Module 2:

Graphs

CO3: Illustrate and justify the concepts of graphs in real life problem solving

Subject In-charge

Ms.Pratibha Rane

Assistant Professor

Class : III SEM SEIT-A & B

Module 2:

Graphs

- Introduction to Graphs: Undirected Graph, Directed Graph,
- graph terminology
- Connectivity in Undirected and Directed Graphs.
- Spanning tree.
- Representation of graph: adjacency matrix, adjacency list,
- Transitive closure of a directed graph and path matrix.
- Traversals: Breadth First Search, Depth First Search.

- **Self-learning Topics: Implementation of BFS, DFS**

- **Chapter 13 : from Reema Thareja**

What is a Graph?

- A graph is an **abstract data structure** that is used to implement the mathematical concept of graphs.
- It is basically a **collection of vertices (also called nodes) and edges** that connect these vertices.
- A graph is often viewed as a **generalization of the tree** structure, where instead of having a purely parent-to-child relationship between tree nodes, any kind of complex relationship can exist.

- **Why are Graphs used?**
  - To model situation where entities or things are related to each other in pairs
  - **Family trees** in which the member nodes have an edge from parent to each of their children.
  - **Transportation networks** in which nodes are airports, intersections, ports, etc. The edges can be airline flights, one-way roads, shipping routes, etc

Graph Definition

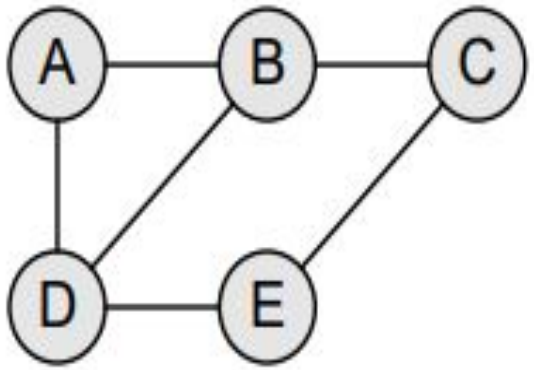


Figure 13.1 Undirected graph

- 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.

Figure 13.1 shows a graph with V(G) = {A, B, C, D and E} and E(G) = {(A, B), (B, C), (A, D), (B, D), (D, E), (C, E)}. Note that there are five vertices or nodes and six edges in the graph.

A graph can be directed or undirected

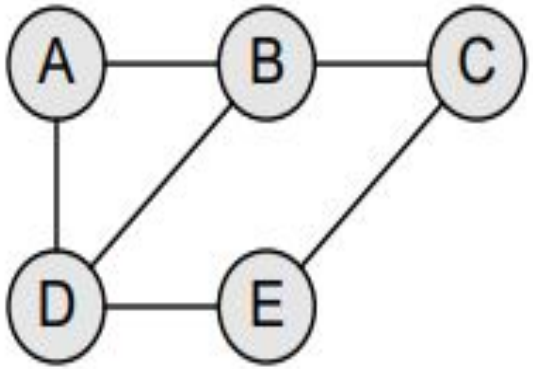


Figure 13.1 Undirected graph

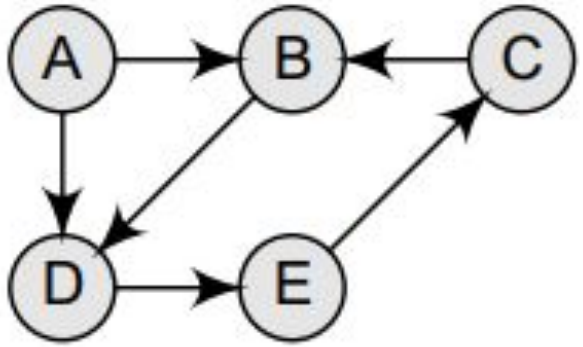


Figure 13.2 Directed graph

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.

- **Regular graph** It is a graph where each vertex has the same number of neighbours. That is, every node has the same degree. A regular graph with vertices of degree k is called a k–regular graph or a regular graph of degree k. Figure 13.3 shows regular graphs.

