Unit-4

Graph: A graph is a set of vertices V and a set of edges E. A graph is a non-linear data structure. then G=(V,E).

Exp:

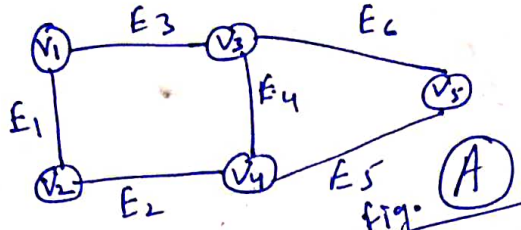


fig. (A)

$V(G) = \{V_1, V_2, V_3, V_4, V_5\}$

$E(G) = \{E_1, E_2, E_3, E_4, E_5, E_6\}$

Pradeep Chauhan
III - Sem
IT

Graph Terminology:

① **Adjacent vertices:** A vertex V_i is said to be adjacent to the vertex V_j if there is an edge between V_i and V_j.

Exp:- In fig. A V_1 is adjacent to V_2 and V_3.

② **Path:** A path is a combination of edges.

Exp:- In fig. A, a path between V_1 and V_4 is $E_3(V_1,V_3) - E_4(V_3,V_4)$ or $E_1(V_1,V_2) - E_2(V_2-V_4)$.

③ **Cycle:** A cycle is a path in which first and last vertices are same. In fig. A $(V_3 - V_5 - V_4 - V_3)$ is a cycle.

④ **Connected graph:** A graph is called connected if there exists a path between any two of its nodes.

⑤ **Complete graph:** A graph is said to be complete if every node is connected to all other nodes.
An undirected complete graph will contain $n(n-1)/2$ edges.
A Directed ——— $n(n-1)$ edges.
where n is total number of nodes in graph.

⑥ Weighted graph: A graph is said to be weighted graph if every edge in the graph is assigned some weight or value.

Exp:-

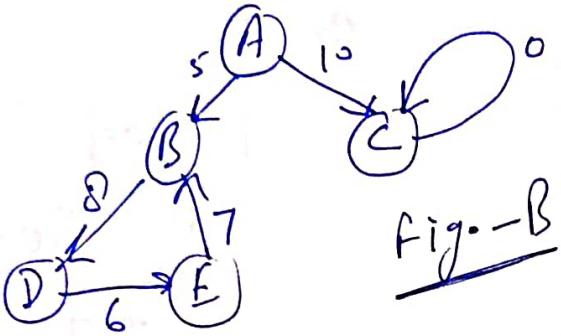


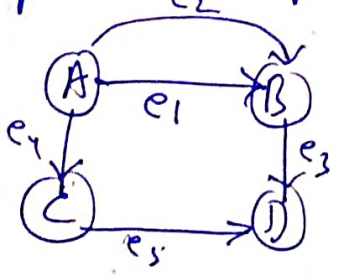
Fig.-B

⑦ Directed Graph: If edge in a graph is assigned a direction then graph is called directed graph or digraph.

Exp:- Fig. B

⑧ self loop:- If there is an edge whose starting and end vertices are same i.e. (c,c) in fig. B. is an edge then it is called a self loop or simply a loop.

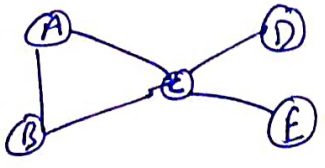
⑨ Parallel edges: If there are more than one edge bet^n the same pair of vertices then they are known as parallel edges.



Exp:- e_1 and e_2 are parallel edges.

⑩ Degree of vertex: The degree of a vertex is the number of edges incident to that node.

Exp:-



Degree of A = 2
B = 2
C = 4
D = 1
E = 1

In a directed graph, there are two degree for every node.

(i) In degree: The indegree of a vertex is the number of edges coming to that vertex or in other words, edges incident to it.

(ii) Out degree of a vertex: The out degree of a vertex is the number of edges going outside from that node or in other words, the edges incident from it.

Exp:-

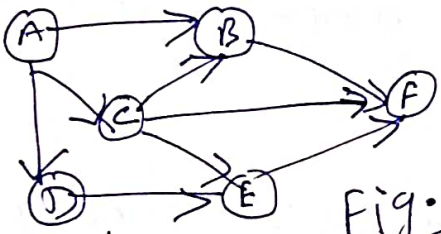


Fig. C

In degree of (A) = 0
(B) = 2
(C) = 1
(D) = 1
(E) = 2
(F) = 3

Out degree of (A) = 3
(B) = 1
(C) = 3
(D) = 1
(E) = 1
(F) = 0

Source: A vertex with zero indegree is called source.

Sink: ——— outdegree is called sink.

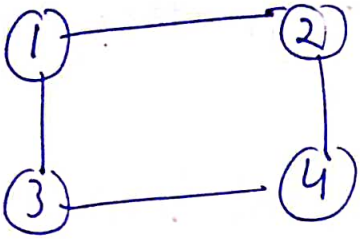
Exp:- In fig-C source is A & F is sink.

(11) Simple Graph: A graph or directed graph which does not have only self-loop or parallel edges is called simple graph.

(12) Multi-graph: A graph has either a self-loop or parallel edges or both is called a multi graph.

(13) Regular graph: A graph is regular if every node is adjacent to the same number of nodes.

Exp:



every node is adjacent to 2 nodes.

(14) cyclic graph:- If graph has cycle then it is cyclic graph.

(15) Acyclic graph:- If graph has no cycle then it is acyclic graph.

(16) Articulation point: If on removing a node from the graph, the graph becomes dis-connected then that node is called the articulation point.

(17) Bridge:- If on removing an edge from the graph, the graph becomes disconnected then that edge is called the bridge.

(18) Biconnected Graph: A graph with no articulation points is called a biconnected graph.

(19) Discrete Graph: A graph has a number of vertices but no edge is called discrete Graph.

Traversing a graph: Each node of graph G will be in one of three states called states of node N:

(i) Status=1: (Ready State) The initial state of the node N.

(ii) Status=2: (waiting State) The node N is on the Queue or stack, waiting to be processed.

(iii) Status=3: (Processed State) the node N has been processed.

Breadth First Search (BFS): complexity: O(V+E)

(1) Initialize all nodes to the ready state.

(2) Put the starting node A in Queue & change its status to the waiting state.

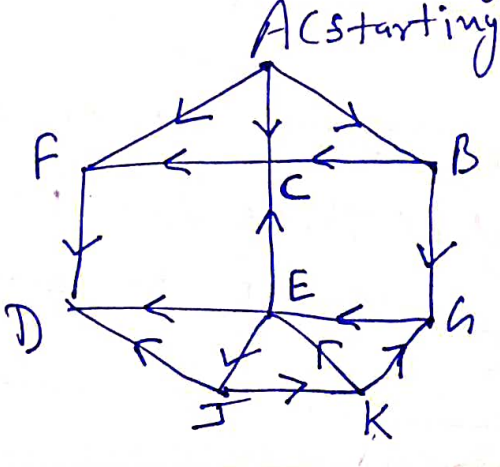
(3) Repeat steps (4) & (5) untill Queue is empty.

(4) Remove the front node N of Queue, process the N and change the status of N to processed state.

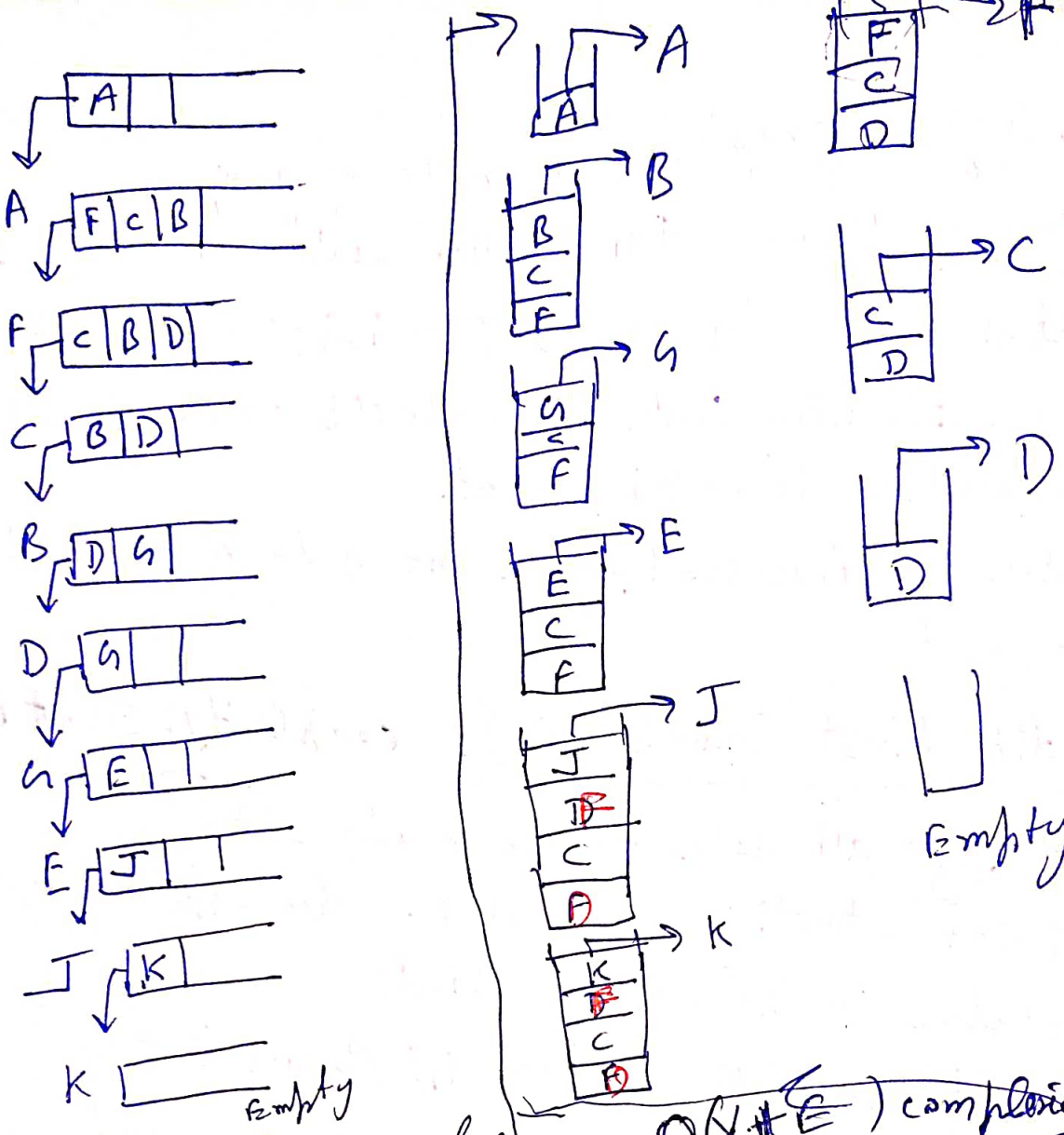
(5) Add to the rear of Queue all the neighbours of N that are in ready state and change their status to the waiting state.

(6) exit

Exp:-



⇒ First make Adjacency matrix



Spanning

cycle

Depth First Search:-

O(V+E) complexity

(1) Initialize all nodes to ready state.

(2) Push the starting node A onto STACK & change its status to waiting state.

(3) Repeat steps (4) & (5) untill STACK is empty.

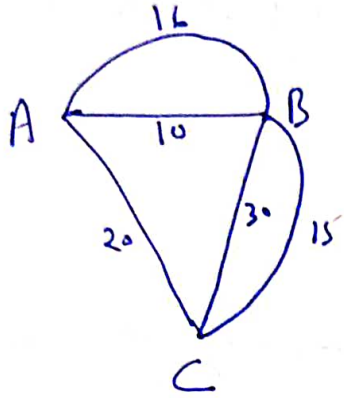
(4) Pop the front node N of STACK, Process N & change the status of N to processed state

(5) Push onto the TOP of STACK all the neighbours of N those are in Ready state and Change their status to waiting state.

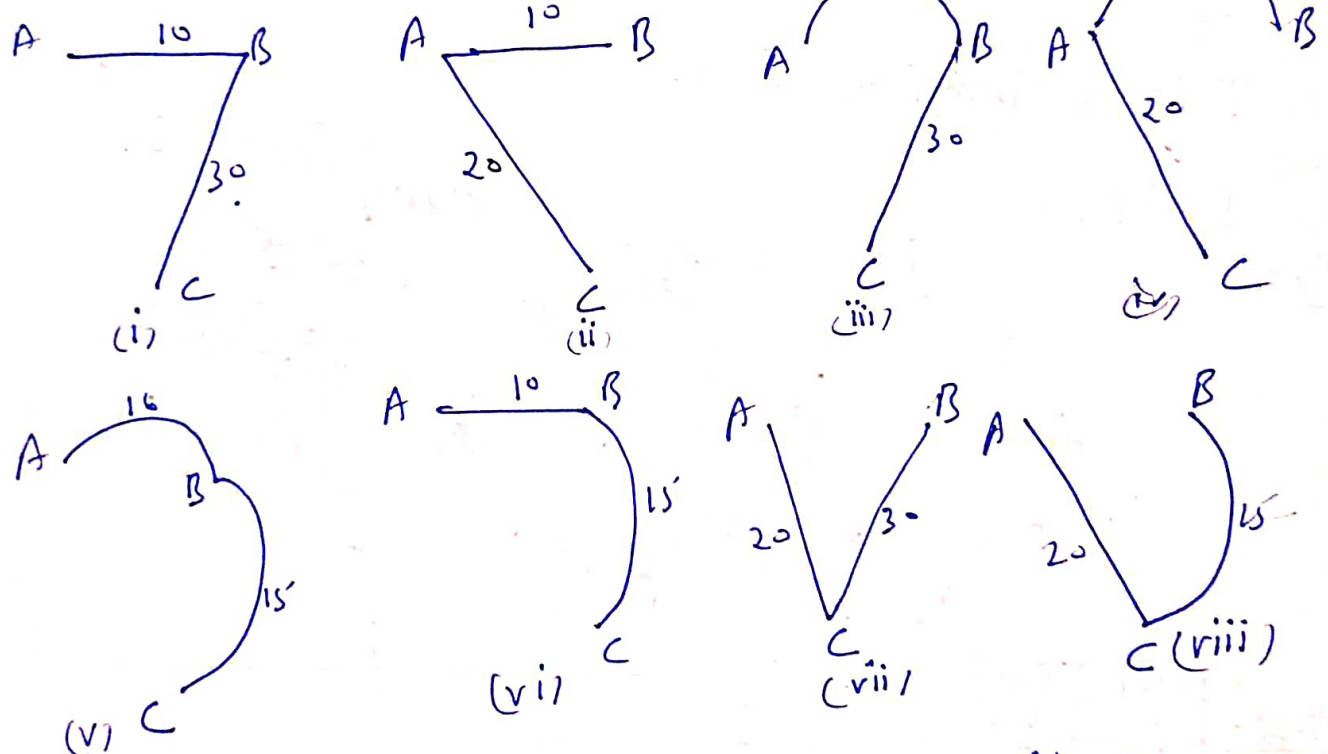
(6) Exit

Spanning Tree: A subgraph which covers all the node without making cycle is called spanning tree.

Exp:-



Different spannig tree are as follows:



Minimum Spanning Tree: **Kruskal's Algo:**

Let E be the set of all edges in graph G.
T is the minimum spanning tree. Then Kruskal's algo for minimum spanning tree is given as:

1. T = Null.
2. while T contains less then N-1 edges and E not empty repeat steps 3, 4 & 5.

③ choose an edge E(U,V) from E of lowest cost

④ Delete edge E(U,V) from E.

⑤ If E(U,V) does not create a cycle in T then add E(U,V) to T.

Else

Discard E(U,V).

⑥ If T contains less than (N-1) edges then print

"No spanning tree".

Else

T is a spanning Tree.

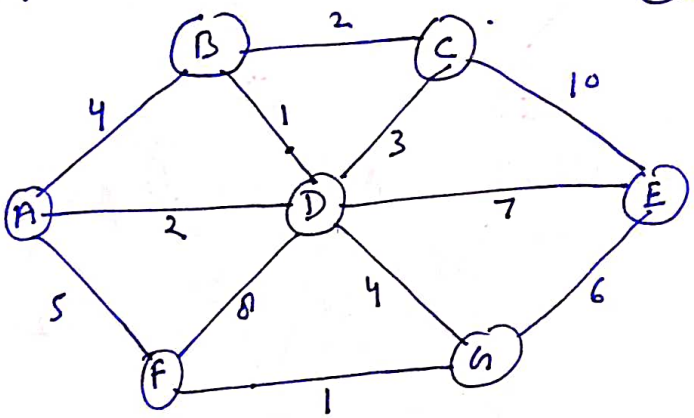
⑦ Exit.

complexity:

best case

O(E log V)

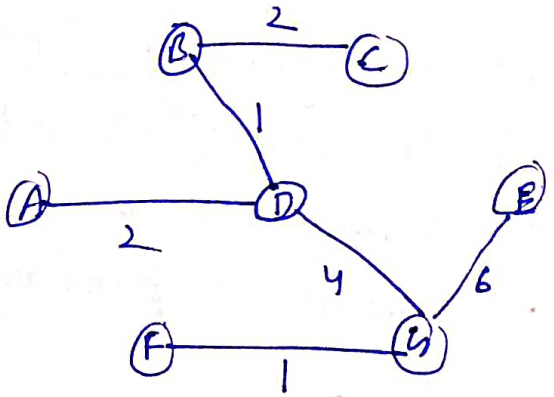
Exp:-



T

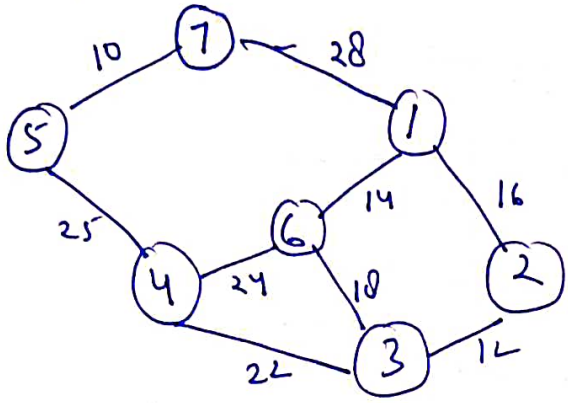
Ans:-

Cost	Edge	Decision
1	B D	select
1	F G	Select
2	A D	Select
2	B C	Select
3	C D	Discard
4	D G	Select
4	A B	Discard
5	A F	Discard
6	G E	Select
7	D E	Discard
8	F D	Discard
10	C E	Discard



Prim's Algorithm:

Exp:-



Initial:-

Nodes	Key	Parent	Status
1	∞	0	Temp
2	∞	0	Temp
3	∞	0	Temp
4	∞	0	Temp
5	∞	0	Temp
6	∞	0	Temp
7	∞	0	Temp

(1)

Nodes	Key	Parent	Status
1	∞	0	T
2	∞	0	T
3	∞	0	T
4	∞	0	T
5	∞	0	T
6	∞	0	T
7	0	0	T

(2)

Suppose node (7) is starting node, now (7) will taken

1	28	7
2	∞	0
3	∞	0
4	∞	0
5	10	7
6	∞	0
7	0	0 Perm

(3)

now we will take node 5

1	28	7	
2	∞	0	
3	∞	0	
4	25	5	
5	~~10~~	7	Perm
6	∞	0	
7	0	0	Perm

Now node 4 will be taken

1	28	7	
2	∞	0	
3	22	4	
4	~~∞~~ 25	5	Perm
5	10	7	Perm
6	24	4	
7	0	0	Perm

Now node 3 will be taken

1	28	7	
2	18	3	
3	22	4	Perm
4	25	5	Perm
5	10	7	Perm
6	18	3	
7	0	0	Perm

Now node 2 will be taken

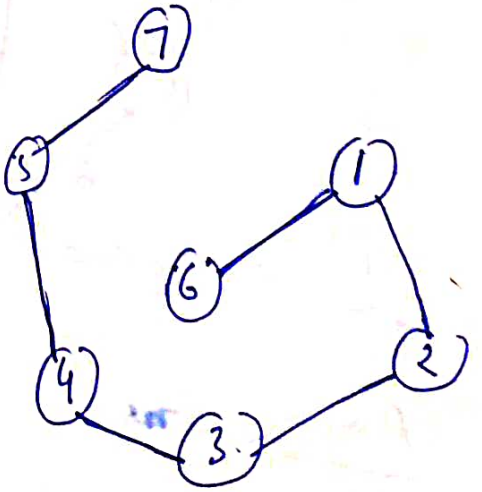
1	16	2	
2	12	3	Perm
3	22	4	Perm
4	25	5	Perm
5	10	7	Per
6	18	3	
7	0	0	Per

