GRAPH

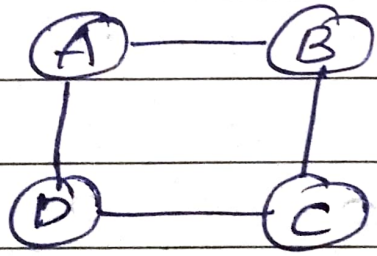
Graph is a non-linear data structure made of finite number of vertices (nodes) and edges connecting them.

$G = \{E, V\}$ E-edges V-vertices

Applications of graph

- used to represent a network
- used in social n/w's like facebook, linkedin etc.

Consider the following graph.



Vertices $V = \{A, B, C, D\}$

Edges $E = \{AB, BC, CD, AD\}$

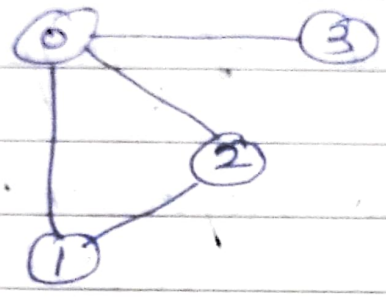
Vertex — Each node in a graph is called vertex

Edge — Edge represents the path b/w two vertices.

Graph Representation

- Incidence matrix
- Adjacency matrix.
- Adjacency list

Consider the following graph.



Graph adjacency matrix representation is as follows.

	0	1	2	3
0	0	1	1	1
1	1	0	1	0
2	1	1	0	0
3	1	0	0	0

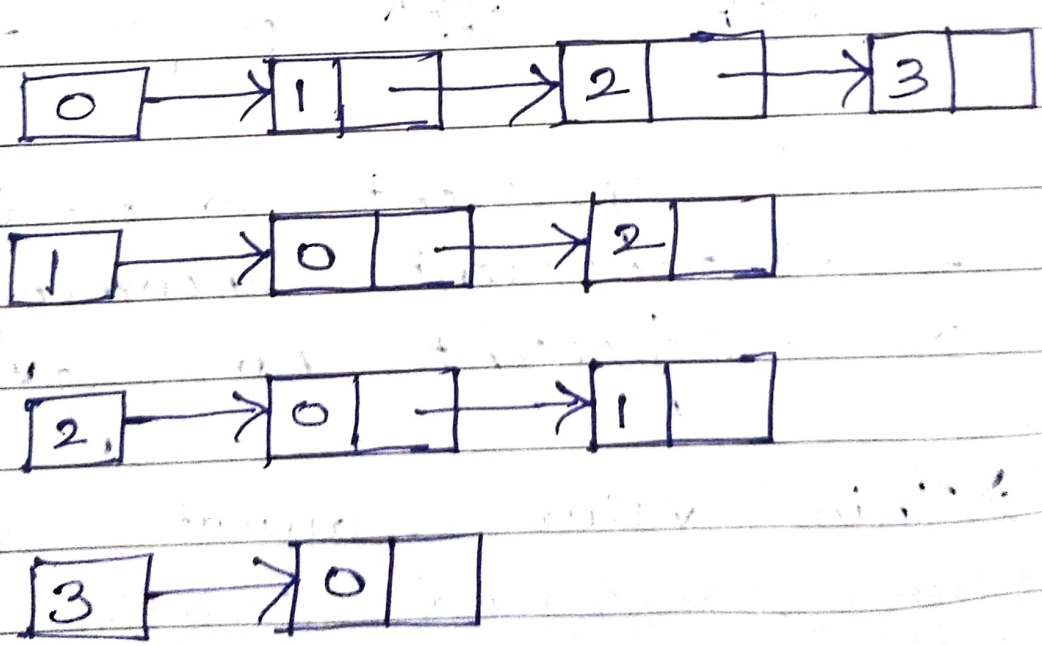
Adjacency matrix is a 2D array of V x V vertices. Each row & column represents a vertex. If there is an edge b/w vertex i & vertex j, then a[i][j] = 1.

Adjacency list

An adjacency list represents a graph as an array of linked lists

The index of the array represents a vertex & each element in its linked list represents the other vertices that form an edge with the vertex.

The adjacency list representation for the above graph is as follows.



