# Graph Theory

Representation of Graphs, Graph Isomorphism, Connectivity and so on

# Part 1: Introduction to Graphs

# The Definition of Graph

We begin with the definition of graph. Consider the following definition.

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

V is a non-empty set of vertices and E is a set of edges connecting vertices. I will show visual examples of graph later.

Remark: A graph with an infinite vertex set or an infinite number of edges is called an infinite graph, and in comparison, a graph with a finite vertex set and a finite edge set is called a finite graph.

We will mostly focus on the finite graph.

# An example of graph

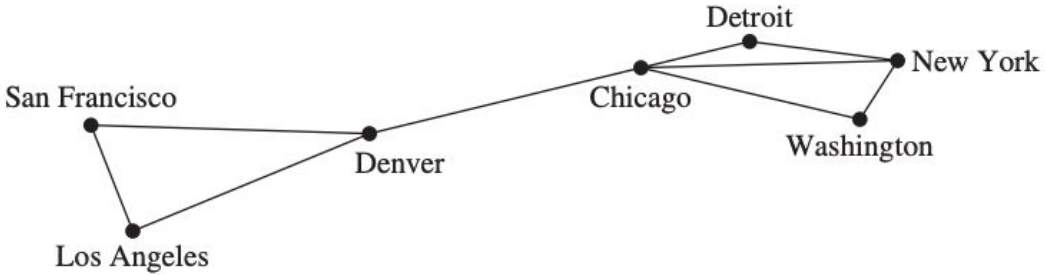



**FIGURE 1** **A computer network.**

In this example, vertices or nodes are cities, and edges are the paths connecting these cities. This graph is also an example of a simple graph.

The definition: A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a simple graph.

# Another Type of Graph: Multigraphs

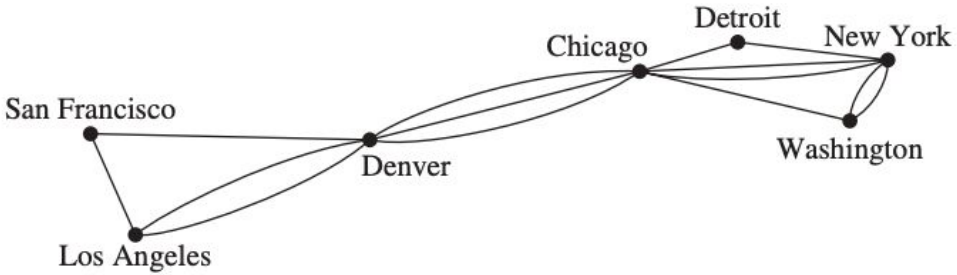



**FIGURE 2** **A computer network with multiple links between data centers.**

Graphs that may have multiple edges connecting the same vertices are called multigraphs. When there are m different edges associated to the same unordered pair of vertices {u, v}, we also say that {u, v} is an edge of multiplicity m.

Useful when there are multiple paths or ways to go from one location to another.

# Loops (or Loop Edges)

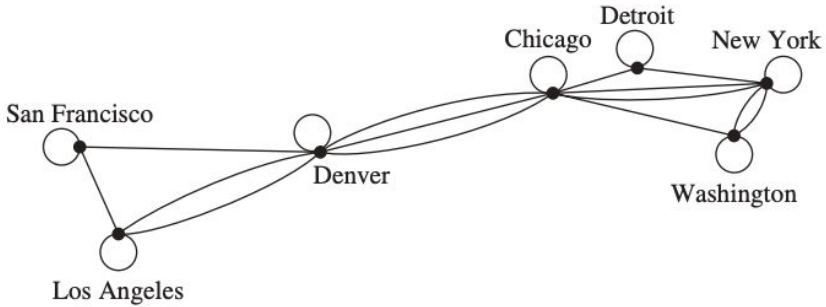



**FIGURE 3** **A computer network with diagnostic links.**

The edges that connect a vertex to itself is called loops. Sometimes we may have more than one loop at a vertex.

Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself, are sometimes called pseudographs.

# Directed and Undirected Graphs

All the examples we have seen so far are undirected graphs. Undirected graph is basically the graph whose edges are not assigned any direction.

We may want to specify directions for some graphs. The graph whose all edges have been assigned directions is called directed graph. The official definition is as follows.

> A *directed graph* (or *digraph*) $(V, E)$ consists of a nonempty set of vertices $V$ and a set of *directed edges* (or *arcs*) $E$. Each directed 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 $v$.

Each directed edge has a start node and end node.

# An Example of Directed Graph

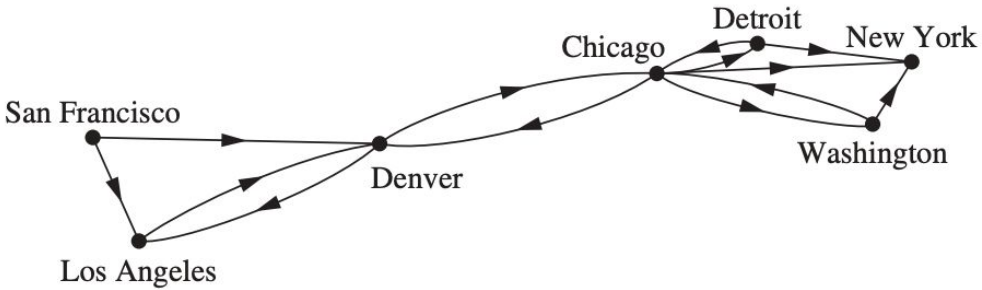



**FIGURE 4** **A communications network with one-way communications links.**

This is an example of directed graph. Each edge is a directed edge, and has start and end vertices.

This graph is also an example of simple directed graphs.

# Directed Multigraphs

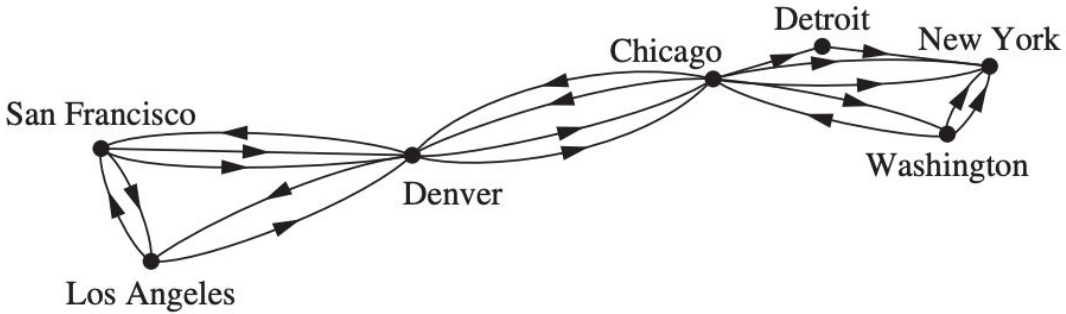



**FIGURE 5** **A computer network with multiple one-way links.**

The graph that has multiple directed edges for the same unordered pair of vertices (u, v) is called directed multigraphs. When there are m directed edges, each associated to an ordered pair of vertices (u, v), we say that (u, v) is an edge of multiplicity m.

# Mixed Graphs

A graph with both directed and undirected edges is called mixed graphs.

# Summary of Basic Types of Graphs

**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 |

Asking yourself the following questions helps you understand graph properties.

- Are the edges of the graph undirected or directed (or both)?
- If the graph is undirected, are multiple edges present that connect the same pair of vertices? If the graph is directed, are multiple directed edges present?
- Are loops present?

# Graph Models - Real World Applications and Examples

From this slide, I will introduce some real world applications and examples of graph. The first example is Social Networks. Notice that vertices or nodes are users or organizations and edges are relationships. Consider the following example.

