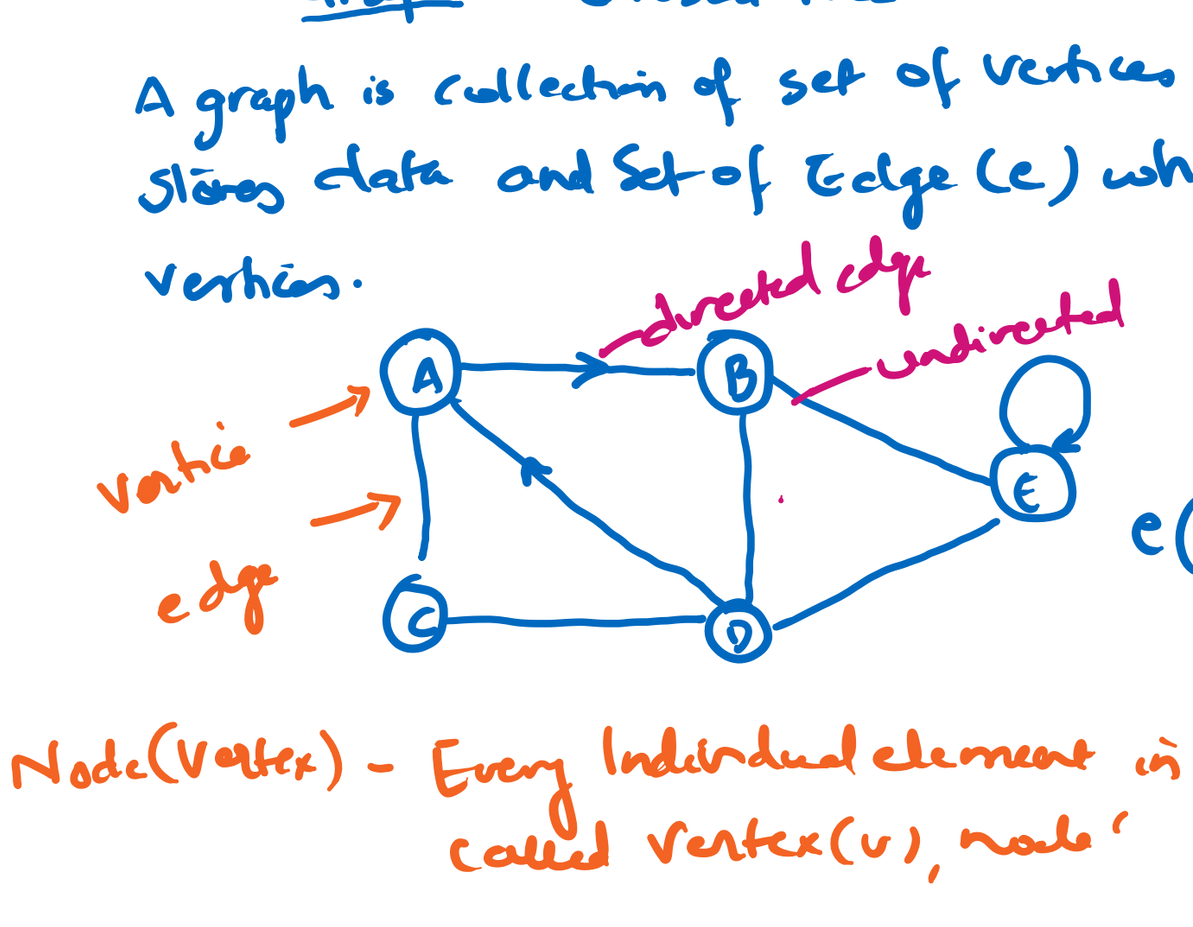
Graphs - closed Tree

A graph is collection of set of vertices (V) which actually stores data and set of Edge (E) which connect those vertices.



