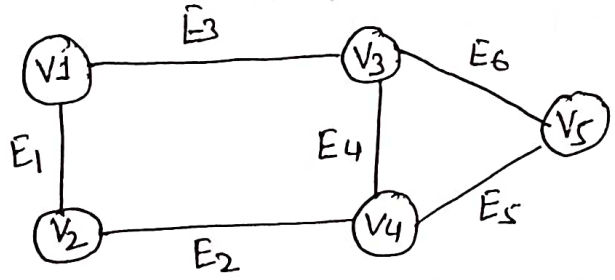
Unit-4

Graph

A graph is a set of vertices V and a set of Edges E.
A graph is a non-Linear data structure.

Graph Terminology :-

Example



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

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

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.

example → V1 is adjacent to V_2 and V_3 but Not V_4 and V_5.

(2) path: A path is a combination of edges.

Example → 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.

example: $(V_3 - V_5 - V_4 - V_3)$ is a cycle.

(4) Connected Graph: A graph is called connected if there exists a path between any two of its vertices.

(5) Complete graph: A graph is said to be complete if every node is connected to all other node (vertex).

★ An undirected complete graph will contain $n(n-1)/2$ edges.

★ A directed complete graph will contain $n(n-1)$ edges. where n is the total number of vertices 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.

Example

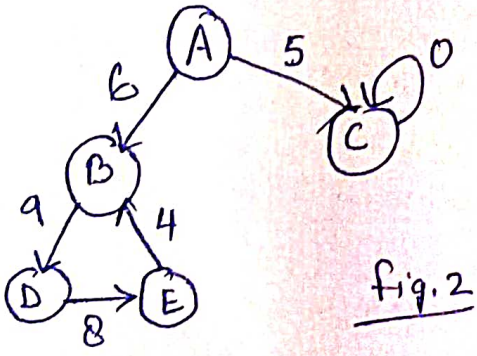


fig. 2

(7) Directed graph :- if an edge in a graph is assigned a direction then graph is called directed graph or diagraph.

(8) Self Loop :- If there is an edge whose starting and end vertices are same i.e. (C, C) in fig 2.

(9) Parallel edges:- If there are more than one edge between the same pair of vertices then they are known as parallel edges.

example →

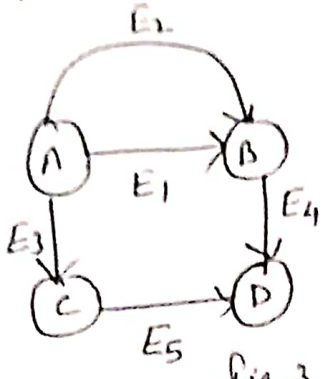


fig-3

E_1 and E_2 are parallel edges.

10) Degree of vertex graph - The degree of vertex is the number of edges incident to that vertex.

Example:-

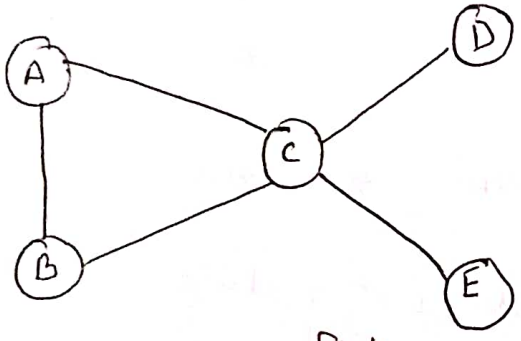


fig 4

Degree(A) = 2 Degree(B) = 2 Degree(C) = 4

Degree(D) = 1 Degree(E) = 1.

In a directed graph there are two degree for every vertex

(i) In degree:- The indegree of vertex is the number of edges coming to that vertex or edges incident to it.

(ii) Out degree: The out degree of a vertex is the

number of edges going outside from vertex or edges incident from it.

Example:-

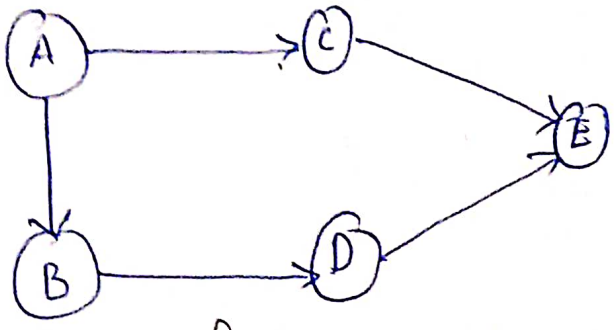


fig 5.

Indegree(A) = 0 outdegree(A) = 2

Indegree(B) = 1 outdegree(B) = 1

Indegree(C) = 1 outdegree(C) = 1

Indegree(D) = 1 outdegree(D) = 1

Indegree(E) = 2 outdegree(E) = 0

Source:- A vertex with zero indegree is called source.

Sink:- A vertex with zero outdegree is called sink.

for example in fig 5 source is A and E is a sink.

11) Simple Graph:- A graph or directed graph which does not have 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 vertex adjacent to same number of vertex.

example

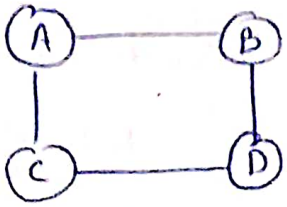


fig 6

In fig 6 Every vertex is adjacent to 2 vertex.

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 vertex from the graph, the graph becomes disconnected then that vertex 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) Bi connected graph- A graph with no articulation points is called a biconnected graph.

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

Traversing a graph:- Each vertex of graph G will be in one of three states called status of Vertex N.

(i) status=1 (Ready state) - The initial state of the Node (vertex) N.

