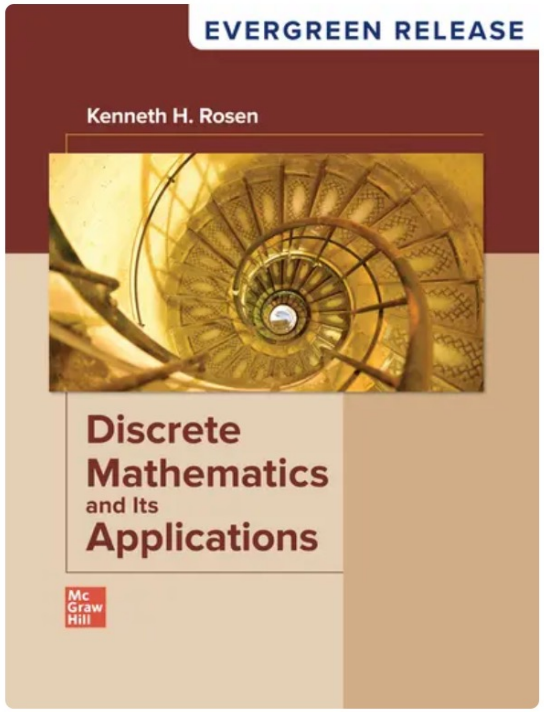
Graphs and Graph Models

Based on Chapter 10
Kenneth Rosen
Discrete Mathematics and Its Applications



ISBN10: 1266045473 | **ISBN13:** 9781266045479

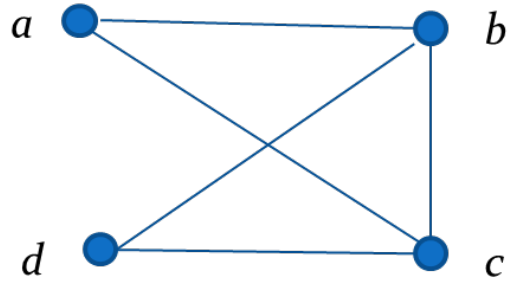
Lecture, 11-09-2025

Graphs

Graphs are mathematical structures that **model pairwise relations between objects.**

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:

- 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 1

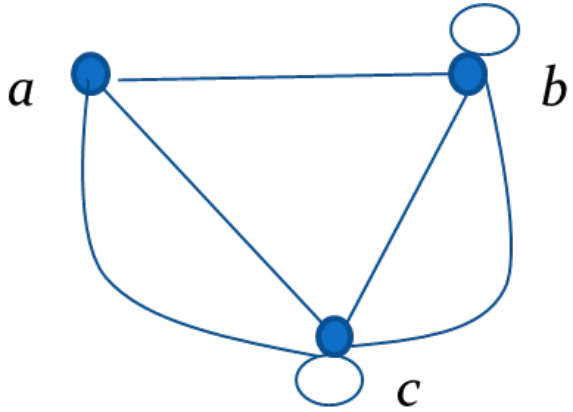
In a ***simple graph*** each edge connects two different vertices and no two edges connect the same pair of vertices.

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[1]

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:

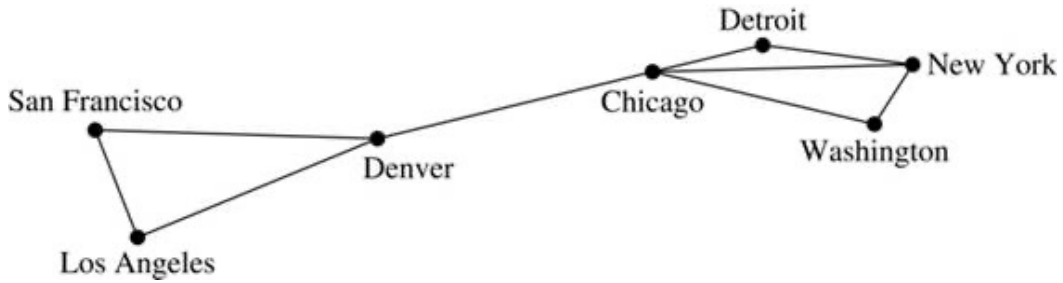
- Graphs where the end points of an edge are not ordered are said to be *undirected graphs*.

Graph Models: Computer Networks[1]

When we build a graph model, we use the appropriate type of graph to capture the important features of the application.