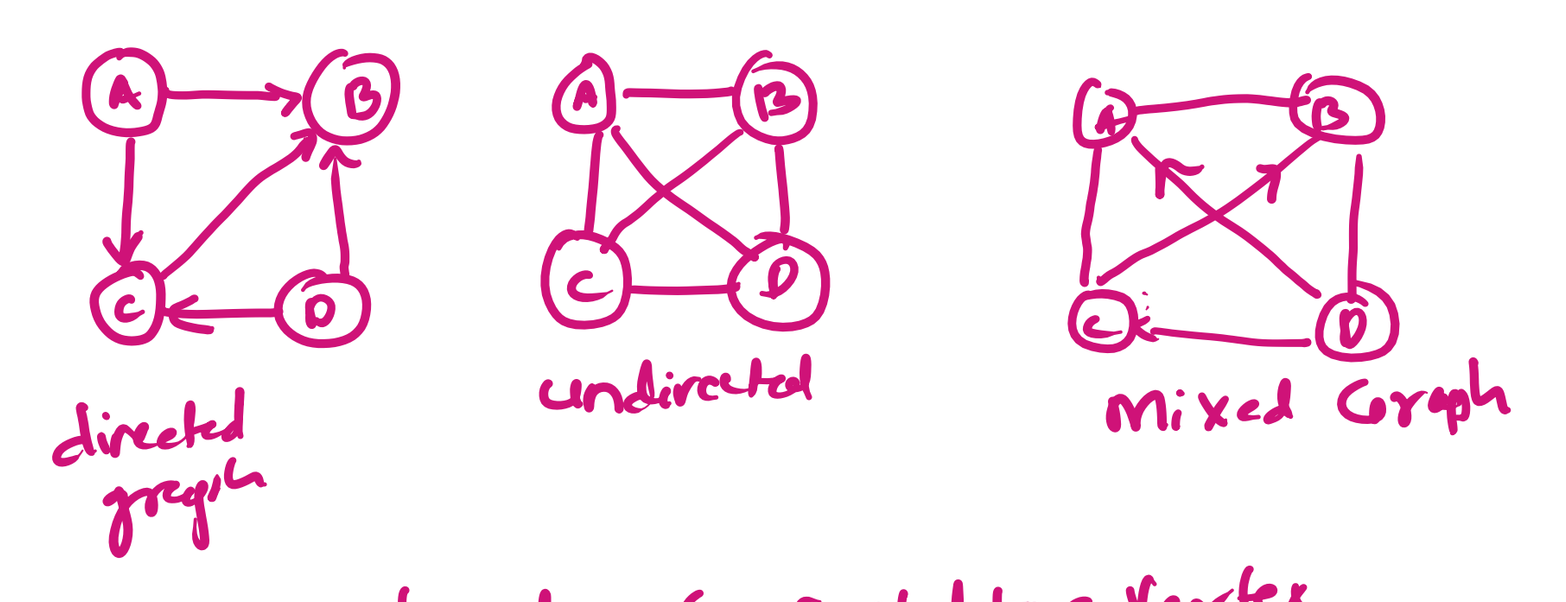
G = (V, E)
V = (A, B, C, D, E)
E (AB, BE, AC, BD etc)

Node (Vertex) - Every individual element in graph called vertex (V), node.

Arc (Edge) - Connecting link between two vertices

Directed edge - The edge with specific direction

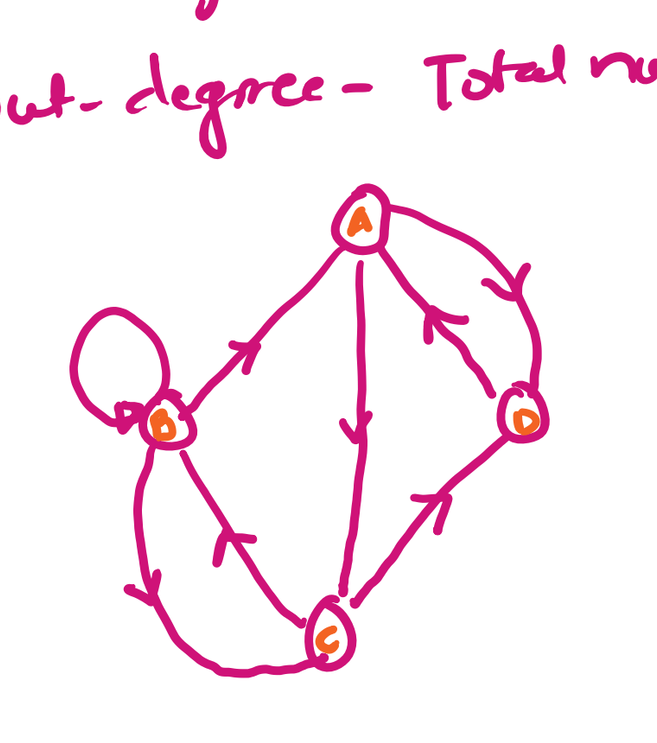
Graph - directed Graph - edges with have specific direction
- undirected Graph - edges/arcs do not have direction



Degree - no. of edges connected to a vertex.

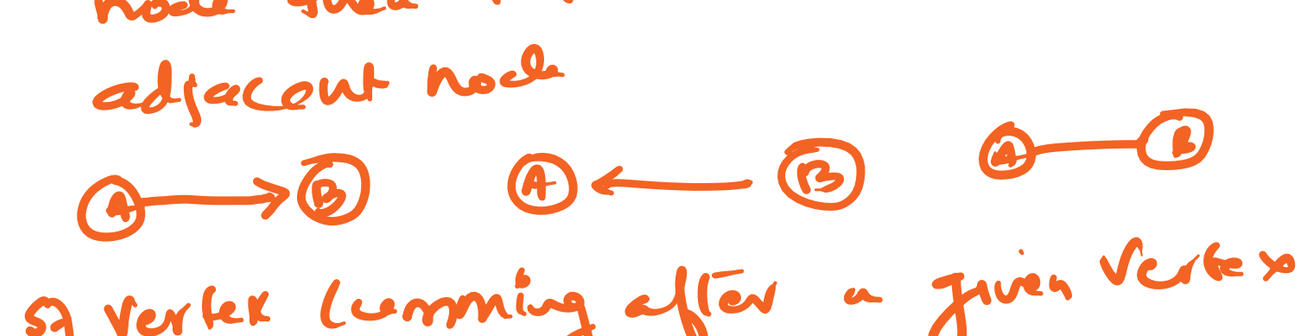
In-degree - Total number of incomming edges

out-degree - Total number of outgoing edges



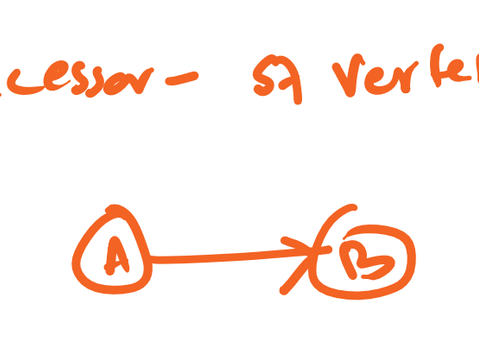
Node	Indegree	outdegree	degree
A	2	2	4
B	2	3	5
C	2	2	4
D	2	1	3

