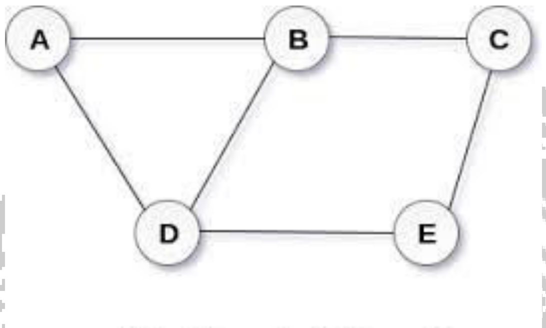
# GRAPHS

## DEFINITION

A graph G is defined as an ordered set (V, E), where V(G) represents the set of vertices and E(G) represents the edges that connect these vertices.



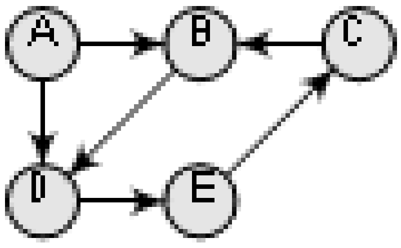
Undirected Graph

In the above graph, V(G) = {A, B, C, D and E} and E(G) = {(A, B), (B, C), (A, D), (B, D), (D, E), (C,E)}.

There are five vertices or nodes and six edges in the graph.

## DIRECTED AND UNDIRECTED GRAPHS

- In an undirected graph, edges do not have any direction associated with them. That is, if an edge is drawn between nodes A and B, then the nodes can be traversed from A to B as well as from B to A.
- In a directed graph, edges form an ordered pair. If there is an edge from A to B, then there is a path from A to B but not from B to A. The edge (A, B) is said to initiate from node A (also known as initial node) and terminate at node B (terminal node).



Directed Graphs

## GRAPH TERMINOLOGY

### Adjacent nodes or neighbours

For every edge, e = (u, v) that connects nodes u and v, the nodes u and v are the end- points and are said to be the adjacent nodes or neighbours.

### Degree of a node

Degree of a node u, deg(u), is the total number of edges containing the node u. If deg(u) = 0, it means that u does not belong to any edge and such a node is known as an isolated node.

### Closed path

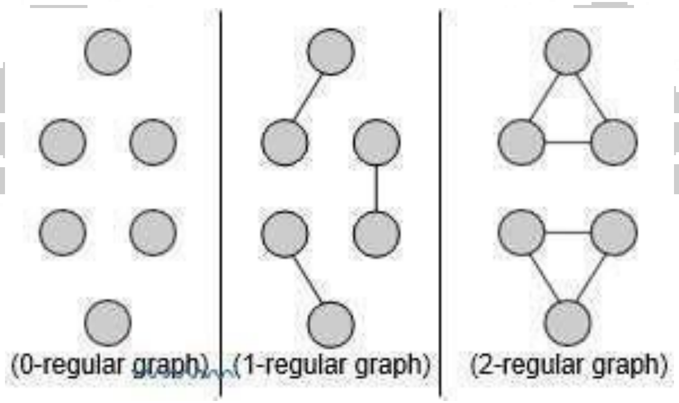
A path P is known as a closed path if the edge has the same end-points. That is, if vo=vn.

### Simple path

A path P is known as a simple path if all the nodes in the path are distinct with an exception that v0 may be equal to vn. If v0= vn then the path is called a closed simple path.

### Cycle

A path in which the first and the last vertices are same. A simple cycle has no repeated edges or vertices (except the first and last vertices).



**Regular Graph**

## Types of Graphs

### Connected graph

A graph is said to be connected if for any two vertices (u, v) in V there is a path from u to v. That is to say that there are no isolated nodes in a connected graph. A connected graph that does not have any cycle is called a tree.

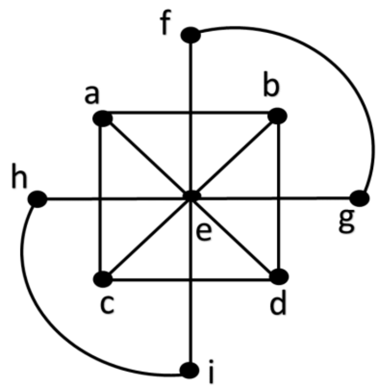


**Fig: Connected graph**

### Complete graph

A graph G is said to be complete if all its nodes are fully connected. That is, there is a path from one node to every other node in the graph. A complete graph has n(n-1)/2 edges, where n is the number of nodes in G.

