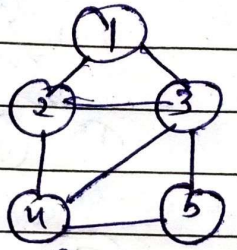
Graph

1) A graph G = (V, E) is a set of vertices (V) and set of edges (e).
   V(G) = Vertices of Graph G
   E(G) = Edges of Graph G.

2) The set V is finite, non-empty set of vertices. The set E is a set of pair of vertices representing edges.

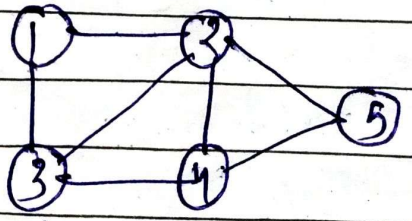
3) Examples:-



V(G) = {1, 2, 3, 4, 5}
E(G) = {(1,2), (1,3), (2,3), (3,4), (3,5), (4,5)} |24

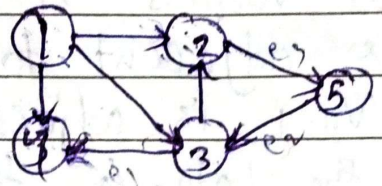
Basic Terminology:-

(i) Undirected Graph:- A graph containing unordered pair of vertices (A, B) and (B, A) represent the same edge.
exp:-



V(G) = {1, 2, 3, 4, 5}
E(G) = {(1,2), (2,5), (4,5), (3,4), (1,3), (2,3), (2,4)}

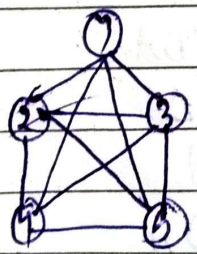
(iii) Directed Graph:- A Graph containing ordered pair of vertices is called directed graph. If an ~~ordered~~ edge is represented using a pair of vertices (V_1, V_3) then the edge is said to be directed from V_1 to V_2. $(V_1, V_2) \neq (V_2, V_1)$



$V(G) = \{1, 2, 3, 4, 5\}$

$E(G) = \{(1,2), (1,3), (1,4), (2,5), (3,2), (3,4), (5,3)\}$

(iv) Complete Graph:- An undirected graph in which every vertex has an edge to all other vertices is called a Complete graph. A Complete graph with in vertices has $\frac{n(n-1)}{2}$ edges

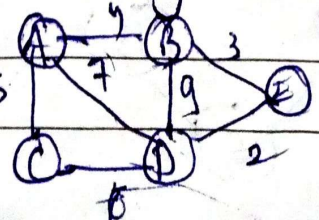


$n = 5$

$$\text{edges} = \frac{n(n-1)}{2} = \frac{5 \times 4}{2} = 10$$

(v) Weighted Graph:- A weighted graph is a graph in which edges are assigned some value. An edge may represent a highway link between two cities. The weight will denote the distance between two Connected cities using highway. weight of an edge is also called its cost

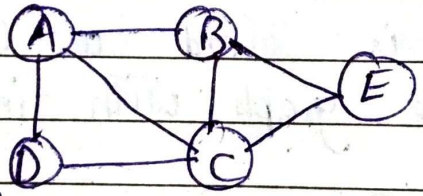
eg:-



(v) Adjacent Nodes / Vertices:- Two vertices v_1 and v_2 are said to be adjacent if there is an edge between v_1 and v_2.

(vi) path:- A path from vertices V_0 to V_n is a sequence of vertices $V_0, V_1, V_2, V_3 \ldots V_{n-1}, V_n$. Here V_0 is adjacent to V_1, V_1 is adjacent to V_2 and V_{n-1} is adjacent to V_n. The length of the path is the number of edges on the path.

(vii) Cycle:- A cycle is a simple path that begins and ends at the same vertices.