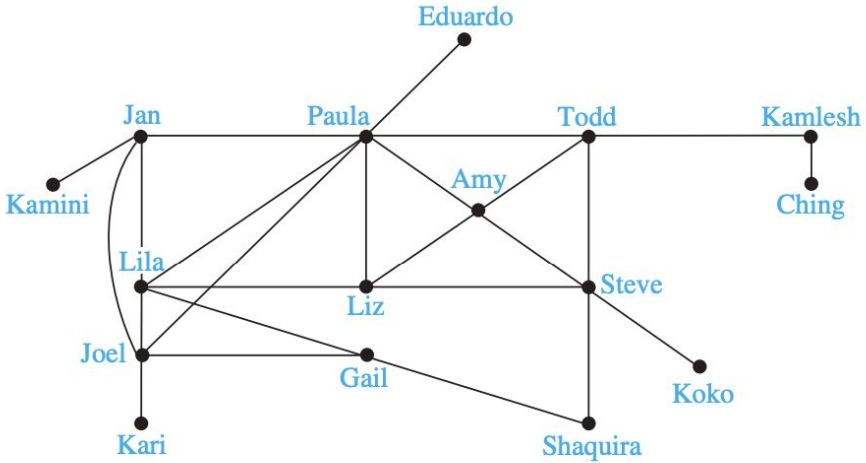



**FIGURE 6** An acquaintanceship graph.

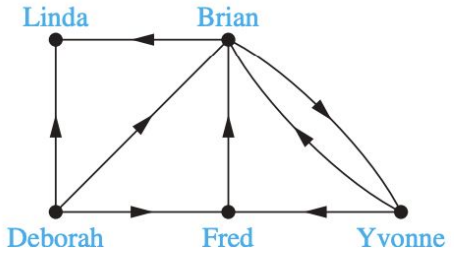



**FIGURE 7** An influence graph.

# More on Social Networks

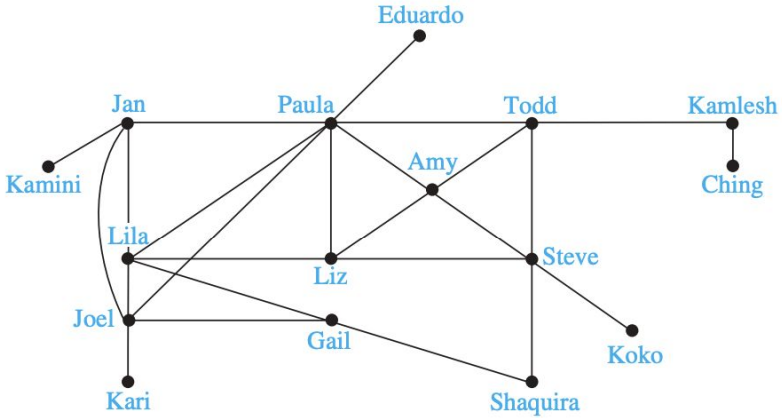



**FIGURE 6** **An acquaintanceship graph.**

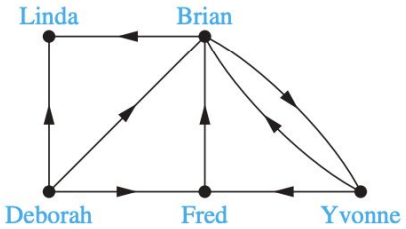



**FIGURE 7** **An influence graph.**

An acquaintanceship graph represents the friendship between users.

An influence graph has directed edges, where there is an edge from user A to user B if user A has the ability to influence user B. We care about influence between users.

# (Telephone) Call Graphs

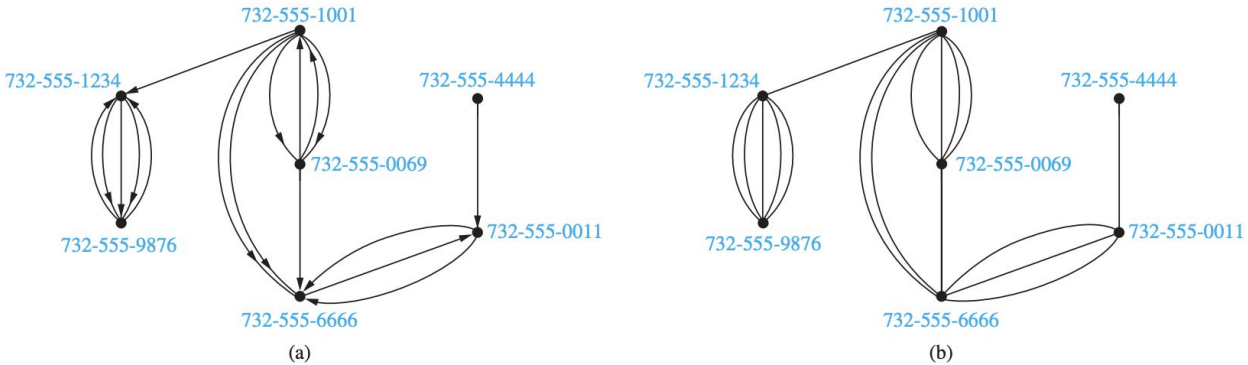



**FIGURE 8** **A call graph.**

Each vertex is a phone number, representing the user, and each edge represents whether the call has been made between users. The directed graph cares about who initiated the call and who got the call. Undirected graph does not care about that, but only care about whether the call has been made between users.

# Web Graph

The vertices are websites and edges are the links between websites.

The graph is directed because the link is basically a jump from one website to another.

For example, if some website uses wikipedia as a reference, then there should be a directed edge from this website to the wikipedia page.

Web Graph changes on a continual basis because many new web links are created every second, or removed.

# Citation Graphs

Vertices are academic papers, and edges are citations.

When the first academic paper cited the second academic paper, then there should be a directed edge from the first paper to the second paper. It is hierarchical in a sense.

Of course, citation graphs are the graphs without multiple edges or loops because no one paper cites itself or cites another paper multiple times.

# Module Dependency Graphs

Context: In software applications, when creating software or system, we separate each important into what is called “modules.” Each module contains important codes that take care of some solid purpose. Some examples of modules are header files, and implementation files. Generally, implementation files depend on (one or multiple) header files.

In this graph, vertices are modules, and edges indicate the dependency between the modules. It is a directed graph.

# An Example of Module Dependency Graphs

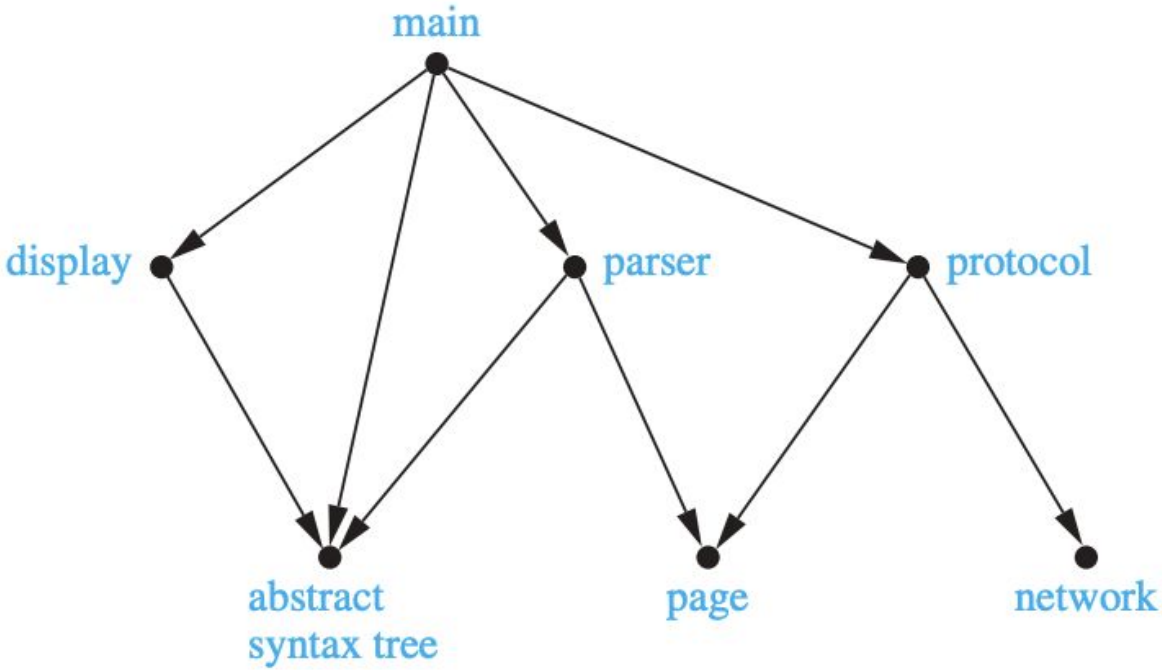



**FIGURE 9** **A module dependency graph.**

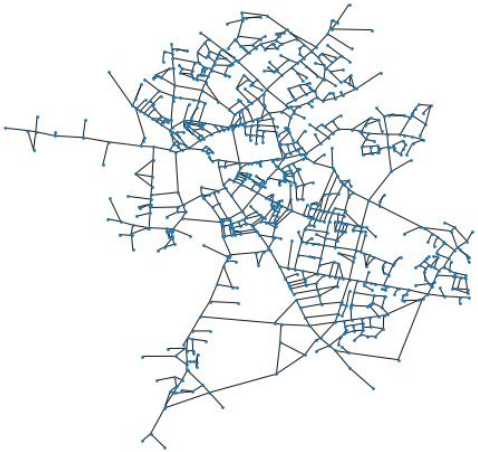
# Airline Routes


Vertices are airports and edges indicate if it is possible to go from one airport to another.