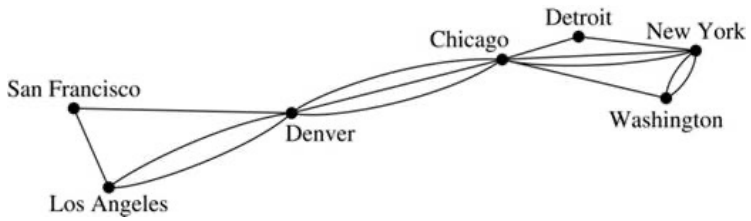
We illustrate this process using graph models of different types of computer networks. In all these graph models, the vertices represent data centers and the edges represent communication links.

To model a computer network where we are only concerned 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.

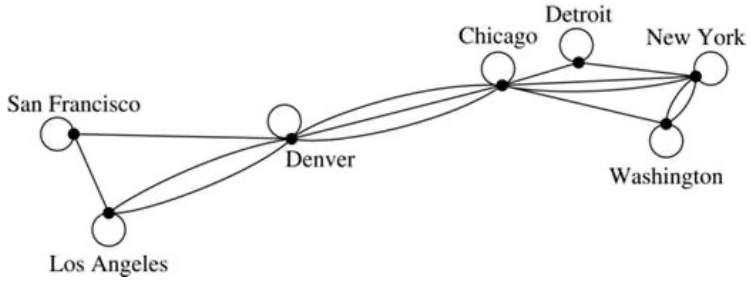


Graph Models: Computer Networks 2

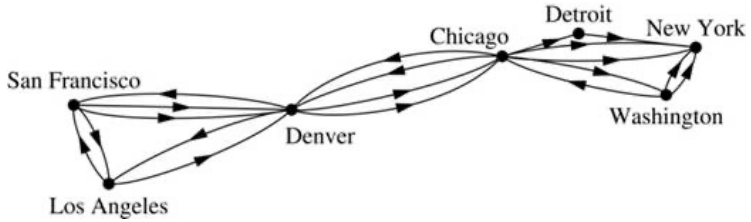
To model a computer network where we care about the number of links between data centers, we use a multigraph.



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



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

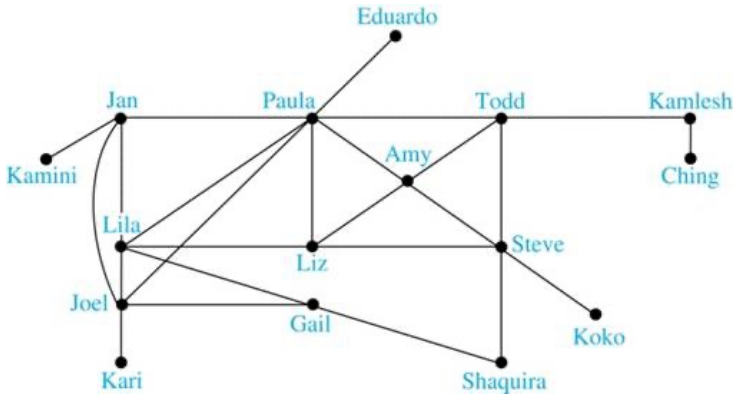
We will illustrate how graph theory can be used in models of:

- Social networks.
- Communications networks.
- Information networks.
- Software design.
- Transportation networks.
- Biological networks.

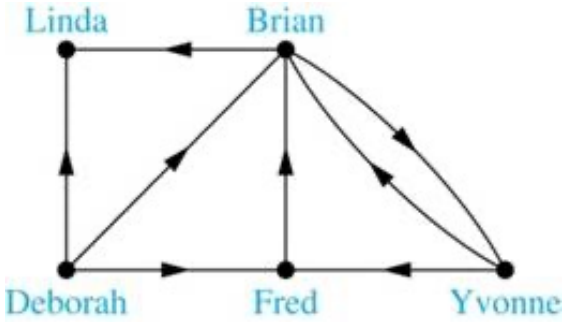
It’s a challenge to find a subject to which graph theory has not yet been applied. Can you find an area without applications of graph theory?

Graph Models: Social Networks 2

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



Example: An influence graph



Examples of Collaboration Graphs

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.
- We will study the academic collaboration graph for mathematicians when we discuss *Erdős numbers – to be explained later*

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[1]

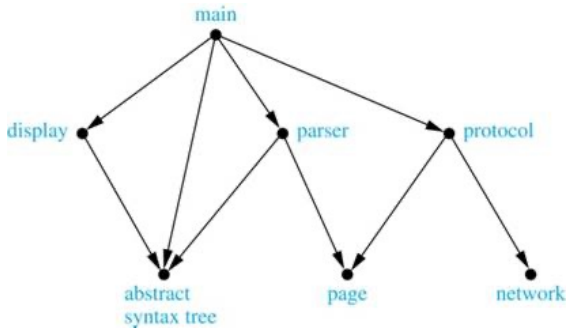
Graph models are extensively used in software design. We will introduce two such models here; one representing the dependency between the modules of a software application and the other representing restrictions in the execution of statements in computer programs.