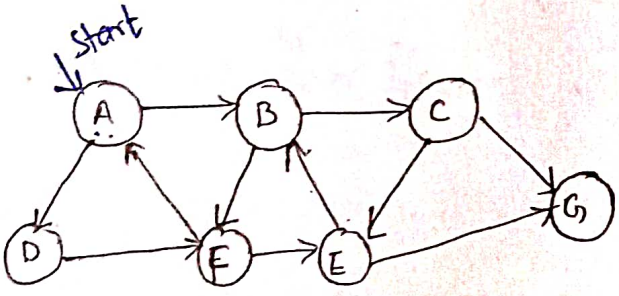
ii) status = 2 (waiting state) - The node N is on the Queue or stack, waiting to be processed.

iii) status=3 (processed state) - The node N has been processed.

Breadth First Search:- Breadth first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes.

BFS( )

1. Initialize all nodes to the ready state.
2. Put the starting node A in Queue and exchange its status to the waiting state.
3. repeat step ④ & ⑤ until 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.



Adjancy List.

A : B, D

B : C, F

C : E, G

G : E

E : B, F

F : A

D : F

b

Queue1 – holds the nodes that are to be processed.

Queue2 – All the nodes that are processed and deleted from Queue1.

Let starting Node A and end node G.

Step 1: First, add A to queue1 and NULL to queue2.

Queue1 = { A }

Queue2 = { NULL }

Step 2: Now delete node A from queue1 and add it into queue2. Insert all neighbours of node A to queue1.

Queue 1 = { B, D }

Queue 2 = { A }

Step 3: Now delete B from Queue1 and add it to queue2. Insert all neighbours of node A to queue 1.

Queue1 = { D, C, F }

Queue 2 = { A, B }

Step 4: Now, delete node D from queue1 and add it to queue2. Insert all neighbours of D to queue1. The only neighbor of D is F, since it is already inserted so it will be not be inserted again.

Queue 1 = { C, F }

Queue2 = { A, B, D }

Step 5: Delete node E from queue1 and add it into queue2. Insert all neighbours of node E ~~are already present we will not insert them again~~. to queue1.

Queue 1 = { F, E, G }

Queue 2 = { A, B, D, C }

Step 6 Delete node F from queue 1 and add its into queue 2. Insert all neighbours of node F to queue 1. Since all neighbours of node F are already present, we will not insert them again.

Queue 1 = { E, G }

Queue 2 = { A, B, D, C, F }

Step 7. Delete node E from queue 1. Since all of its neighbours have already been added, so will not insert them again. Now, all the nodes are visited and the target node E is encountered into queue 2.

Queue 1 = { G }

Queue 2 = { A, B, D, C, F, E }

Step 8. Delete Node G. and add to queue 2. Add all neighbours of G to queue 1. All neighbors are already inserted so will not insert again.

Queue 1 = { NULL }

Queue 2 = { A, B, D, C, F, E, G }

Complexity - O(V+E)

Depth First Search (DFS) - This algorithm starts with the initial node of a graph G and goes deeper untill we find the goal node. or

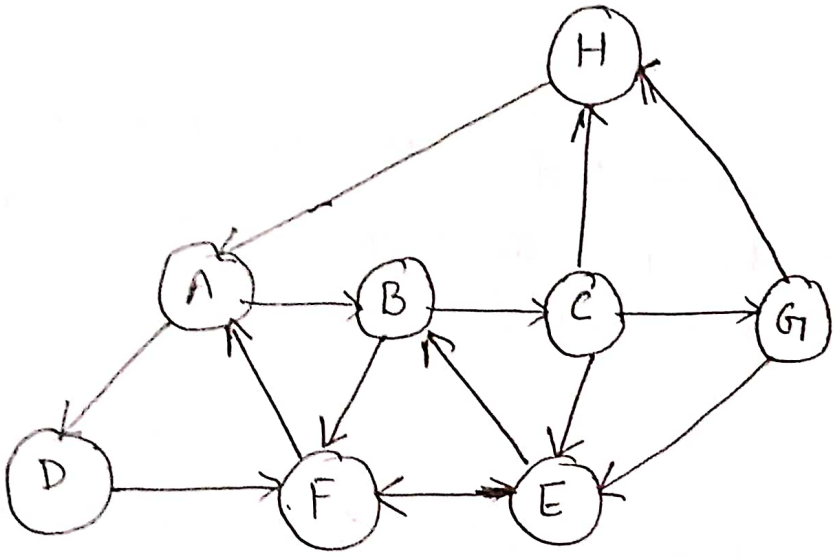
Algorithm - DFS( )

1. Initially all nodes to ready state.
2. Push the starting node A onto stack & change its status to waiting state.
3. repeat step 4 and 5 untill stack is empty.

4. pop the ~~found~~ front node N of stack, process N and 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.



Adjancy matrix

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

Step1: First, H onto the stack

STACK: H

Step2: pop the top element from the stack, and print it. Now push all the neighbours of H onto the stack that are in ready state.

Print H] STACK: A

Step3: pop the top element from the stack, i.e. A, and print it. Now PUSH all the neighbours of A onto the stack that are in ready state.

Print A
STACK: B, D

Step 4: pop the top element from STACK, i.e. A and print it. Now push all the neighbors of D onto the stack that are in ready state.

print D

STACK: B, F

Step 5: pop the top element from the stack, i.e. F and print it. Now push all the neighbors of F onto stack that are in ready state.

print F

STACK: B

Step 6: pop the top element from the stack, i.e B and print it. Now push all neighbors of B onto the stack. that are in ready state.

print B

STACK: C

Step 7: pop top element from STACK, i.e. C and print it. push ~~Add~~ all neighbors of C onto stack that are in ready state.

print C

STACK: E, G

STEP 8: pop top element from STACK. i.e., G and print it. push all neighbors of G onto stack that are in ready state

print G

STACK: E

STEP 9: pop top element from STACK, i.e. E and print it. Add all neighbors of E onto STACK that are in ready state.

print E

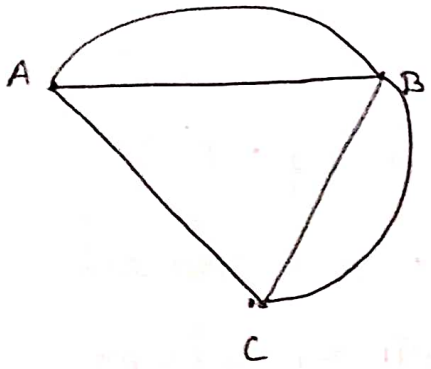
STACK: empty

now all the nodes have been traversed, and the stack is empty.