Adjacent - If there exist an Arc between A & B node then A & B will be called adjacent node

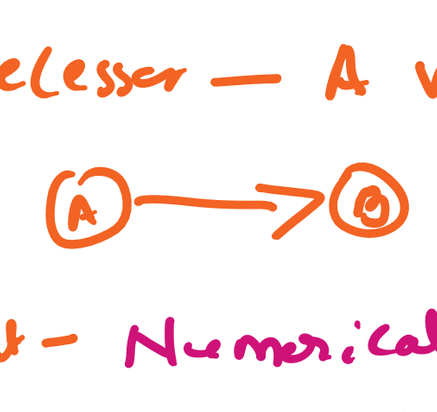


Successor - A vertex comming after a given vertex

B is successor of A

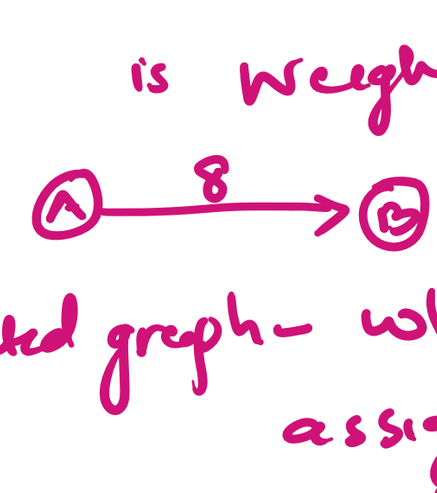


predecessor - A vertex comming before a given vertex

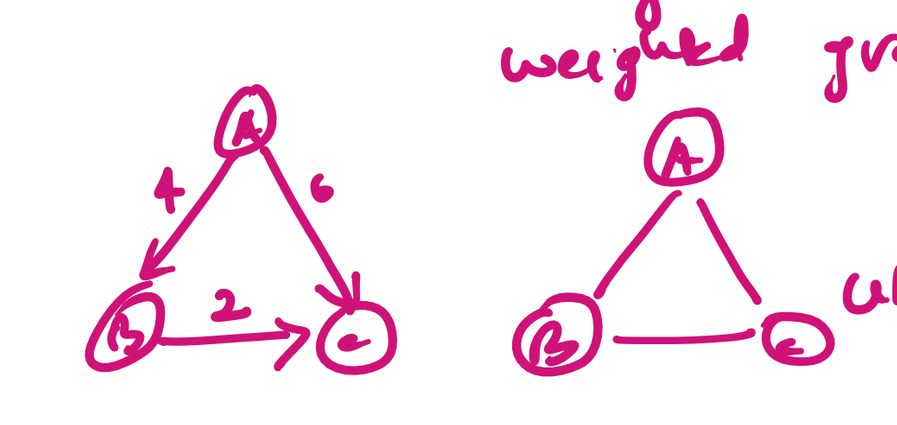


A is predecessor

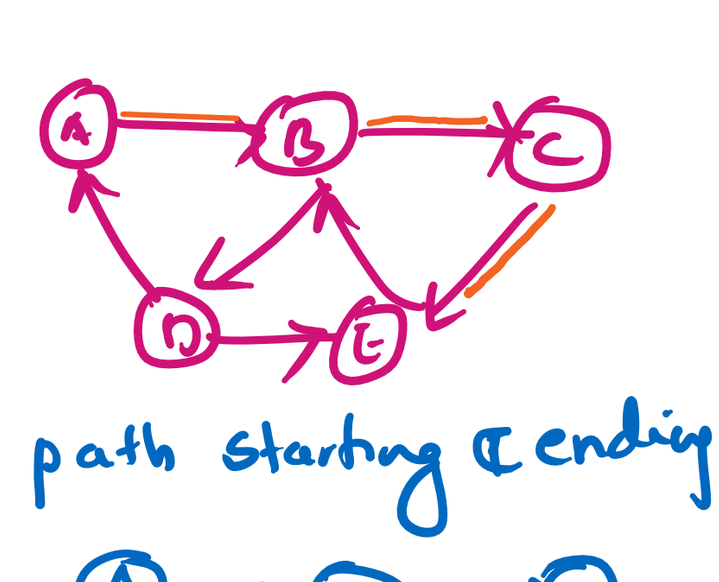
weight - Numerical value assigned to vertex or edge is weight



weighted graph - when every edge of a graph is assigned with weight is called weighted graph

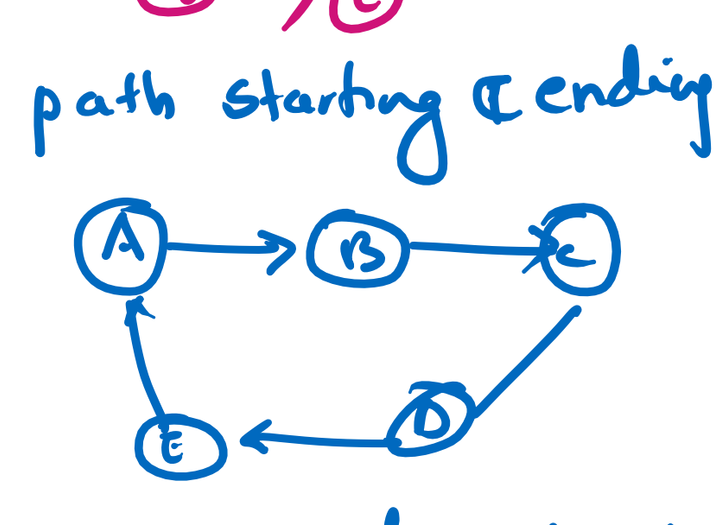


path - A path is a sequence of alternating vertices and edge that starts at a vertex and ends at a vertex, such that each edge is incident to predecessor or successor.

