Book

A Simplified Approach
to

Data Structures

Prof.(Dr.)Vishal Goyal, Professor, Punjabi University Patiala

Dr. Lalit Goyal, Associate Professor, DAV College, Jalandhar

Mr. Pawan Kumar, Assistant Professor, DAV College, Bhatinda

Shroff Publications and Distributors

Edition 2014

Contents for Today's Lecture

- Introduction
- Graphs
- Memory Representation of Graph

Introduction

- This chapter introduces an important non-linear data structure called **graph**. The data structure graph has applications in various fields like electrical and electronics engineering, computer science, games and puzzles, geographical information systems etc.

- Graph theory was originated in **Konigsberg Bridge** problem by **Leonhard Euler**, a mathematician who developed some concepts in solving this problem. Later, these concepts become the basis of the graph theory.

- Frequently, we use graphs as a model for the illustration of various practical situations.

Graph

- A graph **G** consists of finite set of vertices $\mathbf{V_G}$ and finite set of edges $\mathbf{E_G}$ which can be denoted by a tuple $\mathbf{G=(V_G , E_G)}$.
- Here, the set of vertices $\mathbf{V_G}$ represents the entities which has names and some other attributes.
- An edge connects a pair of vertices and represents a relationship between the two entities.
-

Graph (continued)

- A graph may be pictorially represented as shown in the figure:

In this graph, vertices are labeled using letters a, b, c, d and e. Therefore,

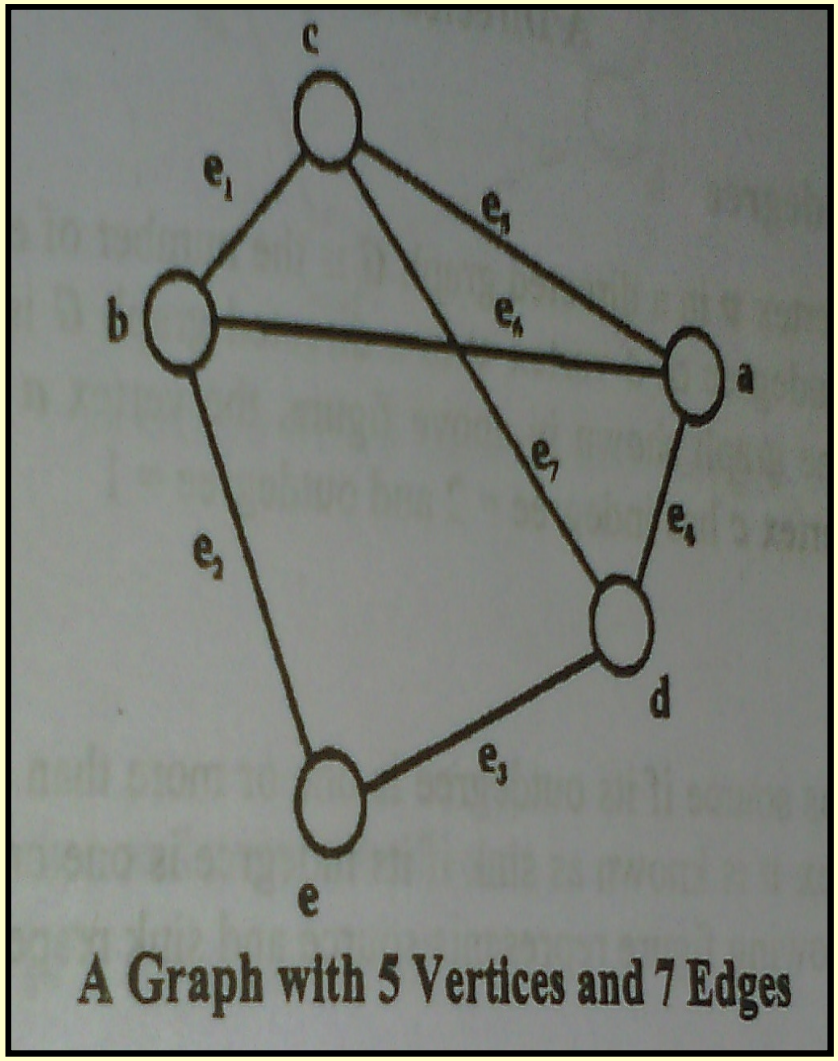
$\mathbf{V_G}$ ={a, b, c, d, e}

$\mathbf{E_G}$ ={ab , be, ed , dc, ca, bc , ad}

or

$\mathbf{E_G}$ ={e1, e2, e3, e4, e5, e6, e7}

In this graph, edge between two vertices can be written in any order. For example, the edge between the vertices **a** and **d** can be written as either **ad** or **da**.



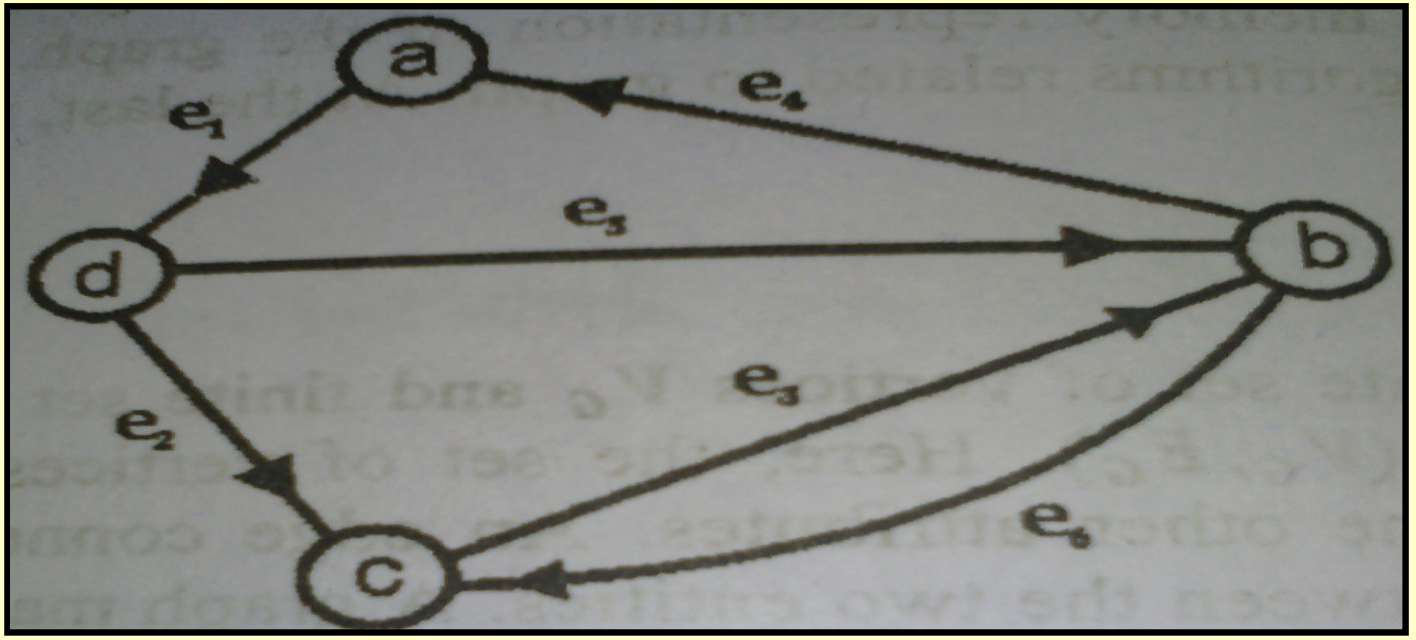
A Graph with 5 Vertices and 7 Edges

Graph Terminology

- **Directed Graph**

  In case of directed graph, each edge is assigned a direction or each edge is identified by an ordered pair of vertices in the graph rather than an unordered pair.