# Road Networks

Cambridge
1521 nodes, 1856 edges

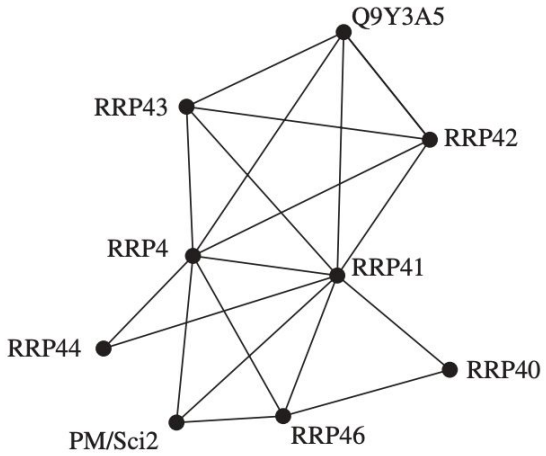
Cambridge (Simplified)
962 nodes, 1281 edges



Vertices are intersections, and edges represent roads.

# Protein Interaction Graphs

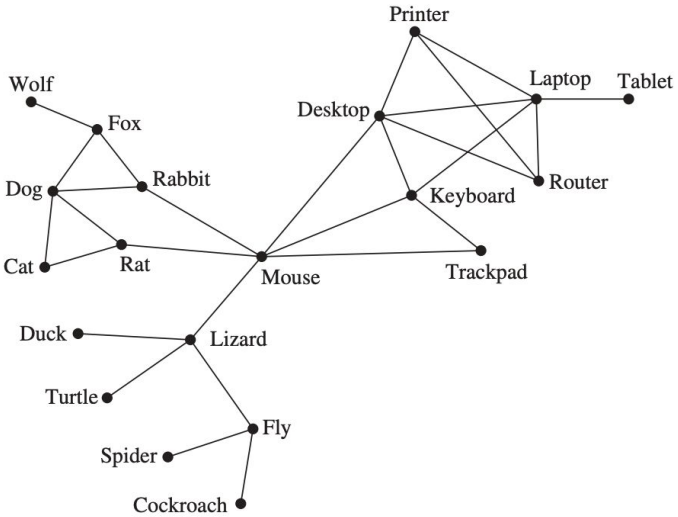
**Protein Interaction Graphs** A protein interaction in a living cell occurs when two or more proteins in that cell bind to perform a biological function. Because protein interactions are crucial for most biological functions, many scientists work on discovering new proteins and understanding interactions between proteins. Protein interactions within a cell can be modeled using a **protein interaction graph** (also called a **protein–protein interaction network**), an undirected graph in which each protein is represented by a vertex, with an edge connecting the vertices representing each pair of proteins that interact. It is a challenging problem to determine genuine




In real world, there are a huge amount of proteins and interactions to consider, so the graph is going to be quite big. Hence, we may want to split the entire graph into smaller graphs to handle the complexity.

# Semantic Networks

In **semantic networks**, vertices are used to represent words, and undirected edges are used to connect vertices when a semantic relation holds between these words. A **semantic relation** is a relationship between two or more words that is based on the meaning of the words. For example, we can build a graph in which vertices represent nouns and two vertices are connected when they have similar meaning. For instance, the names of different countries have similar meaning, as do




This is interesting when one word has multiple meanings (or semantics).

**FIGURE 13** **A semantic network of nouns with similar meaning centered of the word mouse.**

# Tournament Based Graphs

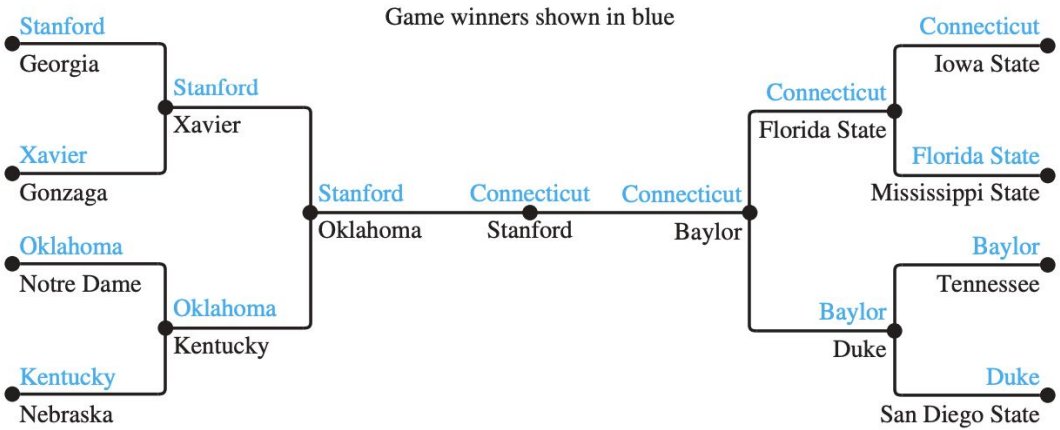



**FIGURE 15** **A single-elimination tournament.**

Vertices are games, and edges are the connections between a previous game and a next game.

# Real World Applications

As we have seen, the graph structure can be applied to various things including biological networks, transportation, software applications, information networks, social networks and so on.

# Part 2: Graph Terminology

# Definition: Adjacent Vertices and Neighbourhood

Two vertices $u$ and $v$ in an undirected graph $G$ are called *adjacent* (or *neighbors*) in $G$ if $u$ and $v$ are endpoints of an edge $e$ of $G$. Such an edge $e$ is called *incident with* the vertices $u$ and $v$ and $e$ is said to *connect* $u$ and $v$.

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

Note that a neighbourhood of some vertex is a set of vertices.

# Degree

The *degree of a vertex in an undirected graph* is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex $v$ is denoted by $\deg(v)$.

The out-degree of a vertex is the number of outgoing edges from this vertex.

The in-degree of a vertex is the number of edges pointing to this vertex.

# Example

What are the degrees and what are the neighborhoods of the vertices in the graphs $G$ and $H$ displayed in Figure 1?

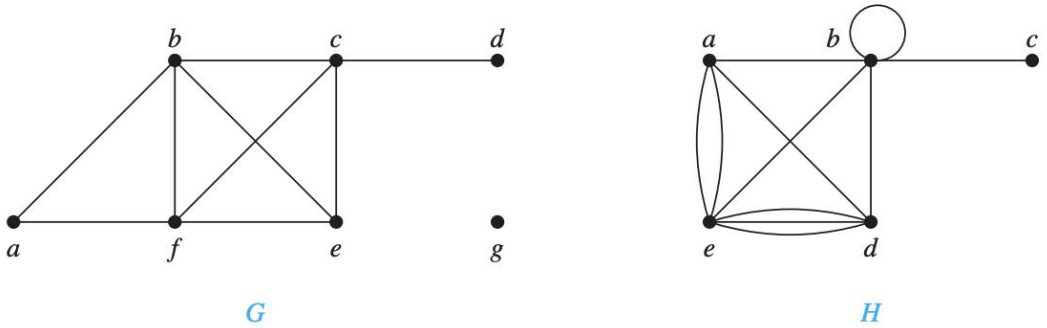



**FIGURE 1** **The undirected graphs $G$ and $H$.**

*Solution:* In $G$, $\deg(a) = 2$, $\deg(b) = \deg(c) = \deg(f) = 4$, $\deg(d) = 1$, $\deg(e) = 3$, and $\deg(g) = 0$. The neighborhoods of these vertices are $N(a) = \{b, f\}$, $N(b) = \{a, c, e, f\}$, $N(c) = \{b, d, e, f\}$, $N(d) = \{c\}$, $N(e) = \{b, c, f\}$, $N(f) = \{a, b, c, e\}$, and $N(g) = \emptyset$. In $H$, $\deg(a) = 4$, $\deg(b) = \deg(e) = 6$, $\deg(c) = 1$, and $\deg(d) = 5$. The neighborhoods of these vertices are $N(a) = \{b, d, e\}$, $N(b) = \{a, b, c, d, e\}$, $N(c) = \{b\}$, $N(d) = \{a, b, e\}$, and $N(e) = \{a, b, d\}$. ◀

# The Handshaking Theorem

**THE HANDSHAKING THEOREM** Let $G = (V, E)$ be an undirected graph with $m$ edges. Then

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

(Note that this applies even if multiple edges and loops are present.)

The degree of a vertex is the number of edges incident with this vertex.

If there is one edge, then two distinct vertices get a new edge incident with them. Hence, the number of overall degrees increase by 2 per edge.

# Application of Handshaking Theorem

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

*Solution:* Because the sum of the degrees of the vertices is $6 \cdot 10 = 60$, it follows that $2m = 60$ where $m$ is the number of edges. Therefore, $m = 30$. ◀

# Some Theorem

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

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

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

Because $\deg(v)$ is even for $v \in V_1$, the first term in the right-hand side of the last equality is even. Furthermore, the sum of the two terms on the right-hand side of the last equality is even, because this sum is $2m$. Hence, the second term in the sum is also even. Because all the terms in this sum are odd, there must be an even number of such terms. Thus, there are an even number of vertices of odd degree. ◁

# Definition: Adjacent Vertices

When $(u, v)$ is an edge of the graph $G$ with directed edges, $u$ is said to be *adjacent to* $v$ and $v$ is said to be *adjacent from* $u$. The vertex $u$ is called the *initial vertex* of $(u, v)$, and $v$ is called the *terminal* or *end vertex* of $(u, v)$. The initial vertex and terminal vertex of a loop are the same.