Cycle:- ABECDA
ABCDA
ABCA
ACDA

Connected graph:- A graph is set to be connected if there exist a path b/w every pair of vertices (v_i and v_j)

eg:-

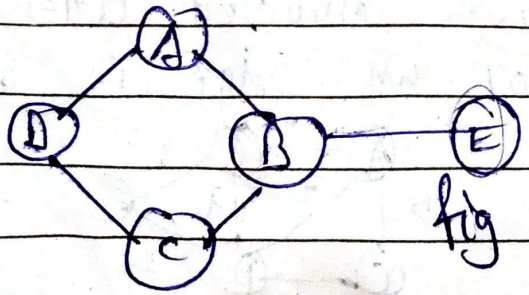


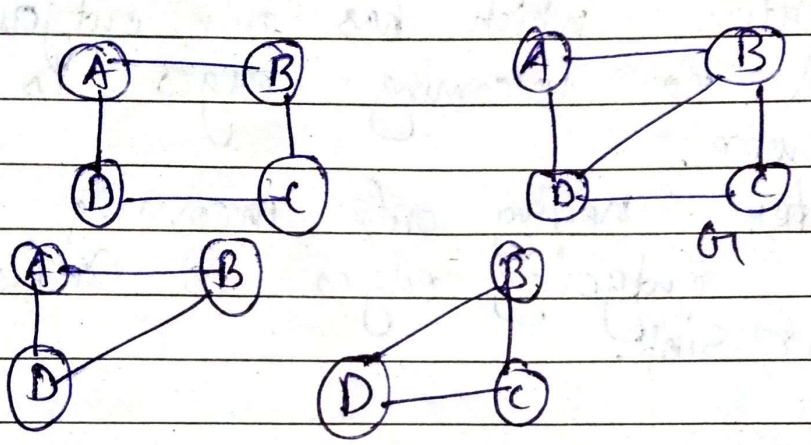
fig of Connected graph

in the vertex ⇒ directed graph
out the vertex ⇒ "

Date / /
Page No.

☆ Sub Graph:- A sub Graph of G_1 is a graph G_1, G_1, such that $V(G_1)$ is a subset of $V(G_1)$ and $E(G_1)$ is a subset of $E(G_1)$.

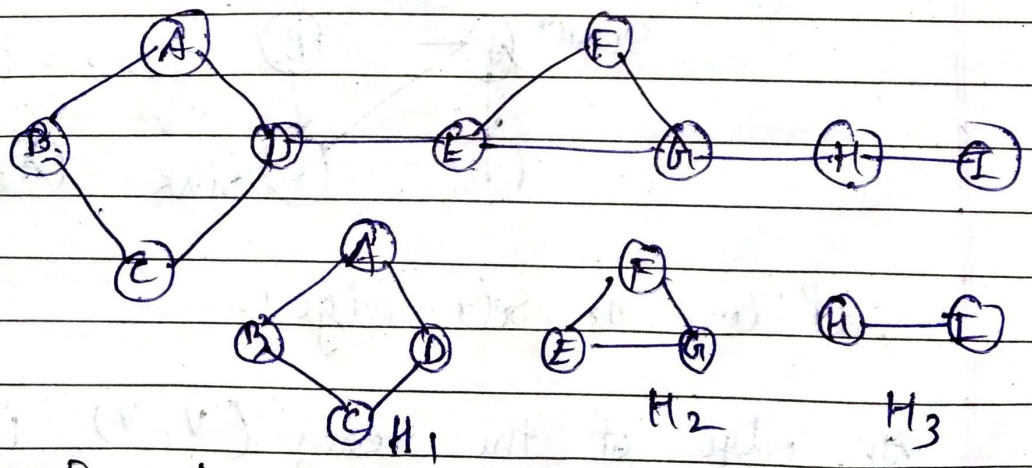
exp:



$V(G_1) = \{A, B, C, D\}$
$E(G_1) = \{(A,B), (A,D), (D,B), (B,C), (D,C)\}$

☆ Component:- A component H of an undirected graph is a maximal connected sub-graph

exp



☆ Degree of vertex:-

→ The total no. of edges linked to a vertex is called it's degree.

→ The in-degree of the vertex is the total no. of edges coming to that vertex.

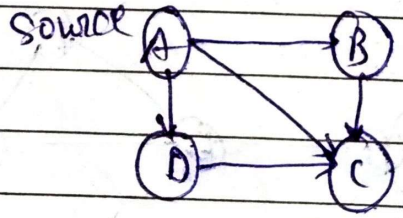
→ The out degree of the vertex is total no. of edges going out from the vertex.

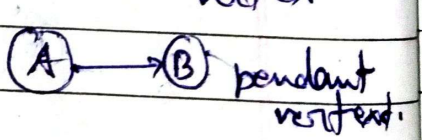
→ A vertex which has only outgoing edges and no incoming edges is called a source.