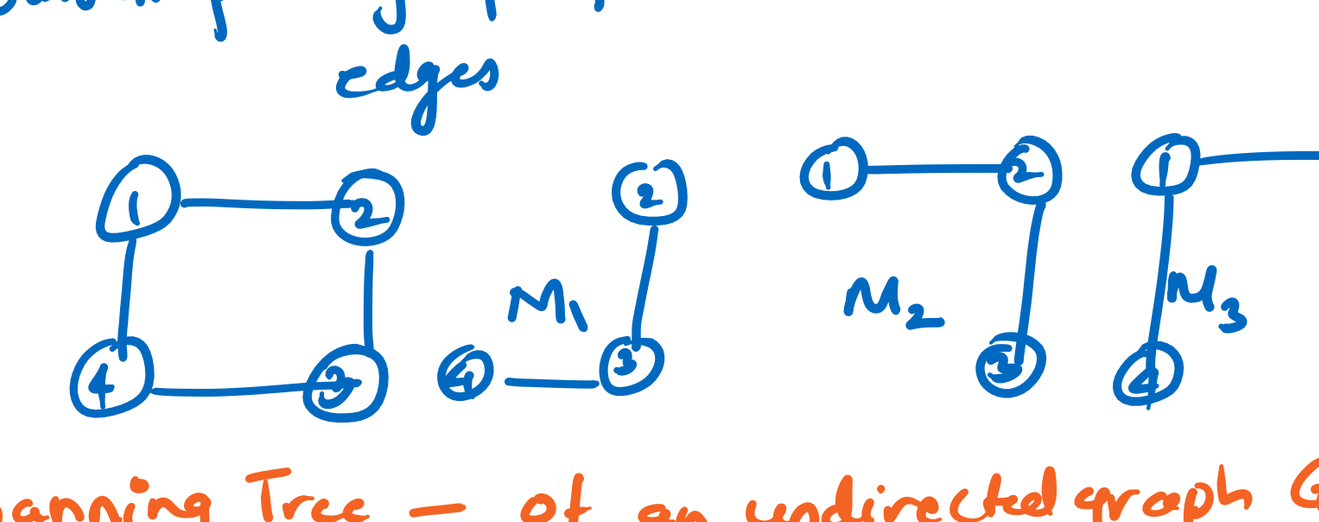


AB, BC, CE
A, B̄, C̄, E

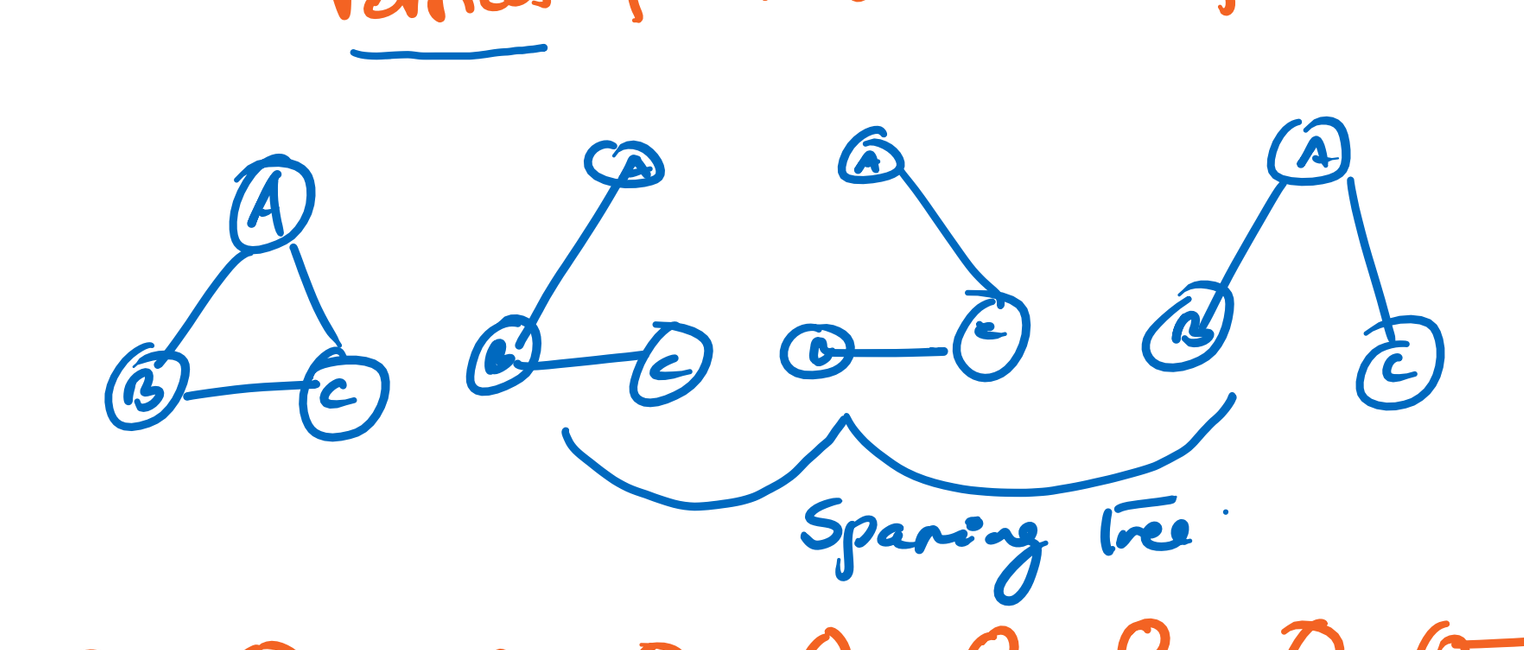
Cycle - path starting & ending at same vertex