Because the edges in graphs with directed edges are ordered pairs, the definition of the degree of a vertex can be refined to reflect the number of edges with this vertex as the initial vertex and as the terminal vertex.

Note that edges are ordered pairs in the directed graph, and edges are unordered pairs in the undirected graph.

# Out-Degree and In-Degree

In a graph with directed edges the *in-degree of a vertex* $v$, denoted by $\deg^-(v)$, is the number of edges with $v$ as their terminal vertex. The *out-degree of* $v$, denoted by $\deg^+(v)$, 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 this vertex.)

# Example

Find the in-degree and out-degree of each vertex in the graph $G$ with directed edges shown in Figure 2.

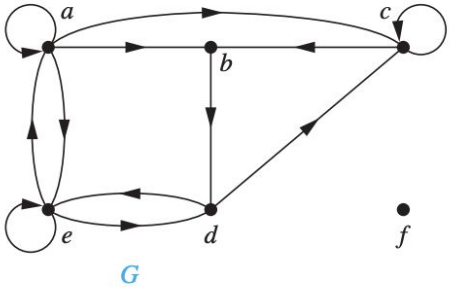



**FIGURE 2** **The directed graph $G$.**

*Solution:* The in-degrees in $G$ are $\deg^-(a) = 2$, $\deg^-(b) = 2$, $\deg^-(c) = 3$, $\deg^-(d) = 2$, $\deg^-(e) = 3$, and $\deg^-(f) = 0$. The out-degrees are $\deg^+(a) = 4$, $\deg^+(b) = 1$, $\deg^+(c) = 2$, $\deg^+(d) = 2$, $\deg^+(e) = 3$, and $\deg^+(f) = 0$.

# A Theorem About the Directed Graph

Because each edge has an initial vertex and a terminal vertex, the sum of the in-degrees and the sum of the out-degrees of all vertices in a graph with directed edges are the same. Both of these sums are the number of edges in the graph. This result is stated as Theorem 3.

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

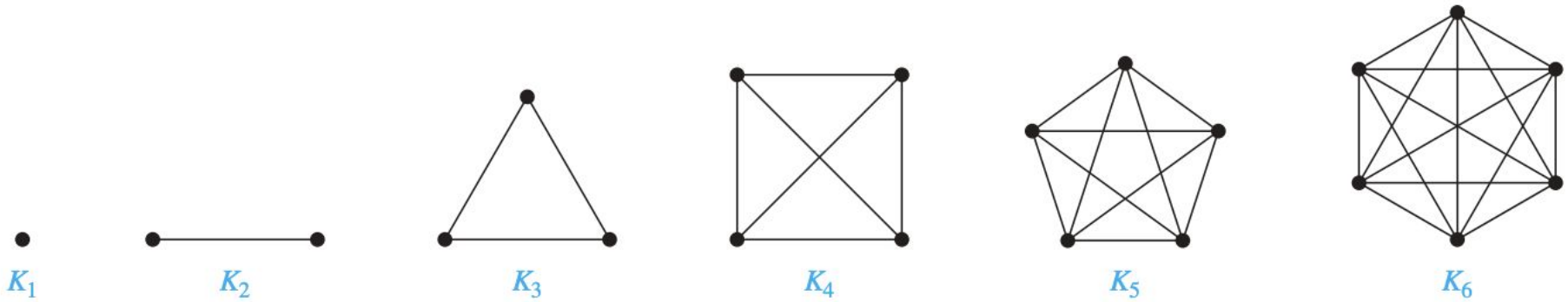
$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |E|.$$

# Special Types of Graphs

There are many special graphs that are given their own names. I will introduce them all in the following slides.

# Complete Graphs

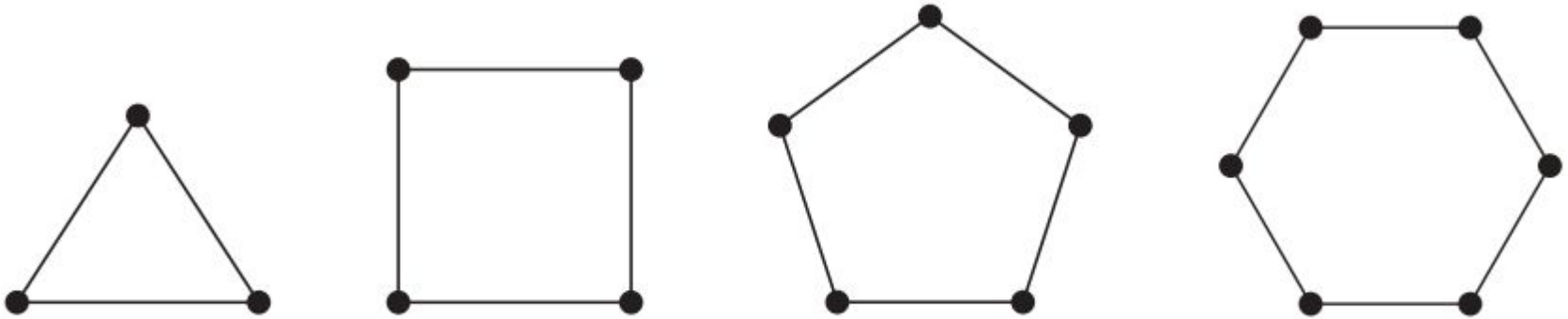
**Complete Graphs** A **complete graph on $n$ vertices**, denoted by $K_n$, is a simple graph that contains exactly one edge between each pair of distinct vertices. The graphs $K_n$, for $n = 1, 2, 3, 4, 5, 6$, are displayed in Figure 3. A simple graph for which there is at least one pair of distinct vertex not connected by an edge is called **noncomplete**.




The complete graph has one edge between any pair of distinct vertices.

# Cycles

**Cycles** A **cycle** $C_n$, $n \geq 3$, consists of $n$ vertices $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, and $\{v_n, v_1\}$. The cycles $C_3$, $C_4$, $C_5$, and $C_6$ are displayed in Figure 4.

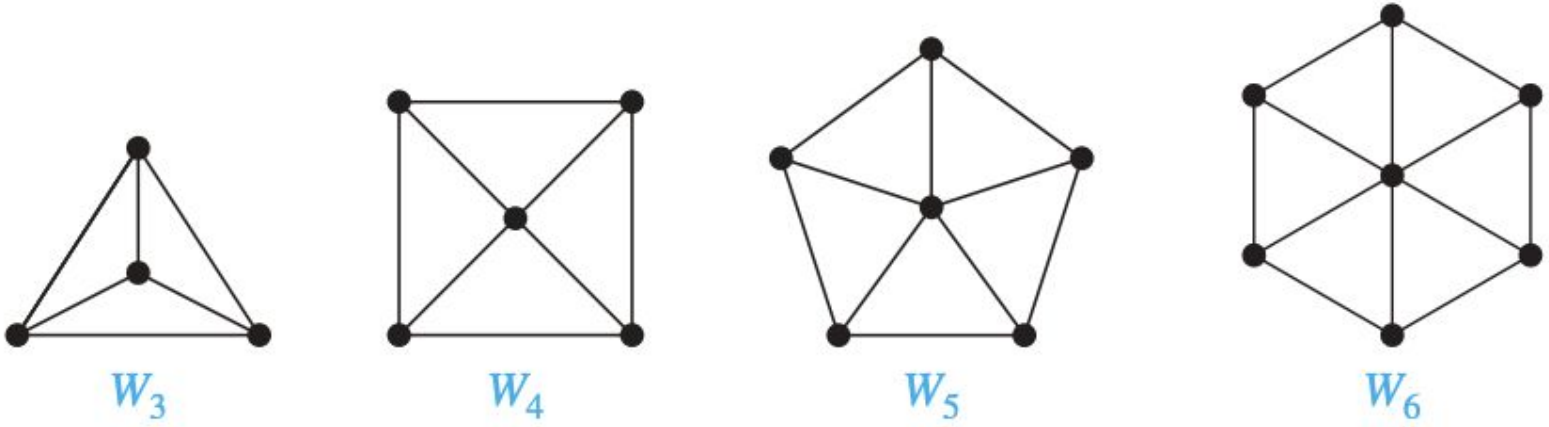



In cycles, there is an edge between the vertices v_i and v_(i+1).

Only defined for n >= 3.

# Wheels

**Wheels** We obtain a **wheel** $W_n$ when we add an additional vertex to a cycle $C_n$, for $n \geq 3$, and connect this new vertex to each of the $n$ vertices in $C_n$, by new edges. The wheels $W_3$, $W_4$, $W_5$, and $W_6$ are displayed in Figure 5.

