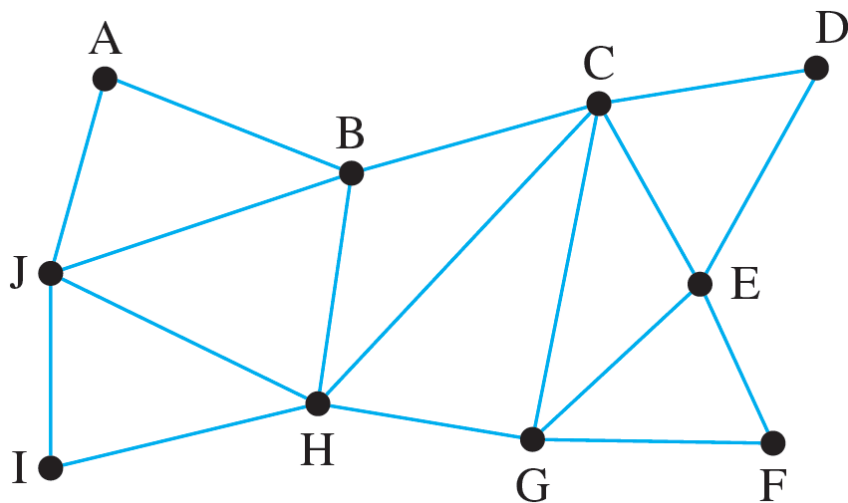
## Example 1.4.9

– Using a Graph to Color a Map

► Imagine that the diagram shown below is a map with countries labeled A–J. Show that you can color the map so that no two adjacent countries have the same color.



# Example 1.4.9 – Solution (1/10)



► Notice that coloring the map does not depend on the sizes or shapes of the countries, but only on which countries are adjacent to which. So, to figure out a coloring, you can draw a graph, as shown below, where vertices represent countries and where edges are drawn between pairs of vertices that represent adjacent countries.

# Example 1.4.9 – Solution (2/10)

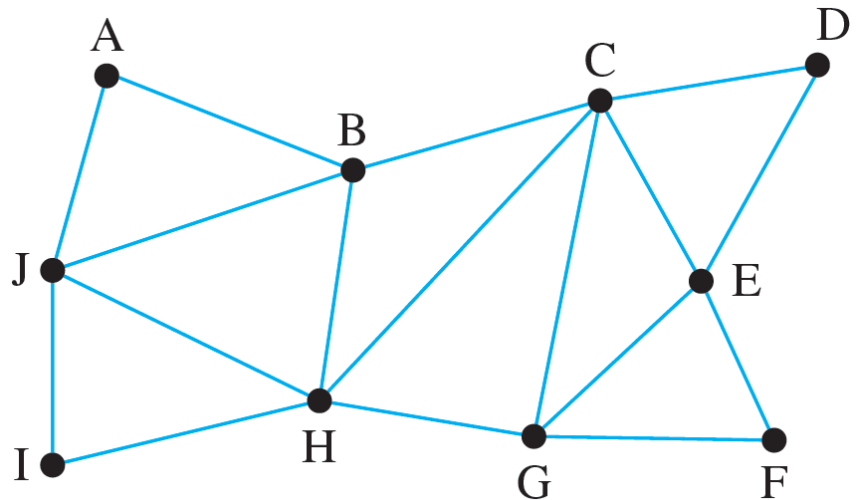
Coloring the vertices of the graph will translate to coloring the countries on the map.

As you assign colors to vertices, a relatively efficient strategy is, at each stage, to focus on an uncolored vertex that has maximum degree, in other words that is connected to a maximum number of other uncolored vertices.



# Example 1.4.9 – Solution (3/10)

► If there is more than one such vertex, it does not matter which you choose because there are often several acceptable colorings for a given graph.



