National University
of computer and emerging sciences

DISCRETE STRUCTURES

COURSE INSTRUCTOR: MUHAMMAD SAIF UL ISLAM

Course Outline

- **Logic and Proofs** (Chapter 1)
- **Sets and Functions** (Chapter 2)
- **Relations** (Chapter 9)
- **Number Theory** (Chapter 4)
- Combinatorics and Recurrence
- **Graphs** (Chapter 10)
- Trees
- Discrete Probability

Lecture Outline

- Graphs and Graph Models
- Graph Terminology and Special Types of Graphs
- Representing Graphs and Graph Isomorphism
- Connectivity
- Euler and Hamiltonian Graphs
- Shortest-Path Problems
- Planar Graphs
- Graph Coloring

Representing Graphs and Graph Isomorphism

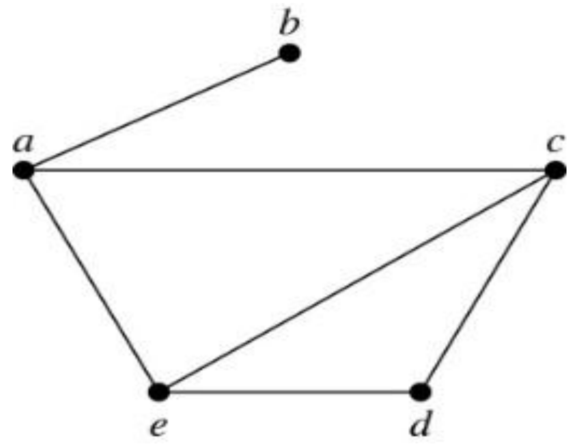
Contents:

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

Representing Graphs: Adjacency Lists

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

Example:



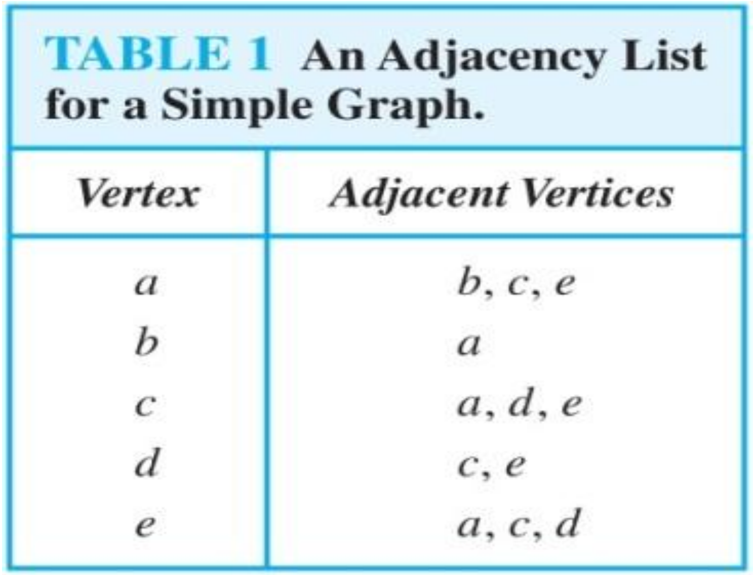
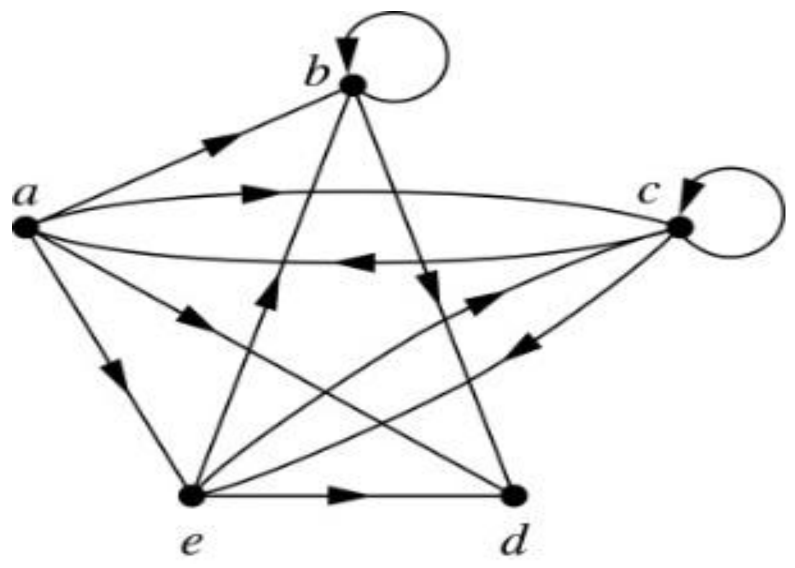


TABLE 1 An Adjacency List for a Simple Graph.

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

Representing Graphs: Adjacency Lists

Example:



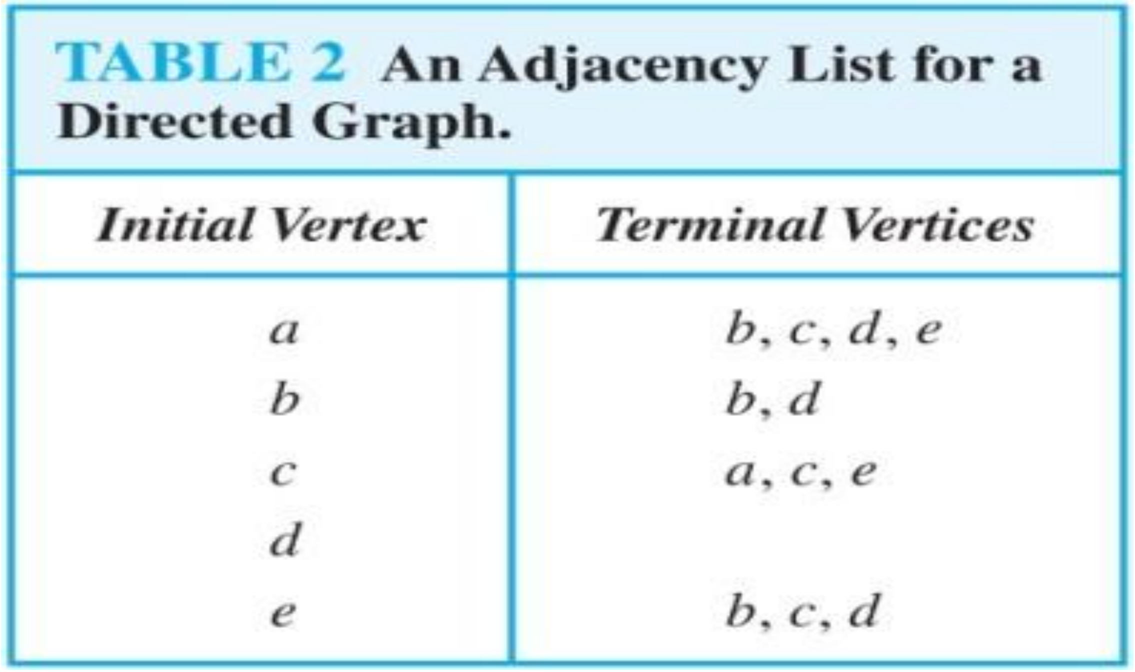


TABLE 2 **An Adjacency List for a Directed Graph.**

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

Representation of Graphs: Adjacency Matrices

Definition:

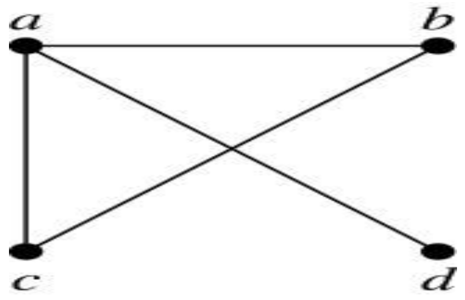
Suppose that $G = (V, E)$ is a simple graph where $|V| = n$. Arbitrarily list the vertices of G as $v_1, v_2, \ldots, v_n$. The *adjacency matrix* $\mathbf{A}_G$ of G, with respect to the listing of vertices, is the $n \times n$ zero-one matrix with 1 as its (i, j)th entry when v_i and v_j are adjacent, 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

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

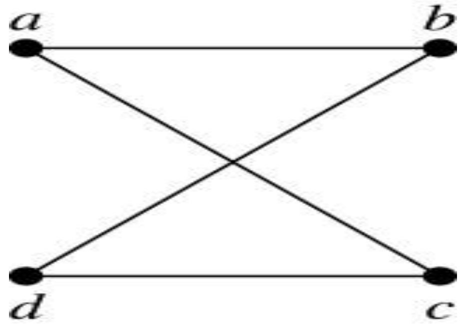
Adjacency Matrices (*continued*)

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

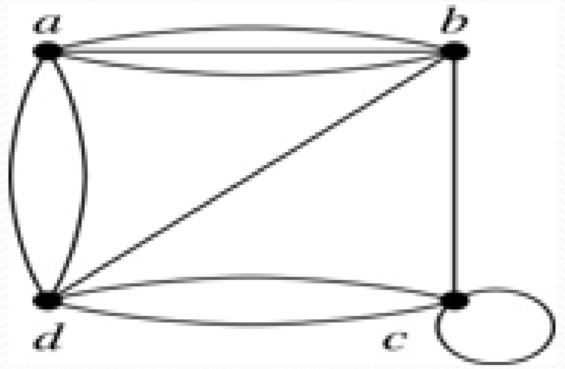
Adjacency Matrices (*continued*)

Adjacency matrices can also be used to represent graphs with loops and multiple edges.

A loop at the vertex vi is represented by a 1 at the (i, j)th position of the matrix.