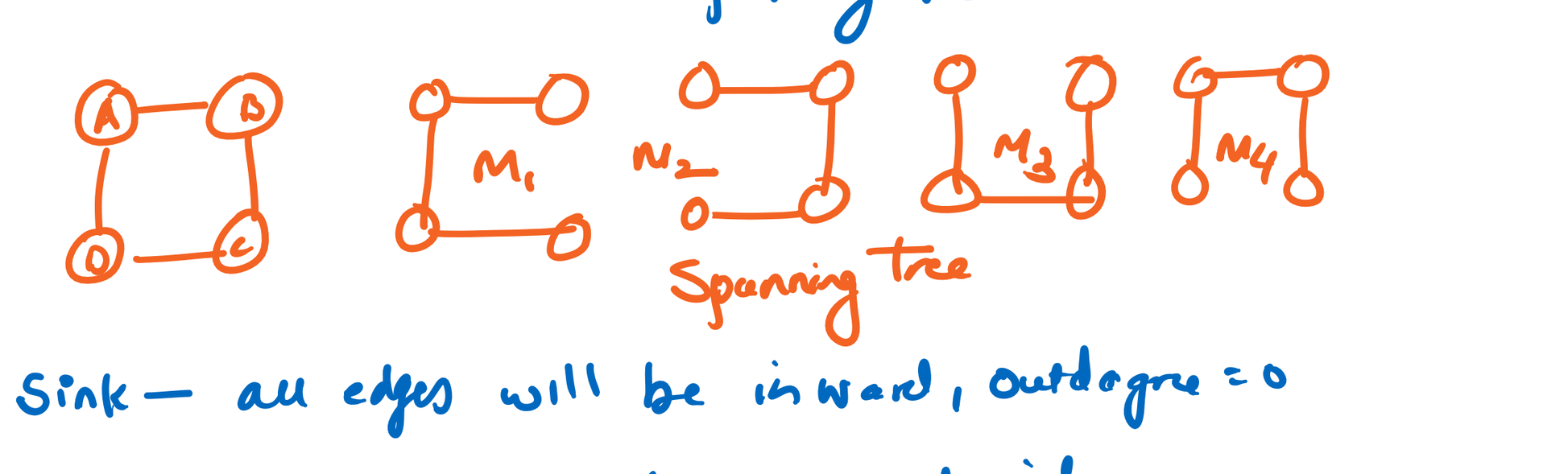


SubGraph - graph formed with subset of vertices & edges



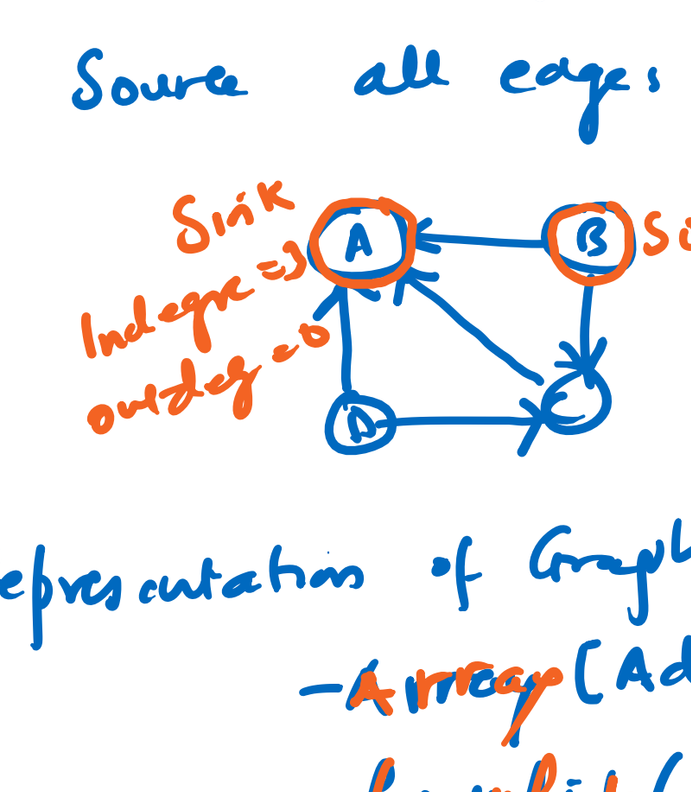
Spanning Tree - of an undirected graph G is a subgraph that is a tree which includes all vertices of G with min possible no of edges