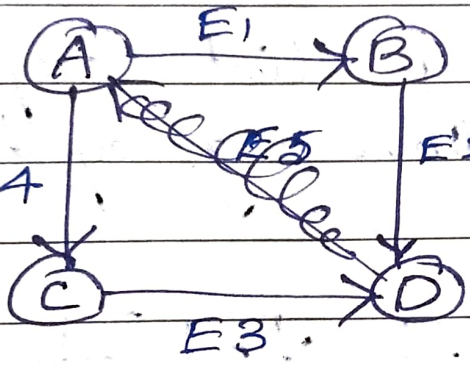
Incidence matrix

In this representation the graph is represented using a matrix of size m×n.

m → no. of. vertices

n → no. of. edges

Consider the following directed graph.



	E1	E2	E3	E4
A	1	0	0	1
B	-1	1	0	0
C	0	0	1	-1
D	0	-1	-1	0

1 → represents outgoing edge

-1 → represents incoming edge

0 → represents no edge.

→ This value cannot come for an undirected graph.

for undirected graph

1 → if there is an edge

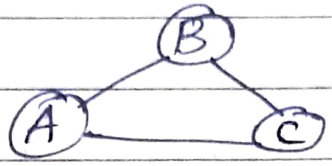
0 → if there is no edge

Types of Graph

1) Undirected graph

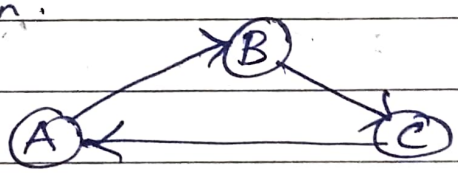
A graph in which the edges are bidirectional i.e., they do not point in any specific direction is called undirected graph.

Eg:-



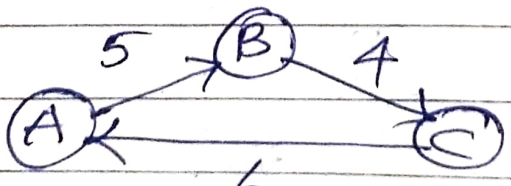
2) Directed graph

A graph in which the edges are uni-directional. The edges point in a single direction.

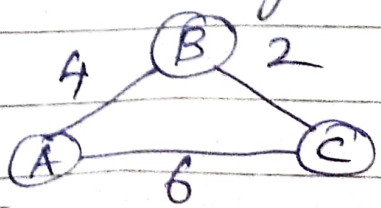


3) Weighted graph

A graph that has value associated with each edge



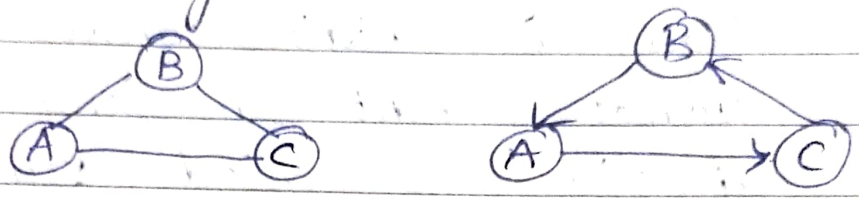
Directed weighted graph



Un-Directed weighted graph

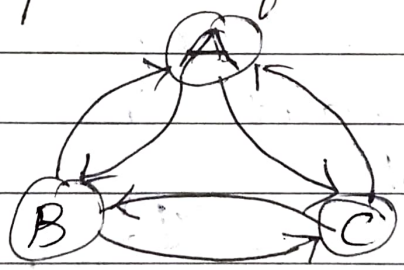
4) unweighted graph

A graph in which there is no value associated with each edge.



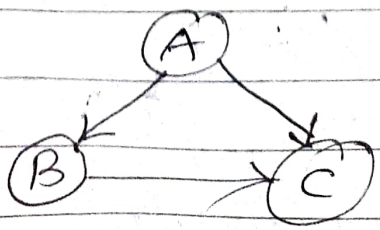
5) Strongly connected graph

The graph is said to be strongly connected if there exist an edge between every pair of vertices.