  The figure below shows a directed graph:

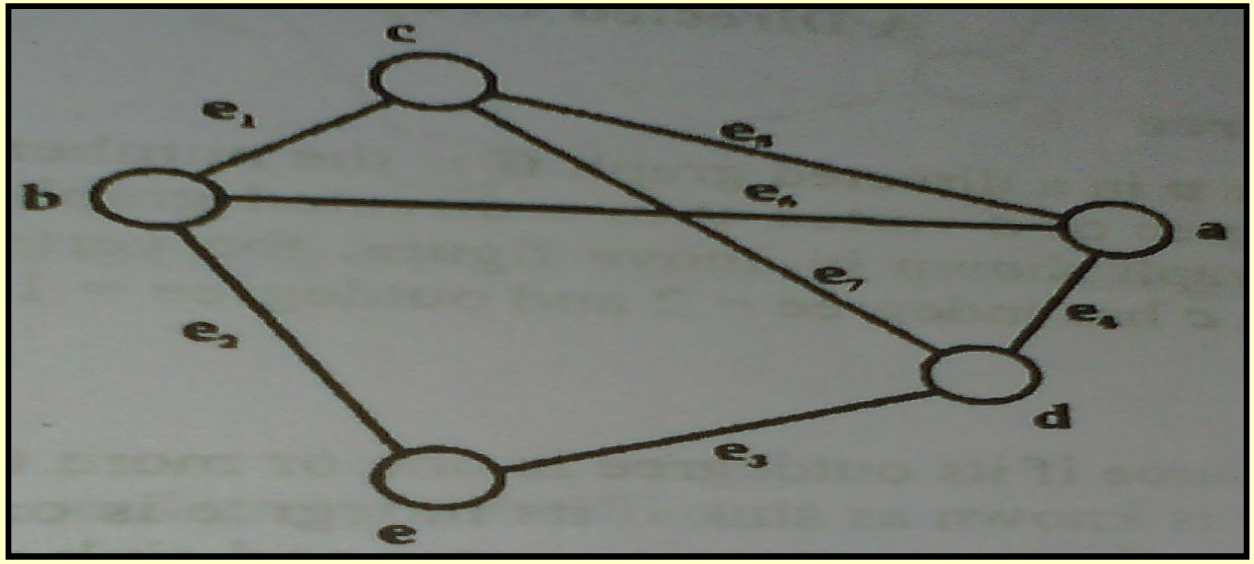


Graph Terminology(continued)

- **Undirected Graph**

  In case of undirected graph, no directions are associated with the edges of the graph. The edges of the undirected graph are represented by unordered pair of vertices.

  The figure below shows a undirected graph:

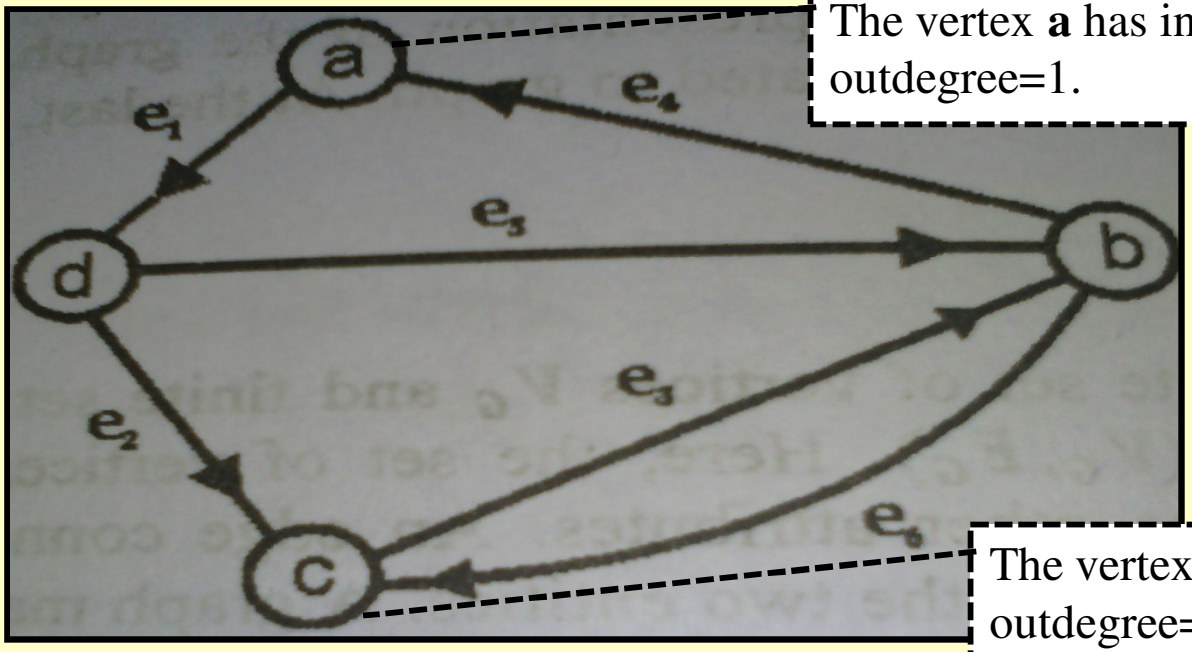


Graph Terminology(continued)

- **Outdegree and Indegree**

  The **outdegree** of a vertex **v** in a directed graph **G** is the number of edges starting from the vertex **v.**

  The **indegree** of a vertex **v** in a directed graph **G** is the number of edges terminating at **v.**



The vertex **a** has indegree=1 and outdegree=1.

The vertex **c** has indegree=2 and outdegree=1.

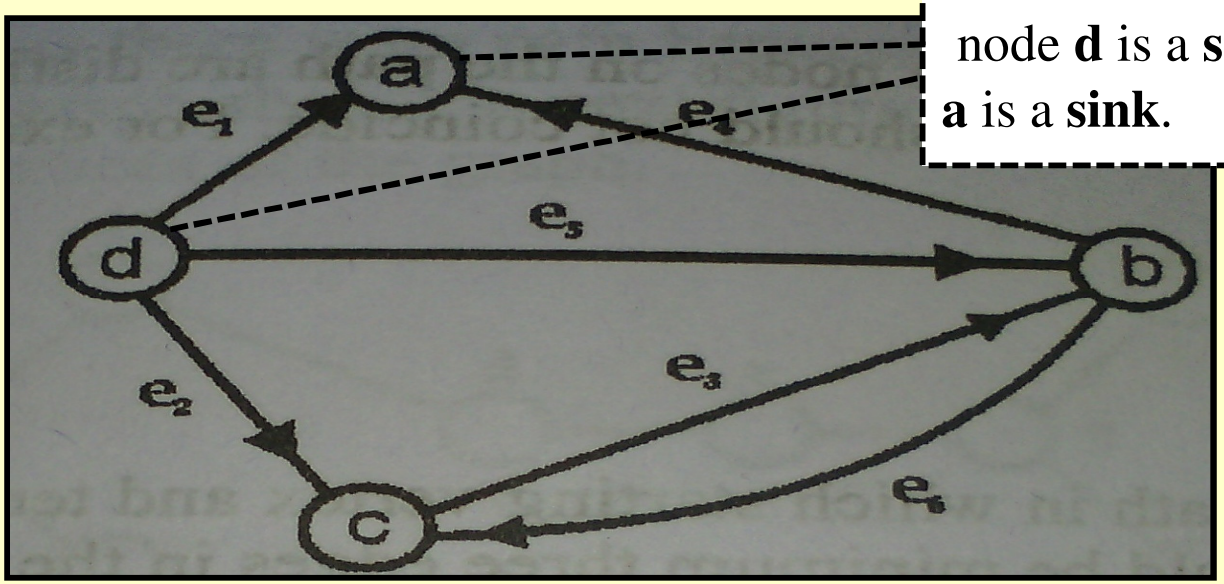
Graph Terminology(continued)

- **Source and Sink**

  A vertex **v** is known as **source** if its outdegree is one or more than one but its indegree is zero.

  A vertex **v** is known as **sink** if its indegree is one or more than one but its outdegree is zero.