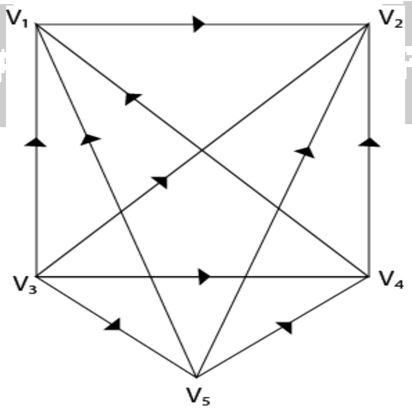


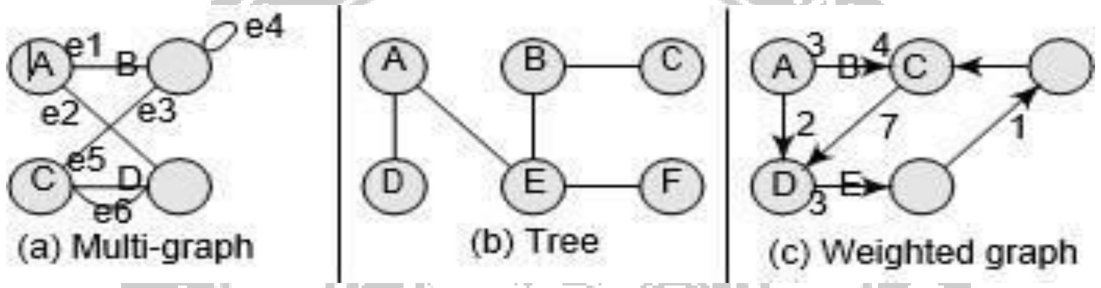
Fig: $K_5$

**Clique**

In a undirected graph G=(V, E), clique is a subset of the vertex, such that for every two vertices in C, there is an edge that connects two vertices.

**Labelled graph or weighted graph**

A graph is said to be labelled if every edge in the graph is assigned some data. In a weighted graph, the edges of the graph are assigned some weight or length. The weight of an edge denoted by w(e) is a positive value which indicates the cost of traversing the edge.



**Multiple edges**

Distinct edges which connect the same end-points are called multiple edges. That is, e = (u, v) and e' = (u, v) are known as multiple edges of G.

**Loop**

An edge that has identical end-points is called a loop. That is, e = (u, u).

**Multi-graph**

A graph with multiple edges and/or loops is called a multi-graph.

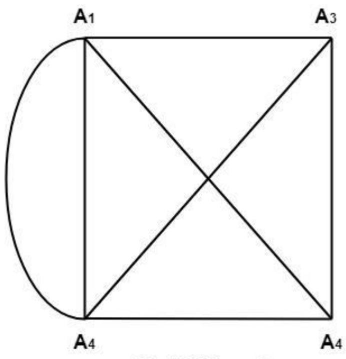


Fig:Multigraph

**Size of a graph**

The size of a graph is the total number of edges in it.

**DIRECTED GRAPHS**

A directed graph G, also known as a digraph, is a graph in which every edge has a direction assigned to it. An edge of a directed graph is given as an ordered pair (u, v) of nodes in G. For an edge (u, v),

- The edge begins at u and terminates at v.
- u is known as the origin or initial point of e. Correspondingly, v is known as the destination or terminal point of e.
- u is the predecessor of v. Correspondingly, v is the successor of u.
- Nodes u and v are adjacent to each other.

**Terminology of a Directed graph**

Out-degree of a node - The out-degree of a node u, written as outdeg(u), is the number of edges that originate at u.

In-degree of a node - The in-degree of a node u, written as indeg(u), is the number of edges that terminate at u.

Degree of a node - The degree of a node, written as deg(u), is equal to the sum of in-degree and out-degree of that node. Therefore, deg(u) = indeg(u) + outdeg(u).