6) Weakly connected:

The graph is weakly connected if there doesn't exist an edge b/w every pair of vertices



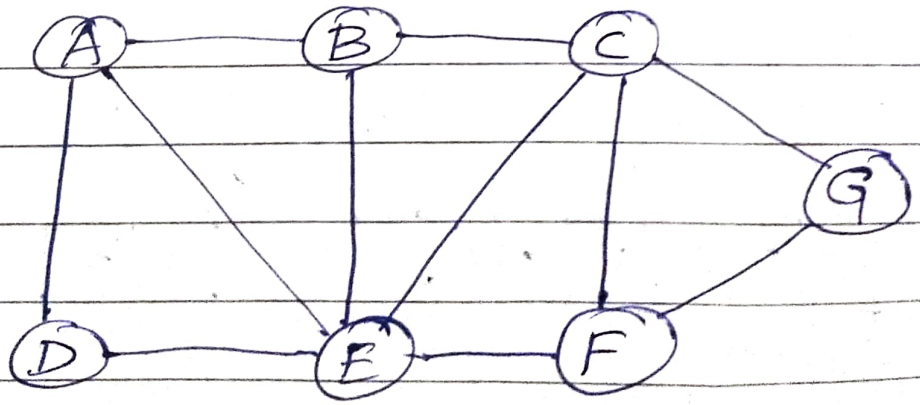
GRAPH TRAVERSAL

- used for searching a vertex in graph.
- There are two types
  * DFS (Depth First Search)
  * BFS (Breadth first Search)

BFS

- produces spanning tree as a final result.
- (spanning tree is a sub-graph without loops)
- implemented using Queue data structure.