  The following figure represents source and sink

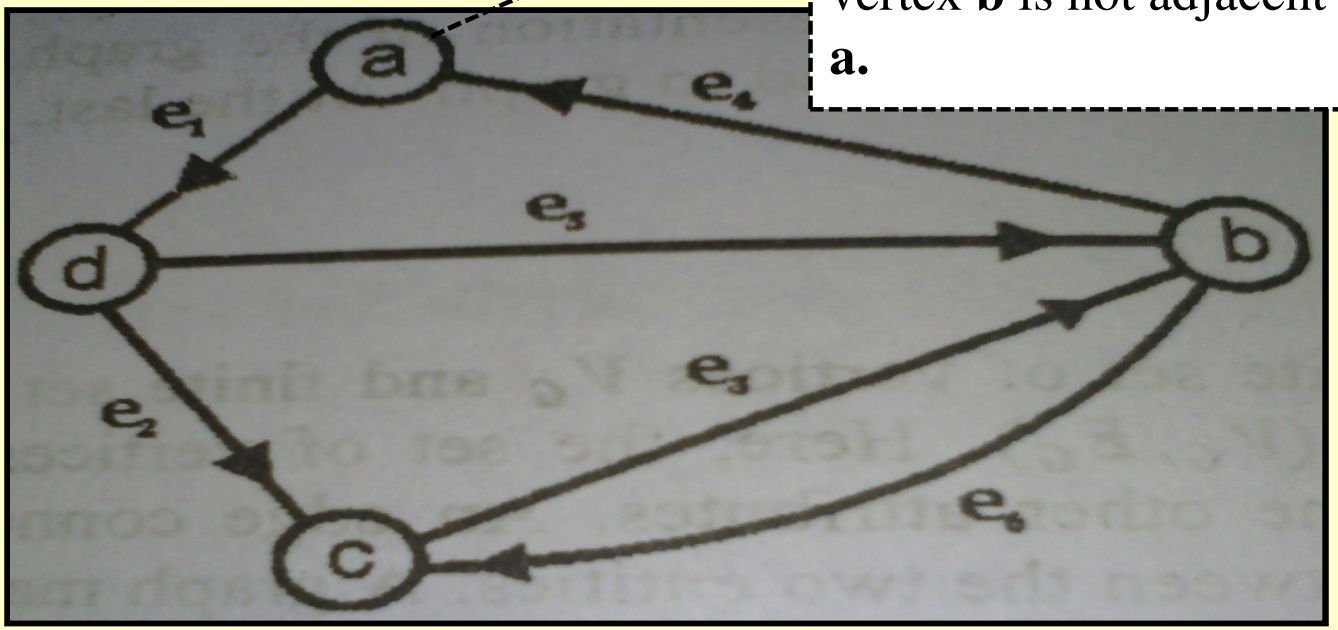


node **d** is a **source** and node **a** is a **sink**.

Graph Terminology(continued)

- **Adjacent Vertices**
  Two vertices are said to be adjacent if there is a direct edge between them. In a graph **G,** the vertex $\mathbf{v_i}$ said to be adjacent to vertex $\mathbf{v_j}$ if there is an edge between $\mathbf{v_j}$ and $\mathbf{v_i}$.

Vertex **a** is adjacent to vertex **b** but vertex **b** is not adjacent to vertex **a.**

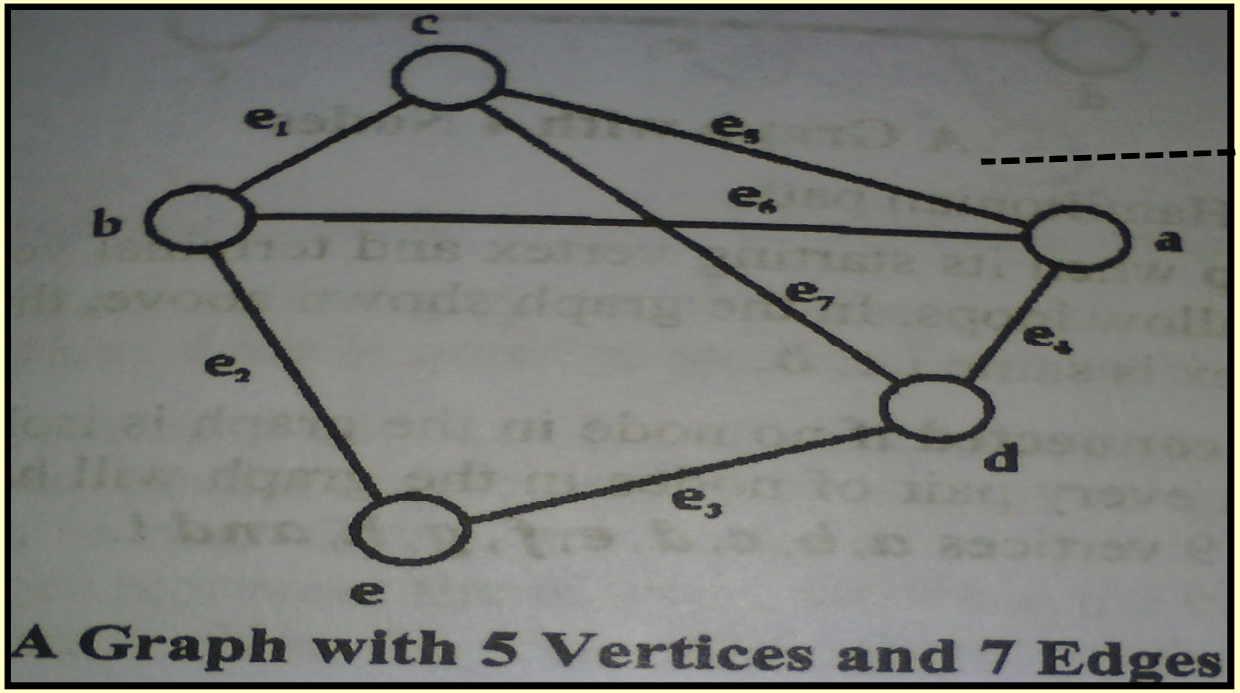


Graph Terminology(continued)

- **Path**

  In a graph, a path from vertex $\mathbf{v_i}$ to $\mathbf{v_j}$ is a sequence of vertices each adjacent to the next. Length of such a path is the number of edges in the path.

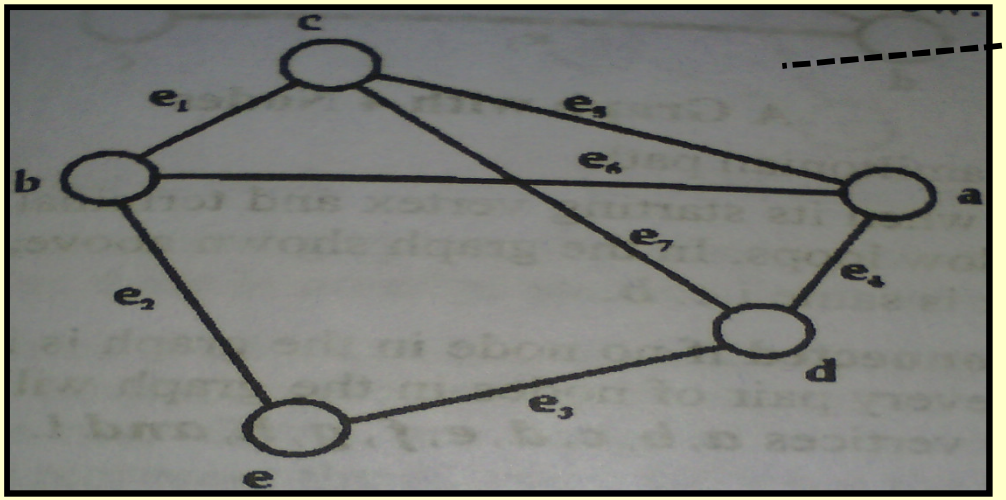
  A path with **n** length will have **n+1** vertices.



In graph:
a b e is a path
a c d e is a path
a c b e is a path
a e d c is not a path as **a** and **e** are not adjacent.
a b d is not a path as **b** and **d** are not adjacent.