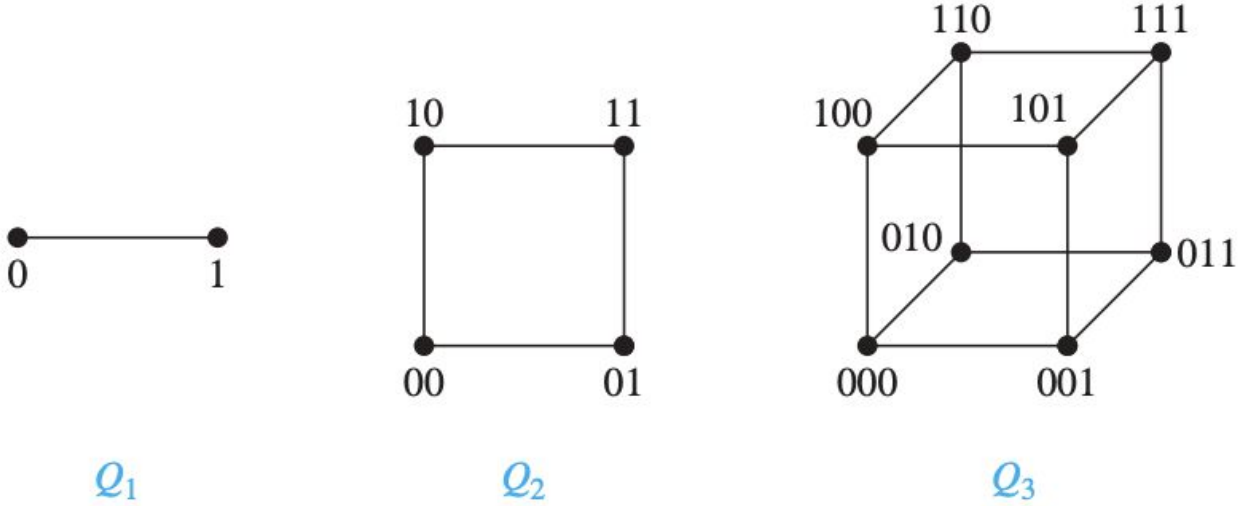



Thus, each wheel has n extra edges and one extra vertex, compared to the corresponding cycles.

Only defined for n >= 3.

# n-Cubes

***n*-Cubes** An ***n*-dimensional hypercube**, or ***n*-cube**, denoted by $Q_n$, is a graph that has vertices representing the $2^n$ bit strings of length $n$. Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position. We display $Q_1$, $Q_2$, and $Q_3$ in Figure 6.



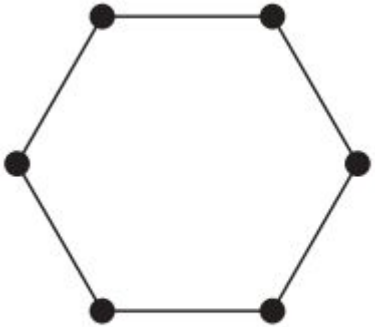

Adjacent vertices differ in exactly one position. Each n-Cube covers all 2^n bit strings. Note that we can construct (n+1) cube by combining two (n) cube.

# Bipartite Graph

A simple graph $G$ is called *bipartite* if its vertex set $V$ can be partitioned into two disjoint sets $V_1$ and $V_2$ such that every edge in the graph connects a vertex in $V_1$ and a vertex in $V_2$ (so that no edge in $G$ connects either two vertices in $V_1$ or two vertices in $V_2$). When this condition holds, we call the pair $(V_1, V_2)$ a *bipartition* of the vertex set $V$ of $G$.

One example is as follows.
Assume that a man and a woman marry together, and not the people of the same gender would marry together. Then, there are two disjoint sets of vertices, men and women. Let the edge represent the marriage condition, every edge in the graph connects a vertex representing a man and a vertex representing a woman.

# Example of the Bipartite Graph



$C_6$

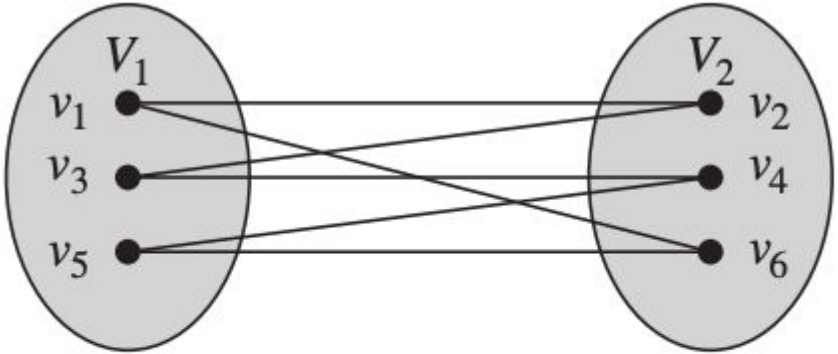
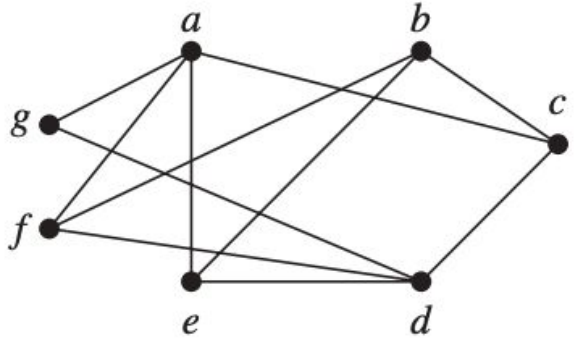



**FIGURE 7** **Showing that $C_6$ is bipartite.**

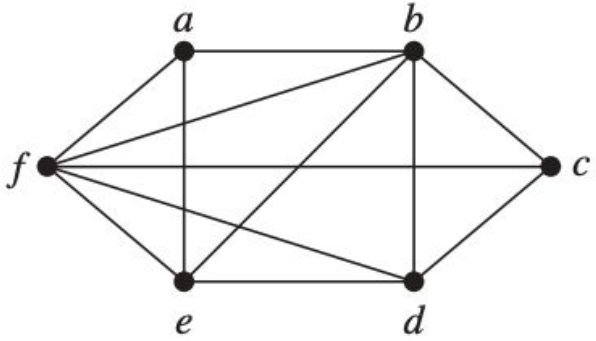
An example of the bipartite graph is C6, the cycle with 6 vertices.

Also any complete graph is not a bipartite graph.

# Are These Graphs Bipartite?




G




H

*Solution:* Graph $G$ is bipartite because its vertex set is the union of two disjoint sets, $\{a, b, d\}$ and $\{c, e, f, g\}$, and each edge connects a vertex in one of these subsets to a vertex in the other subset. (Note that for $G$ to be bipartite it is not necessary that every vertex in $\{a, b, d\}$ be adjacent to every vertex in $\{c, e, f, g\}$. For instance, $b$ and $g$ are not adjacent.)

Graph $H$ is not bipartite because its vertex set cannot be partitioned into two subsets so that edges do not connect two vertices from the same subset. (The reader should verify this by considering the vertices $a$, $b$, and $f$.)

# Theorem about Bipartite Graph

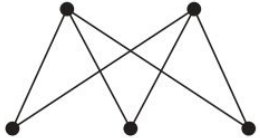
A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color.

***Proof:*** First, suppose that $G = (V, E)$ is a bipartite simple graph. Then $V = V_1 \cup V_2$, where $V_1$ and $V_2$ are disjoint sets and every edge in $E$ connects a vertex in $V_1$ and a vertex in $V_2$. If we assign one color to each vertex in $V_1$ and a second color to each vertex in $V_2$, then no two adjacent vertices are assigned the same color.

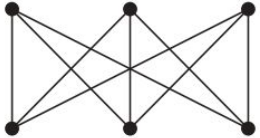
Now suppose that it is possible to assign colors to the vertices of the graph using just two colors so that no two adjacent vertices are assigned the same color. Let $V_1$ be the set of vertices assigned one color and $V_2$ be the set of vertices assigned the other color. Then, $V_1$ and $V_2$ are disjoint and $V = V_1 \cup V_2$. Furthermore, every edge connects a vertex in $V_1$ and a vertex in $V_2$ because no two adjacent vertices are either both in $V_1$ or both in $V_2$. Consequently, $G$ is bipartite. ◁

# Complete Bipartite Graph

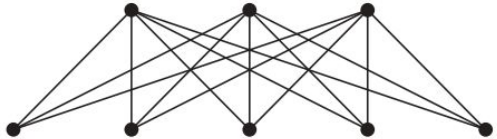
**Complete Bipartite Graphs** A **complete bipartite graph** $K_{m,n}$ is a graph that has its vertex set partitioned into two subsets of $m$ and $n$ vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset. The complete bipartite graphs $K_{2,3}$, $K_{3,3}$, $K_{3,5}$, and $K_{2,6}$ are displayed in Figure 9.