When multiple edges connect the same pair of vertices vi and vj, (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

Definition:

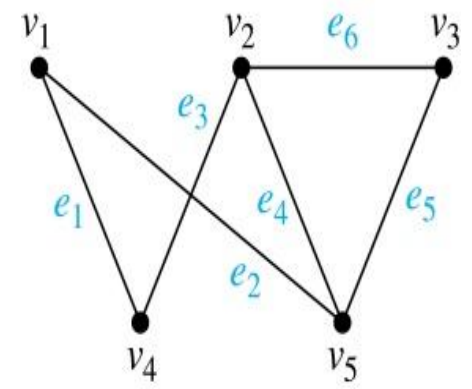
Let $G = (V, E)$ be an undirected graph with vertices where $v_1, v_2, \dots v_n$ and edges $e_1, e_2, \dots e_m$.

The incidence matrix with respect to the ordering of V and E is the $n \times m$ matrix $\mathbf{M} = [m_{ij}]$, where

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

Incidence Matrices (*continued*)

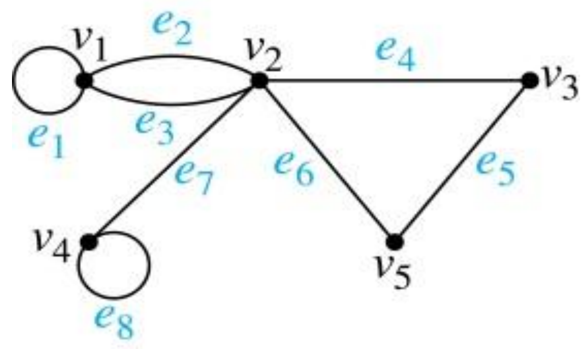
Example: Simple Graph and Incidence Matrix



$$\begin{array}{cc} & \text{edges} \\ \text{vertices} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

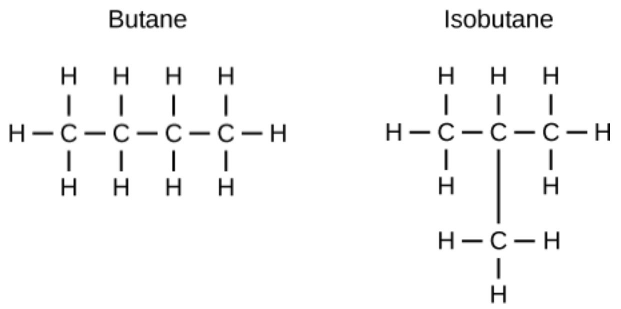
The rows going from top to bottom represent v_1 through v_5 and the columns going from left to right represent e_1 through e_6.

Example: Pseudograph and Incidence Matrix

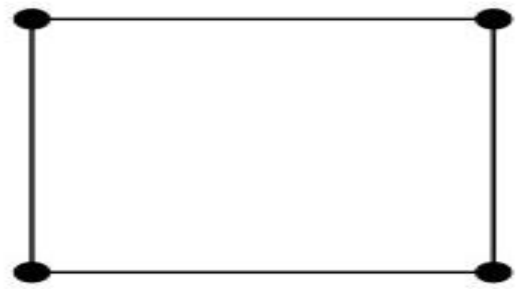


$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

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



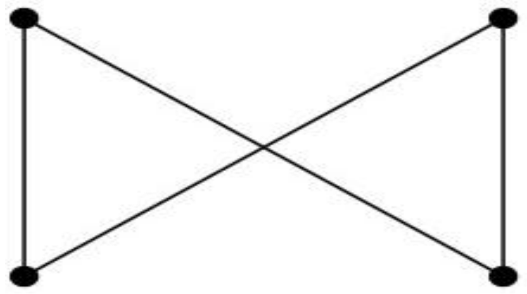
Isomorphism of Graphs



Two graphs G_1 and G_2 are said to be isomorphic if:

- Their number of components (vertices and edges) are same.
- Their edge connectivity is retained.

Note − In short, out of the two isomorphic graphs, one is a tweaked version of the other.



If $G_1 \equiv G_2$ then,

- $|V(G_1)| = |V(G_2)|$
- $|E(G_1)| = |E(G_2)|$
- Degree sequences of G_1 and G_2 are same.
- If the vertices $\{V_1, V_2, .. V_k\}$ form a cycle of length K in G_1, then the vertices $\{f(V_1), f(V_2),\ldots f(V_k)\}$ should form a cycle of length K in G_2.

Isomorphic Invariant

A property P is called an isomorphic invariant if, and only if, given any graphs G and G’, If G has property P and G’ is isomorphic to G, then G’ has property P.

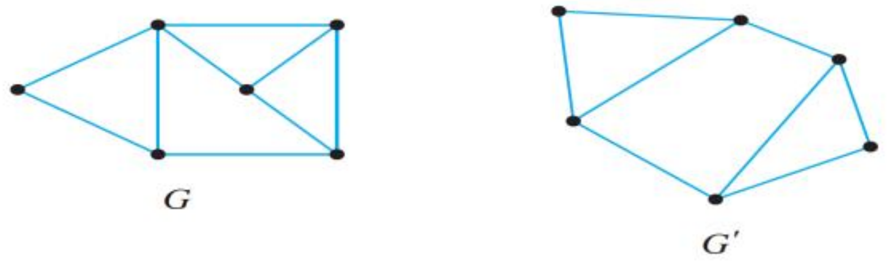
Each of the following properties is an invariant for graph isomorphism, where *n*, *m*, and *k* are all nonnegative integers:

1.has n vertices;	6.has a simple circuit of length k;
2.has m edges;	7.has m simple circuits of length k;
3.has a vertex of degree k;	8.is connected;
4.has m vertices of degree k;	9.has an Euler circuit;
5.has a circuit of length k;	10.has a Hamiltonian circuit.

Isomorphic Invariant: Example

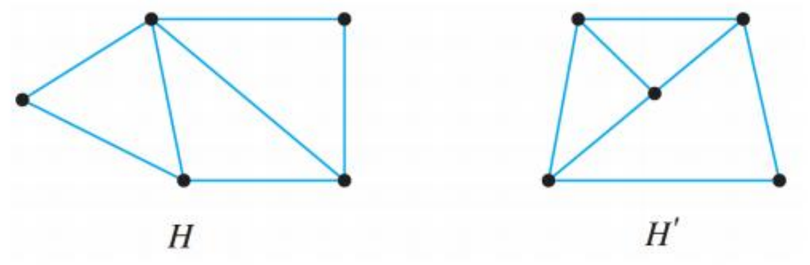
Show that the following pairs of graphs are not isomorphic by finding an isomorphic invariant that they do not share.

a.



a. *G* has nine edges; *G′* has only eight.

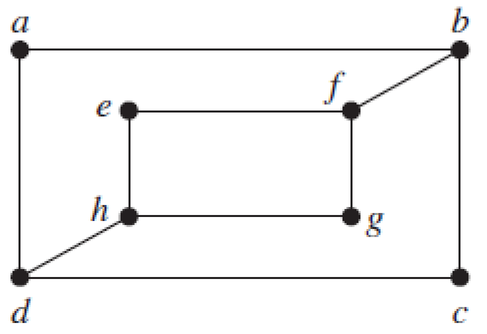
b.

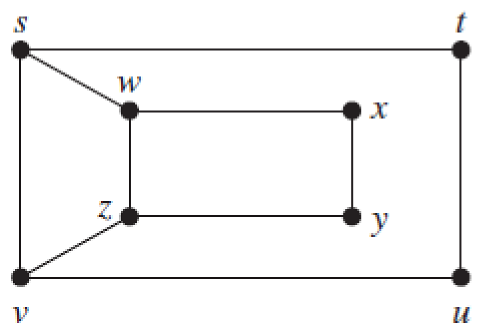


b. *H* has a vertex of degree 4; *H′* does not.

Isomorphic Invariant: Example

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*.





|E| = 10, |V| = 8 |E| = 10, |V| = 8

A=2, B=3, C=2, D=3, E=2, F=3, G=2, H=3 **S=3, T=2, U=2, V=3, W=3, X=2, Y=2, Z=3**

deg(*a*) = 2 in *G*, *a* must correspond to either *t*, *u*, *x*, or *y* in *H*

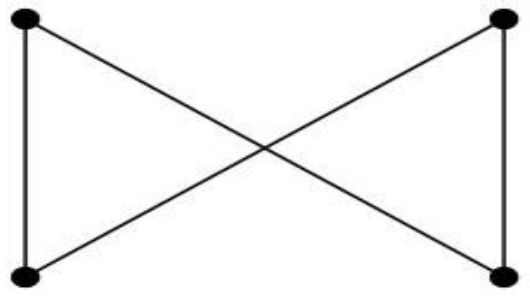
However, each of these four vertices in *H* is adjacent to another vertex of degree two in *H*, which is not true for *a* in *G*.

Paths

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.



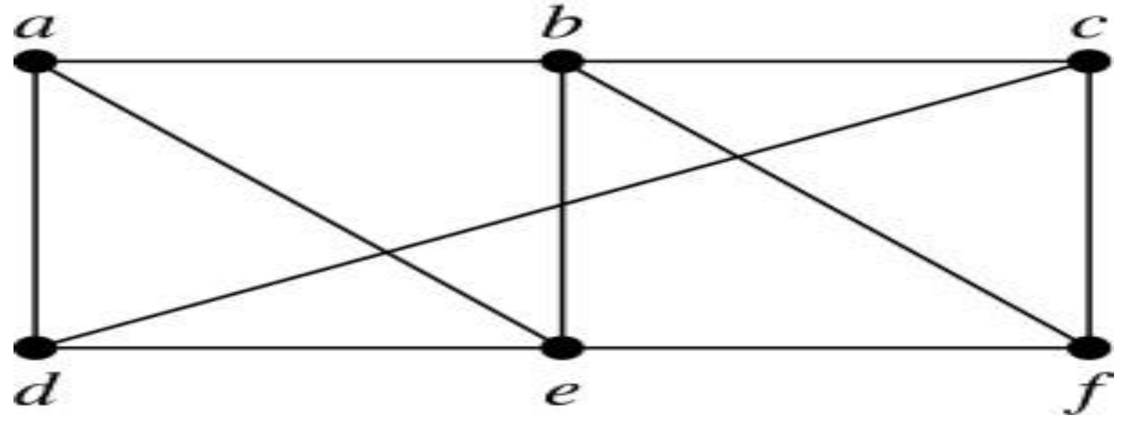
Paths

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 $e_1, \dots, e_n$ of G for which there exists a sequence $x_0 = u, x_1, \dots, x_{n-1}, x_n = v$ of vertices such that e_i has, for $i = 1, \dots, n$, the endpoints x_{i-1} and x_i.