Graph Terminology(continued)

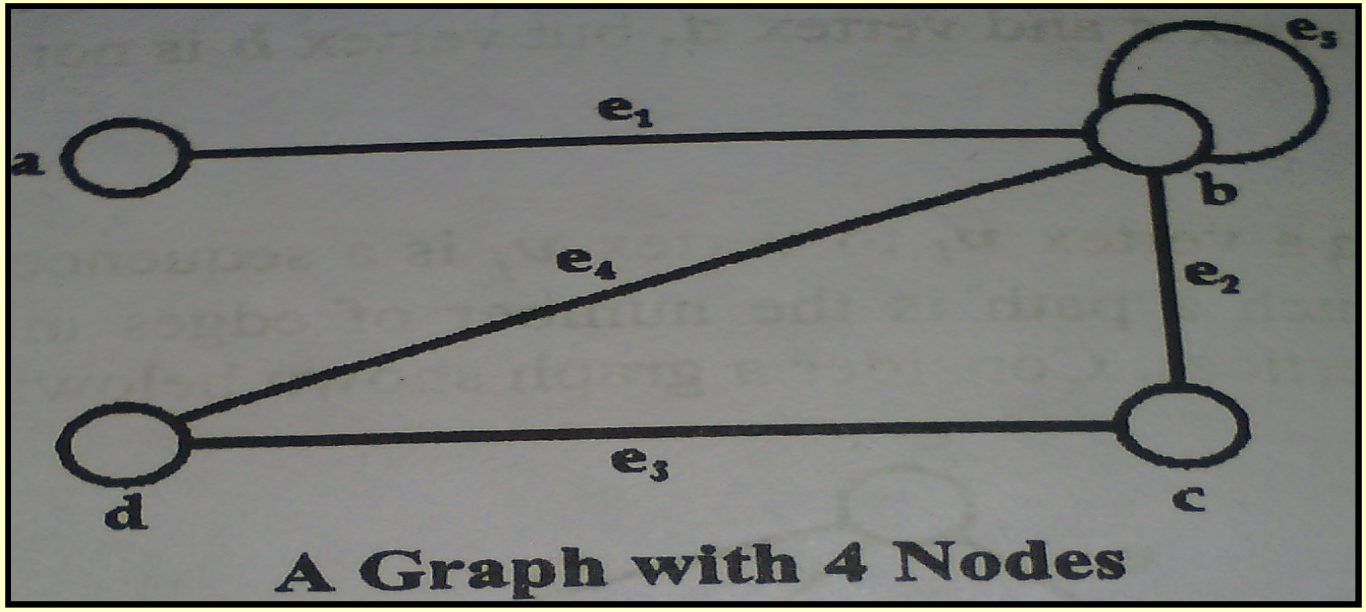
1. A path is said to be **simple** if all the nodes on the path are distinct with the condition that starting vertex and terminal vertex should not coincide.
2. A **closed path** is a simple path in which starting vertex and terminal vertex are the same with the condition that there should be minimum three edges in the path.
   A closed path is also known as **cycle.**



In the graph
a b e d c is a simple path
a d e b is a simple path
a b e is a simple path
a b c a is a cycle.
a b e d c a is a cycle
a c d a is a cycle

Graph Terminology(continued)

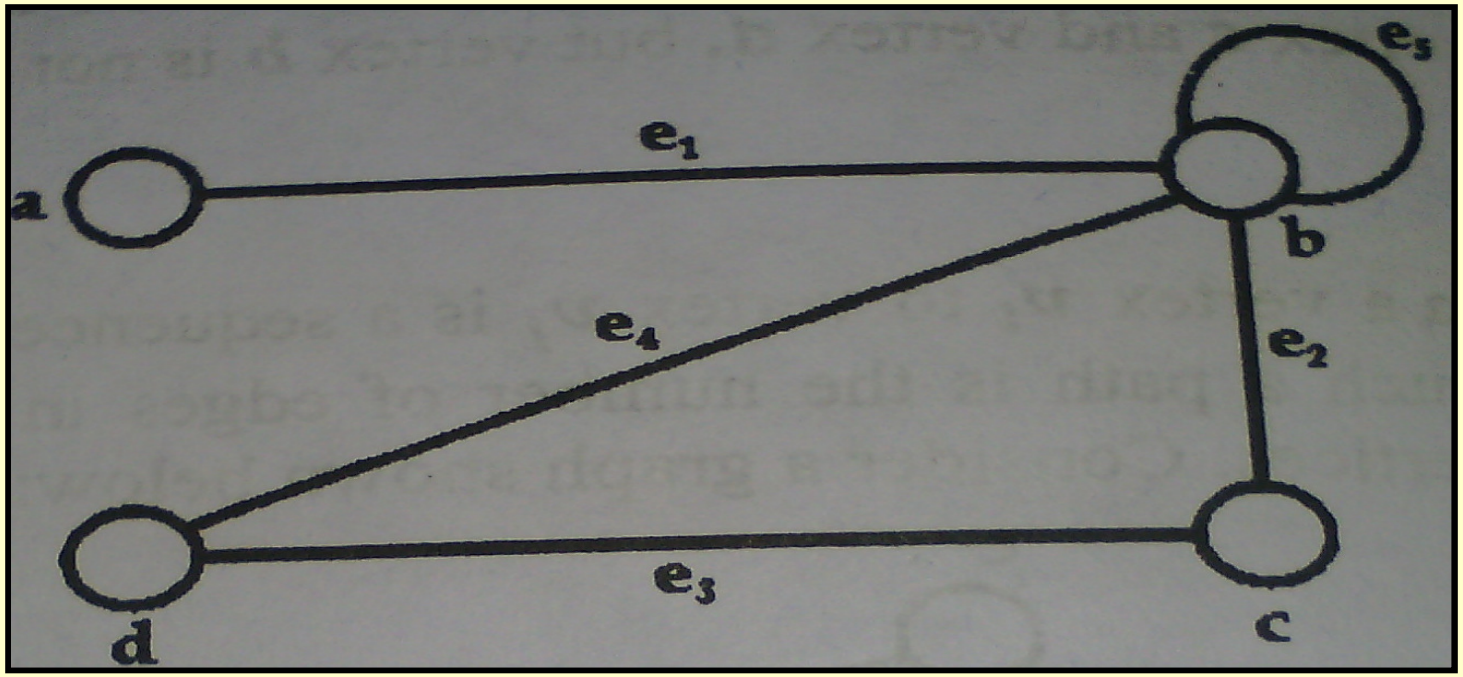
3. A path is said to be a **Hamiltonian path** if it contains all the vertices in the graph.
The following figure represents a Hamiltonian path:



a→ b→ c→ d is a Hamiltonian path.

Graph Terminology(continued)

- An **edge** is called a **loop** when its starting vertex and terminal vertex are the same.
  A simple graph usually does not allow loops.
  In the following graph, the edge $\mathbf{e_5}$ is a loop as its initial and terminal vertex is same i.e. **b.**

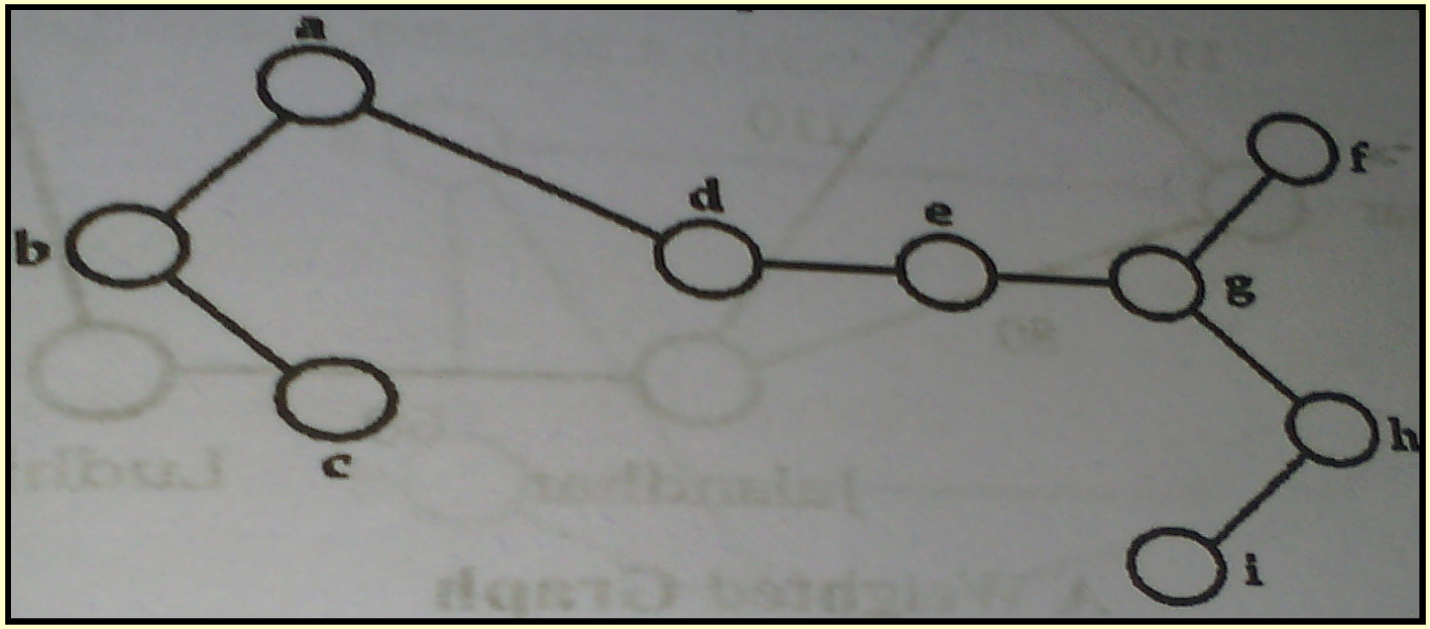


Graph Terminology(continued)