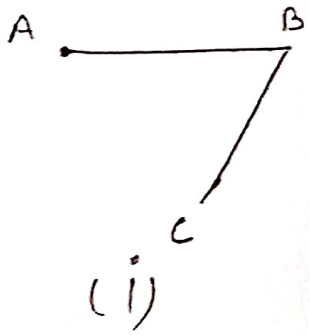
Complexity → $O(V+E)$

Spanning Tree → A subgraph which covers all the nodes without cycle is called spanning tree.

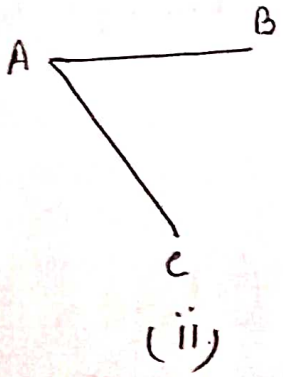
Example-



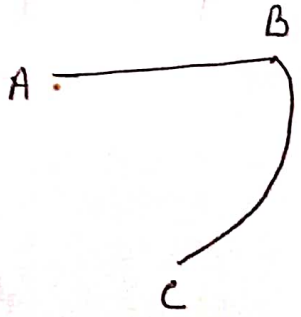
Different spanning tree are as follows:-



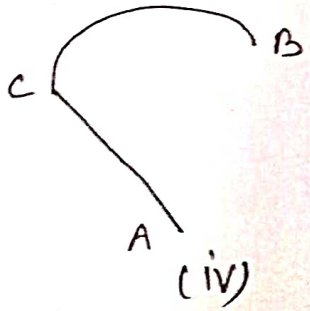
(i)



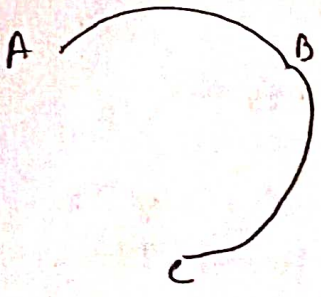
(ii)



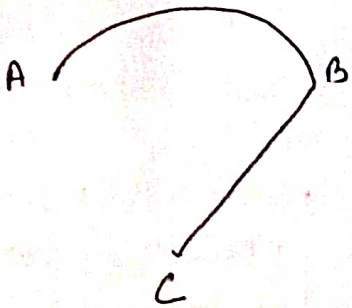
(iii)



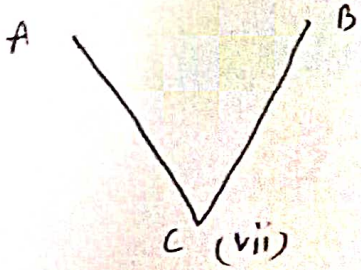
(iv)



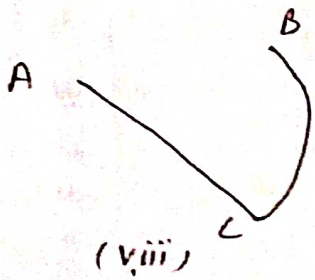
(v)



(vi)



(vii)



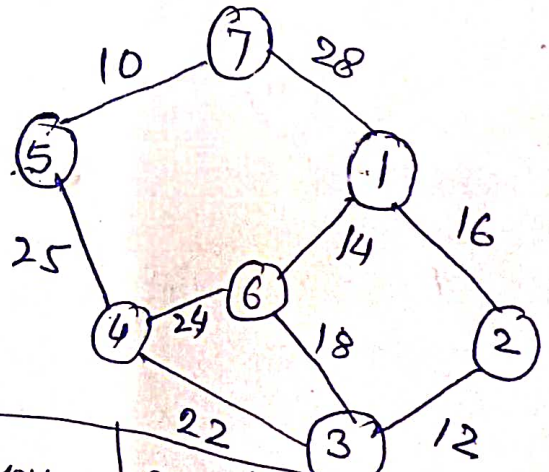
(viii)

Prim's Algorithm –

MST_prim (G, W, r)

1. for each $u \leftarrow V(G)$ repeat step 2 & 3.
2. key[u] $\leftarrow \infty$
3. $\pi(u) \leftarrow$ NIL
4. key[r] $\leftarrow 0$ And T = NULL
5. $Q \leftarrow V(G)$
6. while ($Q \neq 0$) repeat step 7 to 9
7. $u \leftarrow$ Extract_MIN(Q) and add to T
8. for each $V \in Adj(u)$ repeat step 9
9. if $V \in Q$ and $w(u,V) < key[V]$ then.
   $\pi[V] \leftarrow u$ and $key[V] \leftarrow w(u,v)$
10. Exit.