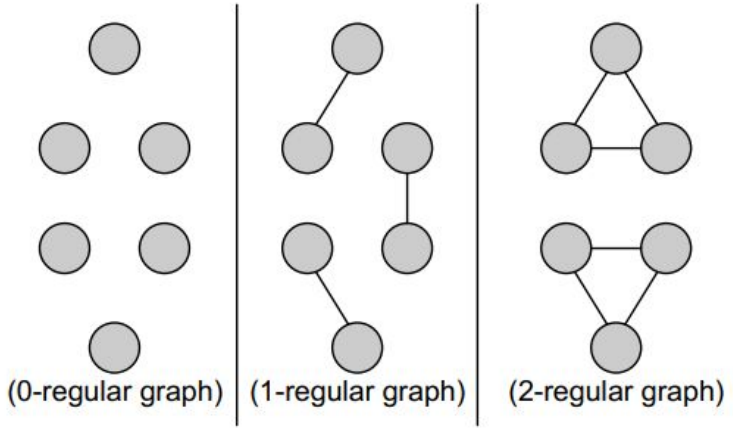


Figure 13.3 Regular graphs

Graph Terminology

- **Path** A path P written as P = {v0 , v1 , v2 , ..., vn }, of length n from a node u to v is defined as a sequence of (n+1) nodes. Here, u = v_0 , v = v_n and v_i–1 is adjacent to v_i for i = 1, 2, 3, ..., n.

- **Closed path** A path P is known as a closed path if the edge has the same end-points. That is, if $v_0 = v_n$.

- **Simple path** A path P is known as a simple path if all the nodes in the path are distinct with an exception that v_0 may be equal to v_n . If $v_0 = v_n$, 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).

Graph Terminology

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

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. Figure 13.4(c) shows a weighted graph

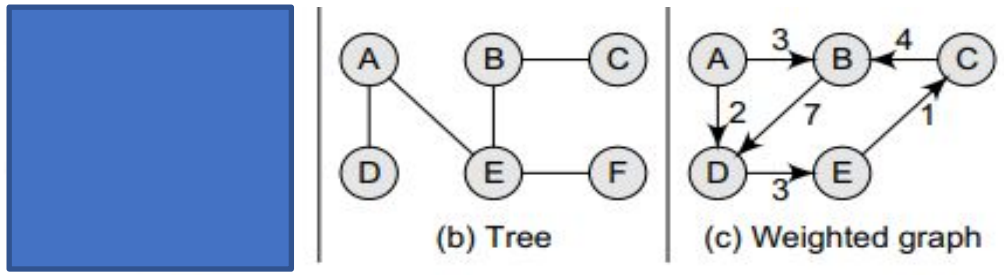


Figure 13.4 Multi-graph, tree, and weighted graph

Graph Terminology

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.

Figure 13.4(a) shows a multi-graph.

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

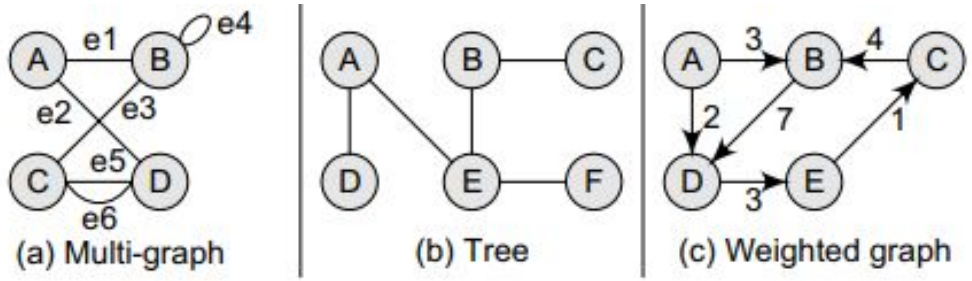


Figure 13.4 Multi-graph, tree, and weighted graph

Directed Graph Terminology

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).