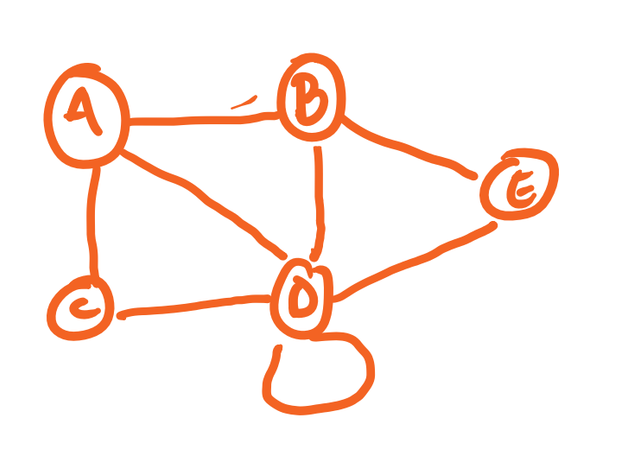
Sink - all edges will be inward, outdegree = 0

Source all edges will be outward indegree = 0

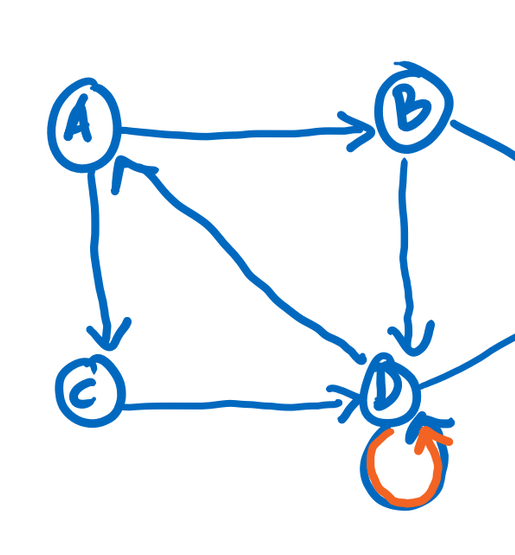


Representation of Graph -
- ~~Array~~ (Adjacency Matrix) [static]
- ~~Linked list~~ (Adjacency list) [dynamic]

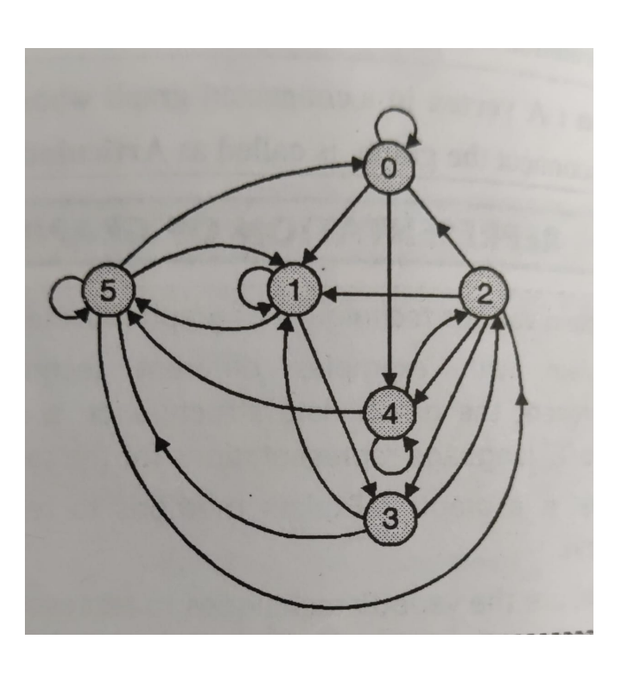
Array Representation



	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

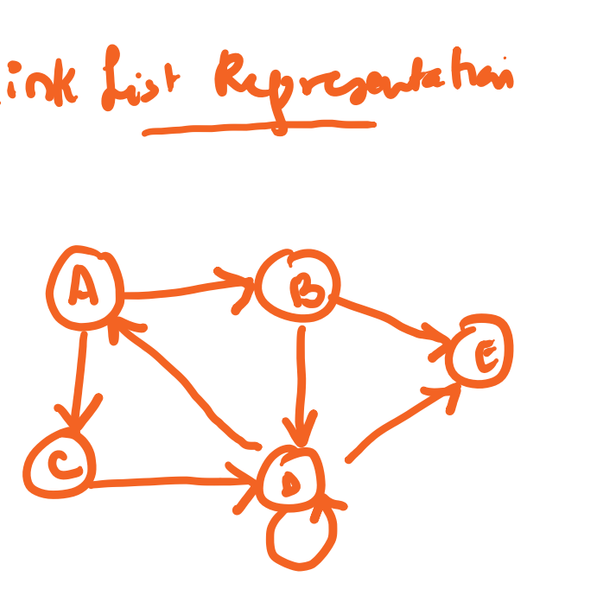


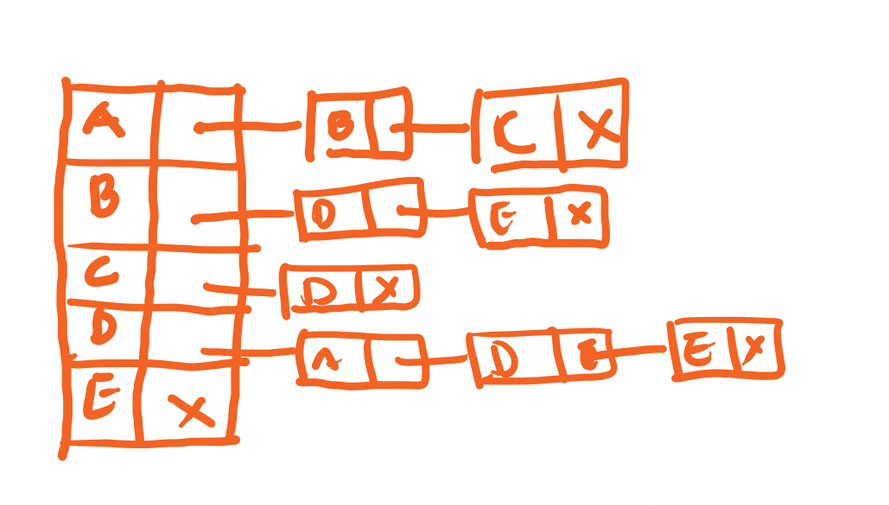
	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

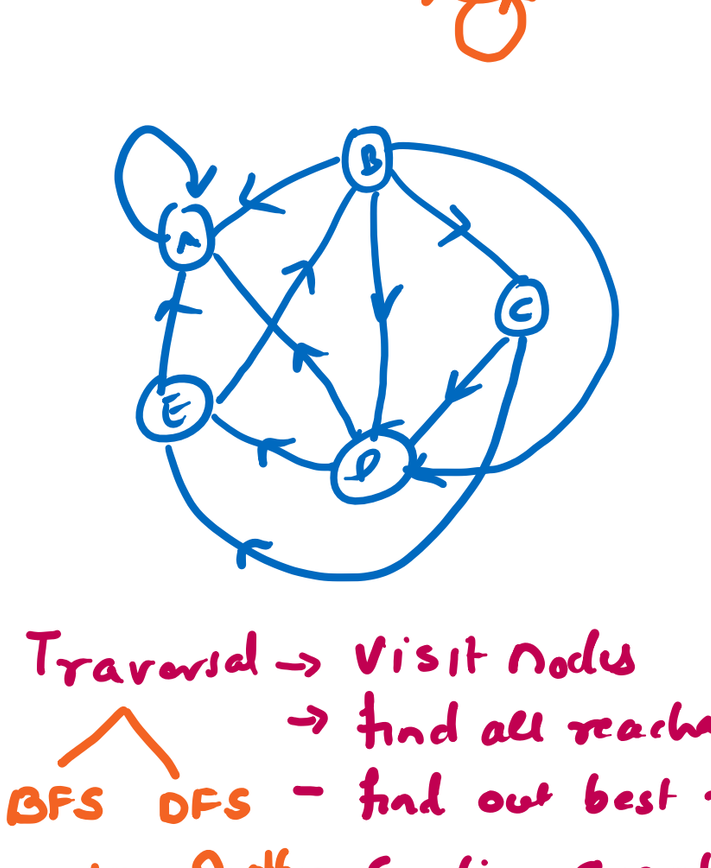


	0	1	2	3	4	5
0	1	1	0	0	1	0
1	0	1	0	1	0	1
2	1	1	0	1	1	0
3	0	1	1	0	0	1
4	0	0	1	0	1	1
5	1	1	1	0	0	1

Link list Representation





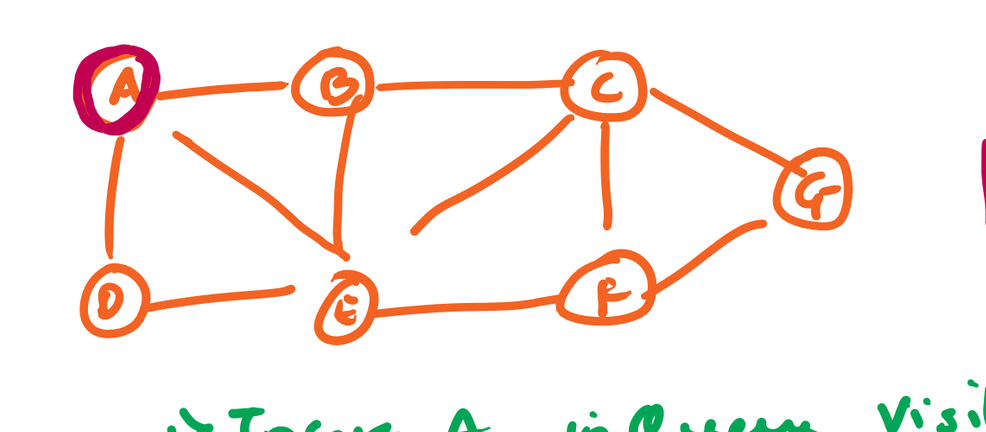


Draw Link Representation for Given graph.

Traversal → visit nodes
→ find all reachable nodes
BFS DFS - find out best reachable path
Breadth, Depth - Sorting graph
First Search

BFS - Breadth first Search [Queue] → size of Queue = total No. of Vertices

- Any vertex can selected as starting point
- Visit that vertex & add that vertex in Queue
- Then visit all adjacent of the vertex, which are not visited
- As soon u reach Adjacent node/vertex remove it predecessor.
- Repeat the process till Queue is empty

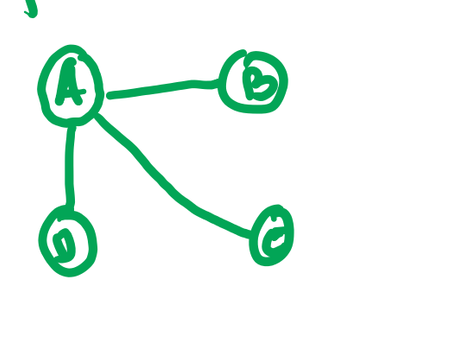


Queue

| A | | | | | | |

1) Insert A in Queue, visit all Adjacent of A
D, E, B

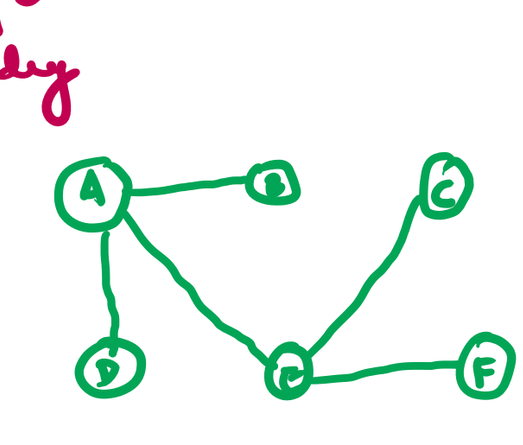
removed A — | D | E | B | | | |



2) Insert D, E, B on Queue

3) Adjacent of D
Remove D from Queue, insert E on queue but E is already present

| · | · | E | B | | | |

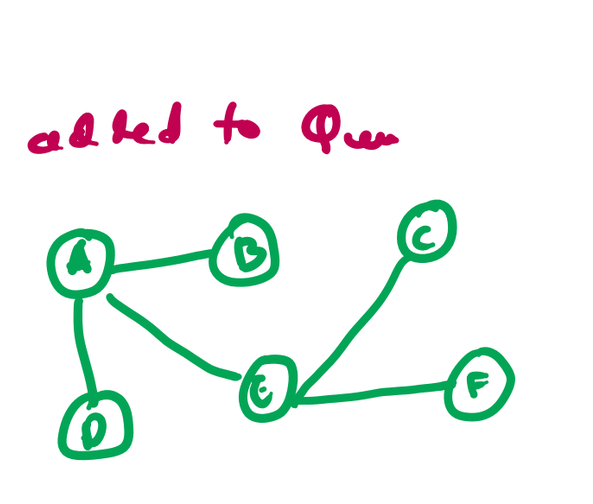


4) Find Adjacent of E & Insert in Queue
B, C, F
B is already on Queue so insert C & F and remove E

| · | · | · | · | B | C | F | |

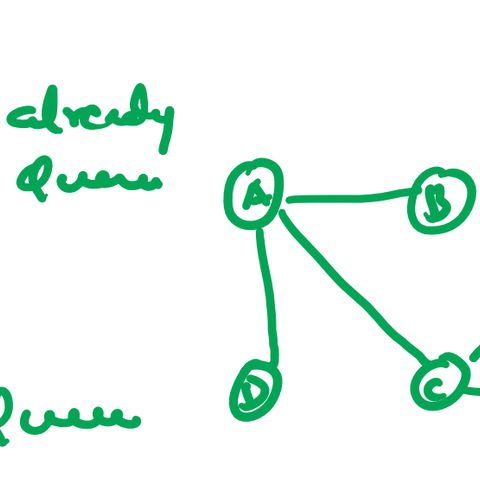
5) find Adjacent of B (C, E) no node added to Queue
Both C & E are visited
remove B

| · | · | · | · | C | F | |



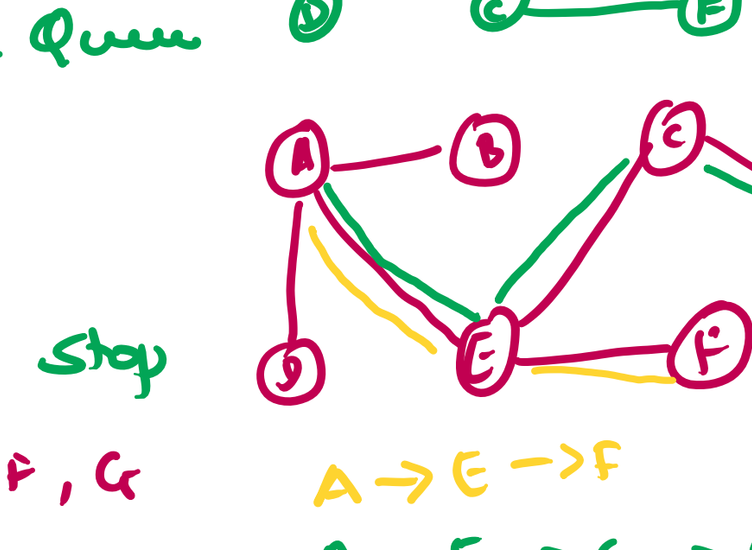
6) find Adjacent of C (G, F)
remove C & insert G on Queue, F is already on Queue

| · | · | · | · | · | F | G |



7) find Adjacent of F (C, G) Both are on Queue
remove F

| · | · | · | · | · | · | G |



8) No Adjacent for G, remove G & stop the process.

A, D, E, B, C, F, G

A → E → F
A → E → C → G

DFS (depth first search) [Stack]