- **Connected Graph**

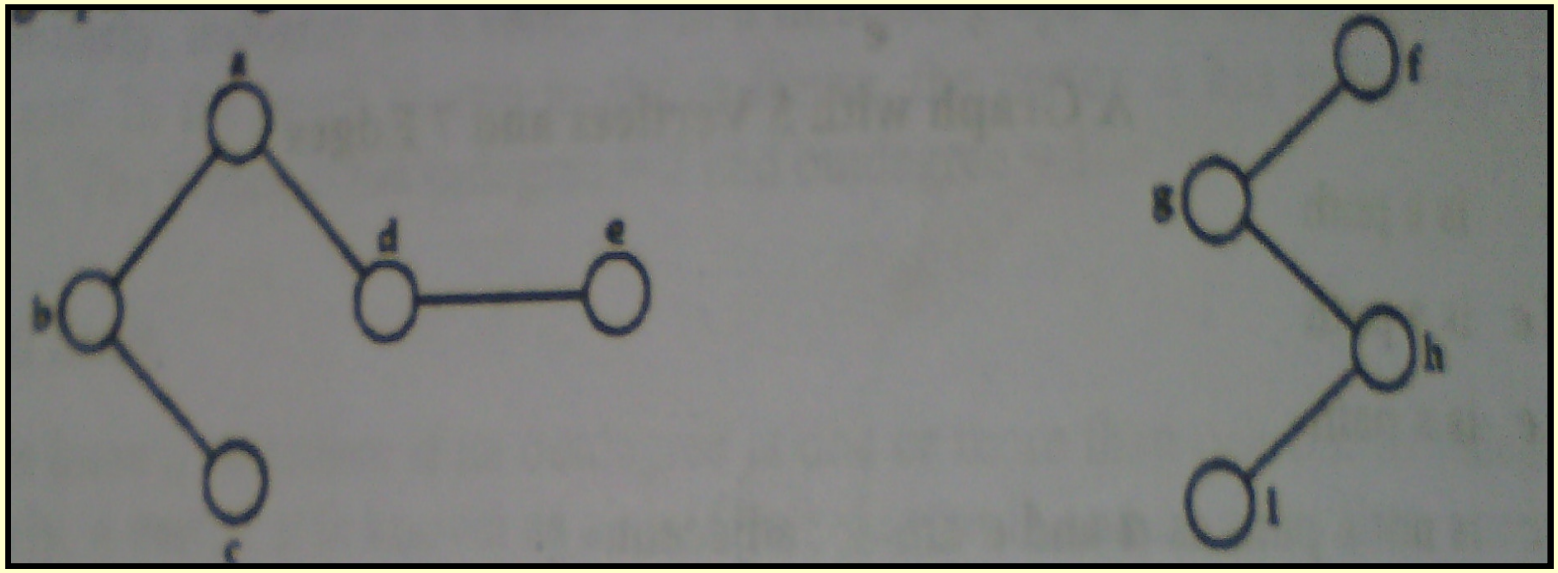
  A graph **G** is said to be **connected** if no node in the graph is isolated.

  If the graph is undirected then every pair of nodes in the graph will have a path. Consider a graph having 9 vertices **a, b, c, d, e, f, g, h and i.**



Graph Terminology(continued)

- **Unconnected Graph**
  A graph **G** is said to be unconnected if there is no path between any of the vertices.
  In the given graph there is no path between the vertices **f, g, h, i and** the vertices **a, b, c, d, e.** thus it is unconnected graph.

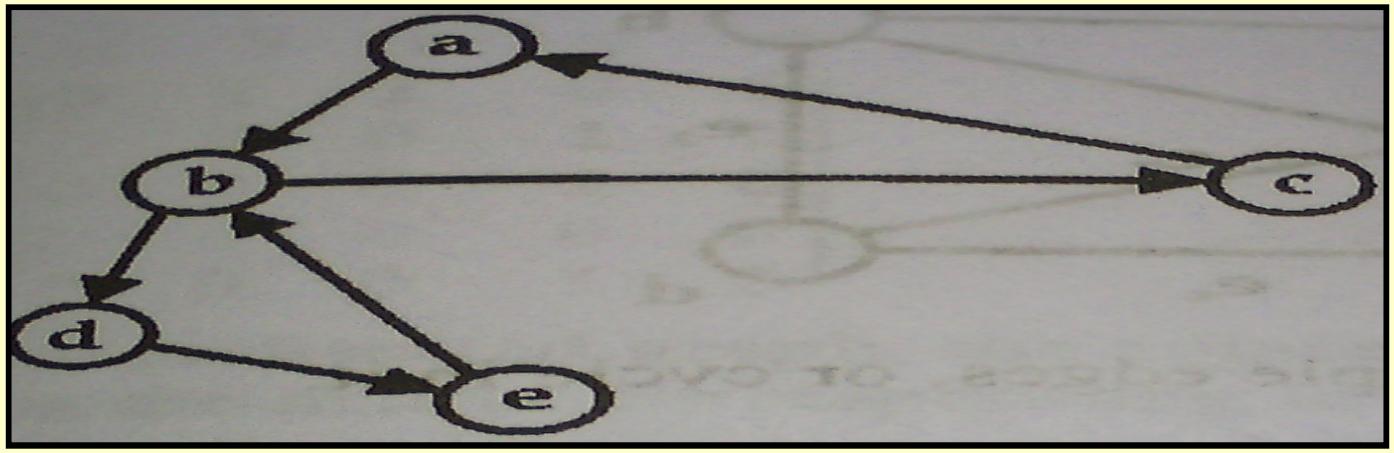


Graph Terminology(continued)

- **Strongly Connected Graph**

  A directed graph **G** is called **strongly connected,** if there exists a path between each pair of its vertices. For example, if there is a path between the vertices $\mathbf{v_i}$ and $\mathbf{v_j}$, then there must also be a path between $\mathbf{v_j}$ and $\mathbf{v_i}$.

  In a strongly connected graph there, there must not be any source or sink.

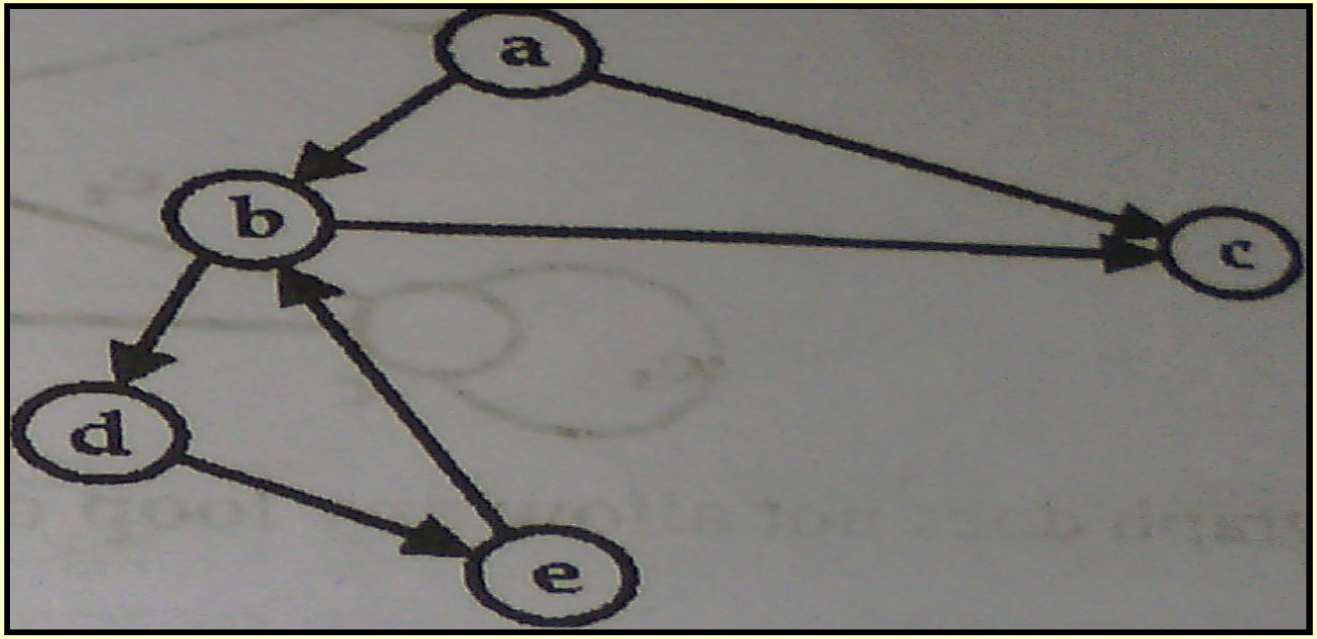


Graph Terminology(continued)

- **Weakly Connected Graph**

  A directed graph **G** is called **weakly connected,** if there does not exists a path between each pair of its vertices.