When a top-down approach is used to design software, the system is divided into modules, each performing a specific task.

We use a *module dependency graph* to represent the dependency between these modules. These dependencies need to be understood before coding can be done.

- 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.

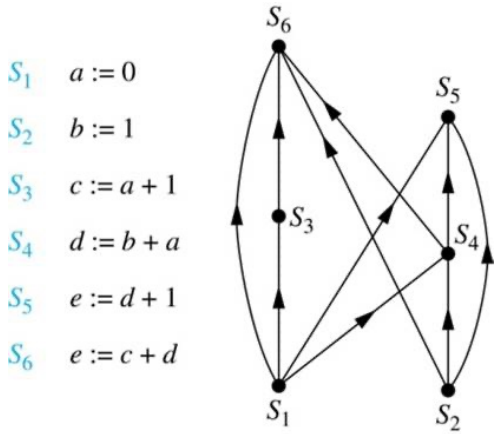


Software Design Applications 2

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 already have been executed before we can execute each of the six statements in the program.



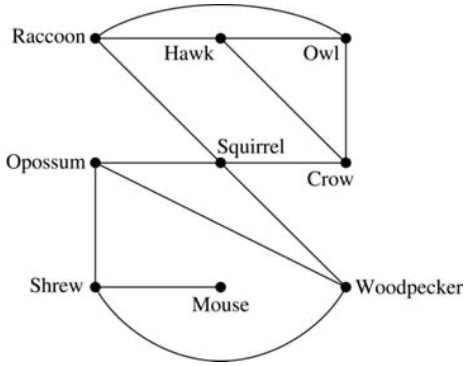
Biological Applications[1]

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 2

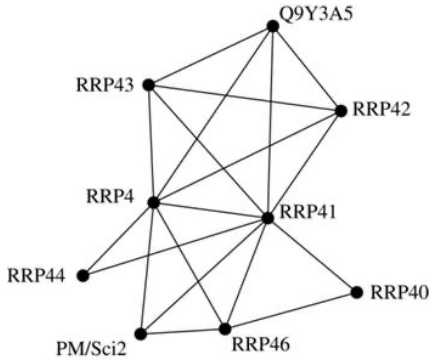
We can model the interaction of proteins in a cell using a *protein interaction network.*

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 Terminology and Special Types of Graphs

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

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,

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 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?

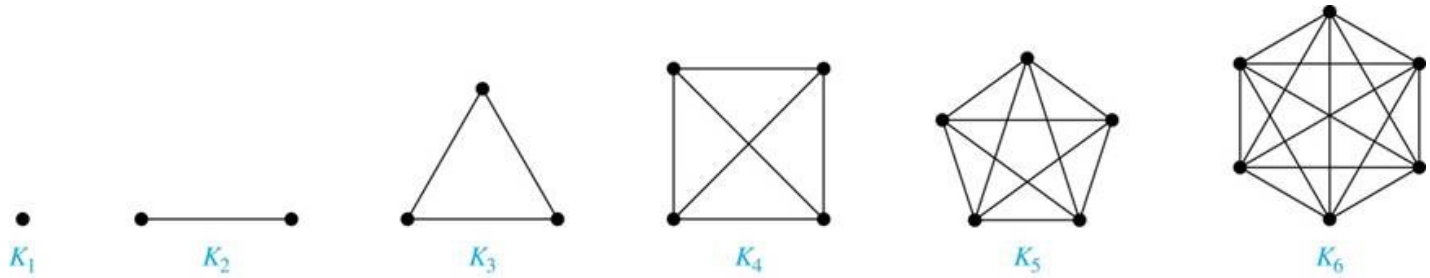
Solution: Because the sum of the degrees of the vertices is $6 \cdot 10 = 60$, the handshaking theorem tells us that $2m = 60$. So the number of edges $m = 30$.

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.