Example:-



Initial

Initial	Nodes	key	Parent	Status
1	1	∞	–	Temp
2	2	∞	–	"
3	3	∞	–	"
4	4	∞	–	"
5	5	∞	–	"
6	6	∞	–	"
7	7	∞	–	"

N	Key	Parent	status
1	∞		Temp
2	∞		Temp
3	∞		Temp
4	∞		Temp
5	∞		Temp
6	∞		Temp
7	0	-	Temp

(2)

min=7 select 7

N	Key	P	S
1	28	7	Temp
2	∞	-	Temp
3	∞	-	Temp
4	∞	-	Temp
5	10	7	Temp
6	∞	-	Temp
7	0	-	Per

(3)

min=5 select 5

N	Key	P	S
1	28	7	Temp
2	∞	-	Temp
3	∞	-	Temp
4	25	5	Temp
5	10	7	Per
6	∞	-	Temp
7	0	-	Per

(4)

min=25 select 4

N	Key	P	S
1	28	7	Temp
2	∞	4	Temp
3	22	4	Temp
4	25	5	Per
5	10	7	Per
6	24	4	Temp
7	0	-	Per

(5)

min=22 select 3

N	Key	P	S
1	28	7	Temp
2	12	3	Temp
3	22	4	Per
4	25	5	Per
5	10	7	Per
6	18	3	Temp
7	0	-	Per

(6)

min=12 select 2

N	Key	P	S
1	16	2	Temp
2	12	3	Per
3	22	4	Temp Per
4	25	5	Per
5	10	7	Per
6	18	3	Temp
7	0	-	Per

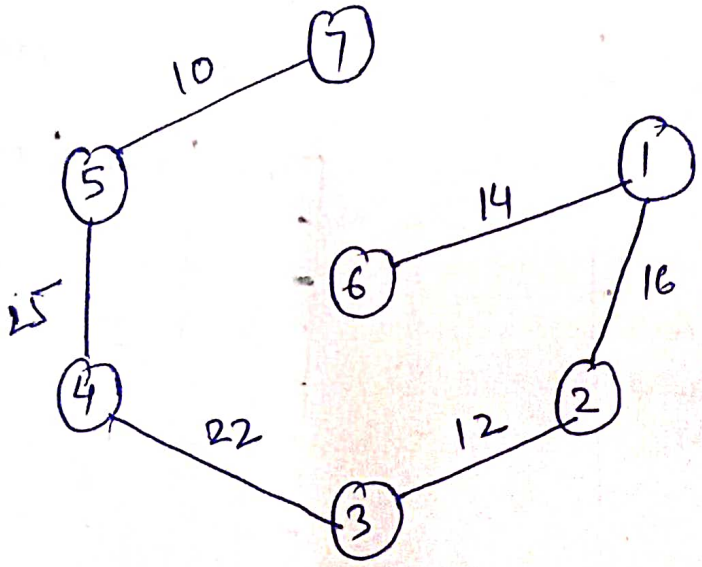
(7)

min=16 select 1

N	Key	P	S
1	16	2	Per
2	12	3	Per
3	22	4	Per
4	25	5	Per
5	10	7	Per
6	14	1	Temp
7	0	-	Per

(8)

N	key	P	S
1	16	2	Per
2	12	3	Per
3	22	4	Per
4	25	5	Per
5	10	7	Per
6	14	1	Per
7	0	-	Per



Kruskal's Algorithm – Kruskal (T, G, W)

let E be the set of all edges in graph G. T is the minimum spanning tree.

1. T = NULL
2. While T contains less than N-1 edges and E is not empty repeat steps (3), (4) & 5.
3. choose an edge E(U, V) from E with min 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

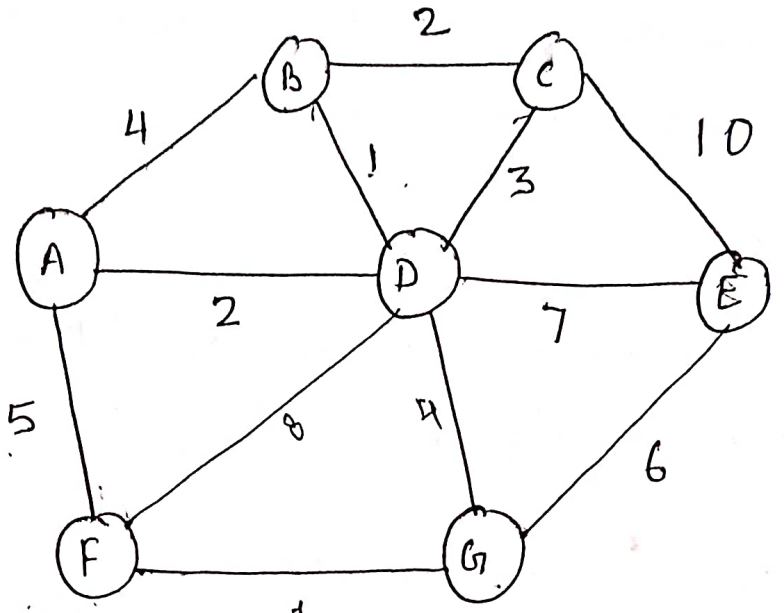
   Discard E(U, V)
6. if T contains less than (N-1) edges then print

   "No spanning tree

   else

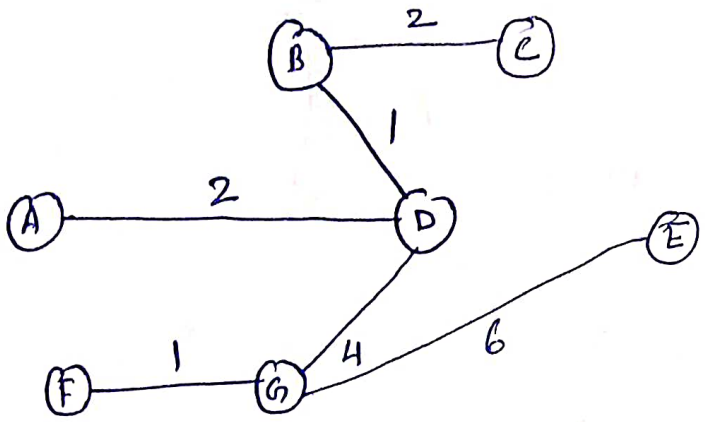
   print "T is spanning tree"
7. Exit.