Isolated vertex - A vertex with degree zero. Such a vertex is not an end-point of any edge.

Pendant vertex (also known as leaf vertex) - A vertex with degree one.

Cut vertex - A vertex which when deleted would disconnect the remaining graph.

Source - A node u is known as a source if it has a positive out-degree but a zero in-degree.

Sink - A node u is known as a sink if it has a positive in-degree but a zero out-degree.

Reachability -A node v is said to be reachable from node u, if and only if there exists a (directed) path from node u to node v.

**Strongly connected directed graph**

A digraph is said to be strongly connected if and only if there exists a path between every pair of nodes in G. That is, if there is a path from node u to v, then there must be a path from node v to u.

**Unilaterally connected graph**

A digraph is said to be unilaterally connected if there exists a path between any pair of nodes u, v in G such that there is a path from u to v or a path from v to u, but not both.

**Weakly connected digraph**

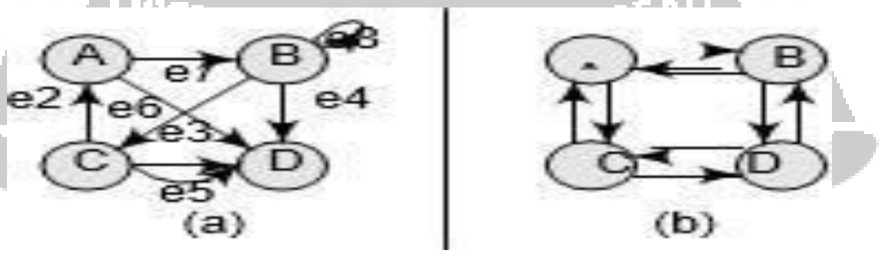
A directed graph is said to be weakly connected if it is connected by ignoring the direction of edges. That is, in such a graph, it is possible to reach any node from any other node by traversing edges in any direction (may not be in the direction they point). The nodes in a weakly connected directed graph must have either out-degree or in-degree of at least 1.

**Parallel/Multiple edges**

Distinct edges which connect the same end-points are called multiple edges. That is, e = (u, v) and e' = (u, v) are known as multiple edges of G.

**Simple directed graph**

A directed graph G is said to be a simple directed graph if and only if it has no parallel edges



a) Directed acyclic graph and b) Strongly connected directed graph

**REPRESENTATION OF GRAPH**

There are three common ways of storing graphs in the computer's memory. They are:

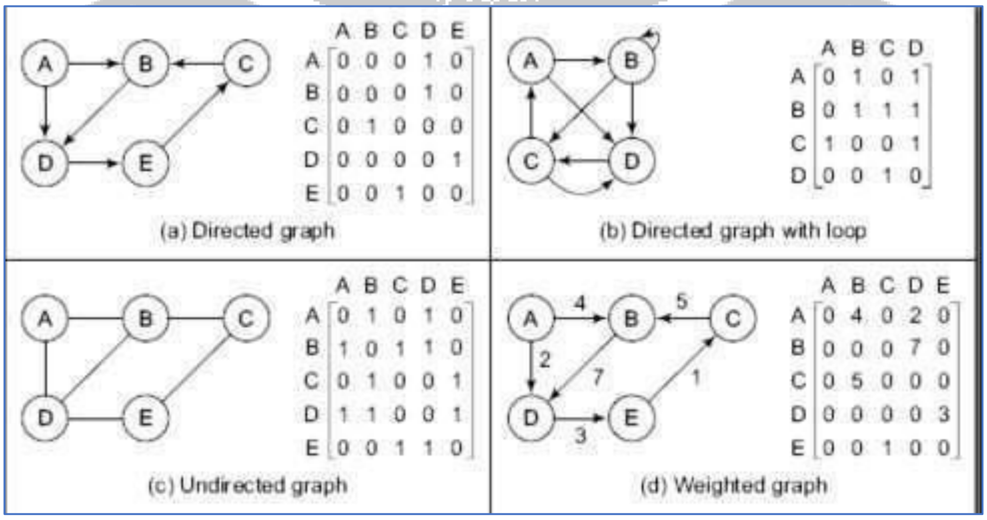
- Sequential representation by using an adjacency matrix.
- Linked representation by using an adjacency list that stores the neighbours of a node using a linked list.
- Adjacency multi-list which is an extension of linked representation