- When the graph is simple, we denote this path by its vertex sequence $x_0, x_1, \dots, x_n$(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 $x_1, x_2, \dots, x_{n-1}$ and *traverse* the edges $e_1, \dots, e_n$.
- 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. (*see text*)

Paths (*continued*)



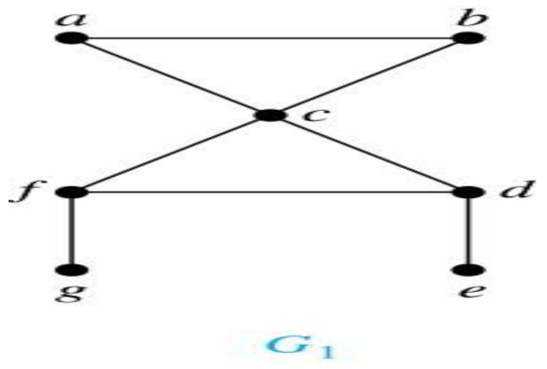
Example: In the simple graph here:

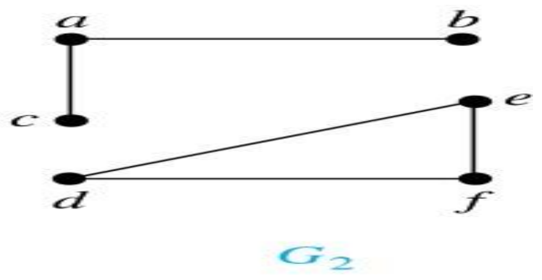
- a, d, c, f, e is a simple path of length 4.
- d, e, c, a is not a path because e is not connected to c.
- b, c, f, e, b is a circuit of length 4.
- a, b, e, d, a, b is a path of length 5, but it is not a simple path.

Connectedness in Undirected Graphs

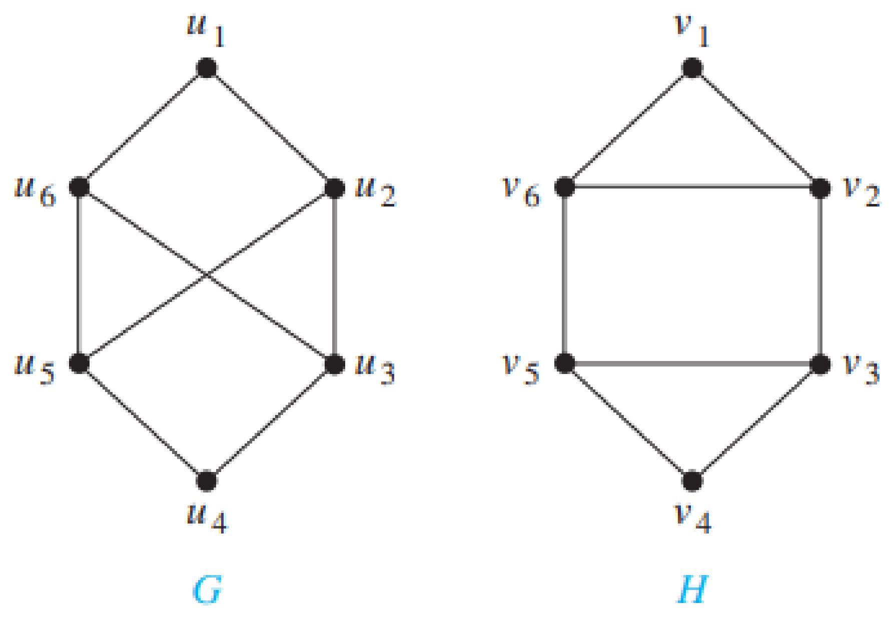
Definition: An undirected graph is called *connected* if there is a path between every pair of vertices. An undirected graph that is not *connected* is called *disconnected*. We say that we *disconnect* a graph when we remove vertices or edges, or both, to produce a disconnected subgraph.

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.





Isomorphic Invariant: Example



Both *G* and *H* have six vertices and eight edges. Each has four vertices of degree three, and two vertices of degree two. So, the three invariants—number of vertices, number of edges, and degrees of vertices—all agree for the two graphs. However, *H* has a **simple circuit** of length three, namely, *v*1, *v*2, *v*6, *v*1, whereas *G* has no simple circuit of length three, as can be determined by inspection (all simple circuits in *G* have length at least four). Because the existence of a simple circuit of length three is an isomorphic invariant, *G* and *H* **are not isomorphic.**

Euler & Hamilton Paths/Circuits

Can we travel along the edges of a graph starting at a vertex and returning to it by traversing each edge of the graph exactly once?

Similarly, can we travel along the edges of a graph starting at a vertex and returning to it while visiting each vertex of the graph exactly once?

The first question, which asks whether a graph has an *Euler circuit*, can be easily answered simply by examining the degrees of the vertices of the graph

The second question asks whether a graph has a *Hamilton circuit*, is quite difficult to solve for most graphs.

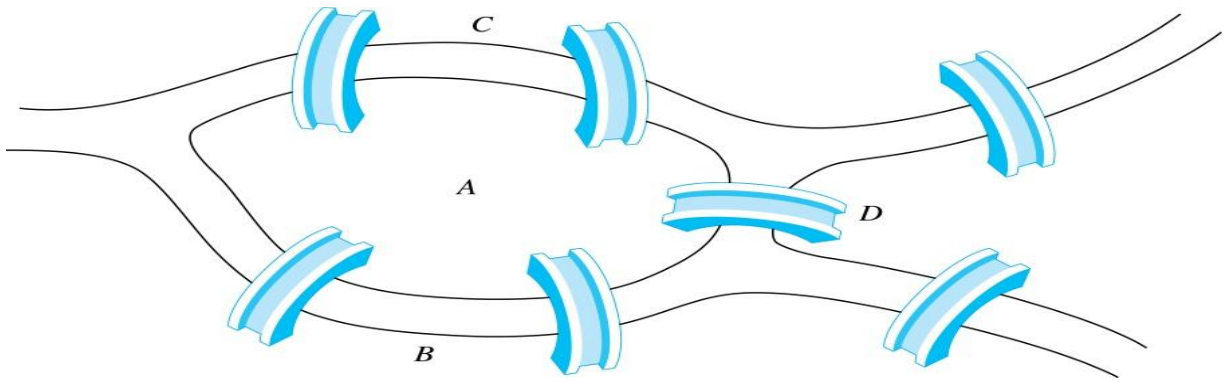
Euler Paths and Circuits

Leonard Euler (1707-1783)

The town of Königsberg, 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.

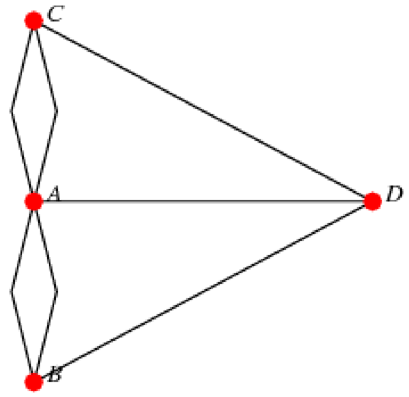
People wondered whether whether it was possible to follow a path that crosses each bridge exactly once and returns to the starting point.

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

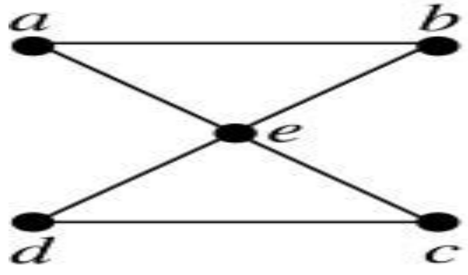
Multigraph Model of the Bridges of Königsberg



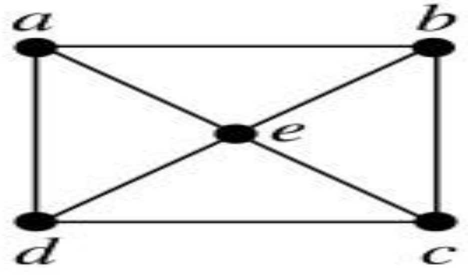
Euler Paths and Circuits

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*.

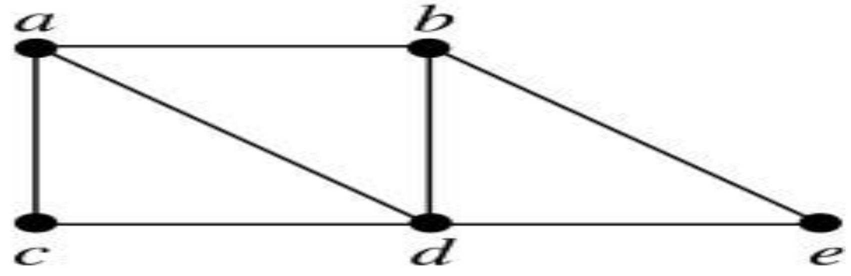
Example: Which of the undirected graphs G_1, G_2, and G_3 has a Euler circuit? Of those that do not, which has an Euler path?



G_1



G_2



G_3

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.

Necessary and Sufficient Conditions for Euler Circuits and Paths

An Euler circuit begins with a vertex *a* and continues with an edge incident with *a*, say {*a*, *b*}. The edge {*a*, *b*} contributes one to deg(*a*).

Each time the circuit passes through a vertex it contributes two to the vertex's degree.

Finally, the circuit terminates where it started, contributing one to deg(*a*). Therefore deg(*a*) must be even. We conclude that the degree of every other vertex must also be even.