  The following graph is a weakly connected graph as there is no edge starting from vertex **c.**



Graph Terminology(continued)

- **Weighted Graph**

  A graph **G** is said to be weighted if edges in it are assigned with weights.

  This is often desired to represent certain physical attributes/properties by means of graph.

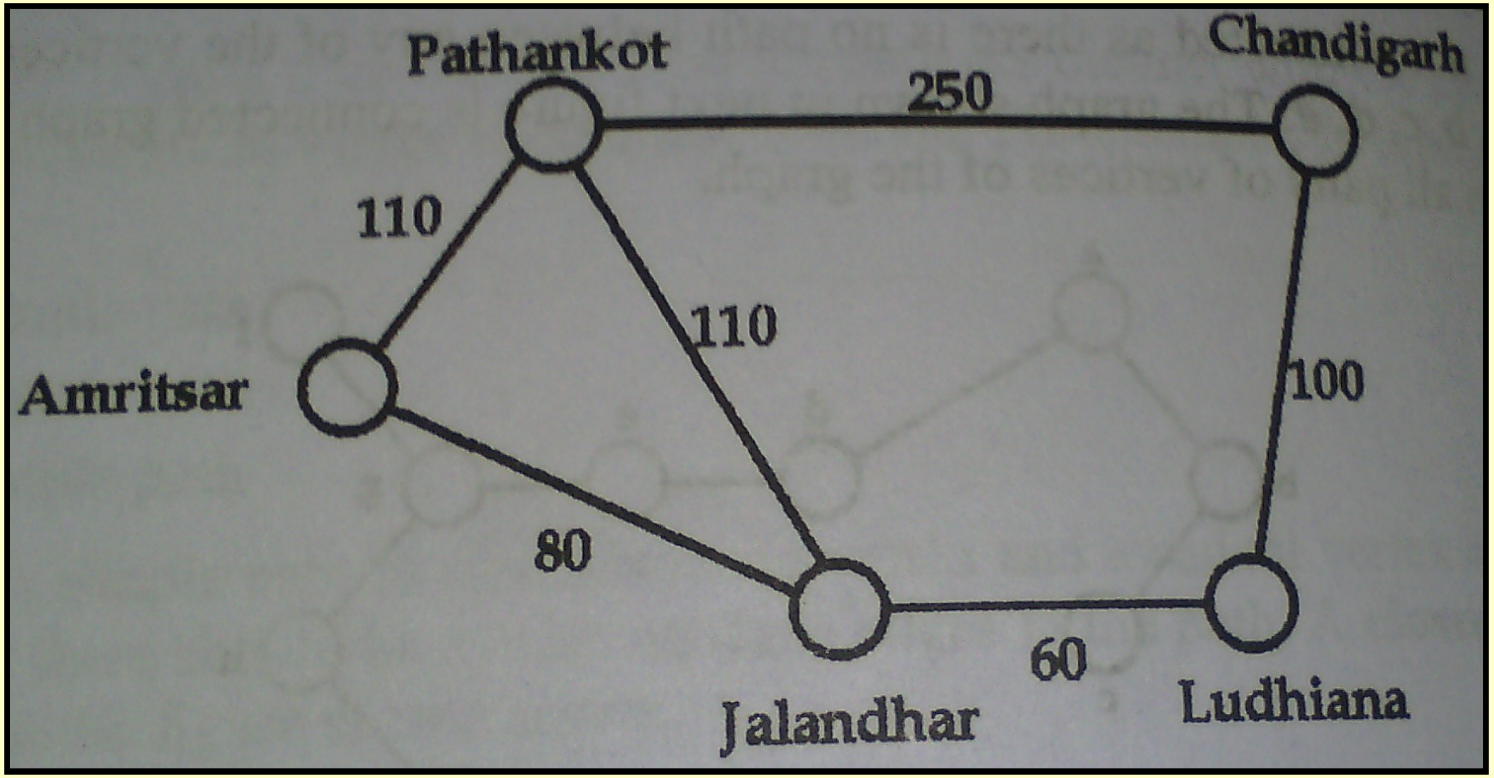
  In case of weighted graph, an edge is represented as:

  e={v_1, v_2, w} , where **v_1, v_2** are the vertices making an edge and **w** is the weight of the edge.

  For example, the edges of the graph in the figure are assigned weights

Graph Terminology(continued)

In the graph, the weights assigned to the edges represents the distance between cities, where the vertices represent different cities.



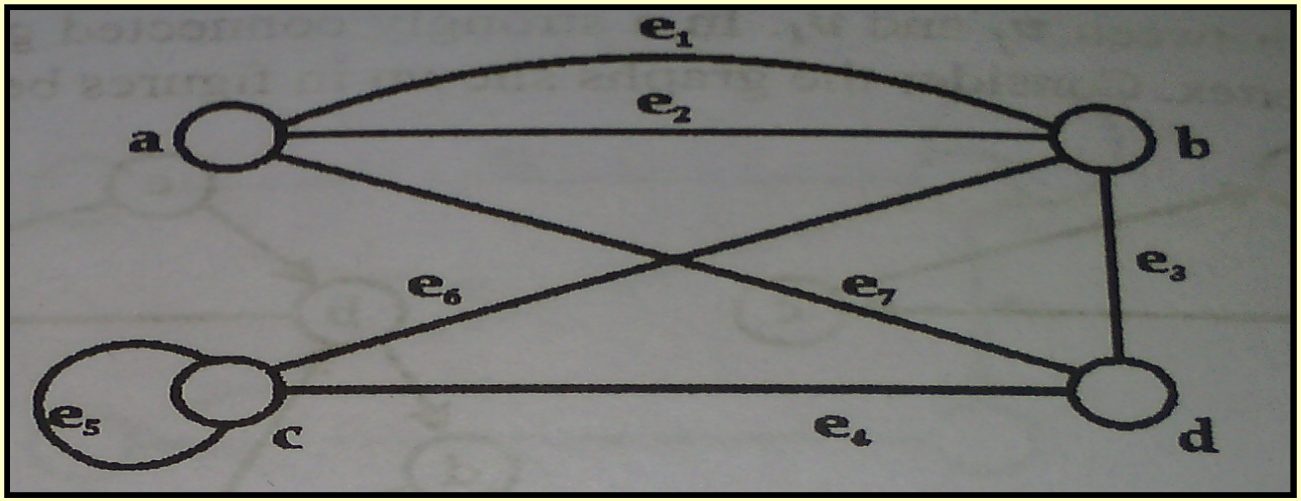
Graph Terminology(continued)

- **Multigraph**

  A graph **G** is said to be multigraph if it contains multiple edges or loop in it.

  For example, the graph below is a multigraph as it contains a loop at vertex **c** and multiple edges between the vertices **a** and **b.**

  A simple graph does not allow any loop or multiple edges, or cycle in it.