**Adjacency matrix Representation**

An adjacency matrix is used to represent which nodes are adjacent to one another. By definition, two nodes are said to be adjacent if there is an edge connecting them. In a directed graph G, if node v is adjacent to node u, then there is definitely an edge from u to v. That is, if v is adjacent to u, we can get from u to v by traversing one edge. For any graph G having n nodes, the adjacency matrix will have the dimension of n ¥ n. In an adjacency matrix, the rows and columns are labelled by graph vertices. An entry aij in the adjacency matrix will contain 1, if vertices vi and vj are adjacent to each other. However, if the nodes are not adjacent, aij will be set to zero

$$a_{ij} \begin{cases} 1 & \text{[if } v_i \text{ is adjacent to } v_j\text{, that is there is an edge } (v_i, v_j)\text{]A} \\ 0 & \text{[otherwise]} \end{cases}$$

Since an adjacency matrix contains only 0s and 1s, it is called a bit matrix or a Boolean matrix. The entries in the matrix depend on the ordering of the nodes in G. therefore, a change in the order of nodes will result in a different adjacency matrix.



$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 0 & 0 & 1 & 0 \\ B & 0 & 0 & 0 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 0 & 0 \end{array}$$

(a) Directed graph

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 0 & 1 \\ B & 0 & 1 & 1 & 1 \\ C & 1 & 0 & 0 & 1 \\ D & 0 & 0 & 1 & 0 \end{array}$$

(b) Directed graph with loop

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 1 & 0 & 1 & 0 \\ B & 1 & 0 & 1 & 1 & 0 \\ C & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 1 & 0 & 0 & 1 \\ E & 0 & 0 & 1 & 1 & 0 \end{array}$$

(c) Undirected graph

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 4 & 0 & 2 & 0 \\ B & 0 & 0 & 0 & 7 & 0 \\ C & 0 & 5 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 3 \\ E & 0 & 0 & 1 & 0 & 0 \end{array}$$

(d) Weighted graph

From the above examples, we can draw the following conclusions:

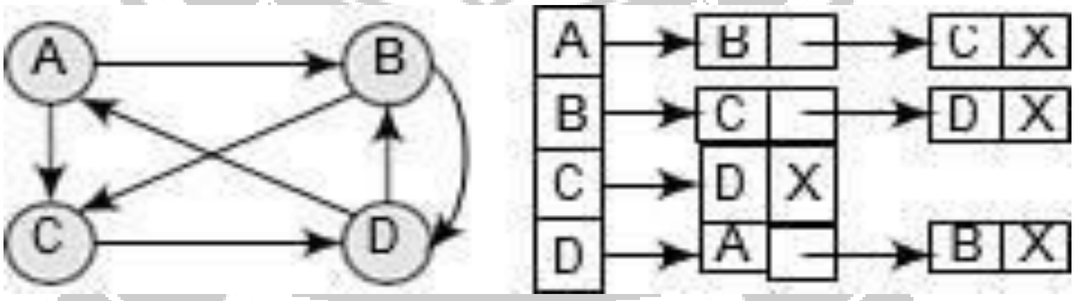
- For a simple graph (that has no loops), the adjacency matrix has 0s on the diagonal.

- The adjacency matrix of an undirected graph is symmetric.
- The memory use of an adjacency matrix is O(n2), where n is the number of nodes in the graph.
- Number of 1s (or non-zero entries) in an adjacency matrix is equal to the number of edges in the graph.
- The adjacency matrix for a weighted graph contains the weights of the edges connecting the nodes.