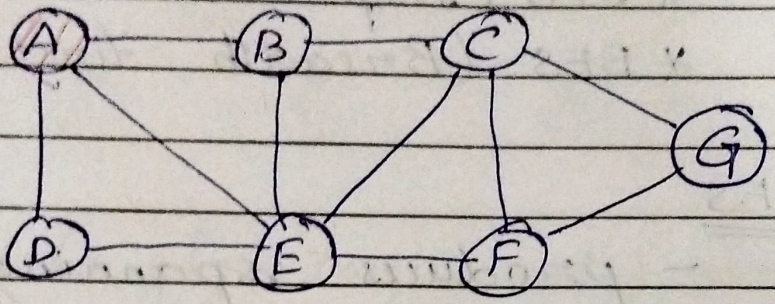
Consider the following graph



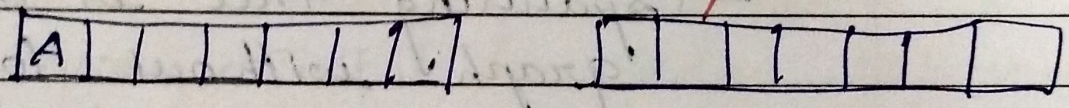
__/__/__

Step 1

Select A as the starting vertex
Insert A into the queue ✓

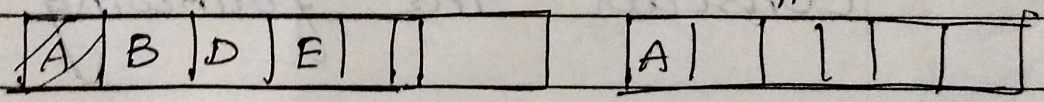


Queue | Output

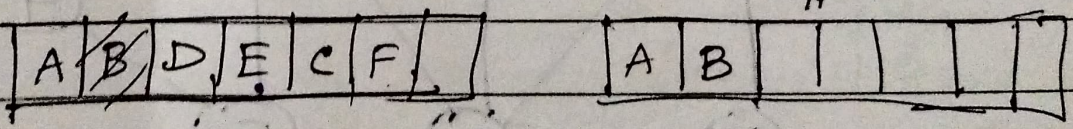


Step 2

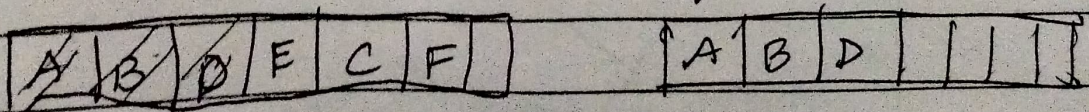
Visit all adjacent vertices of A



Visit adj. vertices of B



Visit adj vertices of D



Visit adj vertices of E

A	B	D	E	C	F	

o/p

A	B	D	E				

Visit adj vertices of C

A	B	D	E	C	F	G

o/p

A	B	D	E	C			

Visit adj vertices of F

A	B	D	E	C	F	G

o/p

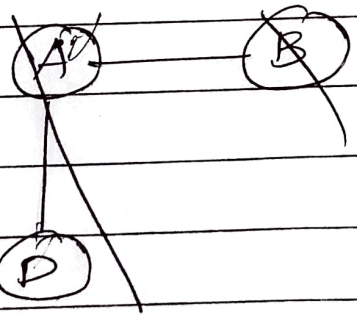
A	B	D	E	C	F	G

visit adjacent vertices of G

A	B	D	E	C	F	G

o/p

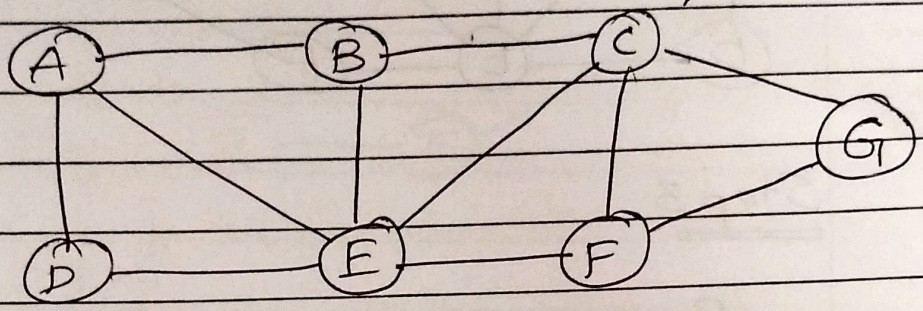
A	B	D	E	C	F	G



__/__/__

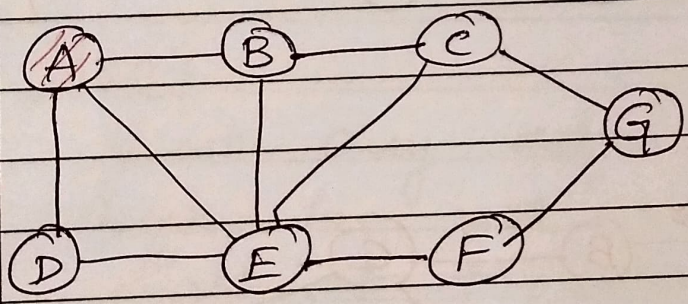
DFS

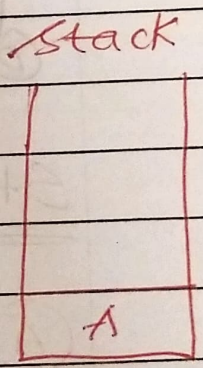
- Implemented using stack.



Step1

- Select A as starting point (visit A)
- Push A to stack.

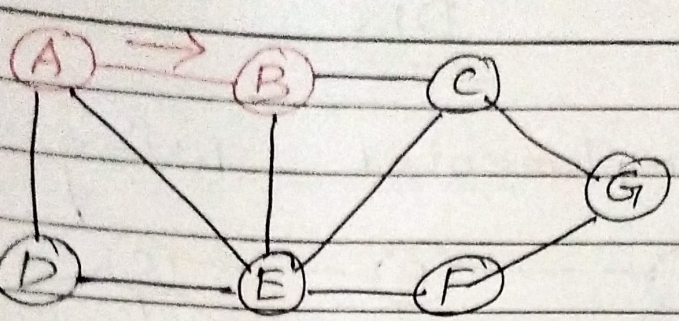


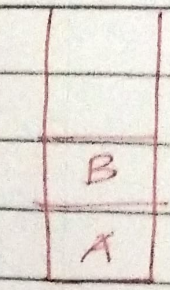


Step 2

- Visit any adjacent vertices of A which is not visited (visit B)
- Push B to stack