By the same reasoning, we see that the initial vertex and the final vertex of an Euler path have odd degree, while every other vertex has even degree. So, a graph with an Euler path has exactly two vertices of odd degree.

In the next slide we will show that these necessary conditions are also sufficient conditions.

Necessary and Sufficient Conditions for Euler Circuits and Paths (*continued*)

Theorem 1:

A connected multigraph has an Euler circuit if and only if each of its vertices has even degree.

Theorem 2:

A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

Algorithm for Constructing an Euler Circuits

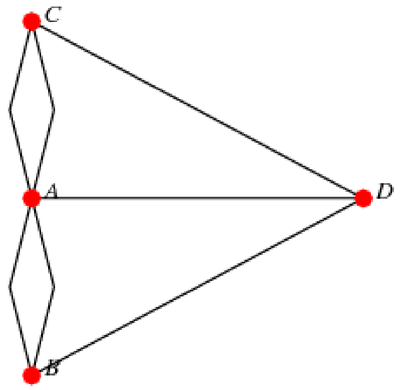
In our proof we developed this algorithms for constructing a Euler circuit in a graph with no vertices of odd degree.

```
  **procedure** *Euler*($G$: connected multigraph with all vertices of even degree)
  *circuit* := a circuit in $G$ beginning at an arbitrarily chosen vertex with edges
                 successively added to form a path that returns to this vertex.
   $H$ := $G$ with the edges of this circuit removed
   **while** $H$ has edges
       *subciruit* := a circuit in $H$ beginning at a vertex in $H$ that also is
                      an endpoint of an edge in circuit.
       $H$ := $H$ with edges of *subciruit* and all isolated vertices removed
       *circuit* := *circuit* with *subcircuit* inserted at the appropriate vertex.
**return** *circuit*{*circuit* is an Euler circuit}
```

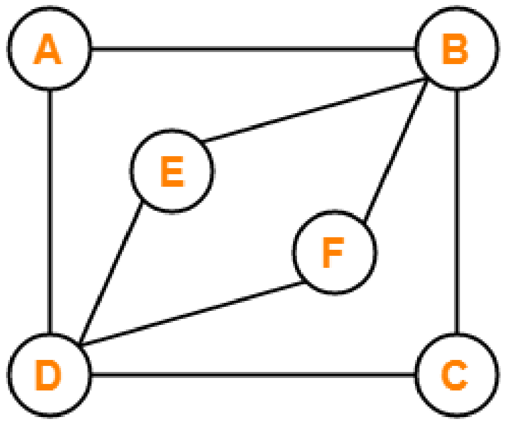
Necessary and Sufficient Conditions for Euler Circuits and Paths (*continued*)

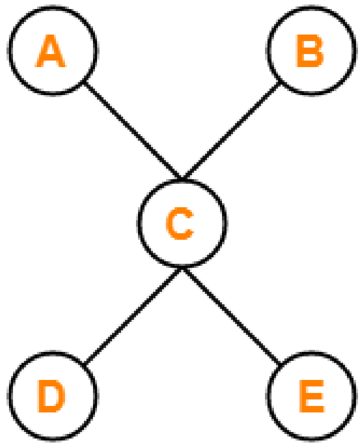
Theorem: A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has an even degree and it has an Euler path if and only if it has exactly two vertices of odd degree.

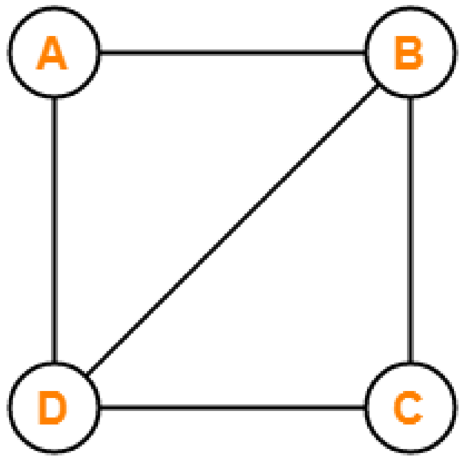
Example: Two of the vertices in the multigraph model of the Königsberg bridge problem have odd degree.   Hence, there is no Euler circuit in this multigraph and it is impossible to start at a given point, cross each bridge exactly once, and return to the starting point.



Euler Paths and Circuits: Exercise

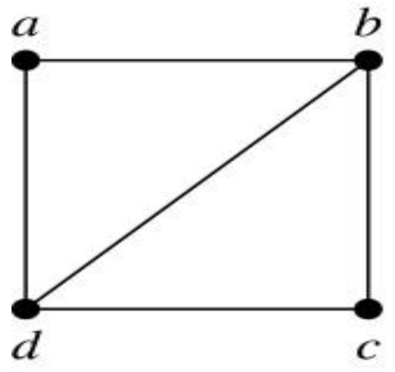




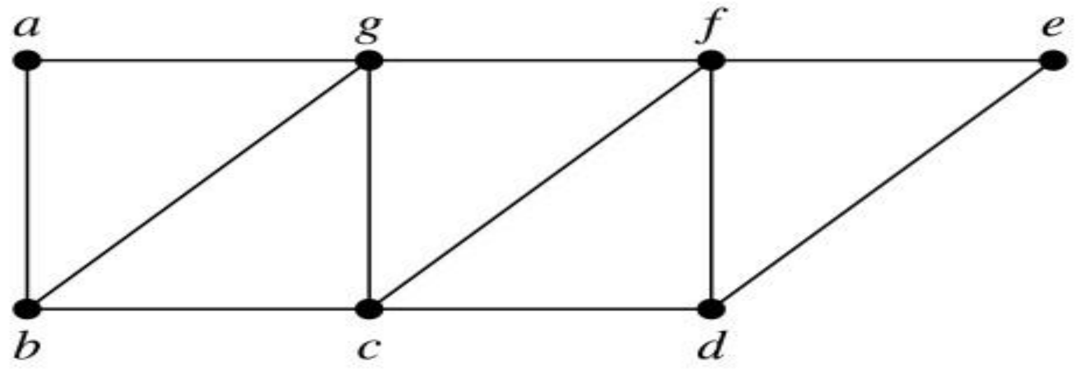


Euler Circuits and Paths

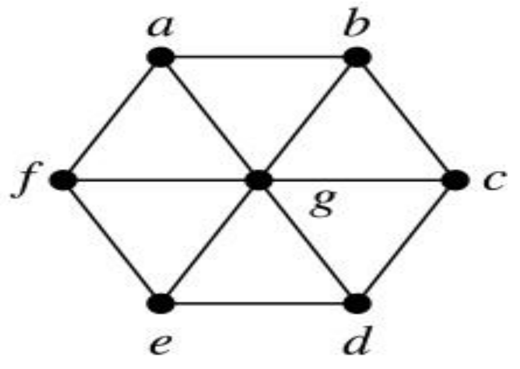
Example:



G_1



G_2



G_3

G_1 contains exactly two vertices of odd degree (*b* and *d*). Hence it has an Euler path, e.g., *d*, *a*, *b*, *c*, *d*, *b*.

G_2 has exactly two vertices of odd degree (*b* and *d*). Hence it has an Euler path, e.g., *b*, *a*, *g*, *f*, *e*, *d*, *c*, *g*, *b*, *c, f*, *d*.

G_3 has six vertices of odd degree. Hence, it does not have an Euler path.

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.

William Rowan Hamilton (1805-1865)

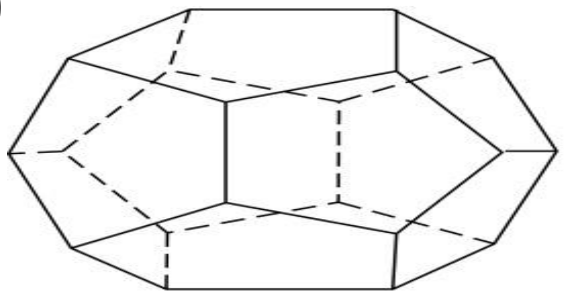
Hamilton Paths and Circuits

Euler paths and circuits contained every edge only once.

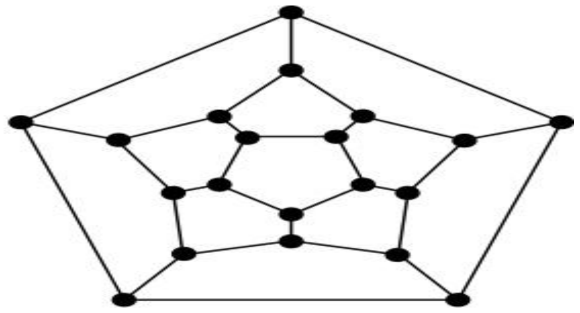
Now we look at paths and circuits that contain every vertex exactly once.

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 used to plot a circuit visiting 20 cities exactly once

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

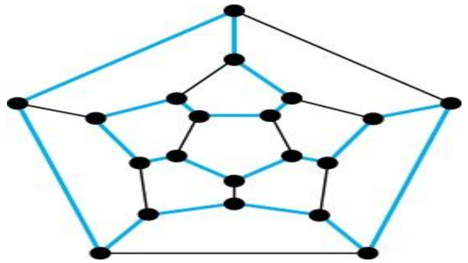


(a)



(b)

The solution (a Hamilton circuit) is given here.



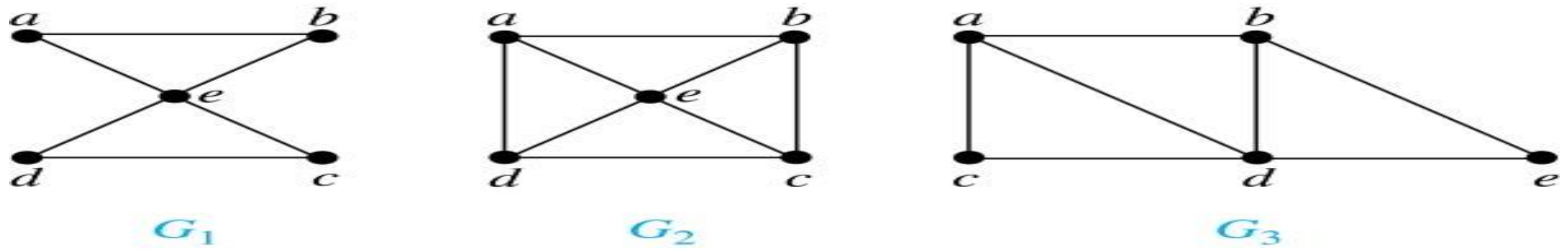
Hamilton Paths and Circuits

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.