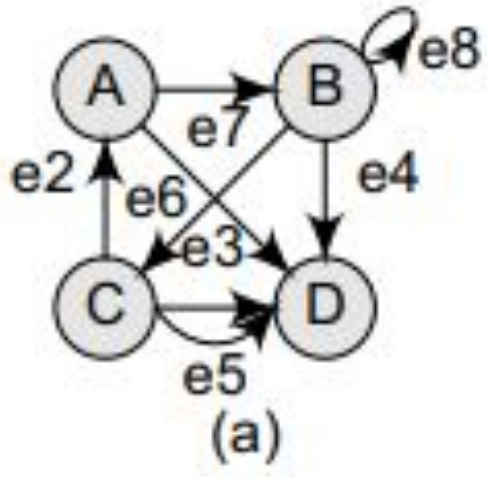
Isolated vertex A vertex with degree zero. Such a vertex is not an end-point of any edge.

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

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

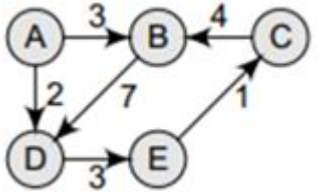
Source A node u is known as a source if it has a positive out-degree but a zero in-degree.

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

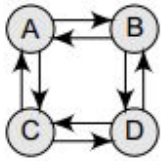


Directed Graph Terminology

- **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. For example, if you consider the directed graph given in Fig. node D is reachable from node A.

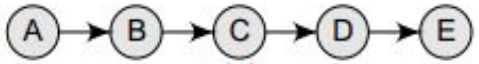


- **Strongly connected directed graph** A digraph is said to be strongly connected if and only if there exists a path between every pair of nodes in G. That is, if there is a path from node u to v, then there must be a path from node v to u.



(b)

- **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.



(a)

Directed Graph Terminology

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. In Fig. 13.5(a), e3 and e5 are multiple edges connecting nodes C and D. Simple directed graph A directed graph G is said to be a simple directed graph if and only if it has no parallel edges. However, a simple directed graph may contain cycles with an exception that it cannot have more than one loop at a given node.

Directed Graph Terminology

Simple directed graph

A directed graph G is said to be a simple directed graph if and only if it has no parallel edges. However, a simple directed graph may contain cycles with an exception that it cannot have more than one loop at a given node.