Example →



E = {(A,B), (B,C), (C,E), (G,E), (F,G), (A,F), (A,D), (D,E), (F,D), (G,D), (B,D), (D,C)}

Cost	Edge	Decision
1	BD	select
1	FG	Select
2	AD	Select
2	BC	select
3	CD	Discard
4	DG	Select
4	AB	Discard
5	AF	Discard
6	GE	Select
7	DE	Discard
8	FD	Discard
10	CE	Discard



Dijkstra's Algorithm:- Dijkstra's shortest path algorithm is ~~based on relaxation~~ a single source shortest path algorithm based on relaxation method.

Algorithm:- Dijkstra (G, w, s)

1. for each vertex $u \in V(G)$ repeat step 2 and 3
2. $d(u) \leftarrow \infty$
3. $P(u) \leftarrow NIL$
4. $d(s) \leftarrow 0$
5. $S \leftarrow \emptyset$ (NULL)
6. $Q \leftarrow V(G)$
7. while ($Q \neq 0$) repeat step 8, 9, 10, 11
8. $u \leftarrow$ Extract_min(Q)
9. $S \leftarrow S \cup u$
10. for each vertex $v \in adj(u)$ repeat step 11
11. if ($d(v) > d(u) + w(u,v)$ And $v \in Q$) then.

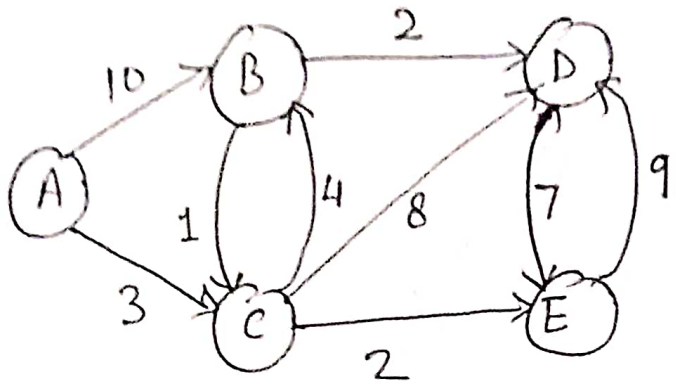
$d(v) = d(u) + w(u,v)$ And $P(v) \leftarrow u$

where $d(u)$ = min distance of u from source

$P(u)$ = predecessor of u

12. Exit.

Example →



Source node is A $S = \{\ \}$ $Q = \{A, B, C, D, E\}$

node	distance	Pred.	status
A	0	–	Temp
B	∞	–	Temp
C	∞	–	Temp
D	∞	–	Temp
E	∞	–	Temp

(1)

A is Extract from Q. so $S = \{A\}$ $Q = \{B, C, D, E\}$

node	distance	Pred	status
A	0	–	Per
B	10	A	Temp
C	3	A	Temp
D	∞	–	Temp
E	∞	–	Temp

(2)

at min = 3 now C is extract from Q. $S = \{A, C\}$, $Q \{B, D, E\}$

node	distance	Pred	status
A	0	-	per
B	7	C	Temp
C	3	A	per
D	11	C	Temp
E	5	C	Temp

(3)

min = 5 now E is extracted from Q. $S = \{A, C, E\}$, $Q = \{B, D\}$

node	dist	Pred	status
A	0	-	per
B	7	C	Temp
C	3	A	per
D	11	C	Temp
E	5	C	per

(4)

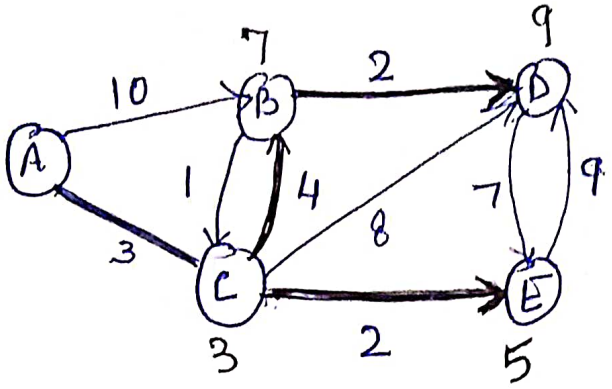
min => now B is extracted from Q. $S = \{A, C, E, B\}$ $Q = \{D\}$

node	dist	pred	status
A	0	-	per
B	7	C	per
C	3	A	per
D	9	B	Temp
E	5	C	per

(5)

now D is extracted from Q. $S = \{A, C, E, B, D\}$ $Q = \{\}$

node	dist	pred	status
A	0	-	per
B	7	C	per
C	3	A	per
D	9	B	Per
E	5	C	per

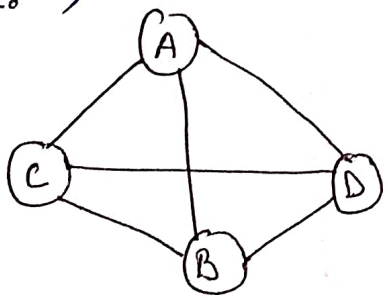


Adjancy matrix → Adjancy matrix is the matrix which keeps the information of adjacent nodes.