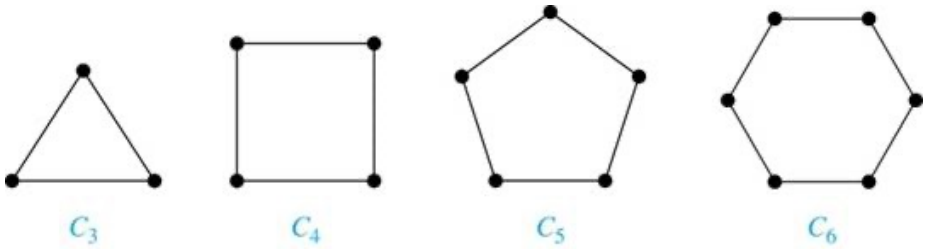
Special Types of Simple Graphs: Complete Graphs

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.

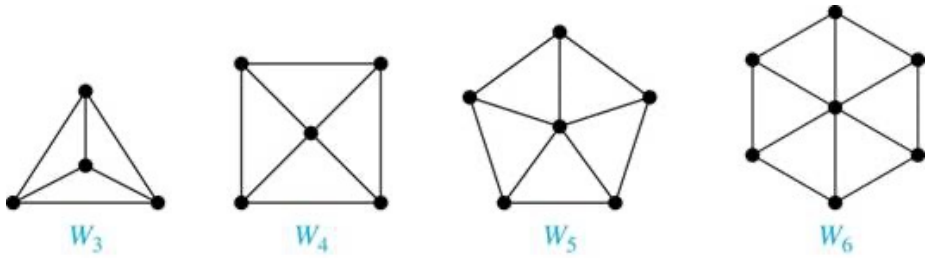


Special Types of Simple Graphs: Cycles and Wheels

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

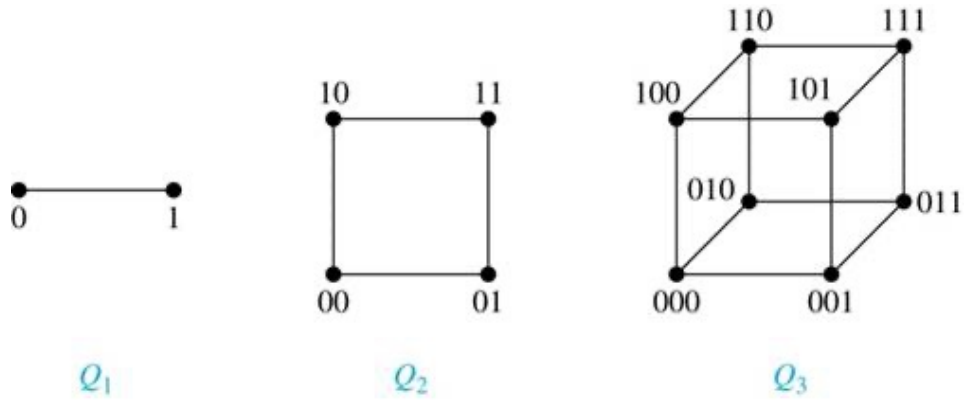


A *wheel* W_n is obtained by adding an additional vertex to a cycle C_n for $n \geq 3$ and connecting this new vertex to each of the n vertices in C_n by new edges.



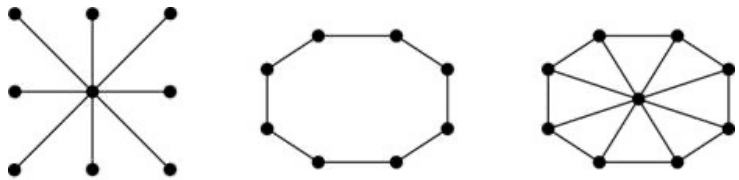
Special Types of Simple Graphs: *n*-Cubes

An *n-dimensional hypercube*, or *n-cube,* $\boldsymbol{Q_n}$, 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.



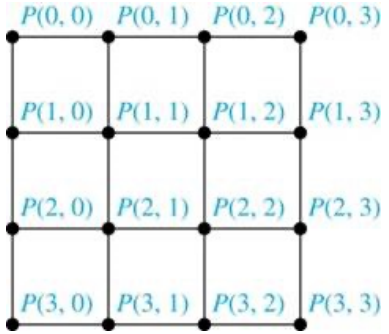
Some local area networks use a *star topology*, which is a complete bipartite graph $K_{1,n}$, as shown in (a). All devices are connected to a central control device.

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.

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,* $\boldsymbol{Q_n}$, 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.

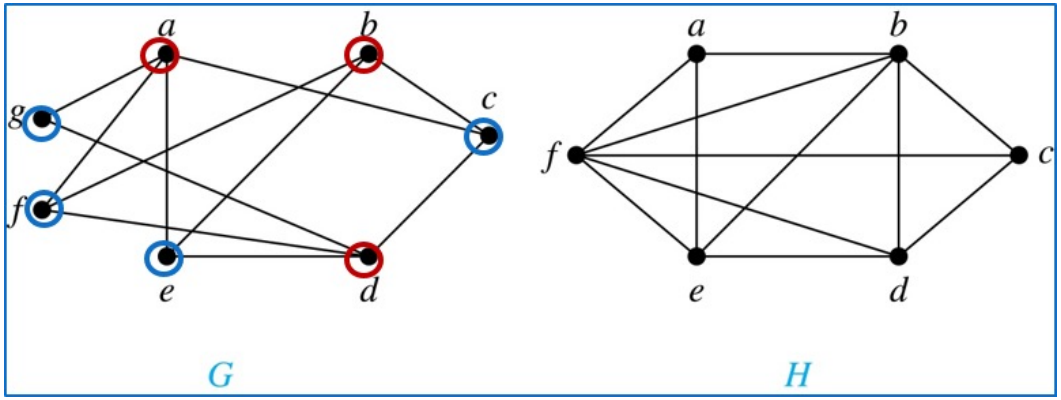


Bipartite Graphs[1]

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.

It is not hard to show that an equivalent definition of a bipartite graph is a graph where it is possible to color the vertices 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.

Representing 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:

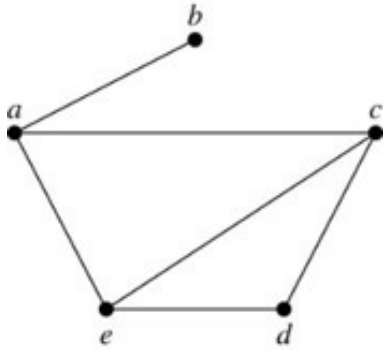


TABLE 1 An Adjacency List for a Simple Graph.

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

Example:

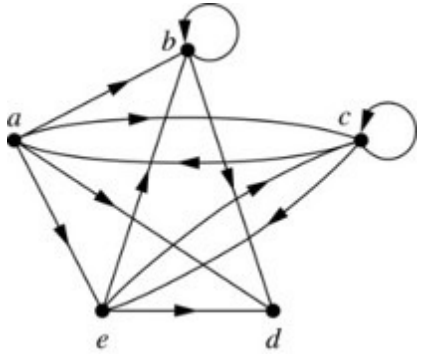


TABLE 2 An Adjacency List for a Directed Graph.

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 1

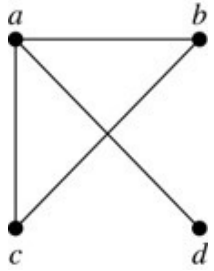
Definition: Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Arbitrarily list the vertices of G as $v_1, v_2, \ldots, v_n$. The *adjacency matrix* $\mathbf{A}_G$ of G, with respect to the listing of vertices, is the $n \times n$ zero-one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent.

- In other words, if the graphs adjacency matrix is $\mathbf{A}_G = [a_{ij}]$, then

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

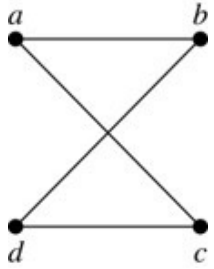
Representation of Graphs: Adjacency Matrices 2

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.

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. 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_{ii} for $i = 1, 2, 3, \ldots, n$, is 0.

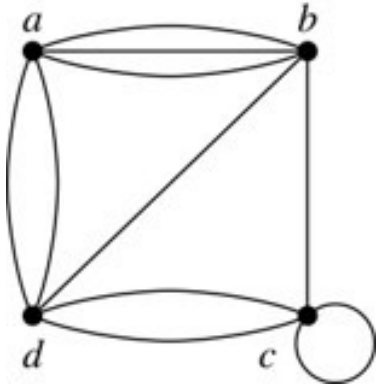
Representation of Graphs: Adjacency Matrices 3

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 1 at the (i, j)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.