REPRESENTATION OF GRAPHS

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

Adjacency Matrix Representation

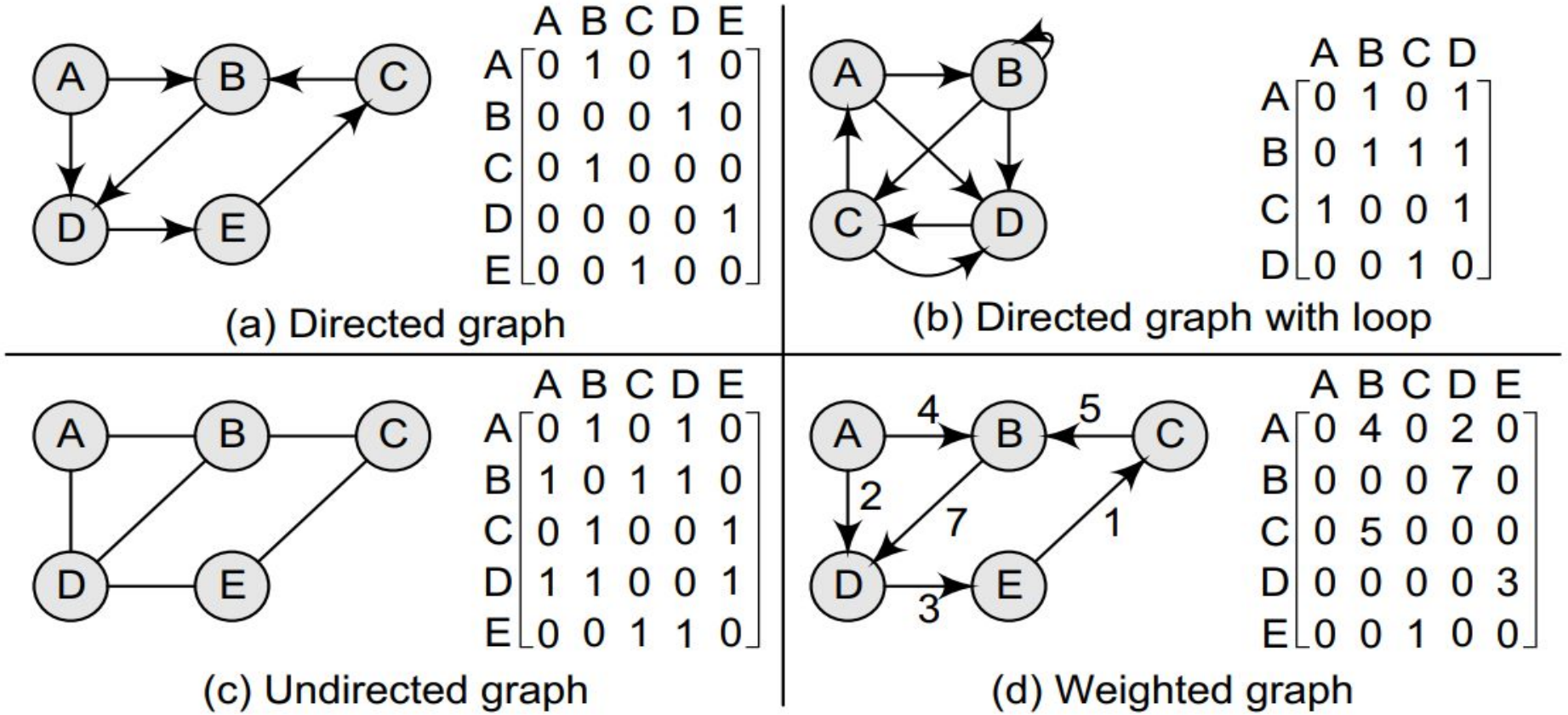
An adjacency matrix is used to represent which nodes are adjacent to one another. By definition, two nodes are said to be adjacent if there is an edge connecting them.

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.

Adjacency Matrix Representation

a_{ij} = 1 [if v_i is adjacent to v_j, that is there is an edge (v_i, v_j)]A
0 [otherwise]

Figure 13.13 Adjacency matrix entry



Graphs and their corresponding adjacency matrices

Adjacency Matrix Representation

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

1. For a simple graph (that has no loops), the adjacency matrix has 0's on the diagonal.
2. The adjacency matrix of an undirected graph is symmetric.
3. The memory use of an adjacency matrix is $O(n^2)$, where n is the number of nodes in the graph.
4. Number of 1's (or non-zero entries) in an adjacency matrix is equal to the number of edges in the graph.
5. The adjacency matrix for a weighted graph contains the weights of the edges connecting the nodes.

Adjacency Matrix Representation

From adjacency matrix A^1 , we can conclude that an entry 1 in the ith row and jth column means that there exists a path of length 1 from vi to vj .

Now consider, A^2 , A^3 , and A^4 .
$(a_{ij})^2 = \Sigma a_{ik}\ a_{kj}$ Any entry $a_{ij} = 1$ if $a_{ik} = a_{kj} = 1$. That is, if there is an edge (vi , vk) and (vk , vj), then there is a path from vi to vj of length 2.