Hamilton Paths and Circuits (*continued*)

Example: Which of these simple graphs has a Hamilton circuit or, if not, a Hamilton path?



Solution: G_2 has a Hamilton circuit: *a*, *b*, *c*, *d*, *e*, *a*.

G_1 does not have a Hamilton circuit (Why?), but does have a Hamilton path : *a*, *b* ,e , *c*, *d*.

G_3 has a Hamilton circuit: a,b,e,d,c,a

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 Hamiton 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

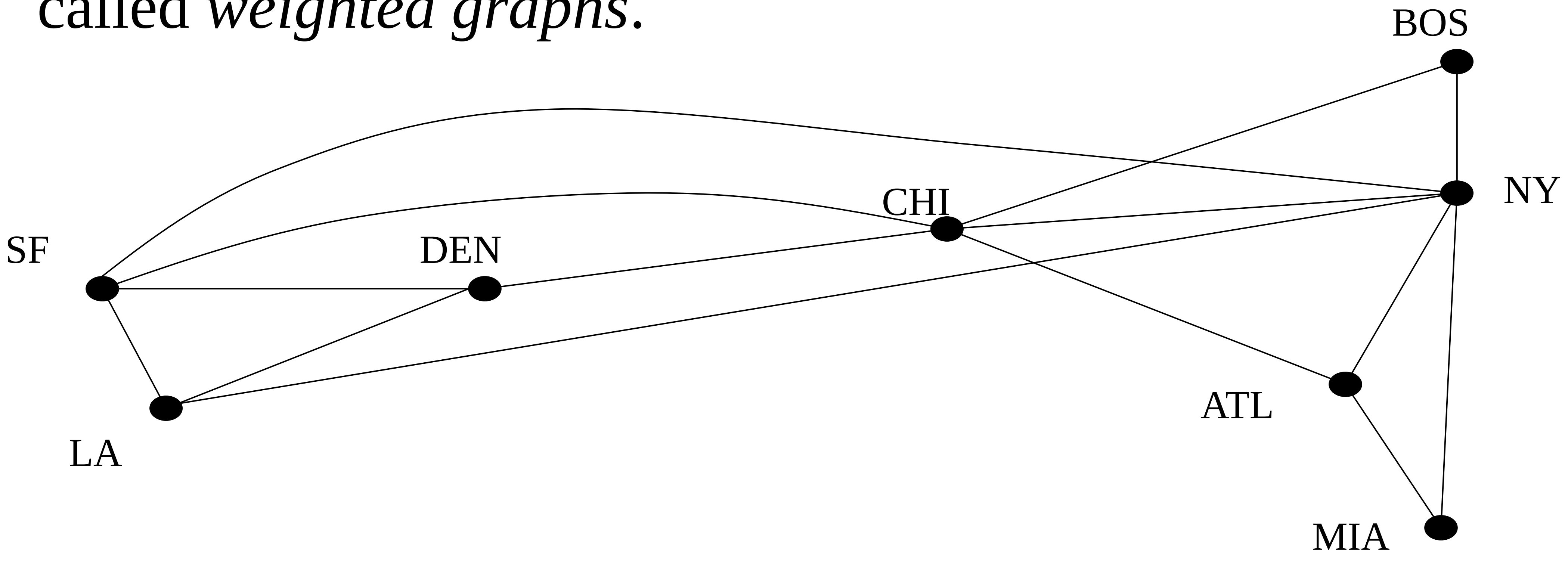
Applications that ask for a path or a circuit that visits each intersection of a city, each place pipelines intersect in a utility grid, or each node in a communications network exactly once, can be solved by finding a Hamilton path in the appropriate graph.

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.

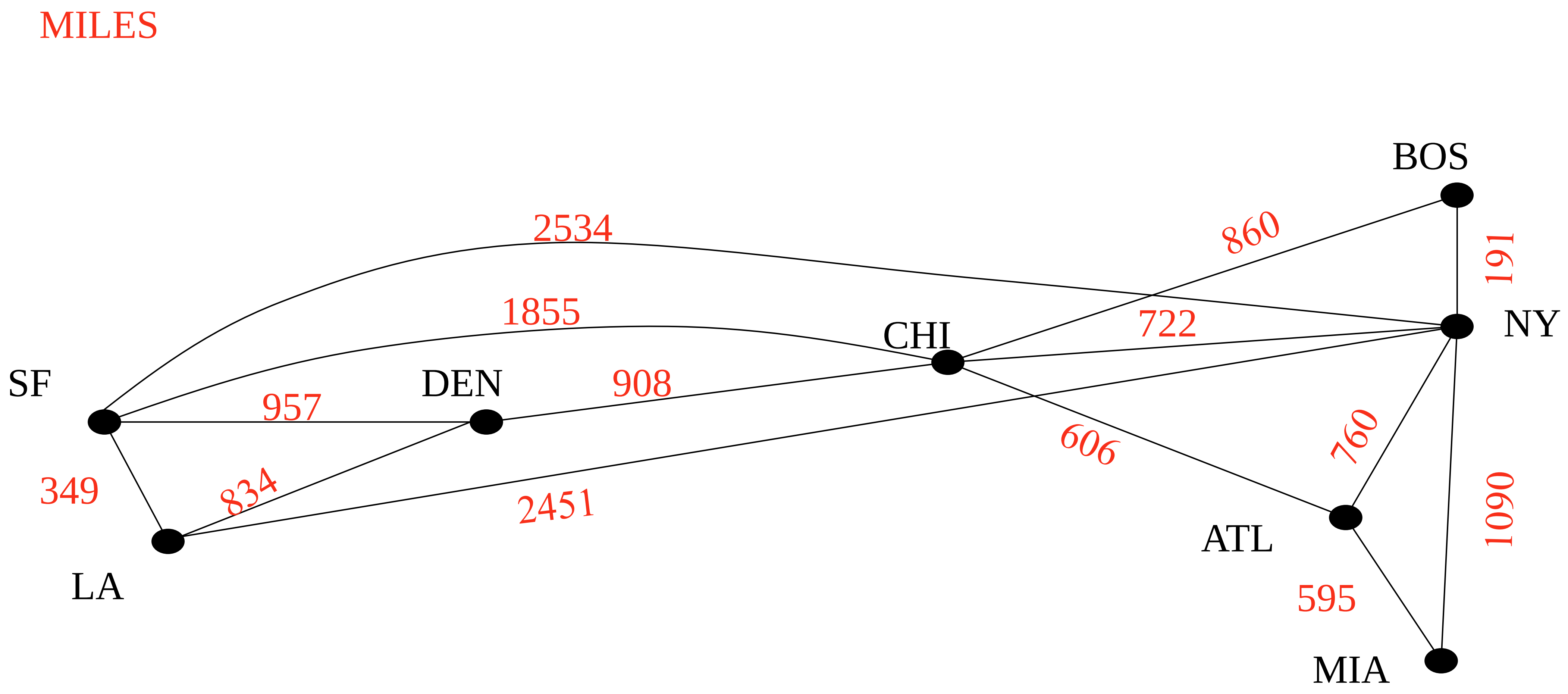
A family of binary codes, known as *Gray codes*, which minimize the effect of transmission errors, correspond to Hamilton circuits in the *n*-cube $Q_{n.}$

Weighted Graphs

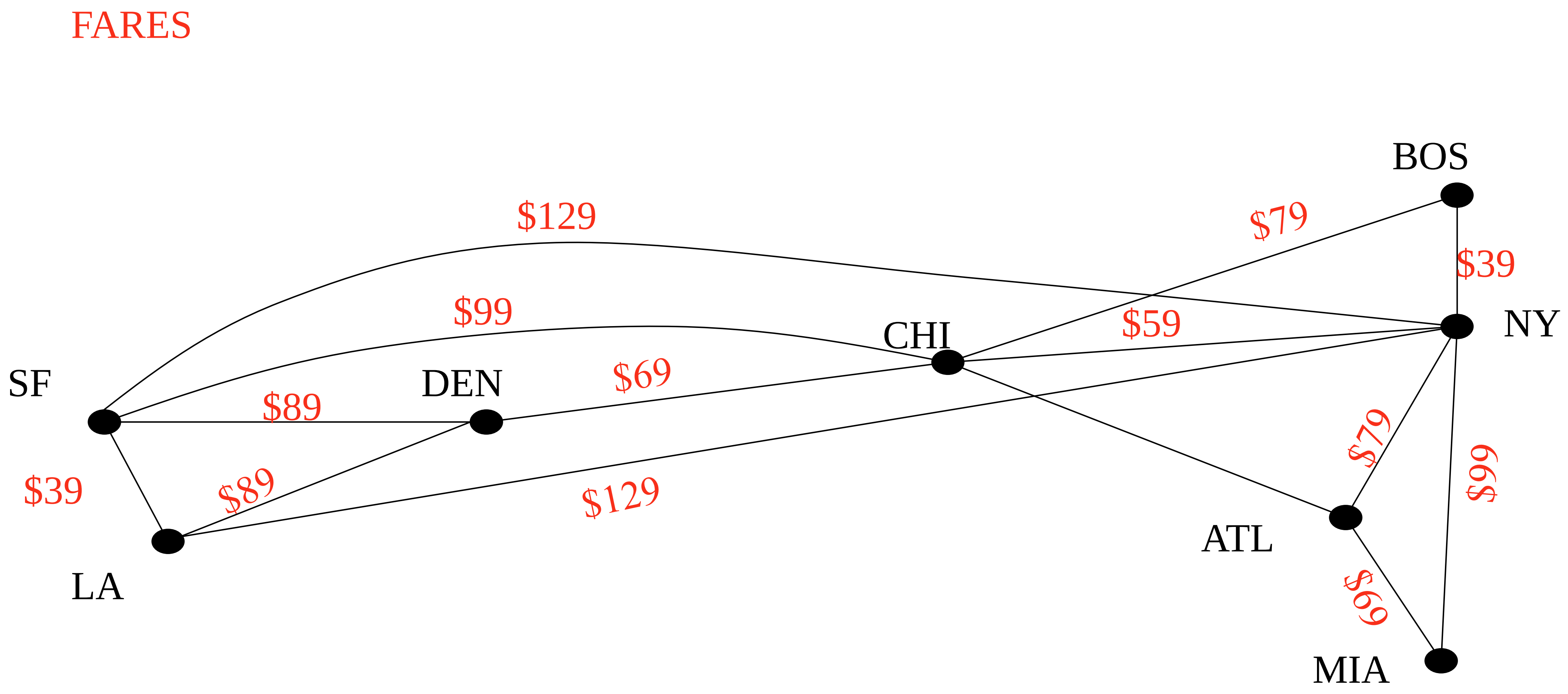
Graphs that have a number assigned to each edge are called *weighted graphs*.