Memory Representation of Graph

Graph can be represented into the computer memory using various ways.
The two standard approaches for representing graph are:

1. Sequential Representation by using the Adjacency Matrix
2. Linked List Representation

Memory Representation of Graph(contd…)

1. Adjacency Matrix Representation of Graph

- Suppose **G = {V_g, E_g}** is a directed graph having **n** nodes. Suppose, vertices are ordered by using **v_1, v_2, v_3, v_4, …, …,…v_n** . Then adjacency matrix **A** for the graph **G** will be a square matrix of order **n** such that:

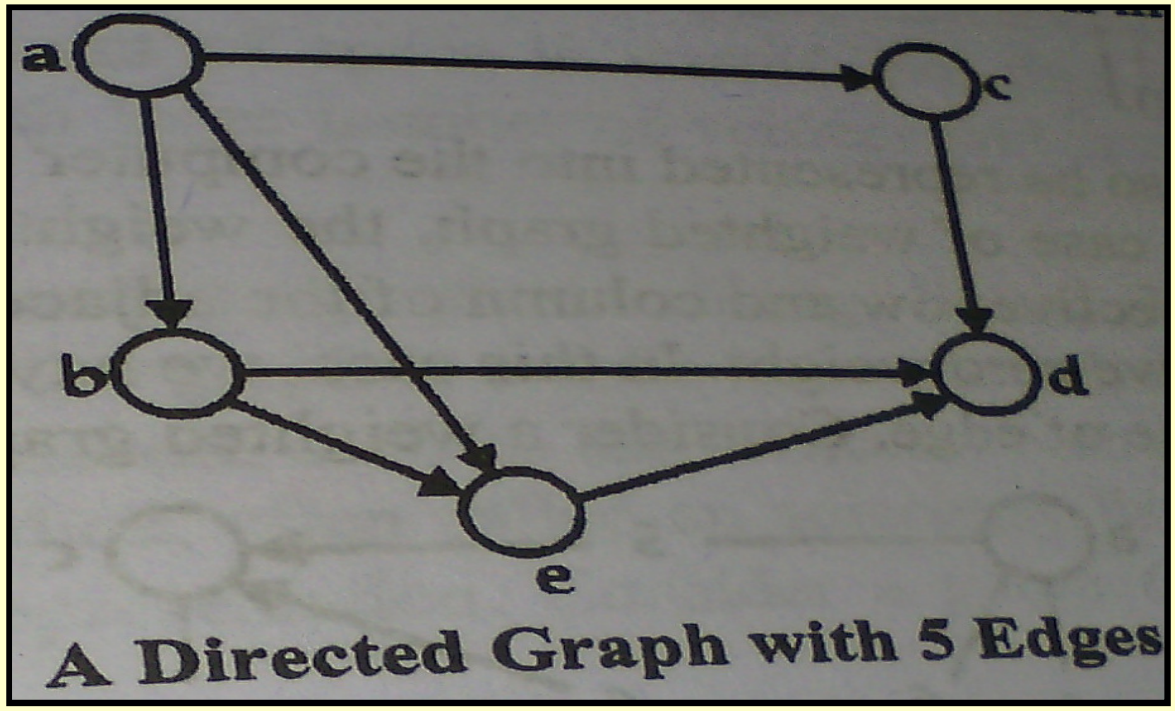
$$aij = \begin{cases} 1 & \text{if an edge lies between the vertices } v_i \text{ and } v_j \\ 0 & \text{if there is no edge between the vertices } v_i \text{ and } v_j \end{cases}$$

The adjacency matrix of a graph depends upon the ordering of its vertices.

If we change the order of vertices in a graph then it will result in a different adjacency matrix.

Memory Representation of Graph(contd…)

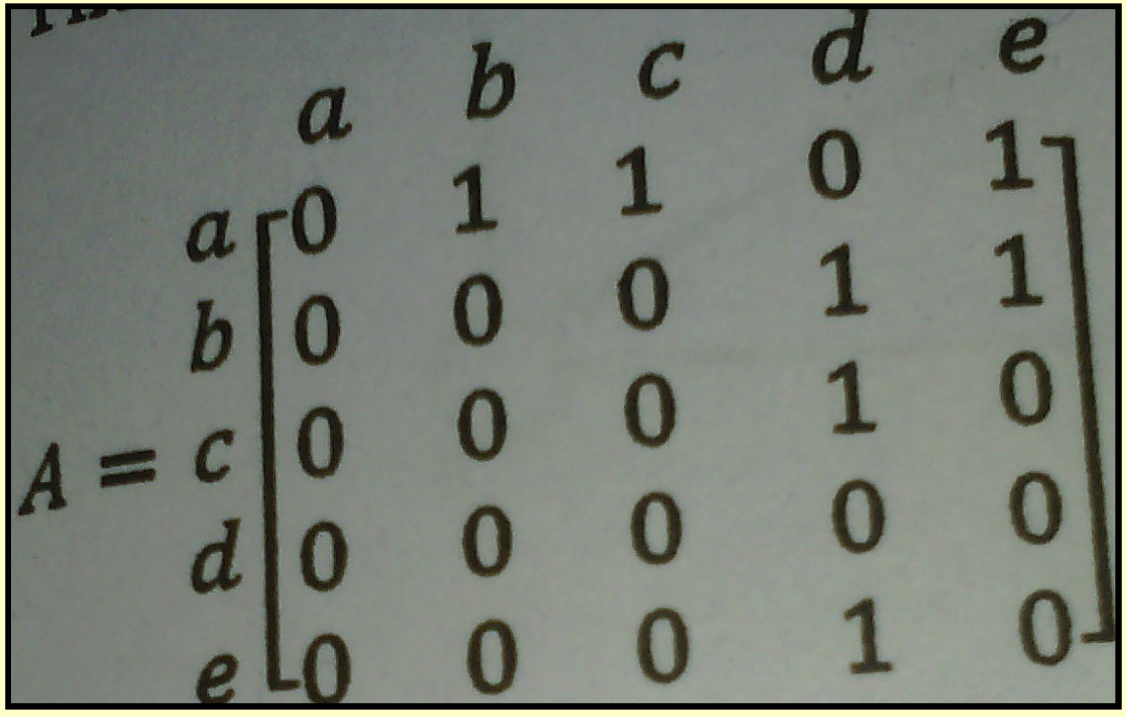
Adjacency matrix formed after changing the order of vertices will be closely related to the preceding one.
Consider, a directed graph **G** having vertices
$\mathbf{V_G = \{a, b, c, d, e\}}$



A Directed Graph with 5 Edges

Memory Representation of Graph(contd…)

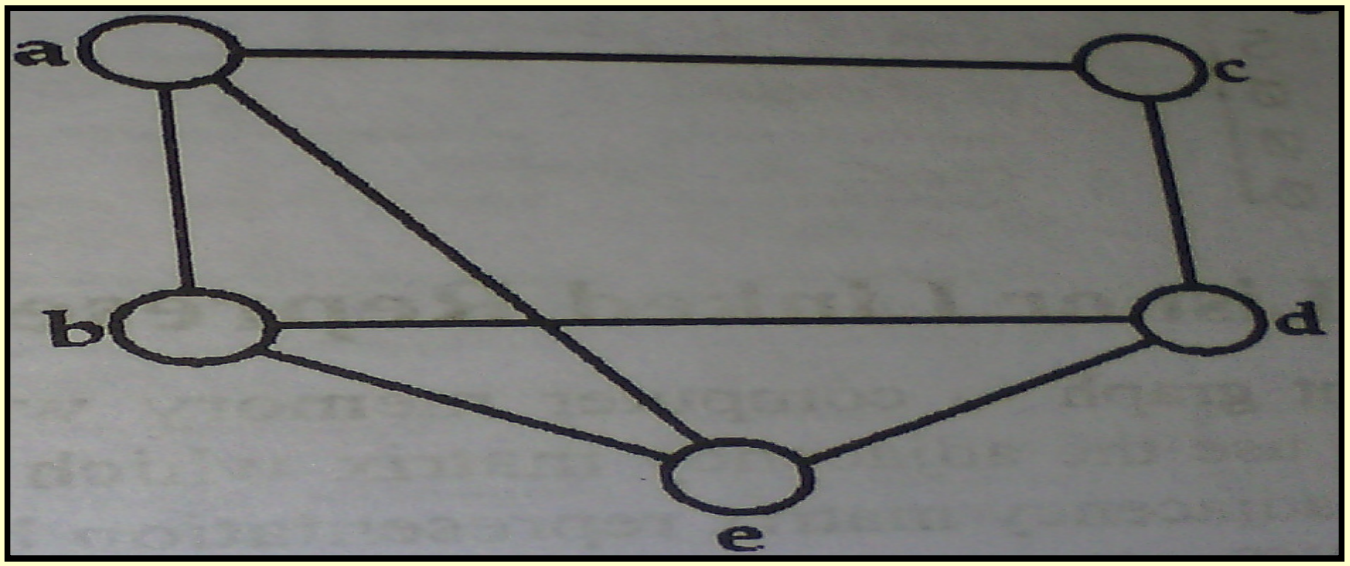
The adjacency matrix corresponding to this ordering sequence will be:



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Memory Representation of Graph(contd…)

- Incase of undirected graph, the adjacency matrix will be **symmetric,** as there will be two entries in the matrix corresponding to each edge in the graph.
  To represent an undirected graph using the adjacency matrix, it is sufficient to store either the upper triangular or the lower triangular matrix.
  Consider an undirected graph:



Memory Representation of Graph(contd…)

The vertices of the above undirected graph, $\mathbf{V_G = \{a, b, c, d, e\}}$.