→ A vertex having only incoming edges and no outgoing edges is called destination or sink.

→ When in-degree of a vertex is 'one' and out-degree is '0'. Then such a vertex is called a pendant vertex.

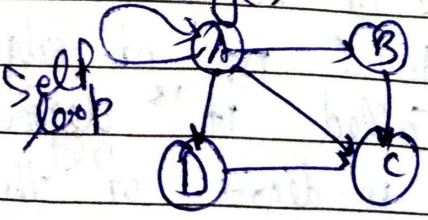
→ When the degree of a vertex is '0' then such a vertex is called isolated vertex.





☆ Self loop or self-edge :-

An edge of the form (v, v) is known as self loop or self-edge.

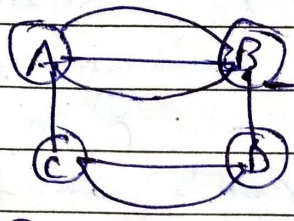


Date / /
Page No.

✱ Multi- Graph:

A graph with multiple occurances of the same edge is known as multi- graph.

exp:-



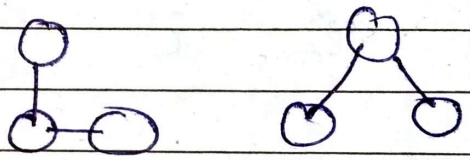
in which 3 edge Connected of A with B

Fig- multi- graph

✱ Tree:-

A tree is a Connected graph without any cycle.

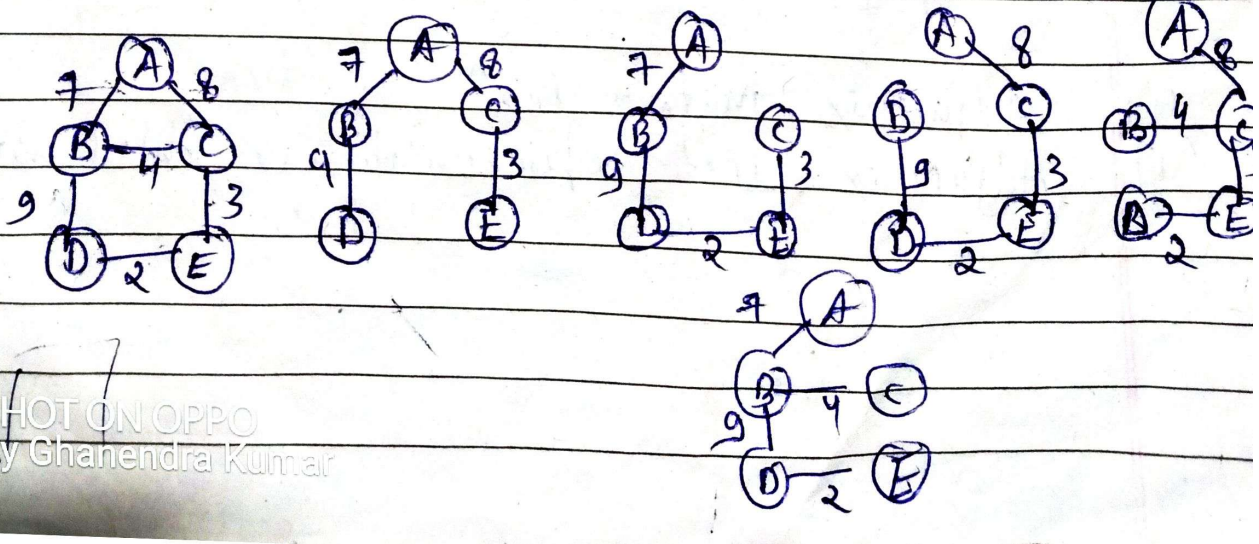
exp:-



✱ Spanning tree :-

A Spanning tree of a graph $G = (V, E)$ is a Sub graph of G having all vertices of G and no cycles in it. If the graph G is not connected than there is no Spanning tree of G.

A Graph may have multiple spanning tree.



★ $n_0 = n_2 + 1$ it is no. of children

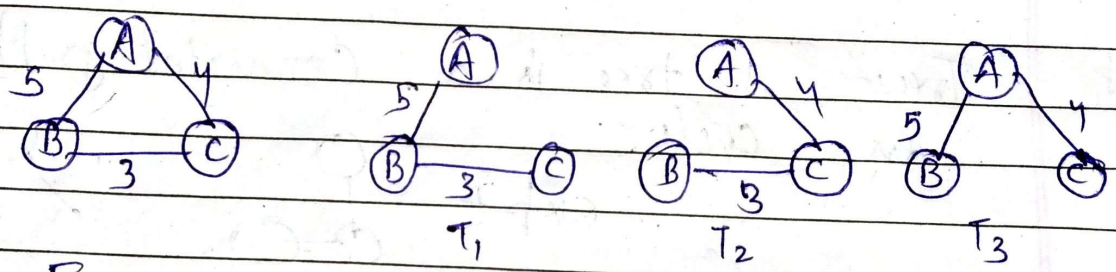
Date 09/11/22
Page No.

★ Minimum spanning tree :-

A minimum spanning tree of a graph G, $G = (V, E)$ is called a minimum cost spanning tree or minimum spanning tree if its cost is minimum.

The cost of a graph is the sum of the cost of the edges in the weighted graph.

Exp:



For weighted Graph

minimum Spanning Tree

Cost of $T_1 = 5 + 3 = 8$

" " $T_2 = 4 + 3 = 7$ — it is minimum

" " $T_3 = 5 + 4 = 9$

★ Representation of Graph :-

The graph may be represented into two ways

(i) Adjacency matrix Repⁿ

(ii) Adjacency list representation or linked list Repⁿ

Date / /
Page No.

(i) Adjacency matrix representation:-

The Adjacency matrix A for a graph G = (V, E) with 'n' vertices is an n×n matrices of bits such that Aij = 1, if there is an edge from v_i to v_j. And Aij = 0, if there is no such edge, we can also write it as:-

$$A[i][j] = \begin{cases} 1 & \text{if and only if } (v_i, v_j) \text{ is in } E(G). \\ 0 & \text{otherwise} \end{cases}$$

exp:-

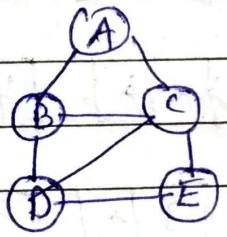


fig. undirected graph

i \ j	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	1	0
C	1	1	0	1	1
D	0	1	1	0	1
E	0	0	1	1	0

exp:-

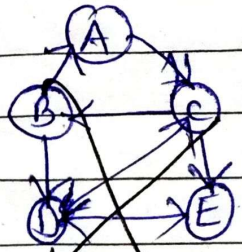
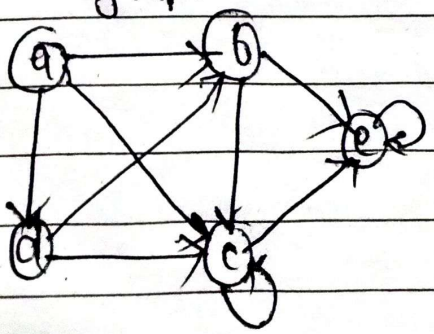


fig. directed graph

i \ j	A	B	C	D	E
A	0	0	1	0	0
B	1	0	0	1	0
C	0	1	0	0	1
D	0	0	1	0	1
E	0	0	0	1	0

exp:-



i \ j	a	b	c	d	e
a	0	1	1	1	0
b	0	0	1	0	1
c	0	0	1	0	1
d	0	1	1	0	0
e	0	0	0	0	1

Adjacency list Repⁿ :-

→ A graph can be repⁿ using a liked list. For each vertex, A list of adjacent vertices is maintained using a link list. It creates a seperate link list for each vertex V_i in the graph $G=(V,E)$

Solution:- List of adjacent vertices to vertex 1.

exp 1

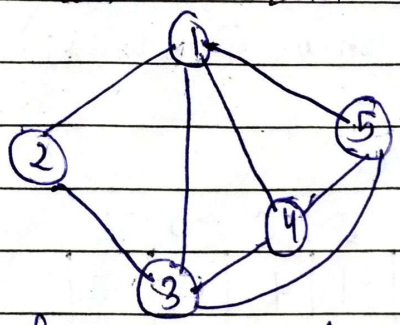
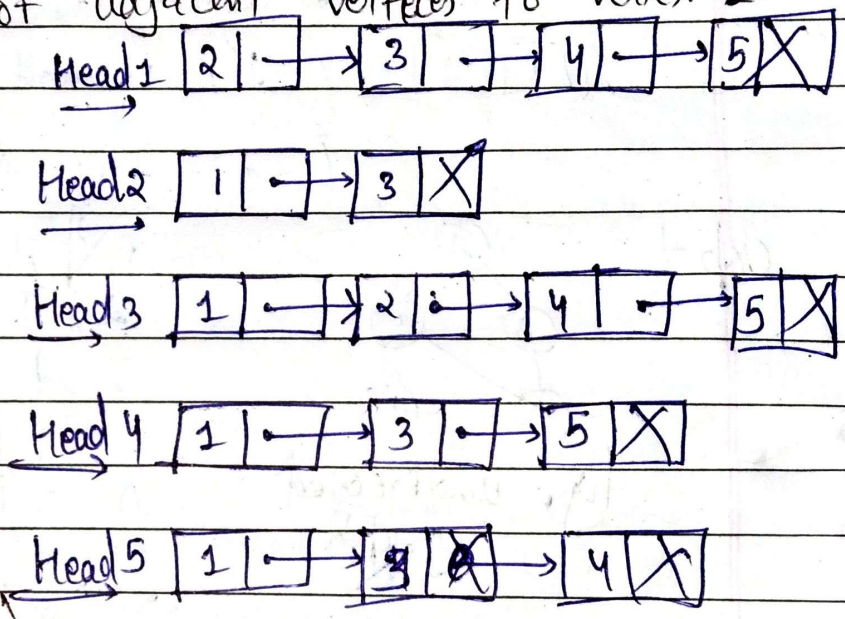
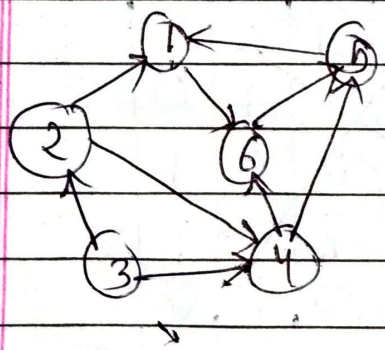


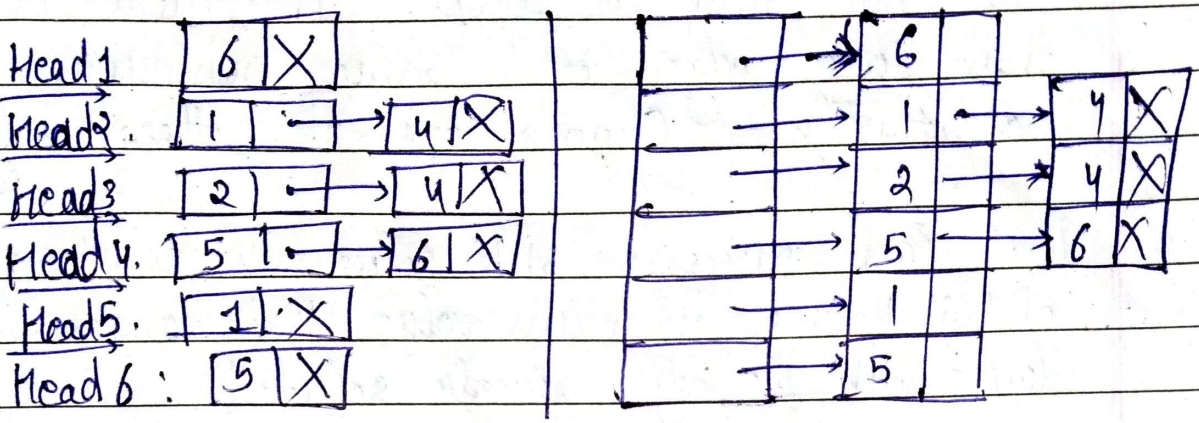
fig - Undirected Graph



exp 2



Sol^{M}:- List of adjacent vertices to vertex 1.



There are two populer techniqus constructing a minimum cost spanning tree from a weighted graph $G=(V,E)$

Sol^{n}

① kruskal's Algorithm
② Prim's Algorithm

i) kruskal's Algorithm:-
It is a method for finding the minimum cost spanning tree of the given graph. In kruskal's Algorithm edges are added to the spanning tree in increasing order of cost. If the edge 'E' form a cycle in the spanning, it is discarded.

- Algorithm:-

Step 1:- List all edges of the graph $G=(V,E)$ in order of increasing or increasing bits.

Height = E. log V | Height = log n

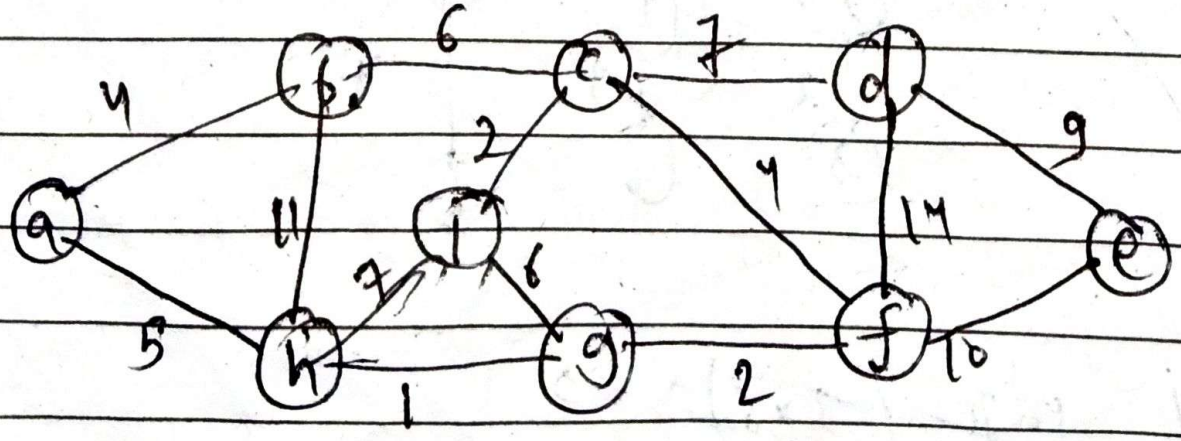
n = no. of vertices.

Step ② Select any edge of minimum value that is not form a loop. If there is more than one edge of same minimum value, arbitrarily Choose one of these edges.

Step ③ For each successive step select any remaining edge of 'G' having minimum value that do not form a loop with the edges already selected.

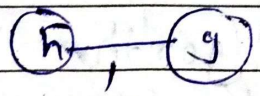
Step ④ Continue Step ③ until (n-1) edges have been selected and these edges will construct the desired shortest spanning tree.

Q→

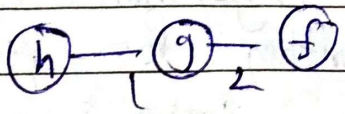


(h,g) - 1
(g,f) - 2
(c,i) - 2
(c,f) - 4
(a,b) - 4
(a,h) - 5
(b,c) - 6 ✗
(i,g) - 6 ✗
(c,d) - 7
(h,i) - 7 ✗
(d,e) - 9
(e,f) - 10 ✗
(h,h) - 4 ✗
(d,f) - 4 ✗

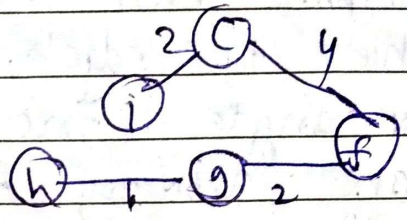
(1)



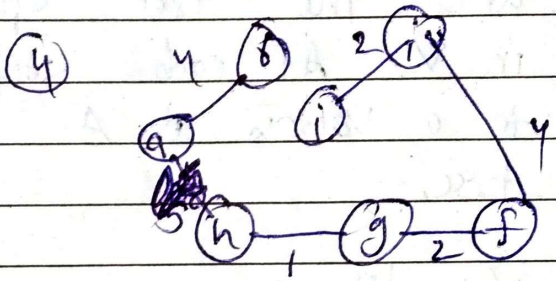
(2)