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

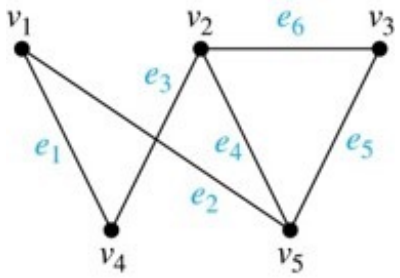
Representation of Graphs: Incidence Matrices 1

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

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

Representation of Graphs: Incidence Matrices 2

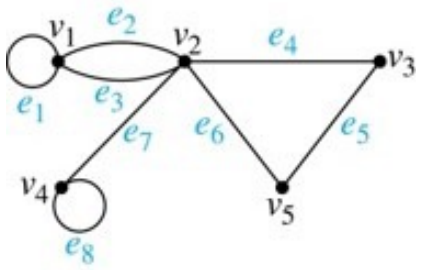
Example: Simple Graph and Incidence Matrix



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

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.

Connectivity

Paths 1

Informal Definition: A *path* is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these.

Applications: Numerous problems can be modeled with paths formed by traveling along edges of graphs such as:

- determining whether a message can be sent between two computers.
- efficiently planning routes for mail delivery.

Paths 2

Definition: Let n be a nonnegative integer and G an undirected graph. A *path* of *length n* from u to v in G is a sequence of n edges $e_1, \ldots, e_n$ of G for which there exists a sequence $x_0 = u, x_1, \ldots, x_{n-1}, x_n = v$ of vertices such that e_i has, for $i = 1, \ldots, n$, the endpoints x_{i-1} and x_i.

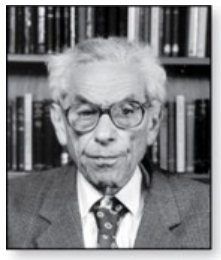
- When the graph is simple, we denote this path by its vertex sequence $x_0, x_1, \ldots, x_n$(since listing the vertices uniquely determines the path).
- The path is a *circuit* if it begins and ends at the same vertex ($u = v$) and has length greater than zero.
- The path or circuit is said to *pass through* the vertices $x_1, x_2, \ldots, x_{n-1}$ and *traverse* the edges $e_1, \ldots, e_n$.
- A path or circuit is *simple* if it does not contain the same edge more than once.

This terminology is readily extended to directed graphs.

Definitions and Representation

- An undirected graph is *connected* if every pair of vertices is connected by a path.
- A *forest* is an acyclic graph, and a *tree* is a connected acyclic graph.
- A *spanning tree* of an undirected graph *G* is a subgraph of *G* that is a tree containing all the vertices of *G*.
- A graph that has weights associated with each edge is called a *weighted graph*.

Erdős numbers



Paul Erdős

Example: *Erdős numbers*.
In a collaboration graph, two people *a* and *b* are connected by a path when there is a sequence of people starting with *a* and ending with *b* such that the endpoints of each edge in the path are people who have collaborated.

- In the academic collaboration graph of people who have written papers in mathematics, the *Erdős number* of a person *m* is the length of the shortest path between *m* and the prolific mathematician Paul Erdős.
- To learn more about Erdős numbers, visit http://www.ams.org/mathscinet/collaborationDistance.html

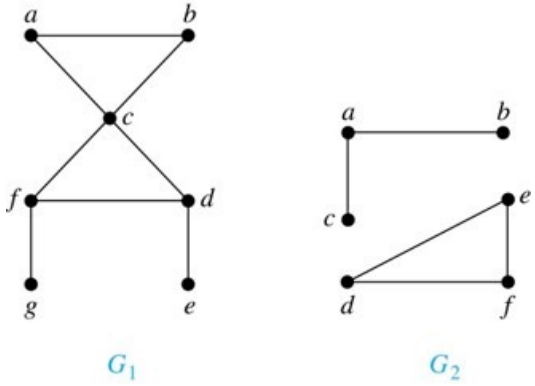
TABLE 1 The Number of Mathematicians with a Given Erdős Number (as of early 2006).

Erdős Number	*Number of People*
0	1
1	504
2	6,593
3	33,605
4	83,642
5	87,760
6	40,014
7	11,591
8	3,146
9	819
10	244
11	68
12	23
13	5

Connectedness in Undirected Graphs

Definition: An undirected graph is called *connected* if there is a path between every pair of vertices. An undirected graph that is not *connected* is called *disconnected*. We say that we *disconnect* a graph when we remove vertices or edges, or both, to produce a disconnected subgraph.

Example: G_1 is connected because there is a path between any pair of its vertices, as can be easily seen. However G_2 is not connected because there is no path between vertices *a* and *f*, for example.