The adjacency matrix for the above graph will be:

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Memory Representation of Graph(contd…)

- A **weighted graph** can also be represented into the computer memory using the adjacency matrix representation.

  In case of weighted graph, the weight of each edge (v_i, v_j) in the graph is stored in the respective row and column of the adjacency matrix.

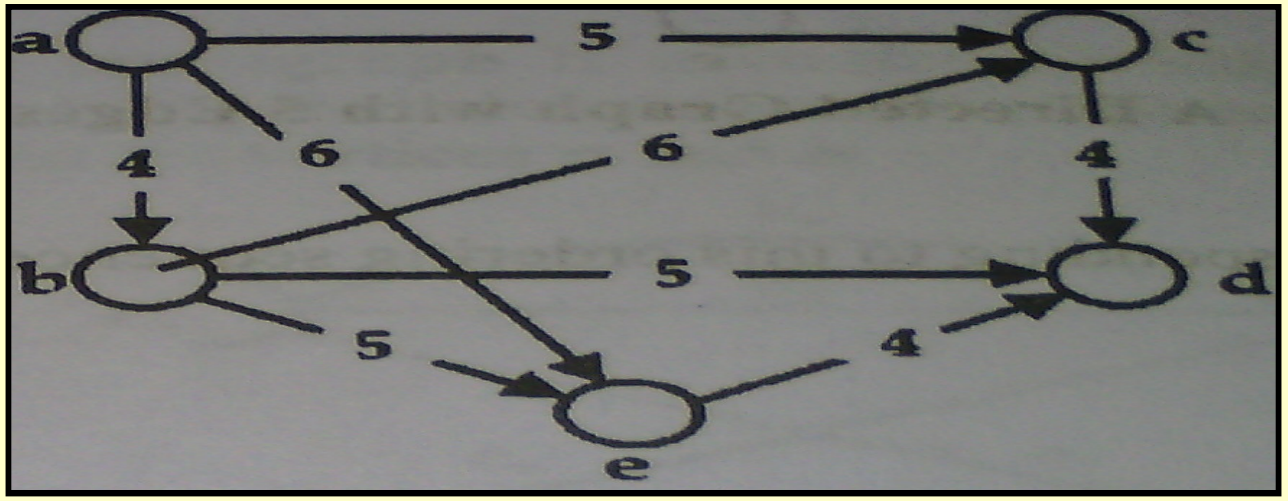
  In the weighted graph, an edge can also have zero weight. In this case, we have to use some sentinel value for representing the absence of edge.

  Consider a weighted graph **G** having vertices

  $V_G = \{a, b, c, d, e\}$.

  The weighted graph and corresponding adjacency matrix is as below:

Memory Representation of Graph(contd…)



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} \emptyset & 4 & 5 & \emptyset & 6 \\ \emptyset & \emptyset & 6 & 5 & 5 \\ \emptyset & \emptyset & \emptyset & 4 & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & 4 & \emptyset \end{bmatrix} \end{array}$$

Memory Representation of Graph(contd…)

The adjacency matrix representation has a number of drawbacks.

- It is difficult to store additional information about the vertices and edges in the graph
- One major problem in this representation is its static nature. Before storing any graph, it is necessary to find the number of vertices in it.
- It is very difficult to insert and delete the vertices from the graph in adjacency matrix representation, as it requires change in the dimensions of the matrix.
- In adjacency matrix representation, the space for each possible edge in the graph is reserved.

Memory Representation of Graph(contd…)

- For example, Consider the case of a graph **G** with **n** vertices and a small number of edges as compared to the $\mathbf{n^2}$ entries in its adjacency matrix **A,** and then the matrix **A** will definitely be sparse. Hence, a huge amount of space is wasted.
- So, this representation is very inefficient for large graphs with large number of vertices which are connected by very small number of edges.

Memory Representation of Graph(contd…)

2. Adjacency List or Linked Representation of Graph

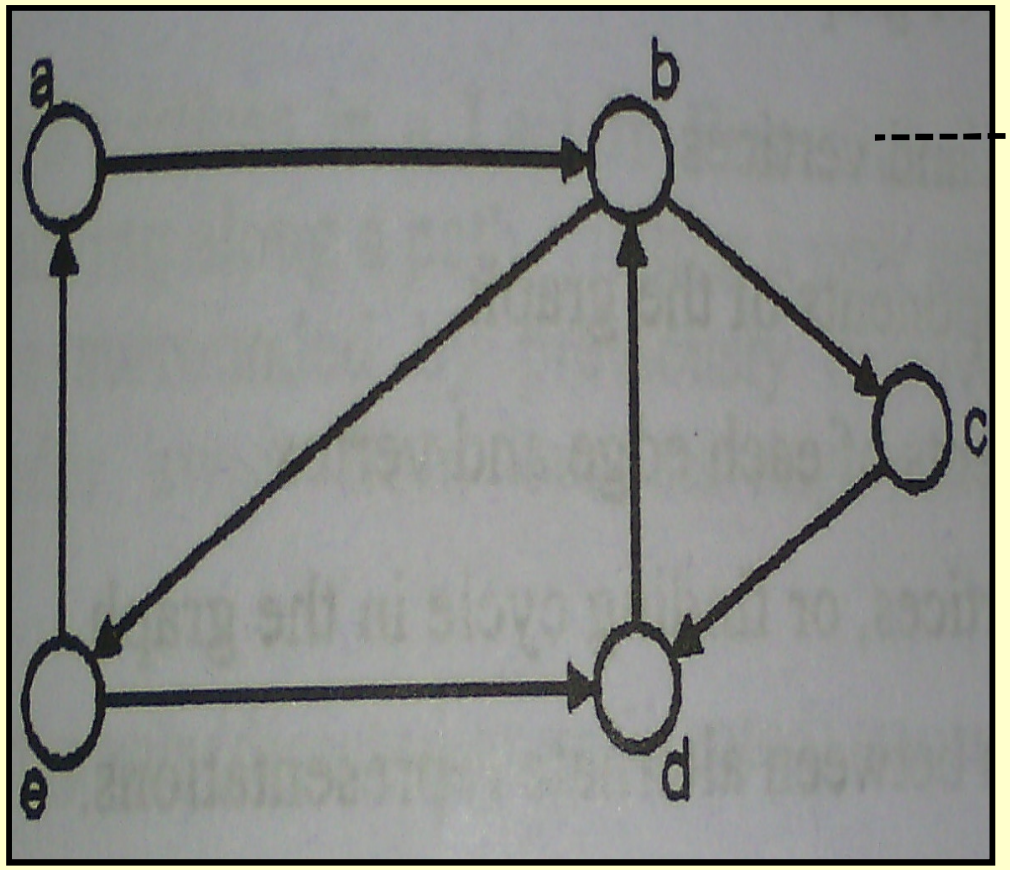
In this representation, each vertex in the graph is a node in a master linked list structure.

Another Linked list starts from each vertex node and denotes the vertices which are directly adjacent to a given source vertex.

This method, often called an adjacency list, is more space efficient than the adjacency matrix representation.

Consider a graph **G** having vertices $\mathbf{V_G = \{a, b, c, d, e\}}$ as in the following figure:

Memory Representation of Graph(contd…)



Adjacency List of Nodes

a ⟶ b
b ⟶ c, e
c ⟶ d
d ⟶ b
e ⟶ a, d

Memory Representation of Graph(contd…)

Adjacency List Representation of Graph shown above is as below: