Graph:

Graph: Graph consists of two things:

A set V of elements called nodes.

A set E of edges such that each edge e in E is identified with a unique (unordered) pair (u,v) of nodes in V, denoted by e = [u,v].

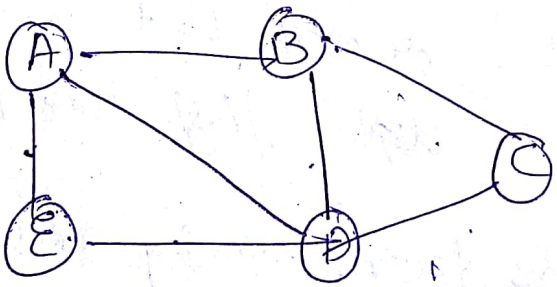
Applications of Graph:

→ Connecting with friends on social media, where each user is vertex. and when users connect they create an edge.

→ Using GPS / Google, Yahoo maps, to find route based on shortest route.

→ To calculate shortest path.

Ex.



V = {A, B, C, D, E}

E = {AB, BC, CD, DE, EA, BA, AD}.

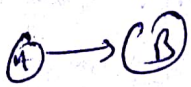
Vertex - Individual data element.

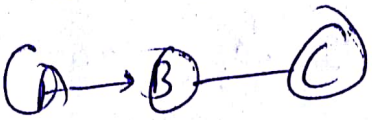
Edge - Path between two vertex.

↳ Undirected ↳ weighted

↳ Directed.

Undirected Graph. 

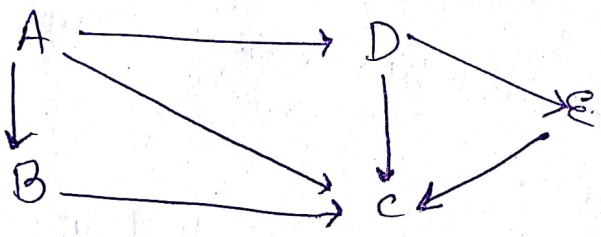
Directed Graph. 

Mixed Graph. 

Degree of Graph: Total no. of edges connected to a vertex.

Representation of Graph -

→ Linked Representation:



Node	List
A	B, C, D
B	C
C	—
D	C, E.
E	C.

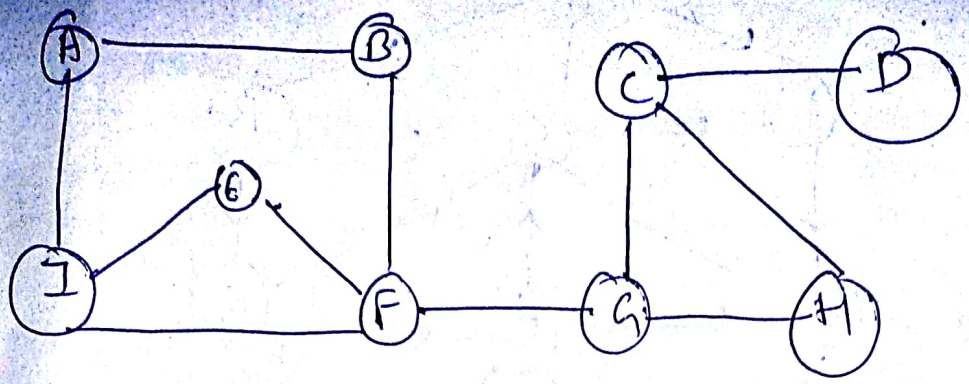
Graph Traversing - Many graph algorithms require one to systematically examine the nodes and edges of a graph G. To do this there are two ways -

→ BFS - breadth first search (Queue)

→ DFS - depth first search. (Stack).

BFS: It begins from starting node A, then all neighbours of A are traversed. No node should be traversed more than once.

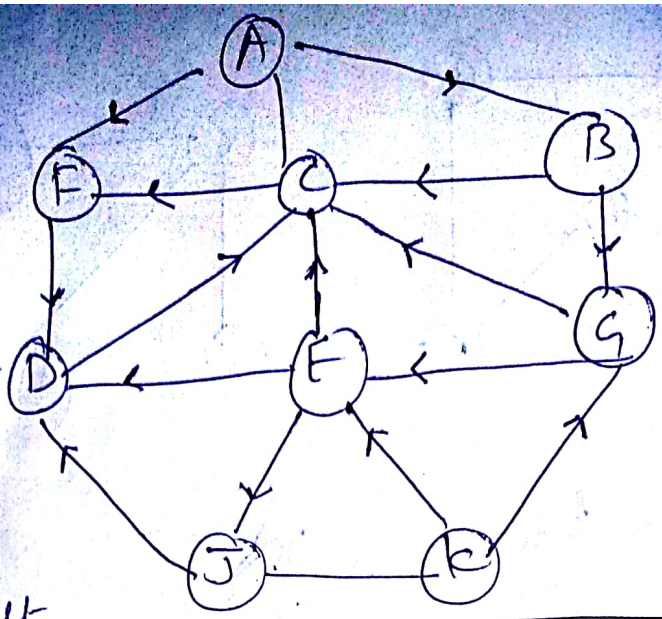
(3)



Operation.	Queue.	Node Traversed
Insert A	A	~~A~~ -
Delete A, add its neighbours	B, I	A
Remove B, add its neighbours	I, F	~~A~~ B
Remove I	F, E	I
Remove F	E, G	F
Remove E	G,	E
Remove G	C, H	G
Remove C	H D.	C
Remove H.	D	H
Remove D	-	D

A B I F E G C H D.

Ex. To be..



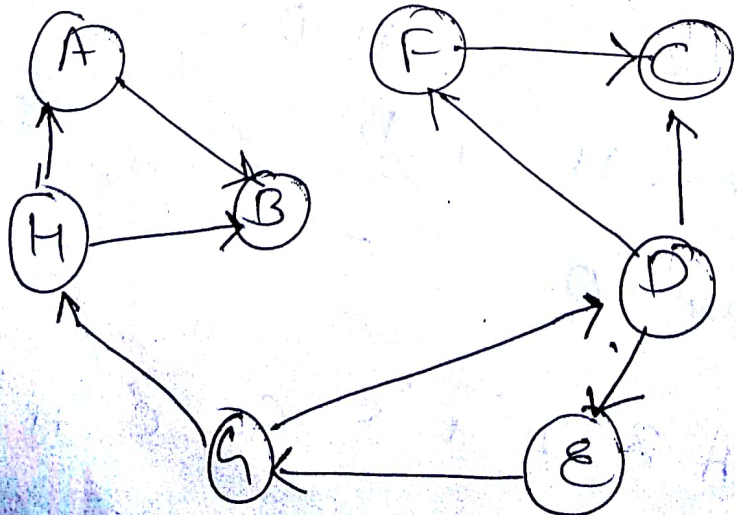
Adjacency List

A
B
C
D
E
F
G
J
K.

Visited Array

A
B
C
D ✓
E
F
G
H

Depth first search:



Visited Array:

A	
B	
C	
D	✓
E	
F	
G	
H	

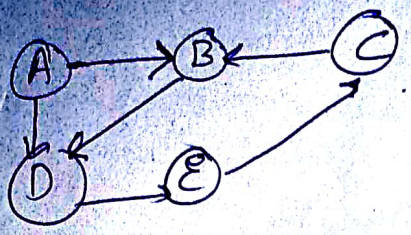
	Operation	Stack	node traversed
(10)	Add nodes of D.	D F	DCEGHMAB
(11)	visit F, Backtrack	D	"
(12)	no unvisited nodes	-	DCEGHMABF

Ans.

Visited.

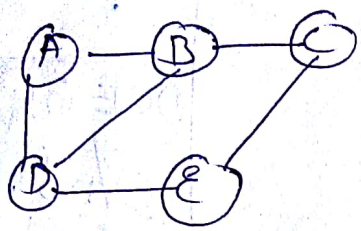
	Operation	Stack	node traversed
(1)	Insert D in stack	D	-
(2)	Remove D add its neighbour.	C	D C
(3)	Remove C, add other neighbours of D	D E	D C E
(4)		D C E	
(5)	Add neighbour of E.	D E G	D C E G
(6)	Add nodes of G	D E G H	D C E G H.
(7)	A, B bothare nodes of G, traverse 'A' first	D E G H A	D C E G H A
(8)	Add nodes of A	D E G H A B	D C E G H A B
(9)	Backtrack (Pop the stack)	D E G H A	D C E G H A B
(10)	Backtrack upto D	D	D C E G H A B.

* Representation of Graph:



Directed

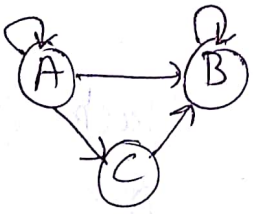
	A	B	C	D	E
A	0	0	0	1	0
B	0	0	0	1	0
C	0	1	0	0	0
D	0	0	0	0	1
E	0	0	1	0	0



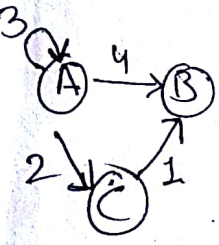
Undirected

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

Directed graph with Loop



	A	B	C
A	1	1	1
B	0	1	0
C	0	1	0



	A	B	C
A	3	4	2
B	0	0	0
C	0	1	0

SHORTEST PATH ALGORITHM:

- → Minimum Spanning Tree
  - → Prims and
  - → kruskal
- → Dijkstra's Algorithm.
- → Warshall's Algorithm.

Minimum Spanning Tree:

Given a connected weighted graph G, and it is often desired to create a spanning tree T for G such that the sum of weights of the tree edges in T is as small as possible. To apply this we have many methods but Prim's and Kruskal's algos. are main.

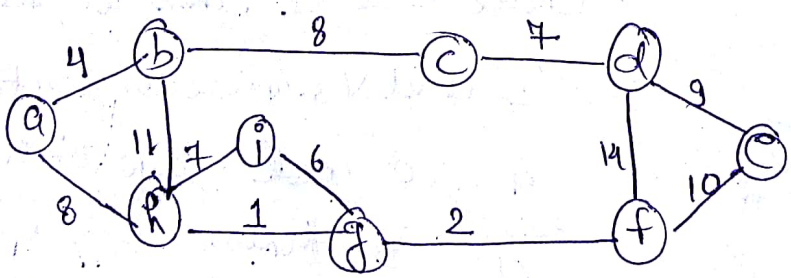
(1) Kruskal's Algorithm –

Step 1. Initially construct a separate tree for each node in graph.

(2) Edges are placed in priority queue, i.e, we take edges in ascending order. We can use a heap for priority queue.

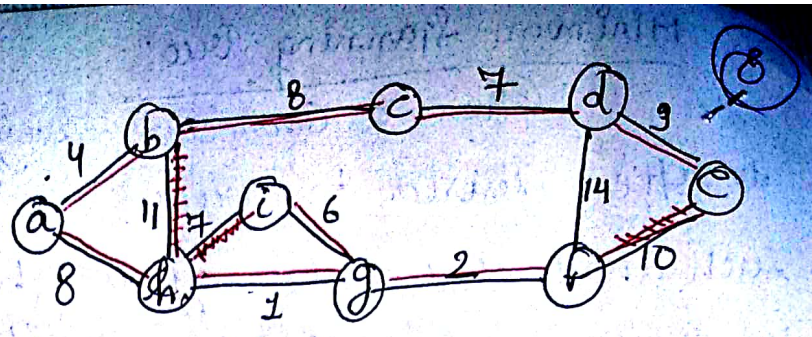
(3) until we have added n−1 edges
- extract cheapest edge.
- if a cycle is formed, reject it.
  else
- add it to tree.

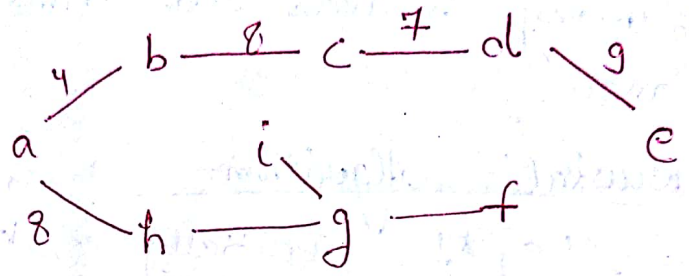
Ex.



(1) Sort and arrange edges in ascending order

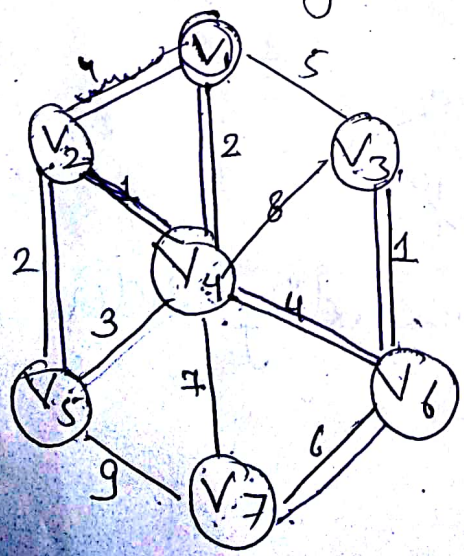
Edge	weight
(h, g)	1 ✓
(g, f)	2 ✓
(a, b)	4 ✓
(i, g)	6 ✓
(h, i)	7 ✗
(c, d)	7 ✓
(b, c)	8 ✓
(a, h)	8 ✓
(d, e)	9 ✓
(e, f)	10 ✓
(b, h)	11 ✗
(d, f)	14 ✗





Prim's Algorithm:

In prim's algorithm, a node is taken then choose other nodes associate to that node and having minimum weight.



→ Choose node V_1, then we see that V_2 and V_3, V_4 are connected to V_4 so ~~node~~ edge with minimum weight is taken i.e, $V_1 - V_4$

→ ~~Again~~ with node V_4 V_1, V_4 & V_5, ... are connected, then V_8 is chosen as it have min weight.

→ $V_8 \rightarrow V_5$, V_5 V_1, ~~V_1 V_4~~.

→ $(V_1, V_4)(V_4, V_2)(V_2, V_5)(V_4, V_6)(V_4, V_3)(V_1, V_7)$

Important

Warshall's Algorithm:-

1.) ~~Begin~~

$P_k[i][j] =$ { 1 If there is a simple path from vertices v_i to v_j which does not use any other node except possibly $v_1, v_2, v_3 \dots v_n$ or this path do not ~~contain~~ use any node numbered higher than k
0 otherwise.

$P_0[i][j] = 1$, if there is a simple path from v_i to v_j

$P_1[i][j] = 1$

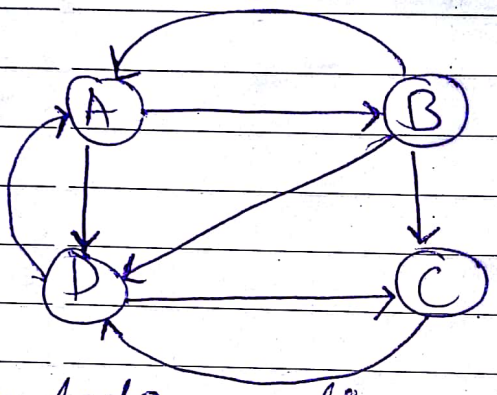
if $P_{k-1}[i][j] = 0$ then $P_k[i][j] = 1$,
only if $P_k[i][k] = 1$ $P_{k-1}[k][j] = 1$.

So, we have two cases where $P_k[i][j] = 1$

Important

$P_k[i][j]$ { $P_{k-1}[i][j]$ or $P_{k-1}[i][k]$ and $P_{k-1}[k][j]$.

Ex



(1) Calculate adjacency matrix

$P_0 =$

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

Now find P_1

Now wherever $P_0[i][j] = 1$, $P_1[i][j] = 1$

And if $P_0[i][j] = 0$, then see $P_0[i][1]$ and $P_0[1][j]$ if both are 1 then $P_1[i][j] = 1$.