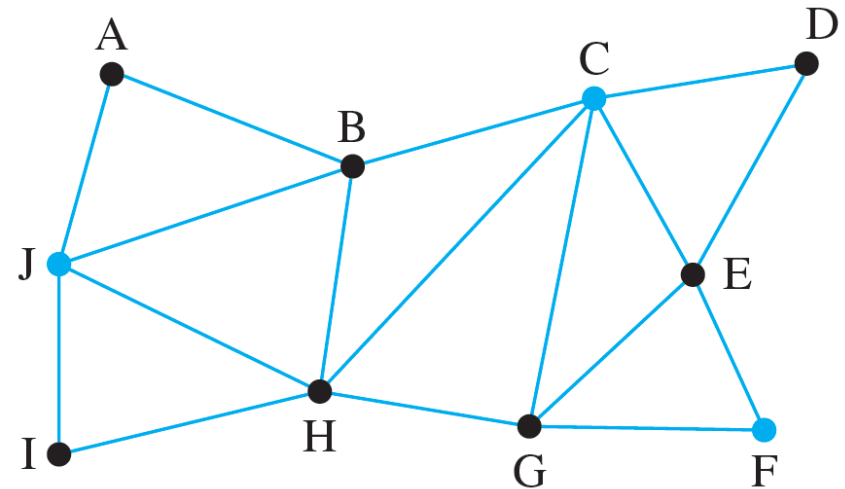
# Example 1.4.9 – Solution

## (4/10)

- ▶ For this graph, both C and H have maximum degree so you can choose one, say, C, and color it, say, blue.
- ▶ Now since A, F, I, and J are not connected to C, some of them may also be colored blue, and, because J is connected to a maximum number of others, you could start by coloring it blue.

# Example 1.4.9 – Solution (5/10)

► Then F is the only remaining vertex not connected to either C or J, so you can also color F blue. The drawing below shows the graph with vertices C, J, and F colored blue.



# Example 1.4.9 – Solution (6/10)

Since the vertices adjacent to  $C$ ,  $J$ , and  $F$  cannot be colored blue, you can simplify the job of choosing additional colors by removing  $C$ ,  $J$ , and  $F$  and the edges connecting them to adjacent vertices. The result is shown in Figure 1.4.4a.

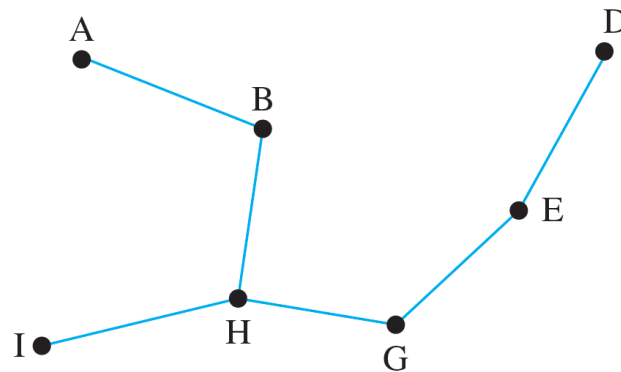


Figure 1.4.4(a)

# Example 1.4.9 – Solution

## (7/10)

- ▶ In the simplified graph again choose a vertex that has a maximum degree, namely H, and give it a second color, say, gray.
- ▶ Since A, D, and E are not connected to H, some of them may also be colored gray, and, because E is connected to a maximum number of these vertices, you could start by coloring E gray.



# Example 1.4.9 – Solution (8/10)

Then  $A$  is not connected to  $E$ , and so you can also color  $A$  gray. This is shown in Figure 1.4.4b.

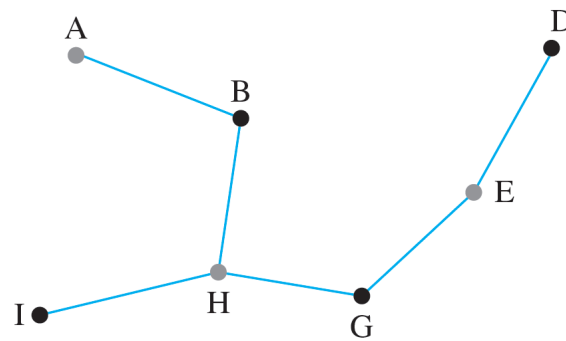
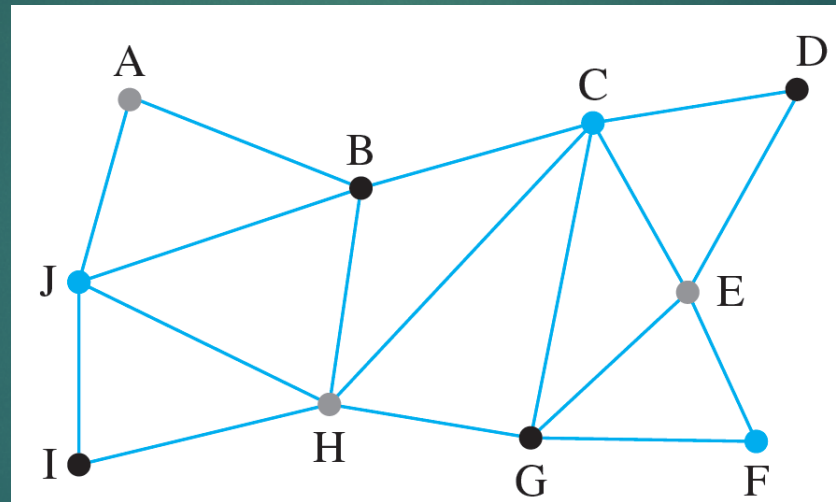