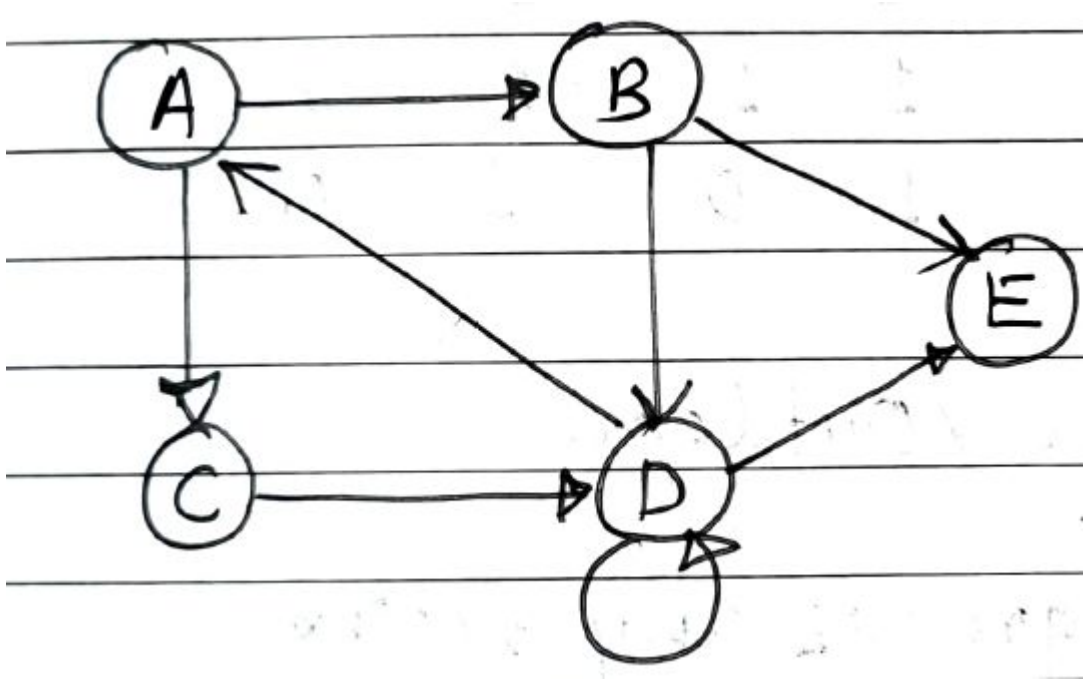
Adjacency Matrix Representation

$A^2 = A^1 * A^1$

$A^3 = A^2 * A^1$

So on……

Find Adjacency Matrix Representation of Directed Graph

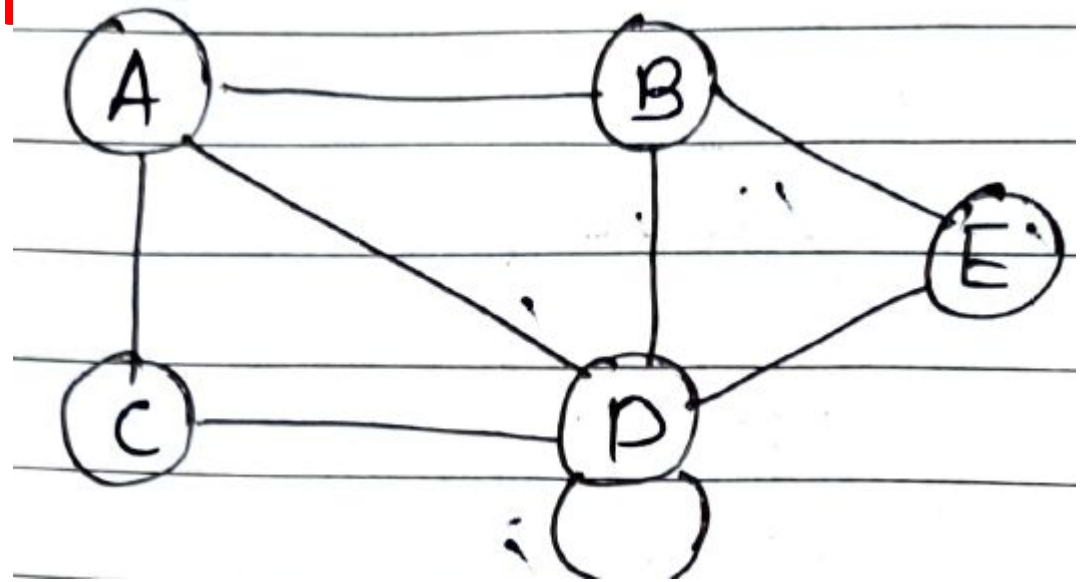


	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

Indegree-No of 1's in Column &
Out Degree-No. of 1's in Row of respective node

Indegree-1,1,1,3,2, Outdegree-2,2,1,3.0

Find Adjacency Matrix Representation of Un-Directed Graph



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

Indegree/outegree-No. of 1's in Row of respective node

Indegree/Outdegree-3,3,2,5,2

Adjacency List Representation

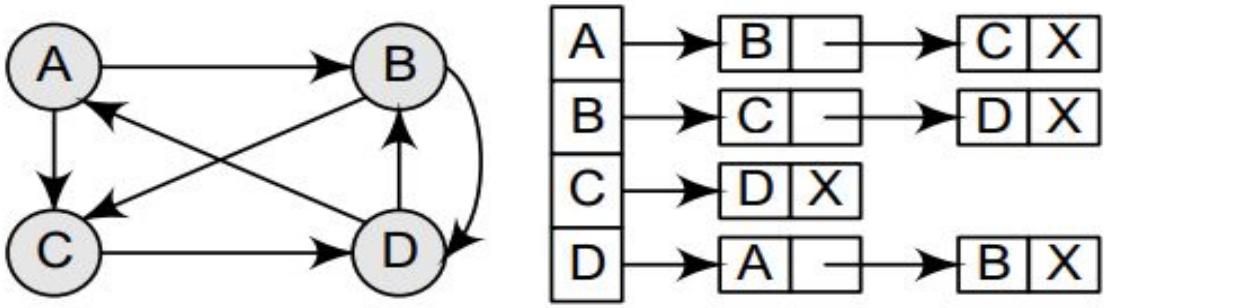


Figure 13.17 Graph G and its adjacency list

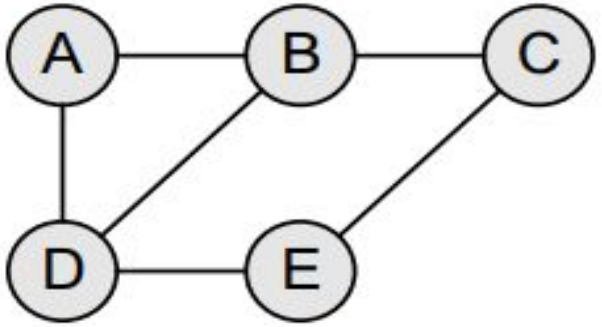
This structure consists of a list of all nodes in G. Furthermore, every node is in turn linked to its own list that contains the names of all other nodes that are adjacent to it.

Adjacency List Representation

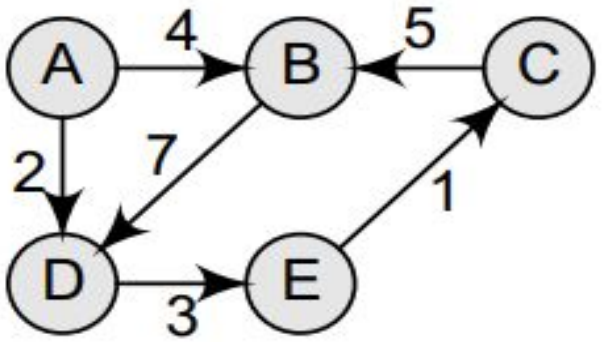
The key advantages of using an adjacency list are:

- 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

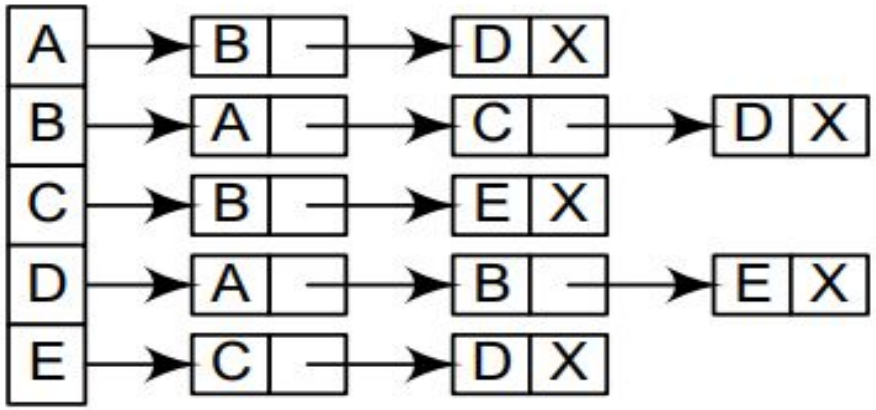
Adjacency List Representation

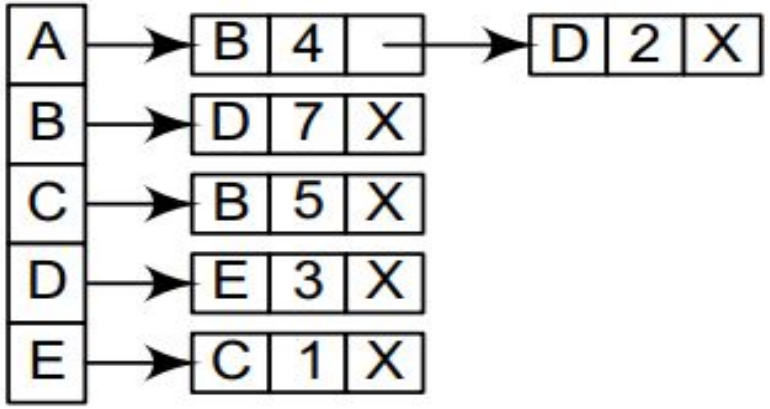


(Undirected graph)



(Weighted graph)





GRAPH TRAVERSAL ALGORITHM

1. **Breadth-first search**:
queue used as an auxiliary data structure to store nodes for further processing

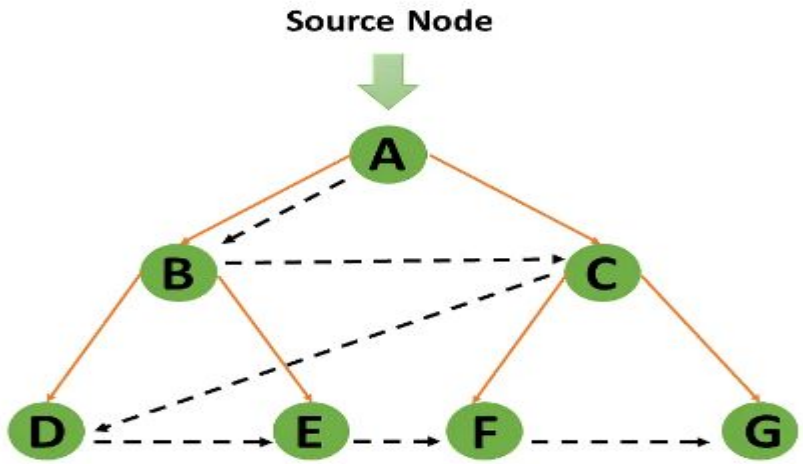
2. **Depth-first search**: scheme uses a stack.

GRAPH TRAVERSAL ALGORITHM-to determine the order in which it visits vertices

1. **Breadth-first search**: It begins at the root of the tree or graph and investigates all nodes at the current depth level before moving on to nodes at the next depth level. You can solve many problems in graph theory via the breadth-first search. For example, finding the shortest path between two vertices a and b is determined by the number of edges.

Steps:

1.Start with the source node.
2.Add that node at the front of the queue to the visited list.
3.Make a list of the nodes as visited that are close to that vertex.
4.And dequeue the nodes once they are visited.
5.Repeat the actions until the queue is empty.



Breadth-first search-Algorithm

Breadth_First_Serach(G, A) // G ie the graph and A is the source node

Let q be the queue

q.enqueue(A) // Inserting source node A to the queue

Mark A node as visited.

While (q is not empty)

B = q.dequeue( ) // Removing that vertex from the queue, which will be visited by its neighbour

Processing all the neighbors of B

For all neighbors of C of B

If C is not visited, q. enqueue(C) //Stores C in q to visit its neighbour

Mark C a visited

Ref:https://www.simplilearn.com/tutorials/data-structure-tutorial/bfs-algorithm

Transitive closure

Refer PDF