Weighted Graphs

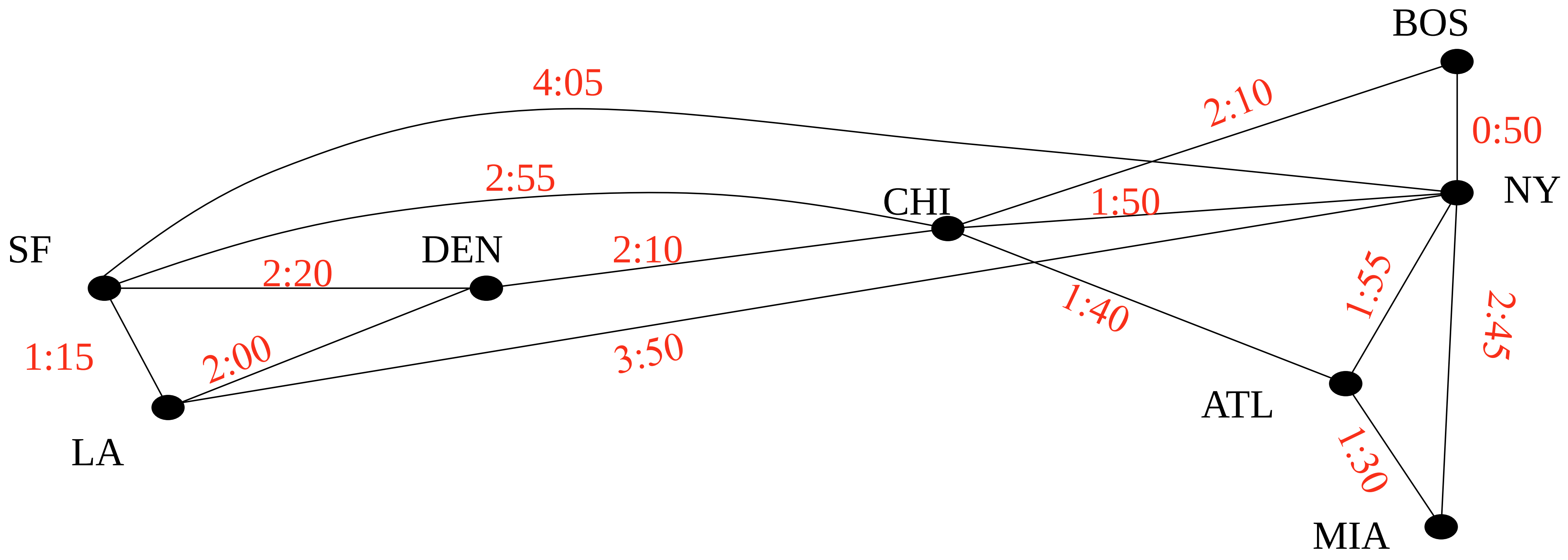


Weighted Graphs



Weighted Graphs

FLIGHT TIMES



Weighted Graphs

A weighted graph is a graph in which each edge (u, v) has a weight w(u, v). Each weight is a real number.

Weights can represent distance, cost, time, capacity, etc.

The length of a path in a weighted graph is the sum of the weights on the edges.

Dijkstra’s Algorithm finds the shortest path between two vertices.

Weighted Graphs Example

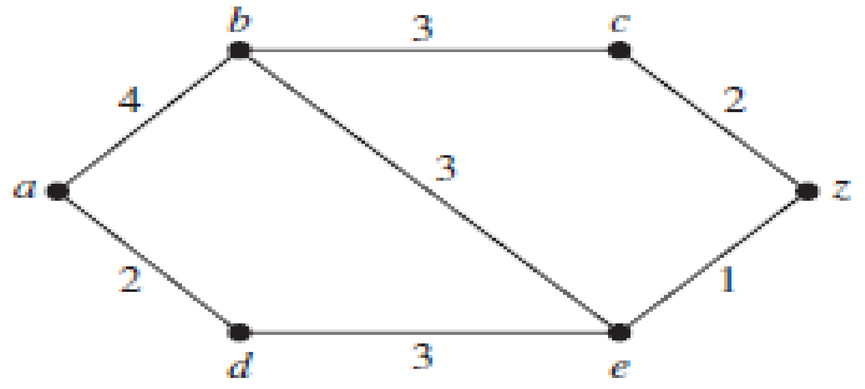


FIGURE 3 A Weighted Simple Graph.

What is the length of a shortest path between *a* and *z* in the weighted graph shown in Figure 3?

Dijkstra’s Algorithm

Dijkstra’s algorithm is used in problems relating to finding the shortest path.

Each node is given a temporary label denoting the length of the shortest path *from* the start node *so far*.

This label is replaced if another shorter route is found.

Once it is certain that no other shorter paths can be found, the temporary label becomes a permanent label.

Eventually all the nodes have permanent labels.

At this point the shortest path is found by retracing the path backwards.

Dijkstra’s Algorithm: Traversing Example

◯ *It means the smallest

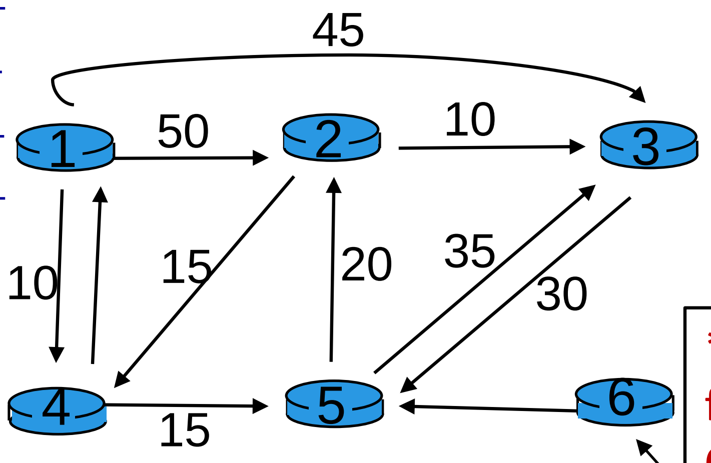
◯ *It means already visited

*We choose the minimum

Step	N'	D(2)	D(3)	D(4)	D(5)	D(6)
0	1	50	45	10	∞	∞
1	4	50	45	10	25	∞
2	5	45	45	10	25	∞
3	2	45	45	10	25	∞
4	3	45	45	10	25	∞
5	6?					

*We update these values only when we get smaller than previous one

5

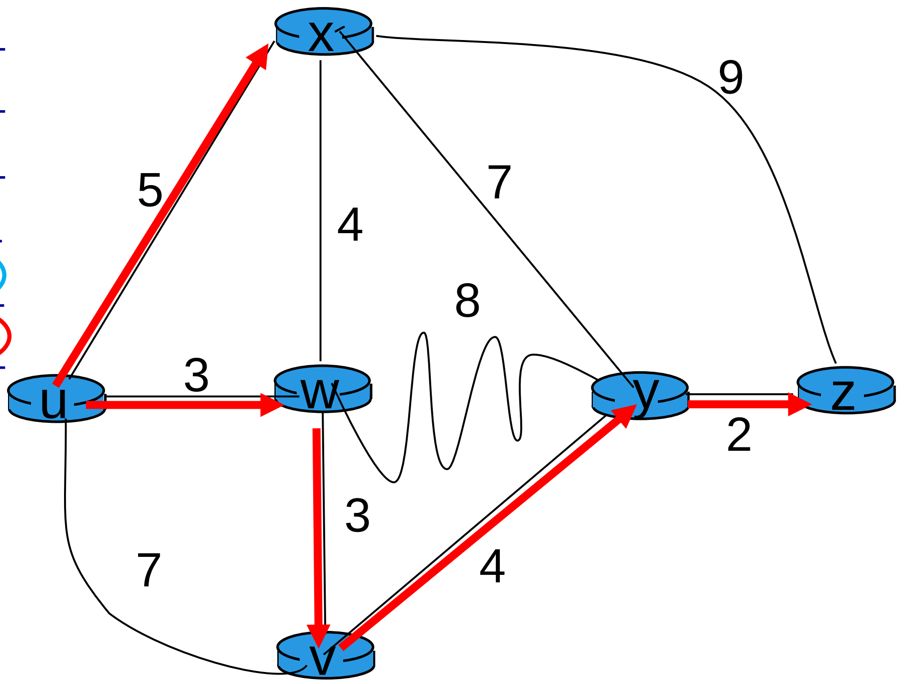


*There’re no edge from any vertex to **6**, So we can not traverse it

Dijkstra’s Algorithm: Shortest Path Example-1

Parent Vertex in order of traversal

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w	3,u	5,u	11,w	∞
2	uwx	6,w	3,u	5,u	11,w	14,x
3	uwxv	6,w	3,u	5,u	10,v	14,x
4	uwxvy	6,w	3,u	5,u	10,v	12,y
5	uwxvyz	6,w	3,u	5,u	10,v	12,y



Notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

Dijkstra’s Algorithm: Constructing Shortest Path

Parent Vertex in order of traversal

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w	3,u	5,u	11,w	∞
2	uwx	6,w	3,u	5,u	11,w	14,x
3	uwxv	6,w	3,u	5,u	10,v	14,x
4	uwxvy	6,w	3,u	5,u	10,v	12,y
5	uwxvyz	6,w	3,u	5,u	10,v	12,y

We will construct shortest path by traversing back from z to u.