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

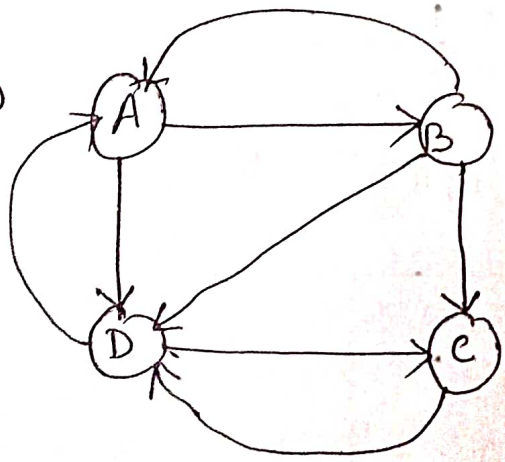
= 0 no edge

Example:- 1)



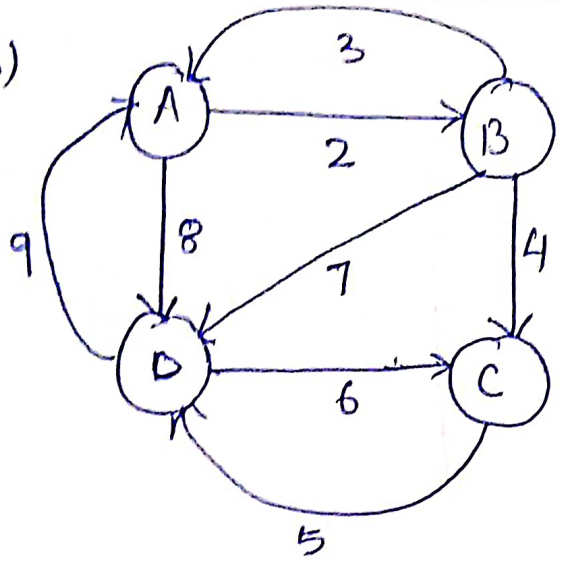
	A	B	C	D
A	0	1	1	1
B	1	0	1	1
C	1	1	0	1
D	1	1	1	0

2)



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

(3)



weighted Adjancy 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

path matrix → Let us take a graph G with n Nodes $V_1, V_2, \ldots V_n$.

The path matrix or rechable matrix of G can be defined as

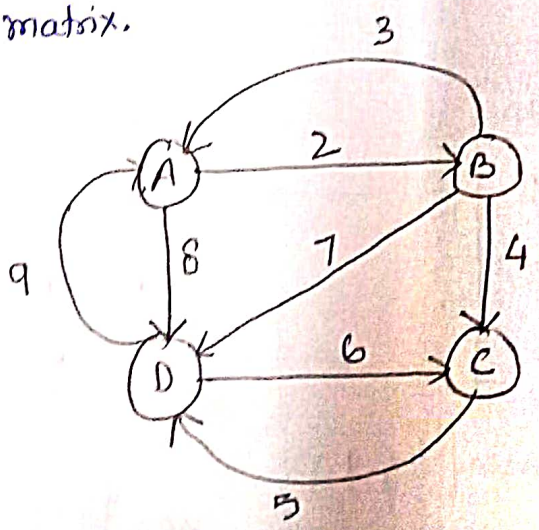
$$P[i][j] = \begin{cases} 1 & \text{if there is a path in between } V_i \text{ and } V_j \\ 0 & \text{otherwise} \end{cases}$$

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 is will be strongly connected if there are no zero entries in path matrix.

Computing path matrix from adjency matrix :-

Example: complete path matrix for graph from its adjancy matrix.



Adjacy matrix 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

$$AM_2 = A^2 = \begin{bmatrix} 2 & 0 & 2 & 1 \\ 1 & 1 & 1 & 2 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 \end{bmatrix}$$

path length = 2

$$AM_3 = AM_2 \times A = A^3 = \begin{bmatrix} 1 & 2 & 1 & 4 \\ 3 & 1 & 3 & 3 \\ 0 & 1 & 0 & 2 \\ 3 & 0 & 3 & 1 \end{bmatrix}$$

path length = 3

$$AM_4 = AM_3 \times A = A^4 = \begin{bmatrix} 6 & 1 & 6 & 4 \\ 4 & 3 & 4 & 7 \\ 3 & 0 & 3 & 1 \\ 1 & 3 & 1 & 6 \end{bmatrix}$$

path length = 4

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

$$X = \begin{bmatrix} 9 & 4 & 9 & 10 \\ 9 & 5 & 9 & 3 \\ 4 & 1 & 4 & 4 \\ 5 & 4 & 5 & 9 \end{bmatrix}$$

replace all no zero entry by 1 →

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

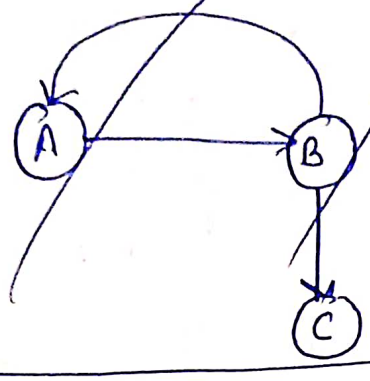
so strongly connected.

Warshall's Algorithm → Warshall's Algorithm 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 take boolean matrices

Example:-



Algorithm →

n = no. of vertices

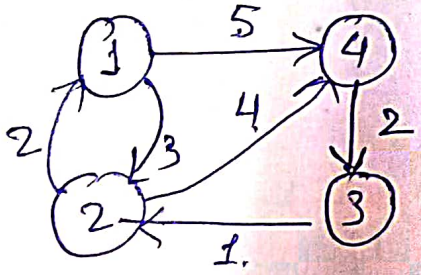
A = matrix of dimension n * n.

```
for ( k=1 to n)
    for ( i=1 to n)
        for (j=1 to n)
```

$A^k[i,j] = \min(A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j])$

return A

example →



$A^0 =$

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

Now create A^1 using matrix A^0. The elements in the first column and the first row are left as they are. Remaining columns are filled in following way

let K → intermediate vertex from S TO D. In this step K = 1

A[i][K] + A[K][j] if (A[i][j] > A[i][K] + A[K][j]).

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & & \\ \infty & & 0 & \\ \infty & & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & \infty & 5 \\ 2 & 0 & 9 & 4 \\ \infty & 1 & 0 & 8 \\ \infty & \infty & 2 & 0 \end{bmatrix} \end{array}$$

Similar A^2 is created using A^1. The elements in second column and second row are left as they are. K = 2

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & & \\ 2 & 0 & 9 & 4 \\ & 1 & 0 & \\ & \infty & & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & \infty & 2 & 0 \end{bmatrix} \end{array}$$

K = 3.

$$A^3 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & & 9 & \\ & 0 & 9 & \\ 3 & 1 & 0 & 5 \\ & & 2 & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 9 & 5 \\ 2 & 0 & 9 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

K = 4

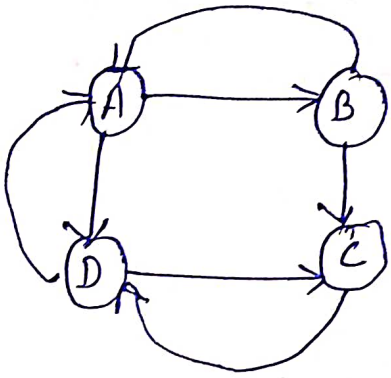
$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & & & 5 \\ & 0 & & 4 \\ & & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array} \longrightarrow \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 7 & 5 \\ 2 & 0 & 6 & 4 \\ 3 & 1 & 0 & 5 \\ 5 & 3 & 2 & 0 \end{bmatrix} \end{array}$$

A^4 gives the shortest path between each pair of vertices.

Transitive closure :- The transitive closure of a Graph G is defined to be graph G' such that G' has 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 of the graph G is precisely the adjacency matrix of its transitive closure G'.

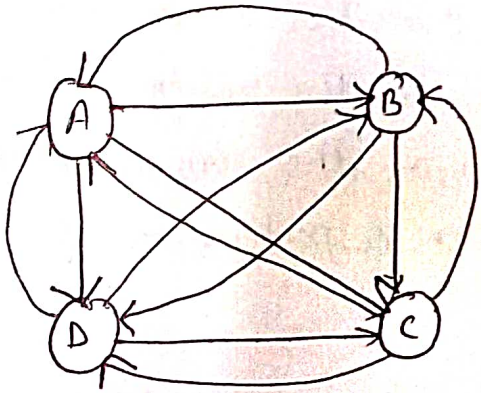
Example →



Path matrix

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

So graph from matrix is

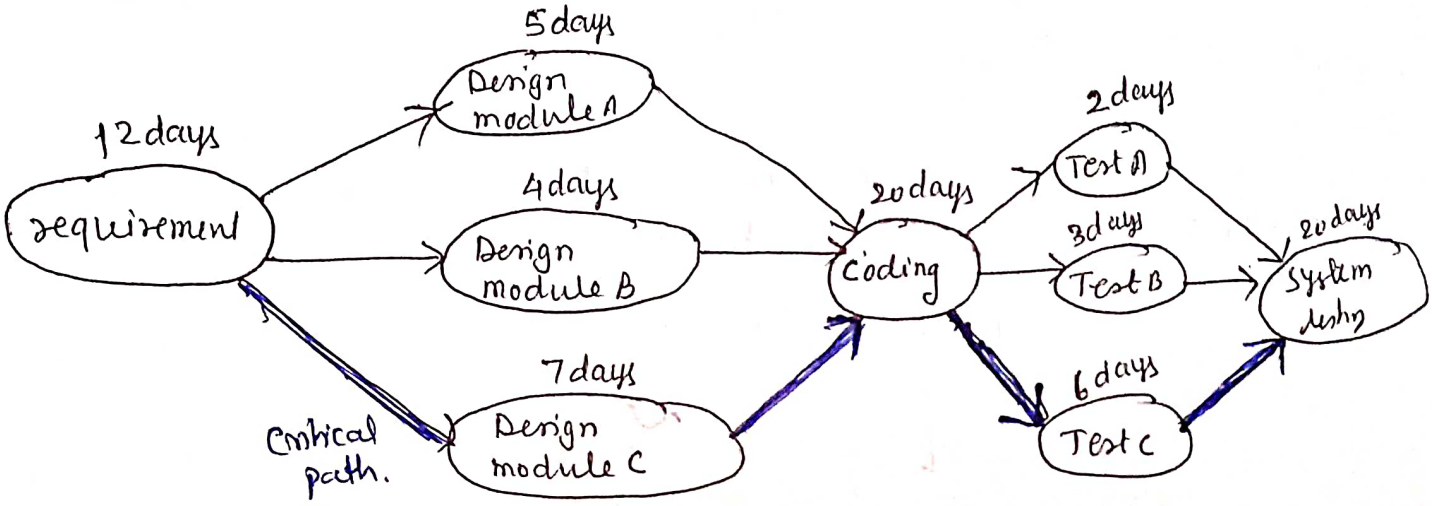


Activity on network- To simplify a project, divide it into several subgraphs called activities.

The successfull completion of these activities will result in completion of entire projects.

The relationship can be represented by a directed graph G in which the vertices represent task or activities and the edges represent procedure or relation between task is an activity on network.

Example :-



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 last vertex. A path ~~from~~ of longest length is called critical path.

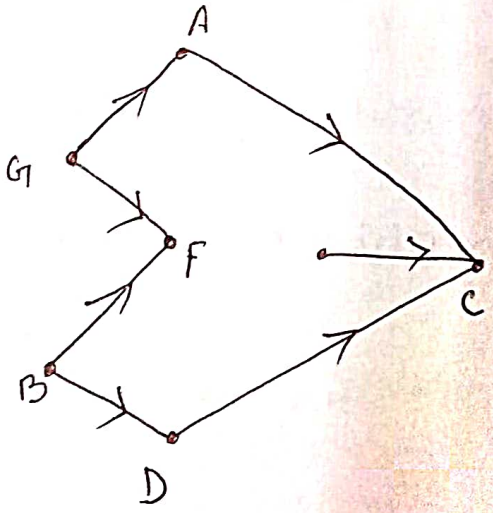
Topological sort → The linear ordering of vertices in a network is called topological order, with the property that if i is a predecessor of j in the network then i precedes j in the linear ordering.

Algorithm–

S is a directed graph without cycle.

1. Find the indegree of each node N of S. (INDEG(N))
2. put in a queue all the nodes with zero indegree.
3. repeat step 4 & 5 until the Queue is empty.
4. remove the front node N of queue.
5. repeat the following for each neighbor M of node N
   a) set INDEG(M) = INDEG(N) − 1
   b) if INDEG(M) = 0 then Add M to the rear of Queue.
6. Exit.

Example →

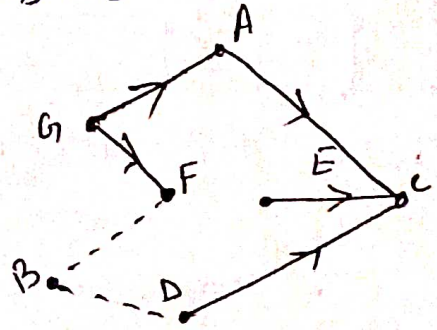


Indeg (A) = 1
Indeg (B) = 0
Indeg (C) = 3
Indeg (D) = 1
Indeg (E) = 0
Indeg (F) = 2
Indeg (G) = 0

Queue.

B	E	G	

remove B from Queue, neighbor D, F

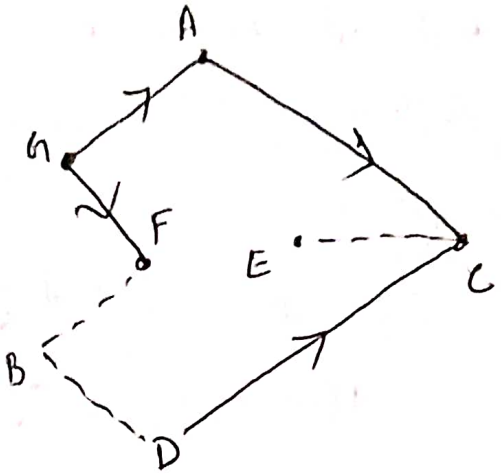


Indeg(D) = 1 - 1 = 0

Indeg(F) = 2 - 1 = 1.

Add D to Queue.

E	G	D	



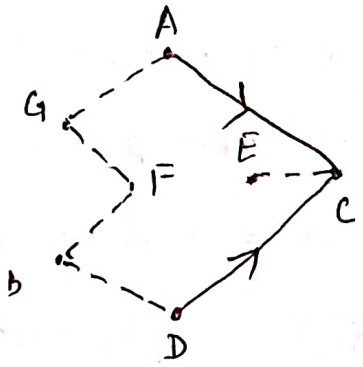
Remove E from Queue

neighbor of E C

Indeg(C) = 3 - 1 = 2

G	D	

Remove G from Queue. neighbor A, F



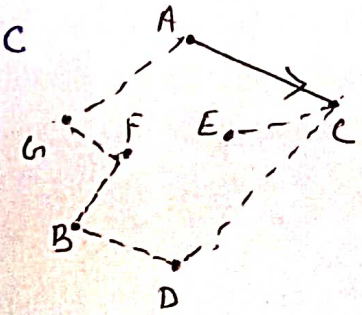
Indeg(A) = 1 - 1 = 0

Indeg(F) = 1 - 1 = 0

Add A, F to Queue

D	A	F	

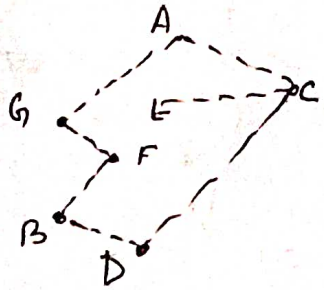
Remove D from Queue, neighbor C



Indeg(C) = 2 - 1 = 1

A	F	

Remove A from Queue. neighbor C.



Indeg(C) = 1 - 1 = 0

Add C to Queue.

F	C	

Remove F from Queue.

C	

Remove C from Queue

So topological order.

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

Representation of Graph in memory -

(1) Sequential representation

- Adjancy matrix
- path matrix

(2) Linked representation - In linked representation, we need two list, a node list NODE and an edge list EDGE as follows.

a) Node list - It has three fields

NODE	NEXT	ADJ

NODE → name or key value of a node

NEXT → pointer to the next node

ADJ → pointer to the first element in the adjancy list of node, which is maintained in the list EDGE.

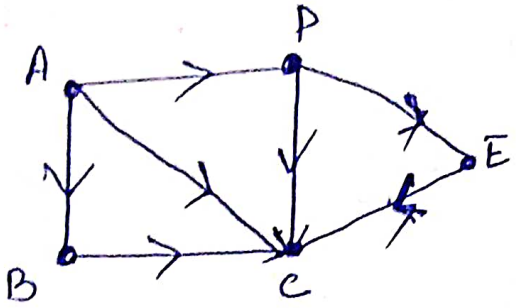
b) Edge List - It has two fields.

DEST	LINK

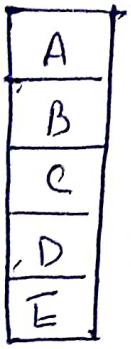
DEST → pointer to the terminal node of the edge in NODE List

LINK → Link together the edges with the same initial node - i.e. the nodes in the same adjancy list.

Example-



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



Head
[Array of pointer]