**ADJACENCY LIST REPRESENTATION**

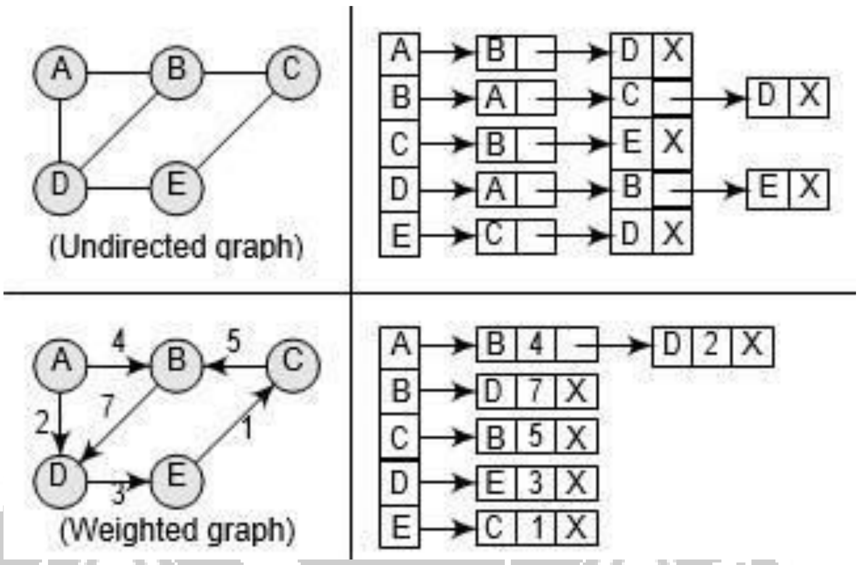
An adjacency list is another way in which graphs can be represented in the computer's memory. This structure consists of a list of all nodes in G. Furthermore, every node is in turn linked to its own list that contains the names of all other nodes that are adjacent to it. The key advantages of using an adjacency list are:

- It is easy to follow and clearly shows the adjacent nodes of a particular node.
- It is often used for storing graphs that have a small-to-moderate number of edges. That is, an adjacency list is preferred for representing sparse graphs in the computer's memory; otherwise, an adjacency matrix is a good choice.
- Adding new nodes in G is easy and straightforward when G is represented using an adjacency list. Adding new nodes in an adjacency matrix is a difficult task, as the size of the matrix needs to be changed and existing nodes may have to be reordered.



Graph G and its Adjacency List

Adjacency list for an undirected graph and a weighted graph



## ADJACENCY MULTI-LIST REPRESENTATION

A multi-list representation basically consists of two parts—a directory of nodes' information and a set of linked lists storing information about edges. While there is a single entry for each node in the node directory, every node, on the other hand, appears in two adjacency lists (one for the node at each end of the edge). For example, the directory entry for node i points to the adjacency list for node i.

In a multi-list representation, the information about an edge (vi , vj ) of an undirected graph can be stored using the following attributes:

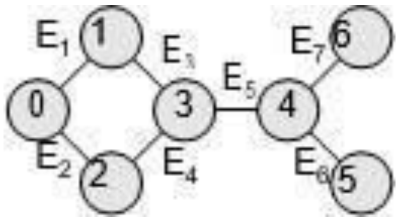
M: A single bit field to indicate whether the edge has been examined or not.

Vj : A vertex in the graph that is connected to vertex vi by an edge.

Vi : A vertex in the graph that is connected to vertex vi by an edge.

Link i for vj : A link that points to another node that has an edge incident on v .

Link j for vi : A link that points to another node that has an edge incident on vi .



Undirected Graph

The adjacency multi-list for the graph can be given as:

| | | | | |
|---|---|---|---|---|
| Edge 1 | 0 | 1 | Edge 2 | Edge 3 |
| Edge 2 | 0 | 2 | NULL | Edge 4 |
| Edge 3 | 1 | 3 | NULL | Edge 4 |
| Edge 4 | 2 | 3 | NULL | Edge 5 |
| Edge 5 | 3 | 4 | NULL | Edge 6 |
| Edge 6 | 4 | 5 | Edge 7 | NULL |
| Edge 7 | 4 | 6 | NULL | NULL |

Using the adjacency multi-list given above, the adjacency list for vertices can be constructed as shown below:

| VERTEX | LIST OF EDGES |
|---|---|
| 0 | Edge 1, Edge 2 |
| 1 | Edge 1, Edge 3 |
| 2 | Edge 2, Edge 4 |
| 3 | Edge 3, Edge 4, Edge 5 |
| 4 | Edge 5, Edge 6, Edge 7 |
| 5 | Edge 6 |
| 6 | Edge 7 |