The Connected Components of the Web Graph

Recall that at any particular instant the web graph provides a snapshot of the web, where vertices represent web pages and edges represent links. According to a 1999 study, the Web graph at that time had over 200 million vertices and over 1.5 billion edges. (The numbers today are several orders of magnitude larger.)

The underlying undirected graph of this Web graph has a connected component that includes approximately 90% of the vertices.

There is a *giant strongly connected component (GSCC)* consisting of more than 53 million vertices. A Web page in this component can be reached by following links starting in any other page of the component. There are three other categories of pages with each having about 44 million vertices:

- pages that can be reached from a page in the GSCC, but do not link back.
- pages that link back to the GSCC, but can not be reached by following links from pages in the GSCC.
- pages that cannot reach pages in the GSCC and can not be reached from pages in the GSCC.

Euler and Hamiltonian Graphs

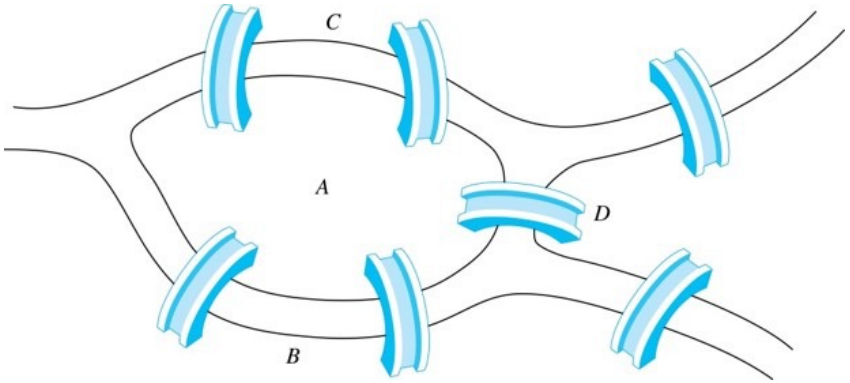
Euler Paths and Circuits[1]

Leonard Euler
(1707-1783)

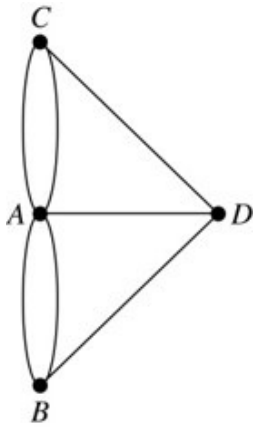
The town of Królewiec (Königsberg in Prussia, now Kalingrad, Russia) was divided into four sections by the branches of the Pregel river. In the 18th century seven bridges connected these regions.

People wondered whether it was possible to follow a path that crosses each bridge exactly once and returns to the starting point.

The Swiss mathematician Leonard Euler proved that no such path exists. This result is often considered to be the first theorem ever proved in graph theory.



The 7 Bridges of Königsberg

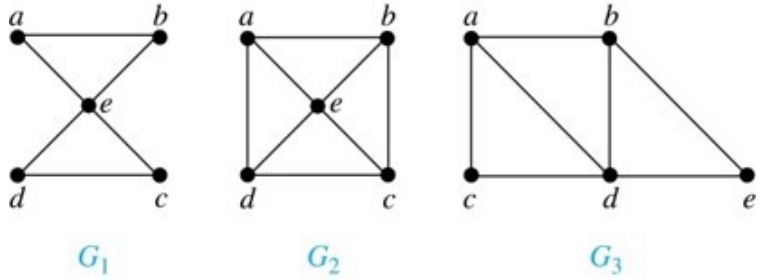


Multigraph Model of the Bridges of Königsberg

Euler Paths and Circuits 2

Definition: An *Euler circuit* in a graph *G* is a simple circuit containing every edge of *G*. An *Euler path* in *G* is a simple path containing every edge of *G*.

Example: Which of the undirected graphs G_1, G_2, and G_3 has a Euler circuit? Of those that do not, which has an Euler path?

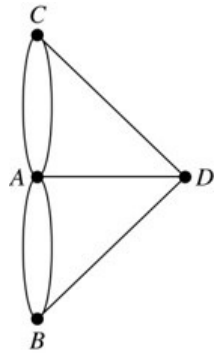


Solution: The graph G_1 has an Euler circuit (e.g., *a*, *e*, *c*, *d*, *e*, *b*, *a*). But, as can easily be verified by inspection, neither G_2 nor G_3 has an Euler circuit. Note that G_3 has an Euler path (e.g., *a*, *c*, *d*, *e*, *b*, *d*, *a*, *b*), but there is no Euler path in G_2, which can be verified by inspection.