1- Start with last selected i.e. **z** 12,**y**

2- Traverse to its parent i.e. **y** we get 10,**v**

3- Traverse to its parent i.e. **v** we get 6,**w**

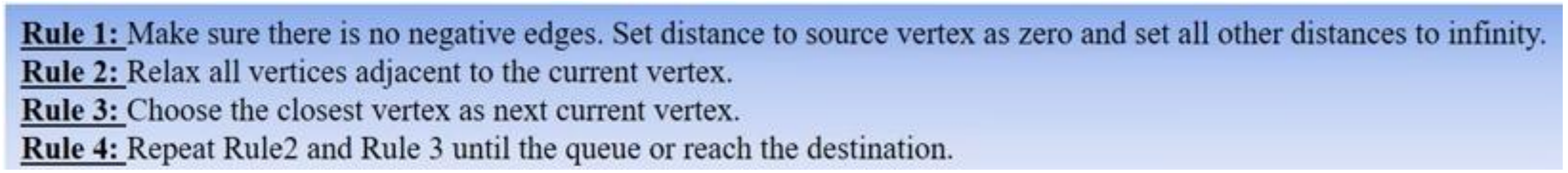
4- Traverse to its parent i.e. **w** we get 3,**u**

5- We get shortest path **u->w->v->y->z**

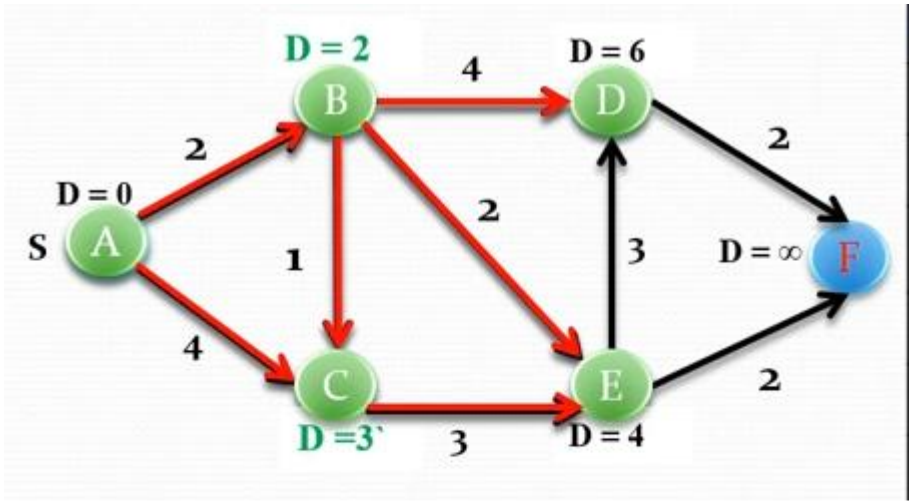
Cost of the path is 12

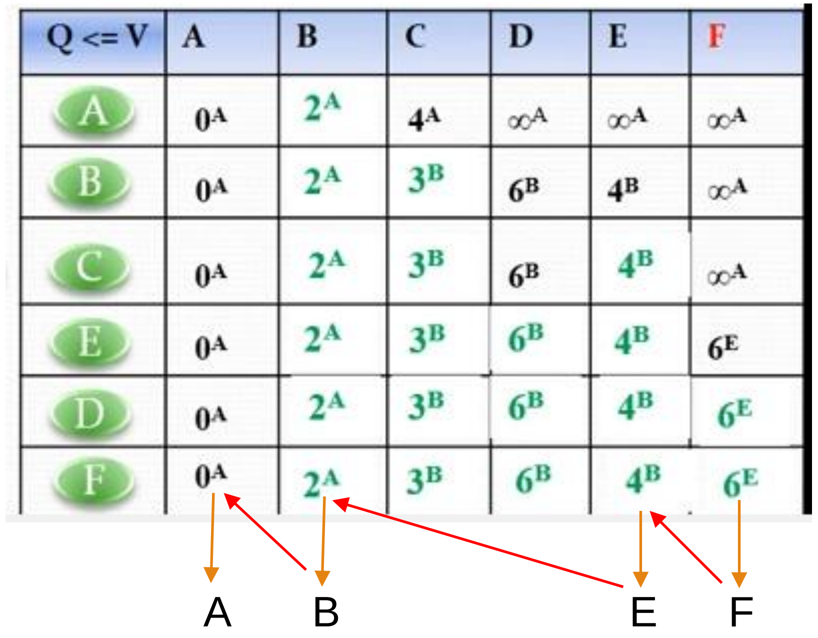
So, the constructed path is **uwvyz**

Dijkstra’s Algorithm: Shortest Path Example-2



Rule 1: Make sure there is no negative edges. Set distance to source vertex as zero and set all other distances to infinity.
Rule 2: Relax all vertices adjacent to the current vertex.
Rule 3: Choose the closest vertex as next current vertex.
Rule 4: Repeat Rule2 and Rule 3 until the queue or reach the destination.

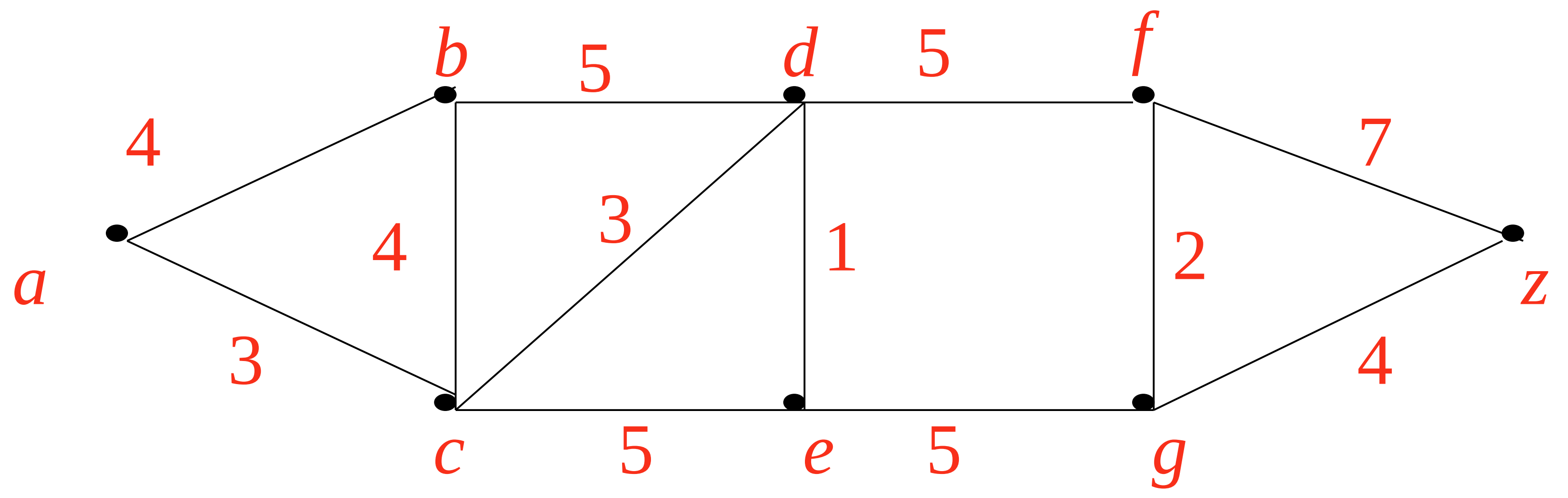




Q <= V	A	B	C	D	E	F
A	0^A	2^A	4^A	∞^A	∞^A	∞^A
B	0^A	2^A	3^B	6^B	4^B	∞^A
C	0^A	2^A	3^B	6^B	4^B	∞^A
E	0^A	2^A	3^B	6^B	4^B	6^E
D	0^A	2^A	3^B	6^B	4^B	6^E
F	0^A	2^A	3^B	6^B	4^B	6^E

So, the shortest path is A->B->E->F of Cost 6

Problem: shortest path from a to z



Solution: shortest path from a to z

S	D(b)	D(c)	D(d)	D(e)	D(f)	D(g)	D(z)
a	4,a	**3,a**	∞	∞	∞	∞	∞
ac	**4,a**		6,c	8,c	∞	∞	∞
acb			**6,c**	8,c	∞	∞	∞
acbd				**7,d**	11,d	∞	∞
acbde					**11,d**	12,e	∞
acbdef						**12,e**	18,f
acbdefg							**16,g**

So, the constructed path is **a->c->d->e->g->z**
And Cost is 16

The Traveling Salesman Problem

The **traveling salesman problem** is one of the classical problems in computer science.

A traveling salesman wants to visit a number of cities and then return to his starting point. Of course he wants to save time and energy, so he wants to determine the **shortest cycle** for his trip.

We can represent the cities and the distances between them by a weighted, complete, undirected graph.

The problem then is to find the **shortest cycle (of minimum total weight that visits each vertex exactly one)**.

Finding the shortest cycle is different than Dijkstra's shortest path.
It is much harder too, no polynomial time algorithm exists!

The Traveling Salesman Problem

Importance:

- Variety of scheduling application can be solved as a traveling salesmen problem.
- Examples:
  - Ordering drill position on a drill press.
  - School bus routing.
- The problem has theoretical importance because it represents a class of difficult problems known as NP-hard problems.

Travelling Salesman problem

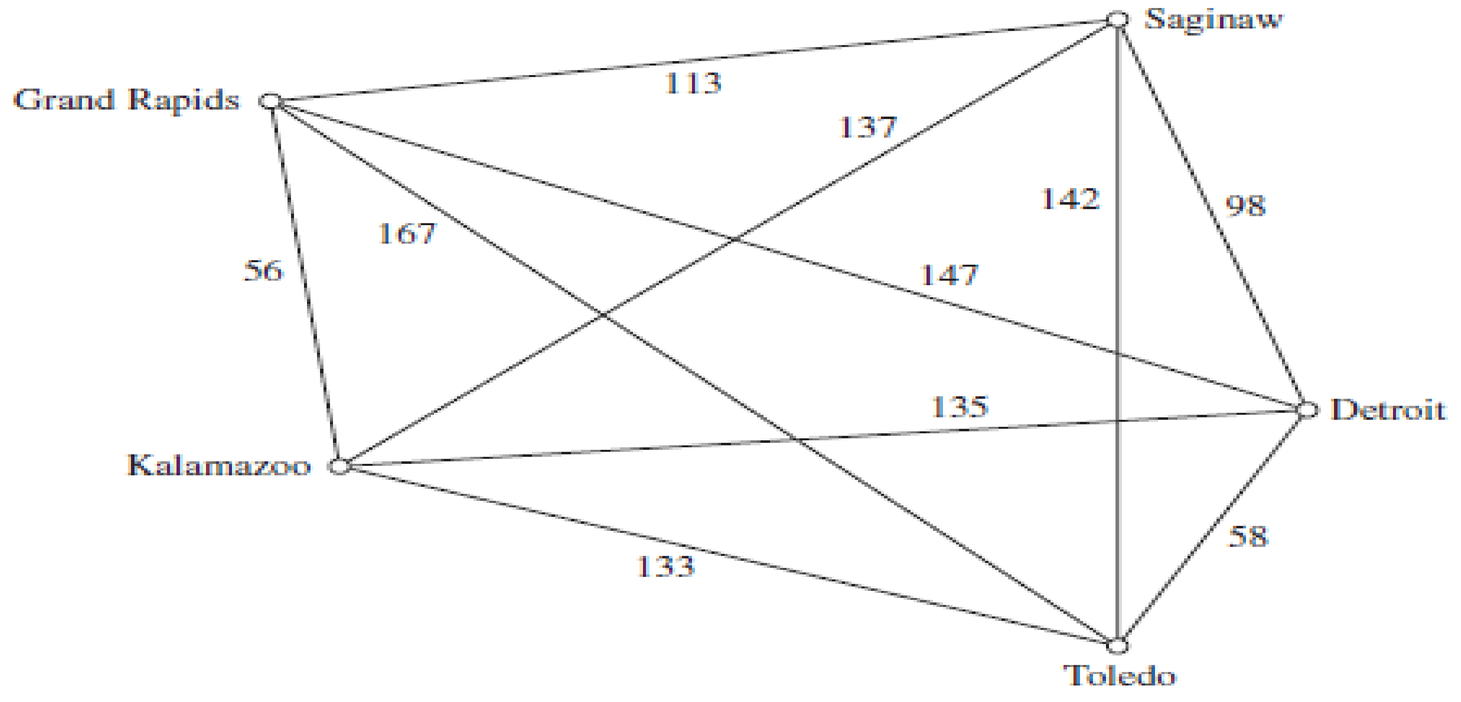
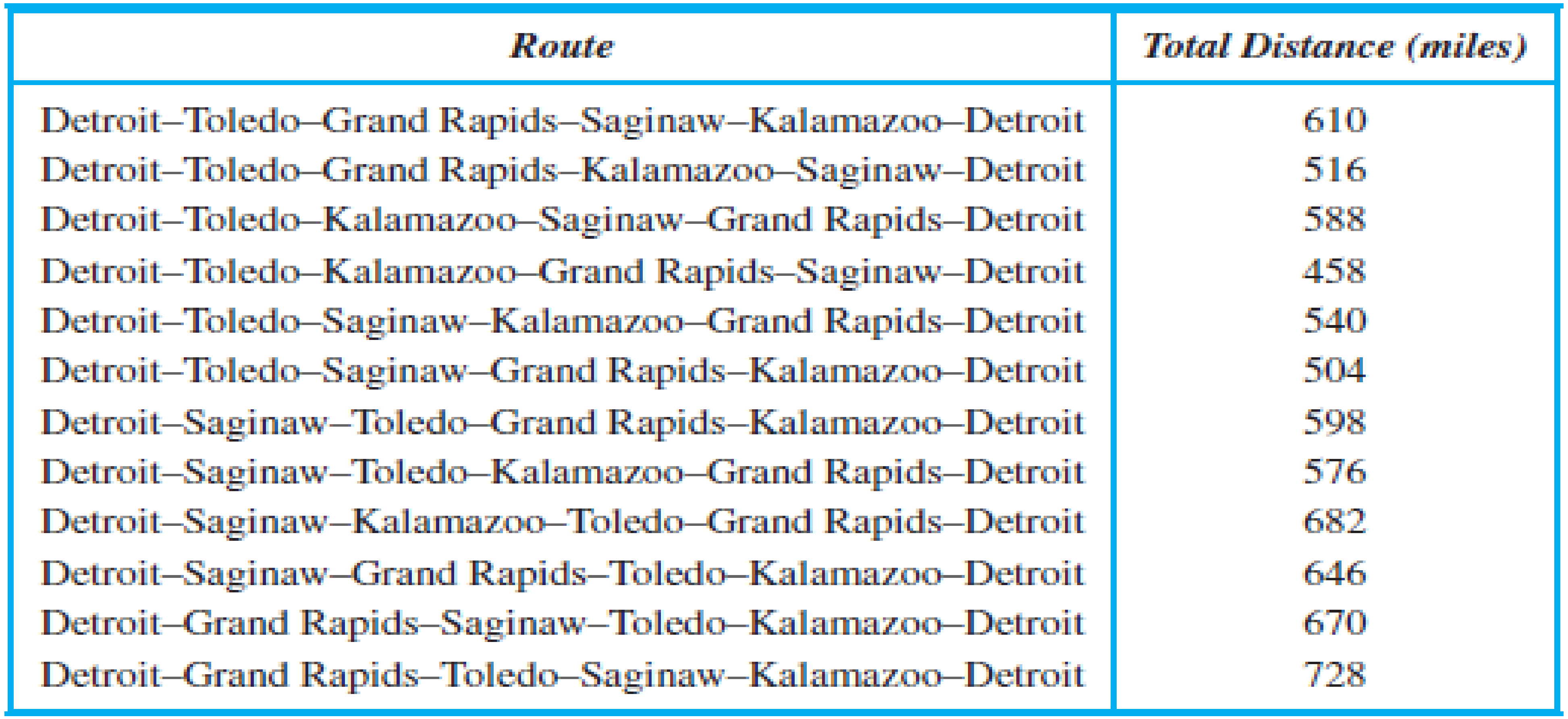


FIGURE 5 **The Graph Showing the Distances between Five Cities.**

Travelling Salesman problem



Route	*Total Distance (miles)*
Detroit–Toledo–Grand Rapids–Saginaw–Kalamazoo–Detroit	610
Detroit–Toledo–Grand Rapids–Kalamazoo–Saginaw–Detroit	516
Detroit–Toledo–Kalamazoo–Saginaw–Grand Rapids–Detroit	588
Detroit–Toledo–Kalamazoo–Grand Rapids–Saginaw–Detroit	458
Detroit–Toledo–Saginaw–Kalamazoo–Grand Rapids–Detroit	540
Detroit–Toledo–Saginaw–Grand Rapids–Kalamazoo–Detroit	504
Detroit–Saginaw–Toledo–Grand Rapids–Kalamazoo–Detroit	598
Detroit–Saginaw–Toledo–Kalamazoo–Grand Rapids–Detroit	576
Detroit–Saginaw–Kalamazoo–Toledo–Grand Rapids–Detroit	682
Detroit–Saginaw–Grand Rapids–Toledo–Kalamazoo–Detroit	646
Detroit–Grand Rapids–Saginaw–Toledo–Kalamazoo–Detroit	670
Detroit–Grand Rapids–Toledo–Saginaw–Kalamazoo–Detroit	728

Summary

Graph
A graph is a mathematical structure consisting of a set of points called **VERTICES** and a set (possibly empty) of lines linking some pair of vertices. It is possible for the edges to oriented; i.e. to be directed edges. The lines are called **EDGES** if they are undirected, and or **ARCS** if they are directed.

Loop and Multiple edges
A **loop** is an edge that connects a vertex to itself. If a graph has more than one edge joining some pair of vertices then these edges are called **multiple edges**.

Simple Graph
A **simple graph** is a graph that does not have more than one edge between any two vertices and no edge starts and ends at the same vertex. In other words a simple graph is a graph without loops and multiple edges.

Adjacent Vertices
Two vertices are said to be **adjacent** if there is an edge (arc) connecting them.

Adjacent Edges
Adjacent edges are edges that share a common vertex.

Degree of a Vertex
The **degree** of a vertex is the number of edges incident with that vertex.

Path
A **path** is a sequence of vertices with the property that each vertex in the sequence is adjacent to the vertex next to it. A path that does not repeat vertices is called a **simple path**.

Circuit
A **circuit** is path that begins and ends at the same vertex.

A Connected Graph
A graph is said to be **connected** if **any two** of its vertices are joined by a path. A graph that is not connected is a **disconnected graph**. A disconnected graph is made up of connected subgraphs that are called **components**.

Bridge
A **bridge** is an edge whose deletion from a graph **increases the number of components** in the graph. If a graph was a connected graph then the removal of a bridge-edge disconnects it.

Euler Path
An **Euler path** is a path that travels through all edges of a connected graph.

Euler Circuit
An **Euler circuit** is a circuit that visits all edges of a connected graph.