Figure 1.4.4(b)

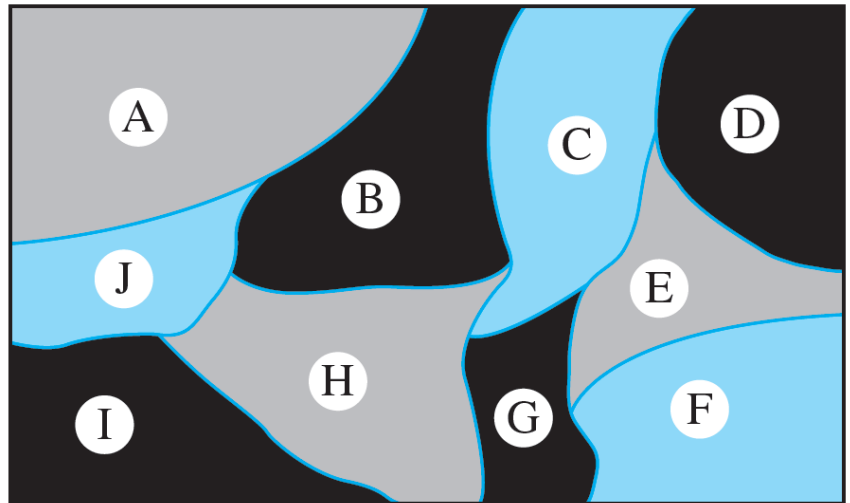
# Example 1.4.9 – Solution (9/10)

► The drawing below shows the original graph with vertices  $C$ ,  $J$ , and  $F$  colored blue, vertices  $H$ ,  $A$ , and  $E$ , colored gray, and the remaining vertices colored black. You can check that no two adjacent vertices have the same color.



# Example 1.4.9 – Solution (10/10)

► Translating the graph coloring back to the original map gives the following picture in which no two adjacent countries have the same color.







# Adjacency Matrices



# Adjacency Matrix Structure

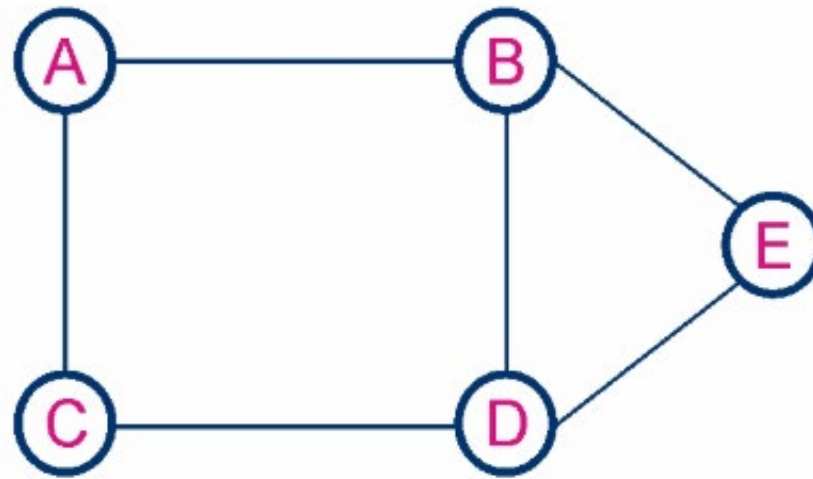
- ▶ Let  $G=(V,E)$  be a graph with  $n$  nodes,  $n>0$ .  $a_{ij}$  is an adjacency matrix of  $G$ . The  $a_{ij}$  is an  $n \times n$  array whose elements are given by

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

The adjacency matrix is a square matrix of size  $N \times N$ , where  $N$  is the number of vertices in the graph.