Now node 1 will be taken

1	16	2	Perm
2	12	3	Perm
3	22	4	Per
4	25	5	Per
5	10	7	Per
6	14	1	
7	0	0	Perm

Now node 6 will be taken

1	16	2	Perm
2	12	3	Perm
3	22	4	Perm
4	25	5	Pe
5	10	7	Per
6	14	1	Per
7	20	0	Pe

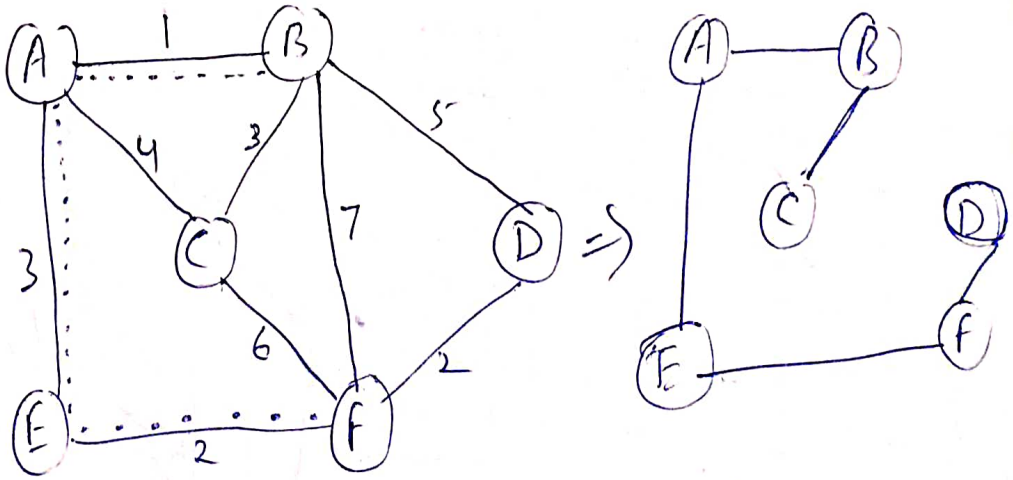


MST-Prim (G, w, r) :- T is MST.

(1) for each u ∈ V(G) repeat steps 2 & 3.
(2) do key[u] ← ∞
(3) π[u] ← NIL
(4) key[r] ← 0 & T = NULL
(5) Q ← V(G)
(6) While (Q ≠ ϕ) repeat steps 7 to 9
(7) do u ← EXTRACT-MIN (Q) and add to T
(8) For each v ∈ Adj (u) Repeat step 9
(9) if v ∈ Q and w(u,v) < key[v] then
π[v] ← u and key[v] ← w(u,v)
(10) Exit.

complexity: $O(V^2)$ $O(E \log V)$ Test

Exp:- 2



Ans:- Ans is shown by dotted line. Ans

DIJKSTRA'S ALGORITHM:

$O(V^2)$
best: $O(V+E \log V)$

Dijkstra's shortest path algorithm is a single source shortest path algo is based on relaxation method.

DIJKSTRA ALGO (G, w, s)

(1) For each vertex $u \in V(G)$ Repeat steps (2) & (3).
(2) do $d(u) \leftarrow \infty$
(3) $\pi(u) \leftarrow NIL$
(4) $d(s) \leftarrow 0$

} Initialize single source (G, s)

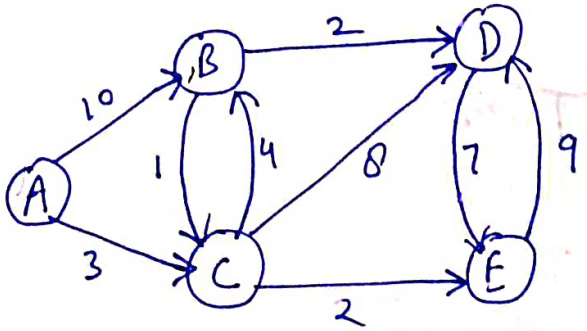
(5) $S \leftarrow \phi$ (NULL)
(6) $Q \leftarrow V(G)$
(7) while $(Q \neq \phi)$ repeat steps (8), (9), (10), (11)
(8) do $u \leftarrow$ EXTRACT-MIN(Q)
(9) $S \leftarrow S \cup u$
(10) for each vertex $v \in adj(u)$ Repeat steps (11)
(11) if $d(v) > d(u) + w(u,v)$ & $v \in Q$
then $d[v] \leftarrow d(u) + w(u,v)$ & $\pi(v) \leftarrow u$

} Relax (u, v, w)

(12) Exit

where $d(u) \overset{min}{=}$ distance of u from source node
$\pi(u)$ = Predecessor of u

Exp:-



Source node is A. $S = \{\}$

node	dist	pred	status
A	0	0	Temp
B	∞	0	Temp
C	∞	0	Temp
D	∞	0	Temp
E	∞	0	Temp

A is Extract from Q so S = {A}

A	0	0	Perm
B	10	A	Temp
C	3	A	Temp
D	∞	0	Temp
E	∞	0	Temp

now C is Extract from Q. S = {A, C}

A	0	0	Perm
B	7	C	Temp
C	3	A	Perm
D	11	C	Temp
E	5	C	Temp

now E is Extract from Q. S = {A, C, B}

A	0	0	Perm
B	7	C	Perm
C	3	A	Perm
D	11	C	Temp
E	5	C	Perm

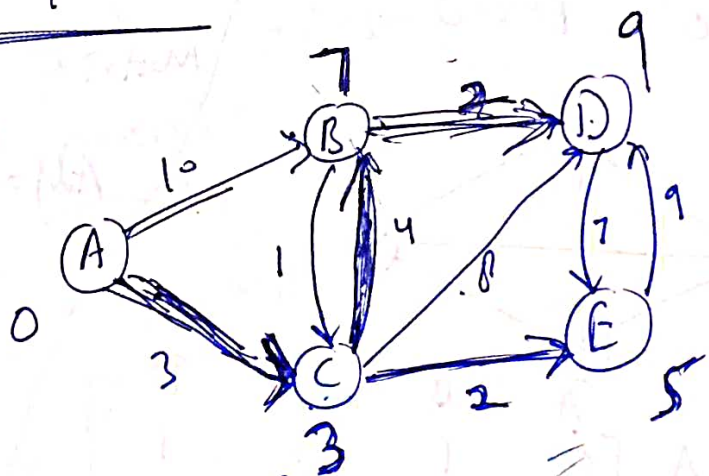
now B is Extract from Q. S = {A, C, E, B}

A	0	0	Perm
B	~~7~~	C	Per
C	3	A	Perm
D	9	B	Temp
E	5	C	Perm

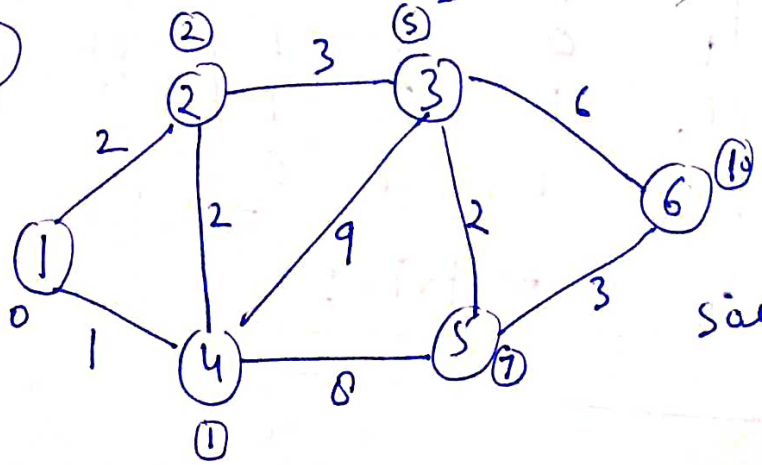
now D is Extract from Q. S = ~~{A, B, C, E, D}~~
S = {A, C, E, B, D}

A	0	0	Perm
B	~~7~~	C	Perm
C	3	A	Perm
D	9	B	Perm
E	5	C	Perm

So ans:-



Exp:- (2)



source is 1.

Adjacency Matrix:

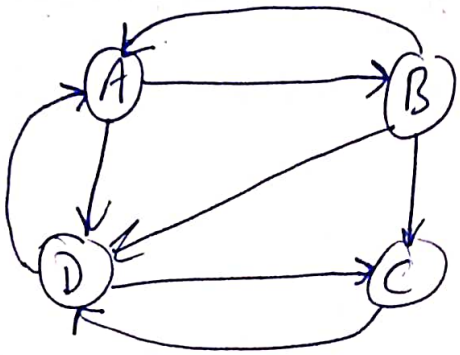
Adjacency matrix is the matrix which keeps the information of adjacent nodes.

Adj [i] [j] = 1 if there is an edge from node i to node j.

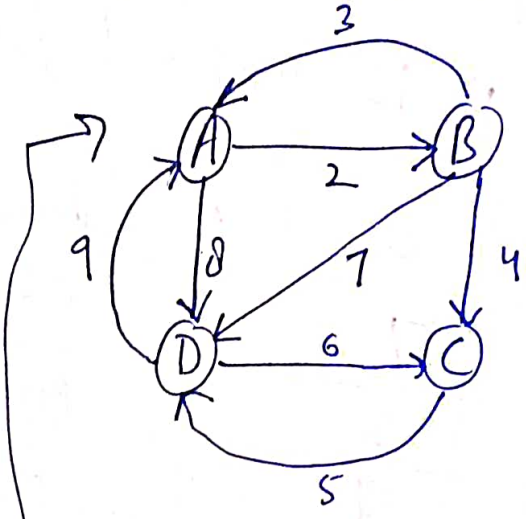
0 ——— no edge ———.

Hence, all the enteries of this matrix bill be either 1 or 0.

Exp:-①



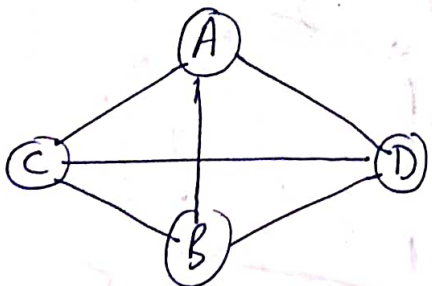
$$Adj = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$



Weighted Adjacency Matrix

$$Adj = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 2 & 0 & 8 \\ 3 & 0 & 4 & 7 \\ 0 & 0 & 0 & 5 \\ 9 & 0 & 6 & 0 \end{bmatrix} \end{array}$$

Exp:-②



Exp:-

$$Adj = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Weighted Adjacency Matrix:

Path matrix:

Let us take a graph G with n nodes $V_1, V_2, \ldots V_n$. The path matrix or reachable matrix of G can be defined as-

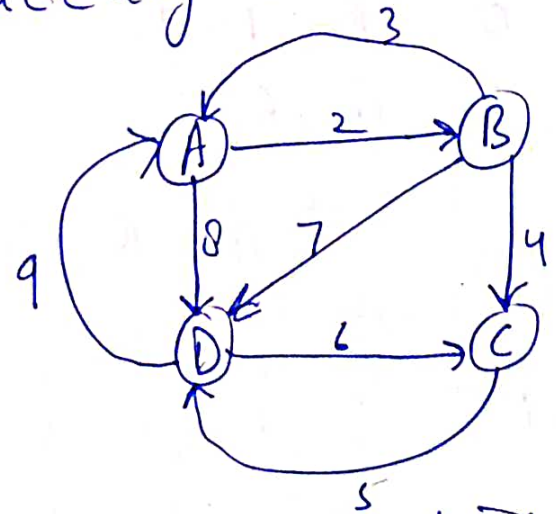
$$P[i][j] = \begin{cases} 1 & \text{if there is a path in between } V_i \text{ to } V_j \\ 0 & \text{otherwise.} \end{cases}$$

If there is a path from V_i to V_j then it can be a simple path from V_i to V_j of length n-1 or less or there can be a cycle of length n or less.

A graph G will be strongly connected if there are no zero enteries in path matrix.

Computing path matrix from adjacency matrix:-

Exp:- Compute path matrix for graph from its adjacency matrix



Weighted Adjacency matrix

	A	B	C	D
A	0	2	0	8
B	3	0	4	7
C	0	0	0	5
D	9	0	6	0

Adjacency metrix is

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

Path length = 1

$$AM_2 = A^2 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 2 & 0 & 2 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix} \end{array}$$

Path length = 2

$$AM_3 = A^2 M_2 \times A = A^3$$

$$= \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 2 & 1 & 4 \\ 3 & 1 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 3 & 1 \end{bmatrix} \end{array}$$

Path length = 3

$$AM_4 = A^3 \times A = A^4 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 6 & 1 & 6 & 4 \\ 4 & 3 & 4 & 7 \\ 3 & 0 & 3 & 1 \\ 1 & 3 & 1 & 6 \end{bmatrix} \end{array}$$

Path length = 4

So $X = A + A^2 + A^3 + A^4$

$$X = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 9 & 4 & 9 & 10 \\ 9 & 5 & 9 & 13 \\ 4 & 1 & 4 & 4 \\ 5 & 4 & 5 & 9 \end{bmatrix} \end{array}$$

we replace all non zero entery by 1

$$P = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

So Strongly connected

Warshall's Algorithm: Warshall's Algo is used to find path matrix:

Let us take a graph G of n vertices $V_1, V_2, V_3, \ldots V_n$.

First we will take boolean matrices $P_0, P_1, P_2, \ldots P_n$

where $P_k[i][j]$ is defined as -

$$P_k[i][j] = \begin{cases} 1 & \text{if there is a simple path from vertices } v_i \text{ to } v_j \text{ which does not use any other node except possibly } v_1, v_2, \ldots v_k \text{ or this path does not use any node numbered higher than } k. \\ 0 & \text{otherwise.} \end{cases}$$

Or we can say,

$P_0[i][j] = 1$ if there is a simple path from V_i to V_j, which does not use any node.

$P_1[i][j] = 1$ if there is a simple path from V_i to V_j which does not use any other nodes except possibly V_1.

$P_2[i][j] = 1$ if there is a simple path from V_i to V_j which does not use any other nodes except V_1, V_2.

$P_k[i][j] = 1$ if there is a simple path from V_i to V_j which does not use any other nodes except $V_1, V_2, \ldots V_k$

$P_n[i][j] = 1$ if there is a simple path from V_i to V_j which does not use any other nodes except $V_1, V_2, \ldots V_n$.

Here P_0 represents the adjacency matrix and P_n represents the Path matrix.

In this algo we will calculate $P_k[i][j]$ from $P_{k-1}[i][j]$.

There can be two cases for $P_{k-1}[i][j]$ -

(i) $P_{k-1}[i][j] = 1$ or (ii) $P_{k-1}[i][j] = 0$

(i) If $P_{k-1}[i][j] = 1$ means there is a path from node i to node j which does not use any other node except $V_1, V_2, \ldots V_{k-1}$.

If we include one more node V_k then definitely there will be a path from node i to node j which does not use any other node except $V_1, V_2, \ldots V_k$.

So | if $P_{k-1}[i][j] = 1$ then $P_k[i][j] = 1$

(ii) If $P_{k-1}[i][j] = 0$ then $P_k[i][j]$ can be 0 or 1.

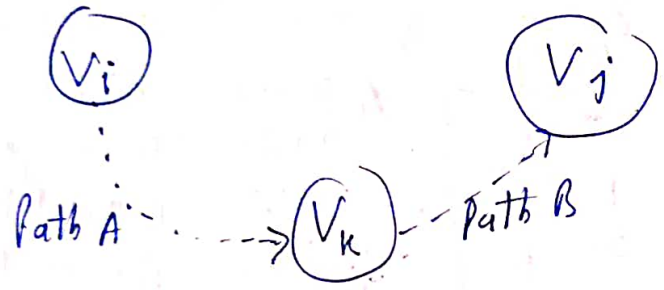
$P_k[i][j]$ will be 1 for $P_{k-1}[i][j] = 0$ when there is a path from node i to node j using nodes $V_1, V_2, \ldots V_k$, but there is no path from V_i to V_j using only nodes $V_1, V_2 \ldots V_{k-1}$. But if we use node V_k then we get a path from V_i to V_j

This means that there is a path from V_i to V_j which definitely passes through V_k and all other nodes in the path can be from nodes $V_1, V_2, -- V_{k-1}$.

Now we can break this path into two paths:

① Path A from V_i to V_k using Nodes $V_1, V_2, -- V_{k-1}$.

② Path B from V_k to V_j using Nodes $V_1, V_2, -- V_{k-1}$.



From Path A we can write that $P_{k-1}[i][k] = 1$

" B " $P_{k-1}[k][j] = 1$

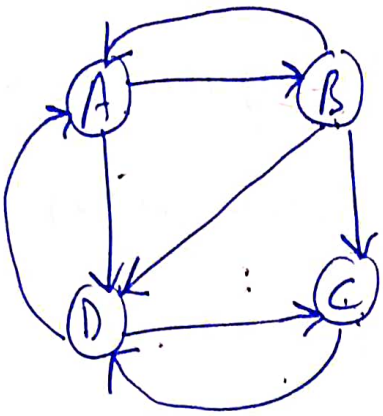
So if $P_{k-1}[i][j] = 0$ then $P_k[i][j]$ can be equal to 1 only if $P_{k-1}[i][k] = 1$ and $P_{k-1}[k][j] = 1$

~~Modified Warshall's Algorithm: It is used to find out the shortest path matrix~~

So the element of Path matrix P_k can be define as

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

Exp:-



P_0 is Adjacency matrix

$$P_0 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Now we have find P_1

$$P_1 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$P_1[1][1] = P_{01} \wedge P_{11}$

$[1][2]$ $P_{11} \wedge$

$[1][3]$ $P_{13} \wedge P_{13}$

$[1][4]$ $P_{11} \wedge$

$$P_2 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$P_3 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

$$P_4 = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

strongly connected because nonzero entry.

Warshall's algo for Path matrix: A directed graph G with N nodes is maintained in memory by its adjacency matrix A. This algo finds the path matrix P of graph G.

(1) Repeat for $i, j = 1, 2, \dots N$ (Find P_0)
  If $A[i][j] = 0$ then set $P[i][j] = 0$
  Else set $P[i][j] = 1$.

(2) Repeat steps (3), (4) & (5) for $k = 1, 2, \dots N$

(3) Repeat steps (4) (5) for $i = 1, 2, \dots N$

(4) Repeat step (5) for $j = 1, 2, \dots N$

(5) Set $P_k[i][j] = P_{k-1}[i][j] \vee (P_{k-1}[i][k] \wedge P_{k-1}[k][j])$

(6) Exit.

Advanced Warshall's algo for find shortest path bet' nodes:

Let G be a weighted graph with N nodes $V_1, V_2, \dots V_N$. Suppose Then G may be maintained in memory by its weight matrix $W = (W_{ij})$, defined as follows:

$$W_{ij} = \begin{cases} W(e) & \text{if there is an edge } e \text{ from } V_i \text{ to } V_j \\ 0 & \text{no edge from } V_i \text{ to } V_j \end{cases}$$

Here we will take matrices as $Q_0, Q_1, Q_2, \dots Q_N$.
$Q_k[i][j]$ is defined as follows:

$$Q_k[i][j] = \begin{cases} \text{length of shortest path from } V_i \text{ to } V_j \text{ using nodes } V_1, V_2, \dots V_k \\ \infty \quad \text{if there is no path from } V_i \text{ to } V_j \text{ using nodes } V_1, V_2, \dots V_k \\ 0 \quad \text{if } i = j \end{cases}$$

There is a path from v_i to v_j using nodes $v_1, v_2 \dots v_k$ in two conditions:

(i) If there is a path from v_i to v_j using $v_1, v_2 \dots v_{k-1}$.

(ii) There is a path from v_i to v_k using $v_1, v_2, \dots v_{k-1}$ and there is a path from v_k to v_j using $v_1, v_2, \dots v_{k-1}$.

So $$Q_k[i][j] = \text{Min}\left(Q_{k-1}[i][j],\ Q_{k-1}[i][k] + Q_{k-1}[k][j]\right).$$

~~We will calculate $Q_k[i][j]$ from $Q_{k-1}[i][k]$.~~

Here we will find Q_n from Q_{n-1}.

The initial matrix Q_0 is same as the weight matrix w except that each 0 in w is replaced by ∞ and if $i = j$ then $Q_{ij} = 0$.

Algo:-

(1) Repeat for $i, j = 1, 2, \dots N$ (~~Initialize~~ find Q_0)
If $i = j$ then $Q[i][j] = 0$
else if $w[i][j] = 0$ then set $Q[i][j] = \infty$
Else set $Q[i][j] = w[i][j]$

(2) Repeat steps (3), (4), (5) for $k = 1, 2, \dots N$.

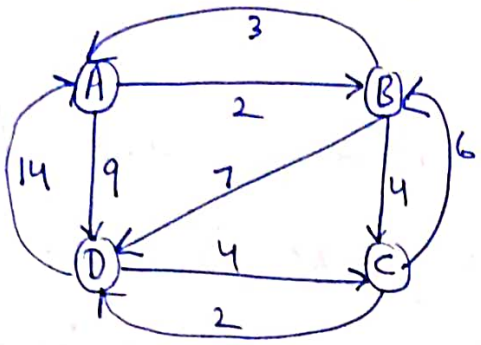
(3) Repeat step (4), (5) for $k = 1, 2, \dots N$

(4) Repeat step (5) for $j = 1, 2, \dots N$

(5) Set $Q_k[i][j] = \text{Min}\left(Q_{k-1}[i][j],\ Q_{k-1}[i][k] + Q_{k-1}[k][j]\right)$

(6) Exit.

Exp:-



$$W = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 0 & 2 & 0 & 9 \\ 3 & 0 & 4 & 7 \\ 0 & 6 & 0 & 2 \\ 14 & 0 & 4 & 0 \end{bmatrix} \end{array}$$

$Q_0 =$

	A	B	C	D
A	0	2	∞	9
B	3	0	4	7
C	∞	6	0	2
D	14	∞	4	0

	A	B	C	D
A	—	AB	—	AD
B	BA	—	BC	BD
C	—	CB	—	CD
D	DA	—	DC	—

$Q_1 =$

	A	B	C	D
A	0	2	∞	9
B	3	0	4	7
C	∞	6	0	2
D	14	16	4	0

—	AB	—	AD
BA	~~BAB~~	BC	BD
—	CB	—	CD
DA	DAB	DC	~~DAD~~

$Q_2 =$

0	2	6	9
3	0	4	7
9	6	0	2
14	16	4	0

~~ABA~~	AB	ABC	AD
BA	~~BAB~~	BC	BD
CBA	CB	~~CBC~~	CD
DA	DAB	DC	~~DAD~~

$Q_3 =$

0	2	6	8
3	0	4	6
9	6	0	2
13	10	4	6

~~ABA~~	AB	ABC	ABCD
BA	~~BAB~~	BC	BCD
CBA	CB	~~CBC~~	CD
DCBA	DCB	DC	~~DCD~~

Not C
14 = 1,3 ; 3,4
AB + CD

$Q_4 =$

0	2	6	8
3	0	4	6
9	6	0	2
13	10	4	6

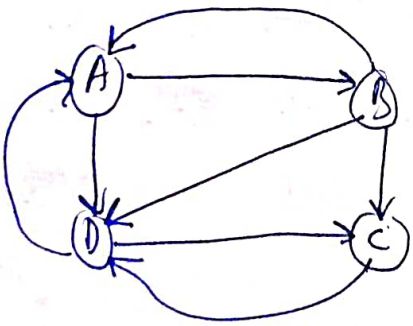
~~ABA~~	AB	ABC	ABCD
BA	~~BAB~~	BC	BCD
CBA	CB	~~CBC~~	CD
DCBA	DCB	DC	~~DCD~~

Transitive closure: The transitive closure of a graph G is defined to be the graph G' such that G' has the same nodes as G and there is an edge (V_i, V_j) in G' whenever there is a path from V_i to V_j in G.

Path matrix P of the graph G is precisely the adjacency matrix of its transitive closure G'.

so we can find out transitive closure G' by obtaining Path matrix using warshall's algo or Power of adjacency matrix.

Exp:-



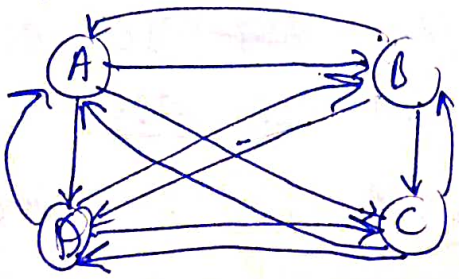
$$\text{Path matrix} = \begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

(G)

using warshall's algo or Power of adjacency matrix method

So graph from matrix is

$G' =$



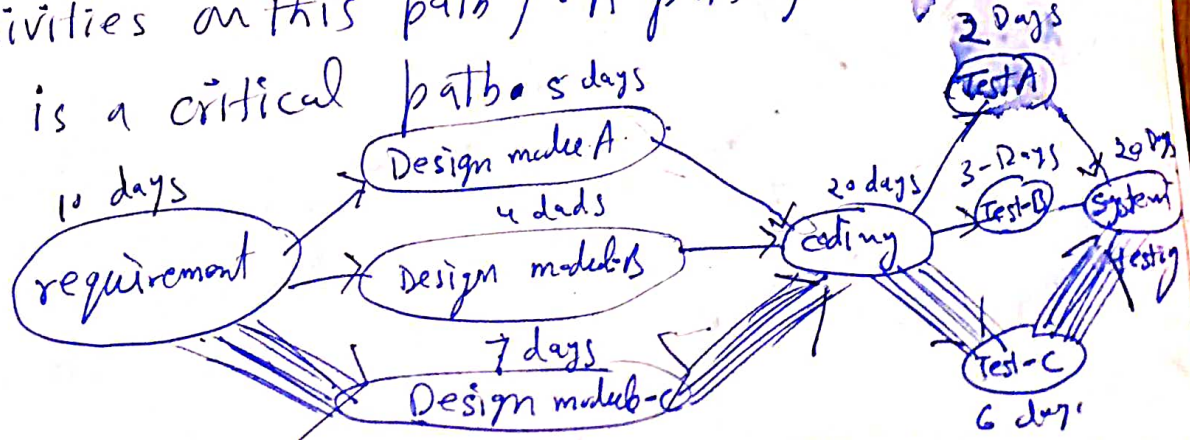
Activity on network:

To simplify a project, divide it into several subgroups called activities. The successful completion of these activities will result in a completion of entire project for e.g., A student can take DS if he had read programming previously but some other paper, he can take which does not depend on it. This relationship can be represented by a direded Graph in which the vertices represent task or activities and the edge represent precedence relations bet^n task is an activity in netwrk.

- Critical path :-

Since the activities in an activity network can be carried out in parallel, the minimum time to complete the project is the length of the longest path from start vertex to the the finish vertex (The length of a path is the sum of the times of activities on this path). A path of longest length is a critical path.

Eg:-



topological sort:

The linear ordering of vertices in a network is called topological order with the property that if i is a predecessor of j in the network then i precedes j in the linear ordering.

topological sort algorithm: (S is directed graph without cycle)

INDEG(N)

(1) Find the indegree of each node N of S.

(2) Put in a queue all the nodes with zero indegree.

(3) Repeat steps (4) and (5) until the queue is empty

(4) Remove the front node N of queue (By setting front = front+1)

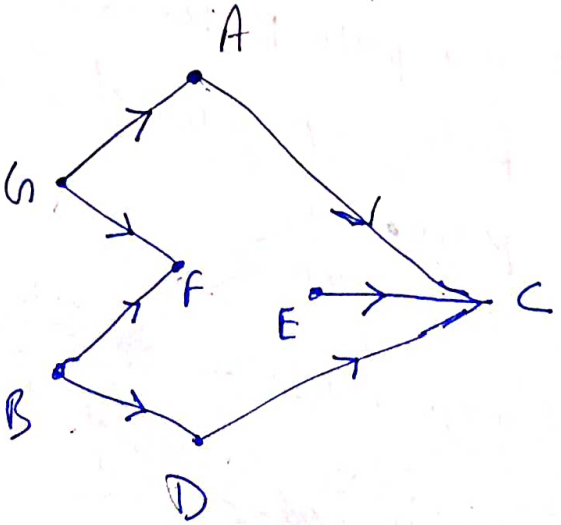
(5) Repeat the following for each neighbor M of node N:

(a) set INDEG(M) = INDEG(M) − 1 (This deletes the edge from N to M)

(b) If INDEG(M) = 0 then Add M to the rear of Queue.

(6) Exit

Exp:-



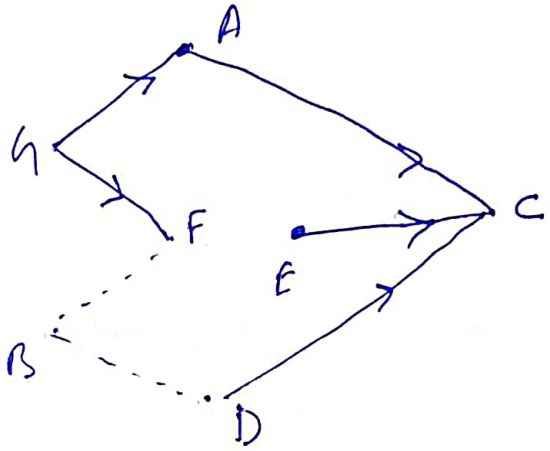
INDEG (A) = 1
——— (B) = 0
——— (C) = 3
——— (D) = 1

INDEG (E) = 0
——— (F) = 2
——— (G) = 0

| B | E | G | |

Queue

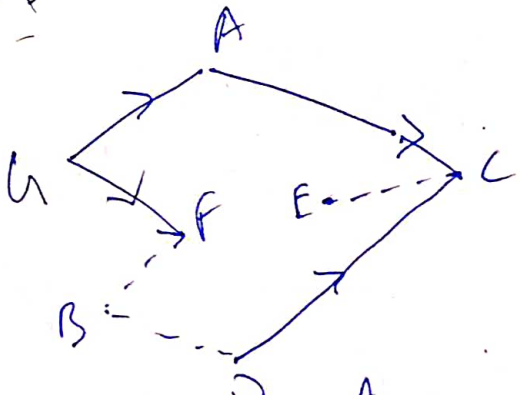
Remove (B) from Queue
neighbor D, F



Indeg (A) = 1
Indeg (C) = 3
——— (D) = 0
~~(E) = 2~~
——— (F) = 1
~~(G) = 2~~
Add D to Queue

| E | G | D | ← |

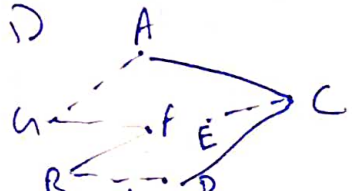
Remove (E) from Queue
neighbors C.



Indeg (A) = 1
——— (C) = 2
——— (F) = 1

| G | D |

Remove (G) from Queue
neighbors A, F



Indeg (A) = 0 , Indeg (C) = 2 , Indeg (F) = 0

Add A, F to Queue

D	A	F

Remove (D)

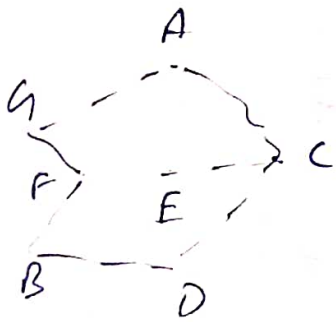
neighbor C

Indeg (C) = 1

A	F

Remove (A) ~~neighbor~~ from Q

neighbor C



Indeg (C) = 0

Add C to Q

F	G

Remove (F) from Q

C

Remove (C) from Q.

Empty.

So topological order | B, E, G, D, A, F, C. |

Representation of Graph in memory:

(1) Sequential Representation of Graph :-

Adjacency Matrix & path matrix.
(Already done)

(2) Linked Representation of Graph :-

In a sequential representation adding or deleting a node is required many changes in matrix. Second if number of edges in Graph is $O(n)$ (order of n) or $O(n \log_2 n)$ then matrix will be sparse. So memory will be wasted. In linked representation we need two lists, a node list NODE and an edge list EDGE, as follows:

(1) Node list: It has three fields

NODE	NEXT	ADJ

Here NODE will be the name or key value of node.
NEXT will be pointer to the next node in the node list.
ADJ will be pointer to the first element in the adjacency list of the node, which is

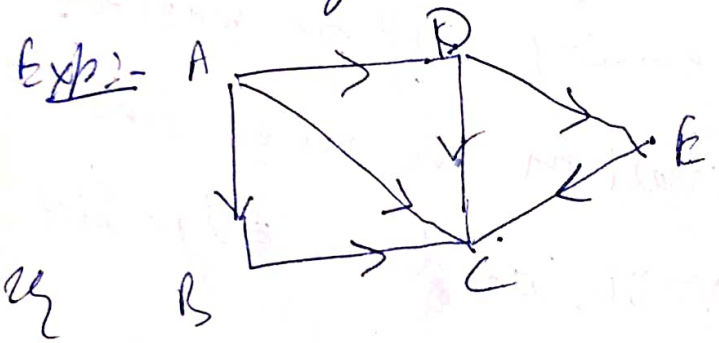
maintained in the list EDGE.

(b) Edge list: It has two fields

DEST	LINK

Here DEST will point to the location of the terminal node of the edge in the Node list.

LINK will link together the edges with the same initial node i.e., the nodes in the same adjacency list.

Ex:- 

Node	Adjacency list
A	B, C, D
B	C
C	—
D	C, E
E	C