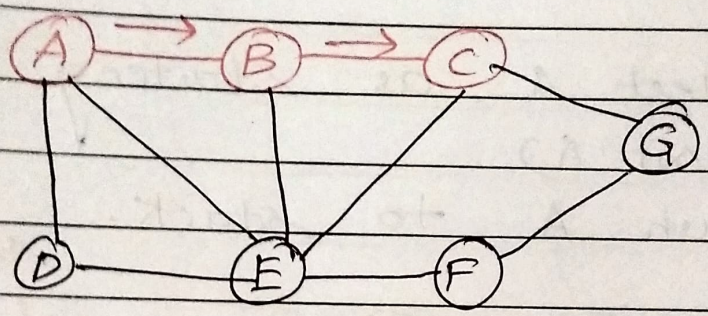
__/__/__

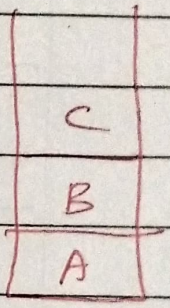




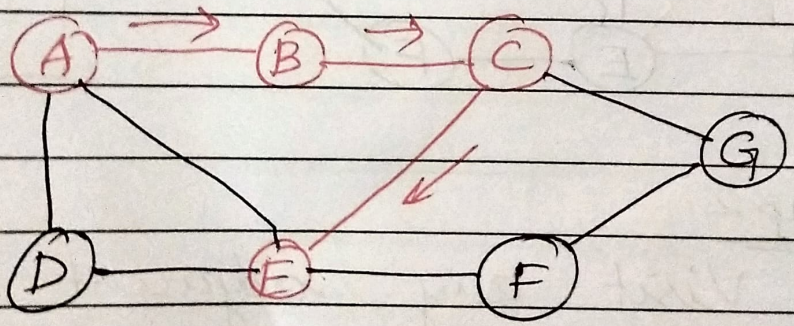
Step 3

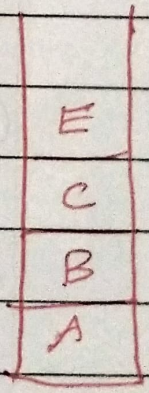
Push C on to stack.



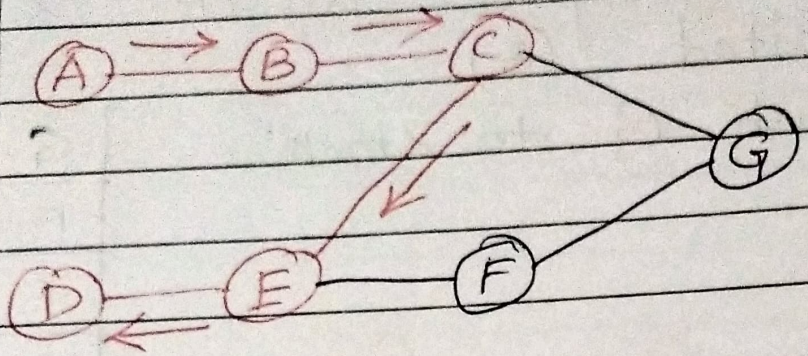


Step 4





Step5



D
E
C
B
A

Step6

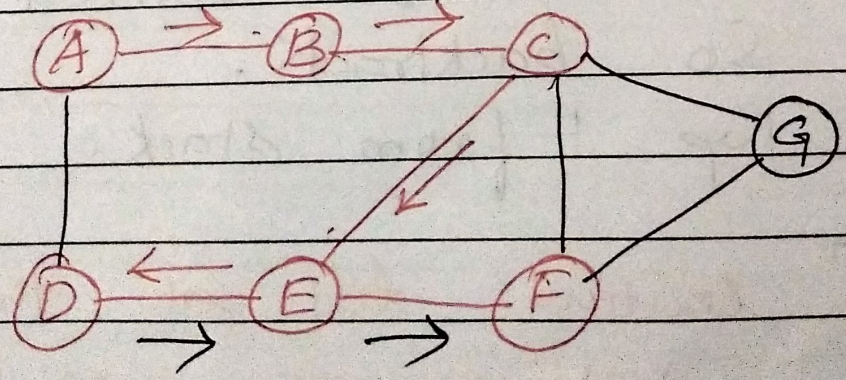
There is no new vertices to be visited from D. So, use backtrack

Pop D from stack.

Step7

visit any adj vertices of E which is not visited.

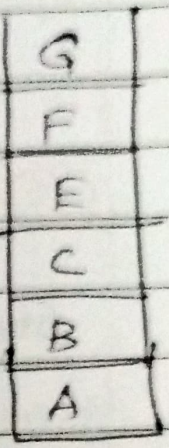
Push F to stack.



F
E
C
B
A

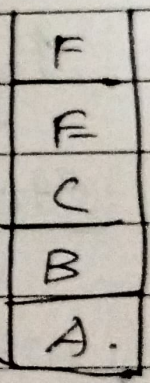
Step 8.

- visit adj (F) which is not visited (G)
- Push G to stack



Step 9

- There is no new vertex to be visited from G.
So, we backtrack
- Pop G from stack



Step 10

- There is no new vertex to be visited from F. So backtracks.
- Pop F from stack.

{ Continue till all the vertices are popped from the stack. }

__/__/__

Final result of DFS is Spanning tree