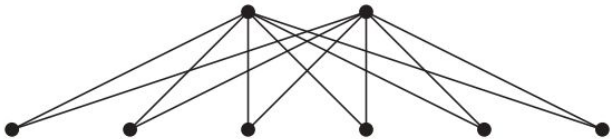
$K_{2,3}$



$K_{3,3}$



$K_{3,5}$

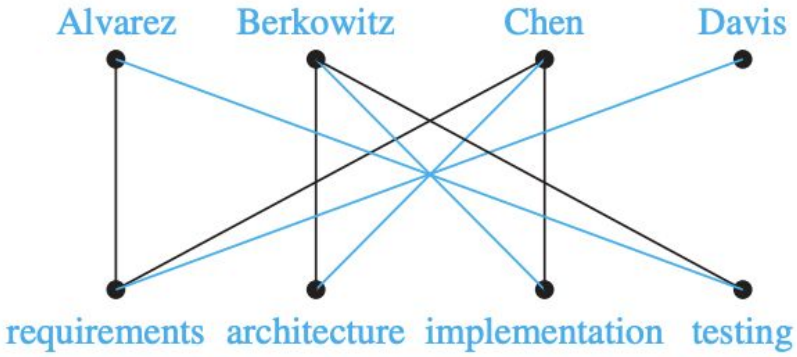


$K_{2,6}$

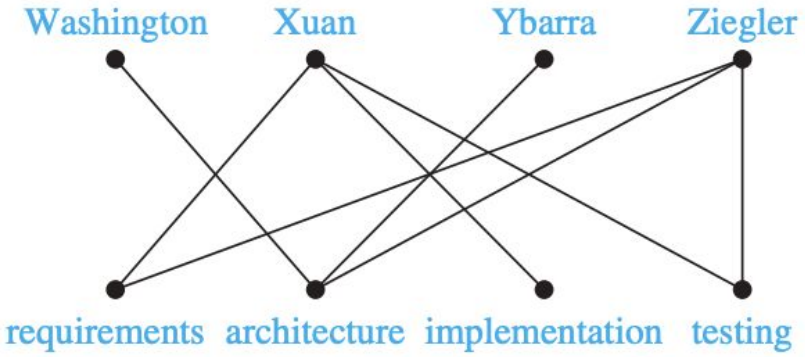
Notice that each complete bipartite graph has (n*m) number of edges.

# Real-World Applications of Bipartite Graph

**Job Assignments** Suppose that there are $m$ employees in a group and $n$ different jobs that need to be done, where $m \geq n$. Each employee is trained to do one or more of these $n$ jobs. We would like to assign an employee to each job. To help with this task, we can use a graph to model employee capabilities. We represent each employee by a vertex and each job by a vertex. For each employee, we include an edge from that employee to all jobs that the employee has been trained to do. Note that the vertex set of this graph can be partitioned into two disjoint sets, the set of employees and the set of jobs, and each edge connects an employee to a job. Consequently, this graph is bipartite, where the bipartition is $(E, J)$ where $E$ is the set of employees and $J$ is the set of jobs. We now consider two different scenarios.




(a)




(b)

# Matching M

Recall the job assignments example.

Finding an assignment of jobs to employees can be thought of as finding a matching in the graph model, where a **matching** $M$ in a simple graph $G = (V, E)$ is a subset of the set $E$ of edges of the graph such that no two edges are incident with the same vertex. In other words, a matching is a subset of edges such that if $\{s, t\}$ and $\{u, v\}$ are distinct edges of the matching, then $s$, $t$, $u$, and $v$ are distinct. A vertex that is the endpoint of an edge of a matching $M$ is said to be **matched** in $M$; otherwise it is said to be **unmatched**. A **maximum matching** is a matching with the largest number of edges. We say that a matching $M$ in a bipartite graph $G = (V, E)$ with bipartition $(V_1, V_2)$ is a **complete matching from** $V_1$ **to** $V_2$ if every vertex in $V_1$ is the endpoint of an edge in the matching, or equivalently, if $|M| = |V_1|$. For example, to assign jobs

In the context of job assignments, a maximum matching would maximize the utilization of capabilities of employees.

# Checking if a complete matching is possible

**HALL'S MARRIAGE THEOREM** The bipartite graph $G = (V, E)$ with bipartition $(V_1, V_2)$ has a complete matching from $V_1$ to $V_2$ if and only if $|N(A)| \geq |A|$ for all subsets $A$ of $V_1$.

***Proof:*** We first prove the *only if* part of the theorem. To do so, suppose that there is a complete matching $M$ from $V_1$ to $V_2$. Then, if $A \subseteq V_1$, for every vertex $v \in A$, there is an edge in $M$ connecting $v$ to a vertex in $V_2$. Consequently, there are at least as many vertices in $V_2$ that are neighbors of vertices in $V_1$ as there are vertices in $V_1$. It follows that $|N(A)| \geq |A|$.

The other direction is quite complicated, and requires induction on the number of the set of vertices V1.

# Subgraph

A *subgraph of a graph* $G = (V, E)$ is a graph $H = (W, F)$, where $W \subseteq V$ and $F \subseteq E$. A subgraph $H$ of $G$ is a *proper subgraph* of $G$ if $H \neq G$.

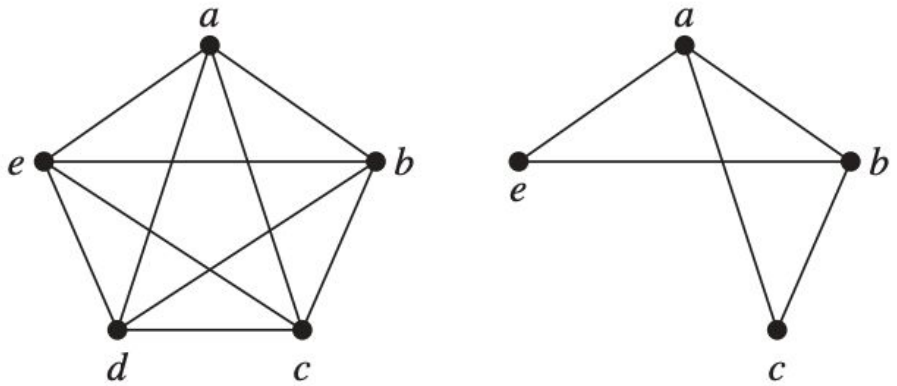



**FIGURE 15** **A subgraph of $K_5$.**

The set of edges and the set of vertices in the subgraph are both subsets of those in the original graph.

# Subgraph induced by a subset of the set of vertices

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 of this edge are in $W$.

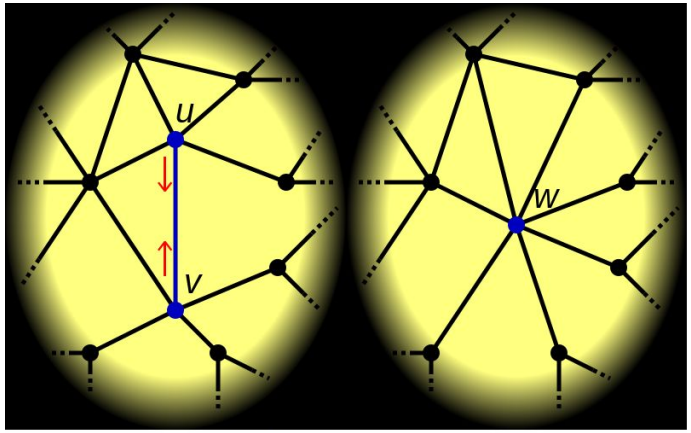
For the graph on the previous page, if we add the edge connecting c and e, we obtain the subgraph induced by W = {a, b, c, e}

# Edge Contractions

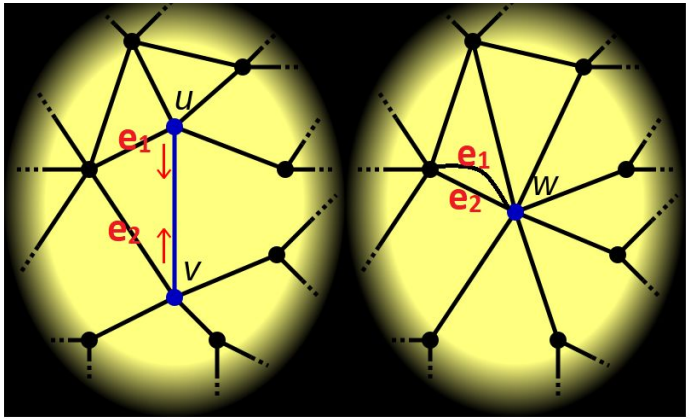
## Formal definition [ edit ]

Let $G = (V, E)$ be a graph (*or directed graph*) containing an edge $e = (u, v)$ with $u \neq v$. Let $f$ be a function that maps every vertex in $V \setminus \{u, v\}$ to itself, and otherwise, maps it to a new vertex $w$. The contraction of $e$ results in a new graph $G' = (V', E')$, where $V' = (V \setminus \{u, v\}) \cup \{w\}$, $E' = E \setminus \{e\}$, and for every $x \in V$, $x' = f(x) \in V'$ is incident to an edge $e' \in E'$ if and only if, the corresponding edge, $e \in E$ is incident to $x$ in $G$.

Contracting an edge without creating multiple edges.



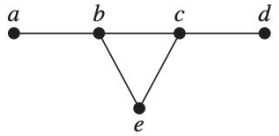

Picture 1



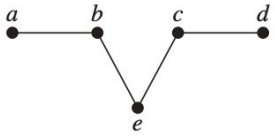

Picture 2

Any edges that were incident to removed vertices are now incident with a new vertex.

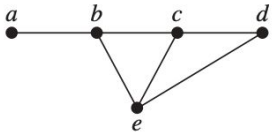
# Visual Illustrations




$G$

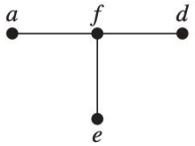



$G - \{b, c\}$

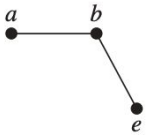



$G + \{e, d\}$

(a) (b)




$G$ contracted by replacing $\{b, c\}$ by $f$




$G - c$

(c) (d)

These are examples of operations on the graph.

# Graph Unions

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 $G_1 \cup G_2$.

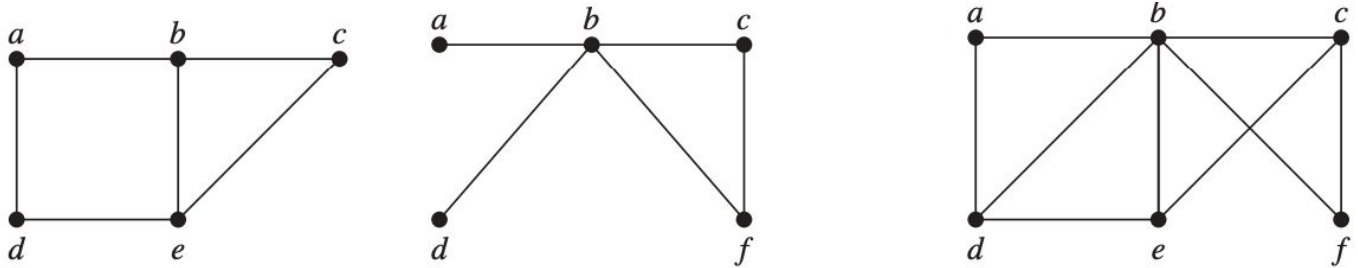



**FIGURE 17** **(a) The simple graphs $G_1$ and $G_2$. (b) Their union $G_1 \cup G_2$.**

# Part 3: Representing Graphs and Graph Isomorphisms

# Adjacency List

Adjacency list is one of the representations of graph. It is useful when coding and using the graph data structure. Adjacency list is an array of vertices, where each entry of array stores the set of adjacent vertices to it.

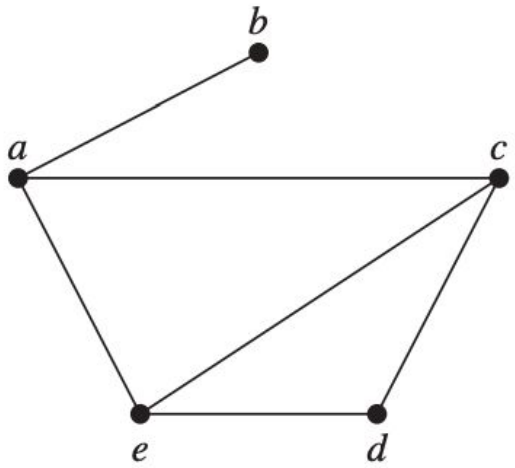



**FIGURE 1** **A simple graph.**

**TABLE 1** **An Adjacency List for a Simple Graph.**

| *Vertex* | *Adjacent Vertices* |
|---|---|
| $a$ | $b, c, e$ |
| $b$ | $a$ |
| $c$ | $a, d, e$ |
| $d$ | $c, e$ |
| $e$ | $a, c, d$ |

# Adjacency List for Directed Graph

For directed graph, we consider the initial vertex and terminal vertex for each edge.

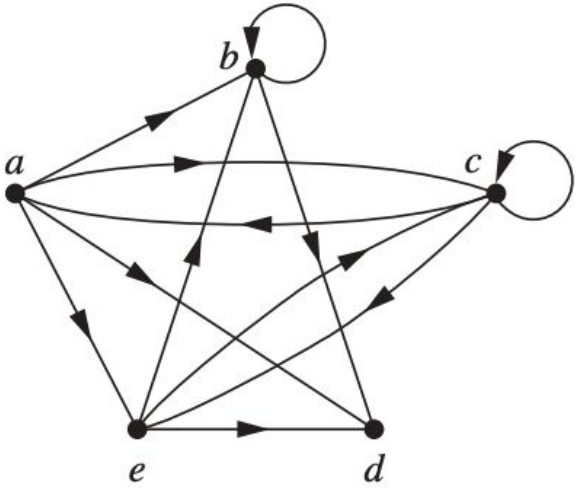



**FIGURE 2** **A directed graph.**

**TABLE 2** **An Adjacency List for a Directed Graph.**

| *Initial Vertex* | *Terminal Vertices* |
|---|---|
| $a$ | $b, c, d, e$ |
| $b$ | $b, d$ |
| $c$ | $a, c, e$ |
| $d$ | |
| $e$ | $b, c, d$ |

# Adjacency Matrix

Adjacency matrix is another representation of graph. For simplicity, suppose each vertex is some non-negative integer, 0, 1, 2, 3….. It is a matrix where each entry is 1 if there is an edge connecting the two vertices, and 0 otherwise.

Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Suppose that the vertices of $G$ are listed arbitrarily as $v_1, v_2, \ldots, v_n$. The **adjacency matrix A** (or $\mathbf{A}_G$) of $G$, with respect to this listing of the vertices, is the $n$ x $n$ zero–one matrix with 1 as its $(i, j)$th entry when $v_i$ and $v_j$ are adjacent, and 0 as its $(i, j)$th entry when they are not adjacent. In other words, if its adjacency matrix is $\mathbf{A} = [a_{ij}]$, then

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

A matrix is usually a 2D-array in the code.

# Examples of Adjacency Matrix

**EXAMPLE 3** Use an adjacency matrix to represent the graph shown in Figure 3.

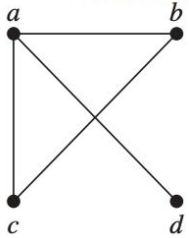



**FIGURE 3**
**A simple graph.**

*Solution:* We order the vertices as $a, b, c, d$. The matrix representing this graph is

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

**EXAMPLE 4** Draw a graph with the adjacency matrix

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

with respect to the ordering of vertices $a, b, c, d$.

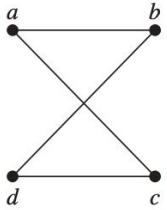



**FIGURE 4**

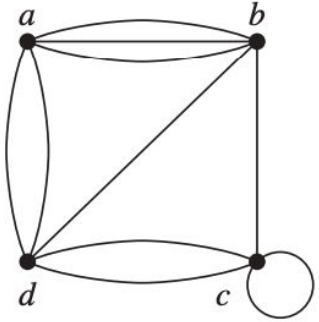
*Solution:* A graph with this adjacency matrix is shown in Figure 4.

Here, see each vertex a, b, c, d as 0, 1, 2, 3 for the sake of zero-indexed matrix.

# More on Adjacency Matrix

If there are multiple edges between the same pair of vertices, then we can store the number of edges connecting the same pair of vertices in the adjacency matrix; consider the following example.

**EXAMPLE 5** Use an adjacency matrix to represent the pseudograph shown in Figure 5.




*Solution:* The adjacency matrix using the ordering of vertices $a, b, c, d$ is

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

Notice that any adjacency matrix is symmetric. The transpose of A = A

# Some Computer Science context

When the graph is sparse, that is, the graph contains relatively few edges, we prefer adjacency list. This is because using an adjacency matrix may result in lots of unused entries in the matrix with zero values, leading to the waste of computer memories.

When the graph is dense, that is, the graph contains a large number of edges, we prefer adjacency matrix. This is because the look-up or access time to adjacency matrix takes O(1) time while the adjacency list may take up to O(deg(v)) time for each vertex v. So, adj matrix improves the time complexity.

# Isomorphism of Graphs

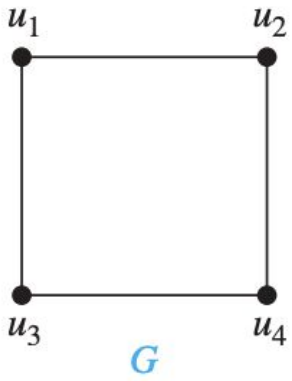
The definition of isomorphism graphs is as follows.

The simple graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are *isomorphic* if there exists 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*.

Since f is injective (one-to-one) and surjective (onto), f is bijective. This implicitly means that V1 and V2 have the same cardinality, and |V1| = |V2|.

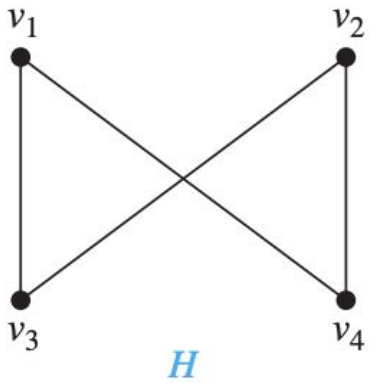
Isomorphism preserves the structure between the two graphs.
Simply put, isomorphic graphs have the same structure.

# Example on Graph Isomorphism




Show that the graphs $G = (V, E)$ and $H = (W, F)$, displayed in Figure 8, 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 $W$. To see that this correspondence preserves adjacency, 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 $u_4$, and 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 $H$.




The idea is to construct an isomorphism between the two graphs that are assumed to be isomorphic to each other.

# Showing Two Graphs Are Not Isomorphic

To see two graphs are not isomorphic, we can find the property that one graph has, but not the other one.

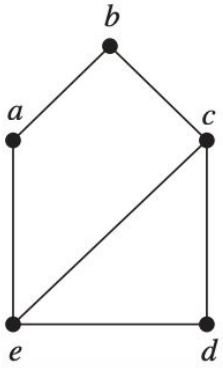
Sometimes it is not hard to show that two graphs are not isomorphic. In particular, we can show that two graphs are not isomorphic if we can find a property only one of the two graphs has, but that is preserved by isomorphism. A property preserved by isomorphism of graphs is called a **graph invariant**. For instance, isomorphic simple graphs must have the same number of vertices, because there is a one-to-one correspondence between the sets of vertices of the graphs.

Isomorphic simple graphs also must have the same number of edges, because the one-to-one correspondence between vertices establishes a one-to-one correspondence between edges. In addition, the degrees of the vertices in isomorphic simple graphs must be the same. That is, a vertex $v$ of degree $d$ in $G$ must correspond to a vertex $f(v)$ of degree $d$ in $H$, because a vertex $w$ in $G$ is adjacent to $v$ if and only if $f(v)$ and $f(w)$ are adjacent in $H$.

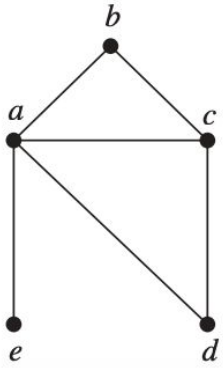
# Examples on Showing Two Graphs Not Isomorphic

Show that the graphs displayed in Figure 9 are not isomorphic.

*Solution:* Both $G$ and $H$ have five vertices and six edges. However, $H$ has a vertex of degree one, namely, $e$, whereas $G$ has no vertices of degree one. It follows that $G$ and $H$ are not isomorphic.




G




H

Isomorphic graphs also need to have the same degree at each vertex as well.

# More examples

Are the following two isomorphic?

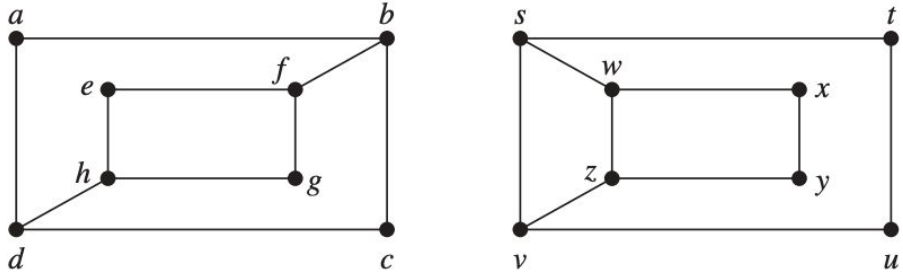



**FIGURE 10** **The graphs $G$ and $H$.**

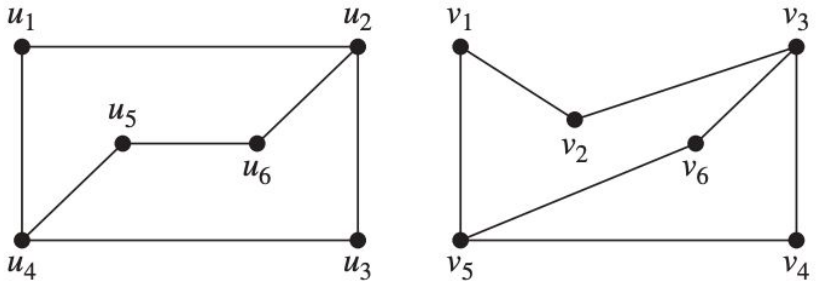
*Solution:* The graphs $G$ and $H$ both have eight vertices and 10 edges. They also both have four vertices of degree two and four of degree three. Because these invariants all agree, it is still conceivable that these graphs are isomorphic.

However, $G$ and $H$ are not isomorphic. To see this, note that because $\deg(a) = 2$ in $G$, $a$ must correspond to either $t$, $u$, $x$, or $y$ in $H$, because these are the vertices of degree two in $H$. However, each of these four vertices in $H$ is adjacent to another vertex of degree two in $H$, which is not true for $a$ in $G$.

Another way to see that $G$ and $H$ are not isomorphic is to note that the subgraphs of $G$ and $H$ made up of vertices of degree three and the edges connecting them must be isomorphic if these two graphs are isomorphic (the reader should verify this). However, these subgraphs, shown in Figure 11, are not isomorphic. ◀

# Constructing an Isomorphism using Adjacency Matrices

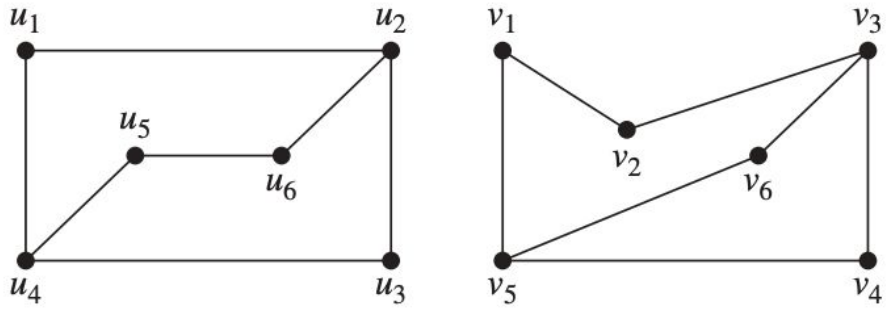
Consider whether the following two graphs are isomorphic to each other, and if so, try to construct an isomorphism.




*G* *H*

*Solution:* Both $G$ and $H$ have six vertices and seven edges. Both have four vertices of degree two and two vertices of degree three. It is also easy to see that the subgraphs of $G$ and $H$ consisting of all vertices of degree two and the edges connecting them are isomorphic (as the reader should verify). Because $G$ and $H$ agree with respect to these invariants, it is reasonable to try to find an isomorphism $f$.

# Constructing an Isomorphism using Adjacency Matrices




$G$ $H$

Notice that the following two matrices are the same. So, preserving edge relations correctly.

$$\mathbf{A}_G = \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{array} \begin{array}{c} \begin{array}{cccccc} u_1 & u_2 & u_3 & u_4 & u_5 & u_6 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array},$$

$$\mathbf{A}_H = \begin{array}{c} \\ v_6 \\ v_3 \\ v_4 \\ v_5 \\ v_1 \\ v_2 \end{array} \begin{array}{c} \begin{array}{cccccc} v_6 & v_3 & v_4 & v_5 & v_1 & v_2 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}.$$

# Constructing an Isomorphism using Adjacency Matrices

Now, we conclude as below.

one correspondence between the vertex set of $G$ and the vertex set of $H$, namely, $f(u_1) = v_6$, $f(u_2) = v_3, f(u_3) = v_4, f(u_4) = v_5, f(u_5) = v_1, f(u_6) = v_2$. To see whether $f$ preserves edges, we examine the adjacency matrix of $G$,

Because $\mathbf{A}_G = \mathbf{A}_H$, it follows that $f$ preserves edges. We conclude that $f$ is an isomorphism, so $G$ and $H$ are isomorphic. Note that if $f$ turned out not to be an isomorphism, we would *not* have established that $G$ and $H$ are not isomorphic, because another correspondence of the vertices in $G$ and $H$ may be an isomorphism.

# Connectivity