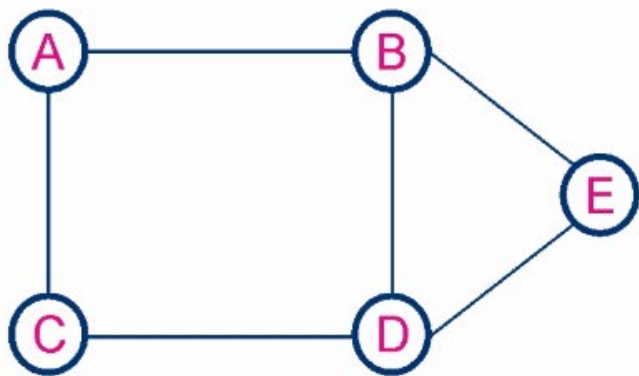
- Each element  $(i, j)$  of the matrix represents the presence (1) or absence (0) of an edge between vertex  $i$  and vertex  $j$ .
- For **undirected graphs**, the matrix is symmetric, meaning  $\mathbf{a[i][j]} = \mathbf{a[j][i]}$ .
- For **directed graphs**, the adjacency matrix is not necessarily symmetric, meaning  $\mathbf{a[i][j]} \neq \mathbf{a[j][i]}$ .

# Adjacency Matrix



Undirected Graph

# Adjacency Matrix

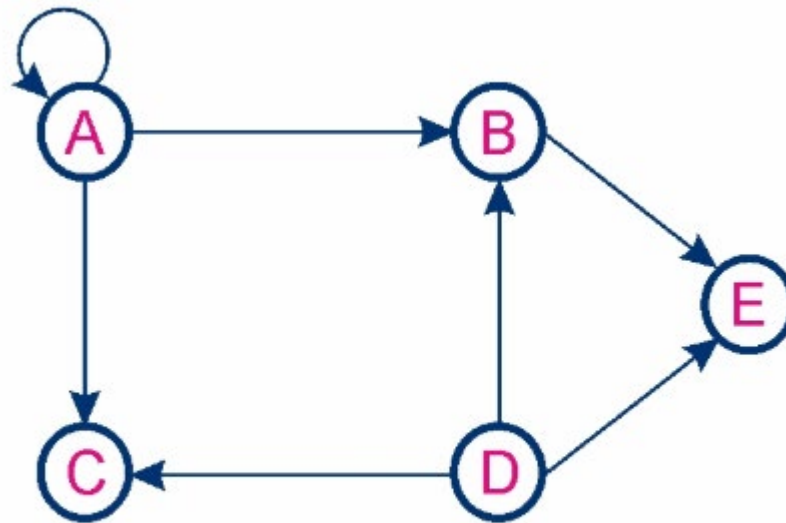


Undirected Graph



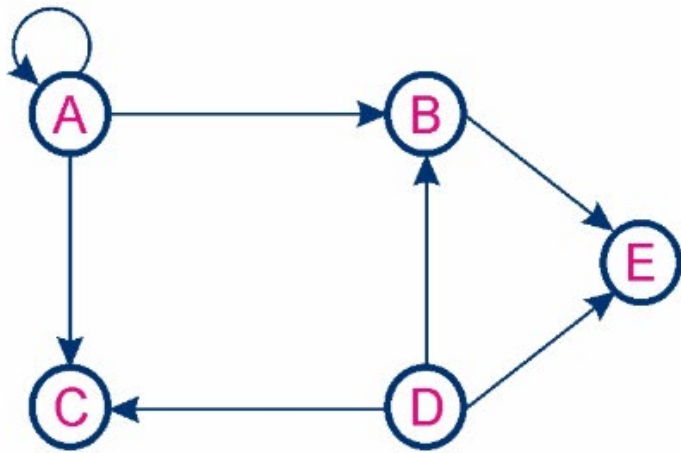
|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 0 |
| B | 1 | 0 | 0 | 1 | 1 |
| C | 1 | 0 | 0 | 1 | 0 |
| D | 0 | 1 | 1 | 0 | 1 |
| E | 0 | 1 | 0 | 1 | 0 |