Algorithm for Constructing an Euler Circuits 2

Theorem: A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has an even degree and it has an Euler path if and only if it has exactly two vertices of odd degree.

Example: Two of the vertices in the multigraph model of the Königsberg bridge problem have odd degree. Hence, there is no Euler circuit in this multigraph and it is impossible to start at a given point, cross each bridge exactly once, and return to the starting point.



Applications of Euler Paths and Circuits

Euler paths and circuits can be used to solve many practical problems such as finding a path or circuit that traverses each

- street in a neighborhood,
- road in a transportation network,
- connection in a utility grid,
- link in a communications network.

Other applications are found in the

- layout of circuits,
- network multicasting,
- molecular biology, where Euler paths are used in the sequencing of DNA.

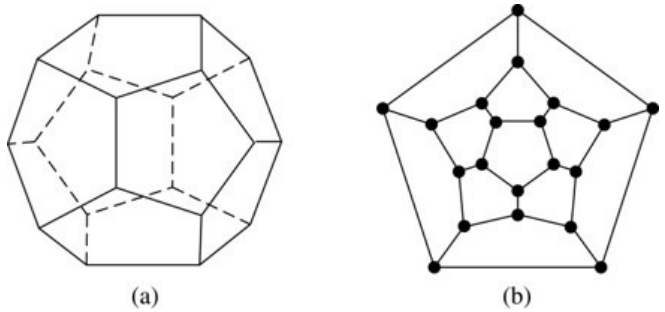
Hamilton Paths and Circuits[1]

William Rowan Hamilton (1805- 1865)

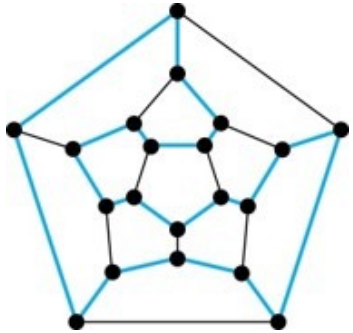
Euler paths and circuits contained every edge only once. Now we look at paths and circuits that contain every vertex exactly once.

William Hamilton invented the *Icosian puzzle* in 1857.
It consisted of a wooden dodecahedron (with 12 regular pentagons as faces), illustrated in (a), with a peg at each vertex, labeled with the names of different cities. String was used to plot a circuit visiting 20 cities exactly once

The graph form of the puzzle is given in (b).



The solution (a Hamilton circuit) is given here.



Hamilton Paths and Circuits 2

Definition: A simple path in a graph *G* that passes through every vertex exactly once is called a *Hamilton path*, and a simple circuit in a graph *G* that passes through every vertex exactly once is called a *Hamilton circuit.*

That is, a simple path $x_0, x_1, \ldots, x_{n-1}, x_n$ in the graph $G = (V, E)$ is called a Hamilton path if $V = \{x_0, x_1, \ldots, x_{n-1}, x_n\}$ and $x_i \neq x_j$ for $0 \leq i < j \leq n$, and the simple circuit $x_0, x_1, \ldots, x_{n-1}, x_n, x_0$ (with $n > 0$) is a Hamilton circuit if $x_0, x_1, \ldots, x_{n-1}, x_n$ is a Hamilton path.

Necessary Conditions for Hamilton Circuits

Gabriel Andrew Dirac
(1925-1984)

Unlike for an Euler circuit, no simple necessary and sufficient conditions are known for the existence of a Hamilton circuit.

However, there are some useful necessary conditions. We describe two of these now.

Dirac's Theorem: If G is a simple graph with $n \geq 3$ vertices such that the degree of every vertex in G is $\geq n/2$, then G has a Hamilton circuit.

Ore's Theorem: If G is a simple graph with $n \geq 3$ vertices such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices, then G has a Hamilton circuit.

Øysten Ore
(1899-1968)

Applications of Hamilton Paths and Circuits

Applications that ask for a path or a circuit that visits each intersection of a city, each place pipelines intersect in a utility grid, or each node in a communications network exactly once, can be solved by finding a Hamilton path in the appropriate graph.

The famous ***traveling salesperson problem*** **(*TSP*)** asks for the shortest route a traveling salesperson should take to visit a set of cities. This problem reduces to finding a Hamilton circuit such that the total sum of the weights of its edges is as small as possible.

A family of binary codes, known as ***Gray codes***, which minimize the effect of transmission errors, correspond to Hamilton circuits in the n-cube Q_n.