# Adjacency Matrix



Directed Graph

# Adjacency Matrix



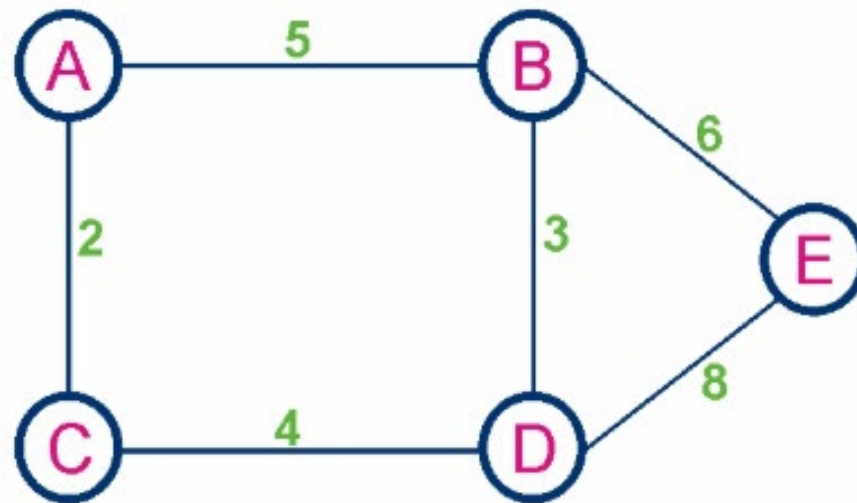
Directed Graph



|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 1 | 1 | 1 | 0 | 0 |
| B | 0 | 0 | 0 | 0 | 1 |
| C | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

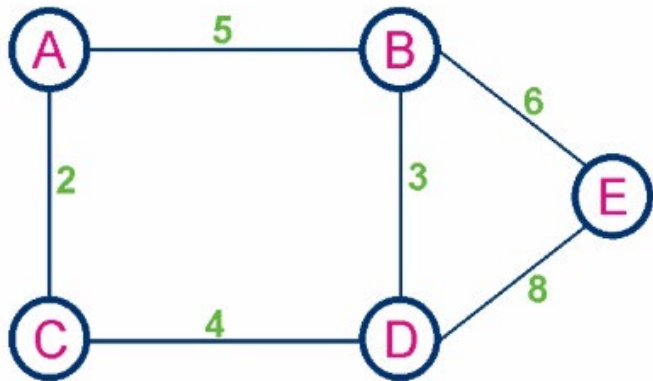


# Adjacency Matrix



Weighted Graph

# Adjacency Matrix



Weighted Graph



|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 5 | 2 | 0 | 0 |
| B | 5 | 0 | 0 | 3 | 6 |
| C | 2 | 0 | 0 | 4 | 0 |
| D | 0 | 3 | 4 | 0 | 8 |
| E | 0 | 6 | 0 | 8 | 0 |

# Operations on Adjacency Matrix

- ▶ **Creating an Adjacency Matrix:** An adjacency matrix represents a graph where the rows and columns correspond to nodes, and the entries in the matrix indicate whether pairs of vertices are adjacent or not in the graph.
- ▶ **Adding an Edge:** To add an edge between two vertices, you set the value at the corresponding row and column to 1 (or the weight of the edge in the case of a weighted graph).
- ▶ **Removing an Edge:** To remove an edge, you set the value at the corresponding row and column to 0.

# Operations on Adjacency Matrix

- ▶ **Checking for an Edge:** To check if there is an edge between two vertices, you look at the value in the matrix at the row and column corresponding to the two vertices.
- ▶ **Finding Neighbors:** To find all neighbors of a vertex, you can look at all the columns in a given row. If the entry is 1 (or any positive number in a weighted graph), there is an edge between those vertices.
- ▶ **Graph Traversal:** Techniques like Depth-First Search (DFS) and Breadth-First Search (BFS) can be implemented using the adjacency matrix to visit all the nodes/vertices in a graph.
- ▶ **Calculating the Degree of a Vertex:** The degree of a vertex in an undirected graph can be calculated by summing the values in the corresponding row (or column, since the matrix is symmetric).

# Python Operations

```
# Create a 4x4 adjacency matrix for an undirected graph
adj_matrix = [[0] * 4 for _ in range(n)]

# Function to add edges
def add_edge(v1, v2):
    adj_matrix[v1][v2] = 1
    adj_matrix[v2][v1] = 1 # Because the graph is undirected

# Function to remove edges
def remove_edge(v1, v2):
    adj_matrix[v1][v2] = 0
    adj_matrix[v2][v1] = 0

# Function to check the existence of an edge
def has_edge(v1, v2):
    return adj_matrix[v1][v2] == 1
```



# Python Operations

```
# Adding edges
```

```
add_edge(0, 1)
```

```
add_edge(1, 2)
```

```
add_edge(2, 3)
```

```
add_edge(3, 0)
```

```
# Print adjacency matrix
```

```
print("Adjacency Matrix:")
```

```
for row in adj_matrix:
```

```
    print(row)
```

```
# Check for edge
```

```
print("Edge between 0 and 3:", "Yes" if has_edge(0, 3) else "No")
```

```
# Removing an edge
```

```
remove_edge(1, 2)
```

```
print("Adjacency Matrix after removing an edge:")
```

```
for row in adj_matrix:
```

```
    print(row)
```