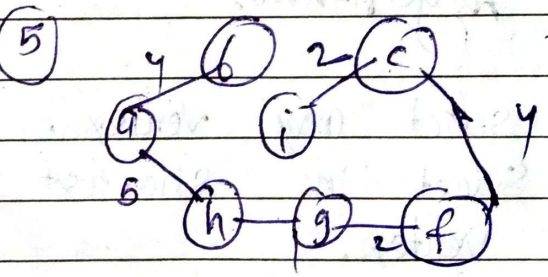
(3)



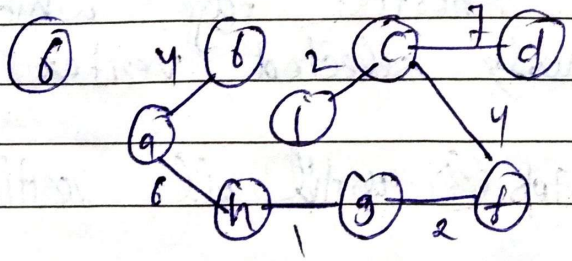
(4)



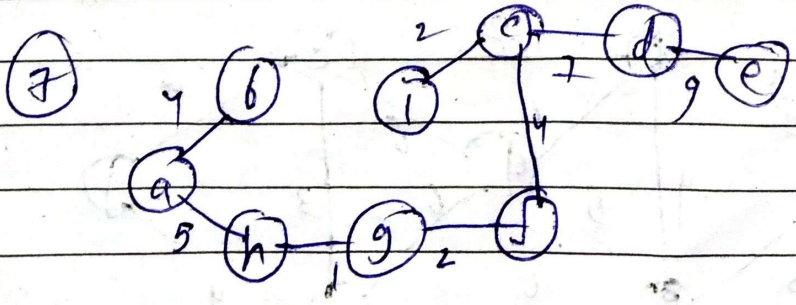
(5)



(6)



(7)



Total Cost of MST = 1+2+2+4+5+7+9+4 =
= 34

Date 14/1/22
Page No.

★ Prim's Algorithm :-

Prim's Algorithm operates much like Dijkastra's Algorithm for finding shortest path in a graph. Prim's Algorithm has the property that the edges in a set A always forms a single tree. The tree starts from an arbitrary root vertex 'r' and grows until the tree spans all the vertices in V. At each step a light edge connecting to a vertex in A form a minimum spanning tree.

★ Algorithm:-

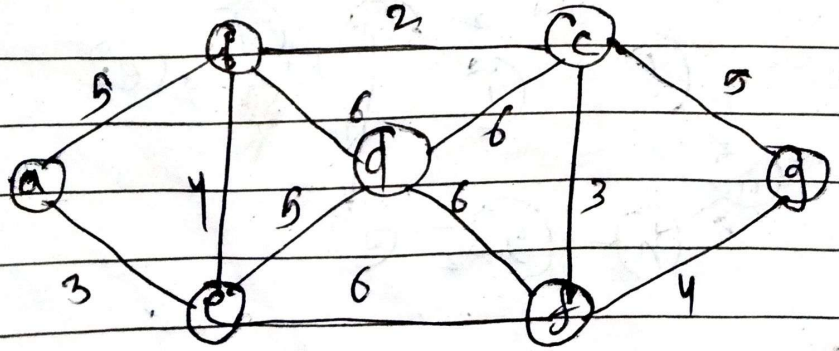
Step 1 Select any vertex.

Step 2 Select the shortest edge connected to that vertex.

Step 3 Select the shortest edge which connects a previously selected vertex to a new vertex.

Step 4 Repeat step (3) until all vertices have been Connected

Q)

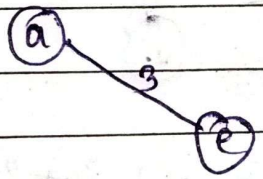


Sol:

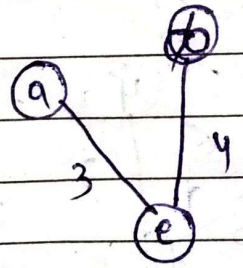
	a	b	c	d	e	f	g
a	0	5	∞	∞	(3)	∞	∞
b	5	0	(2)	6	(4)	∞	∞
c	∞	(2)	0	6	∞	(3)	5
d	∞	6	6	0	(5)	6	∞
e	(3)	(4)	∞	(5)	0	6	∞
f	∞	∞	(3)	6	6	0	(4)
g	∞	∞	5	∞	∞	(4)	0

7X7

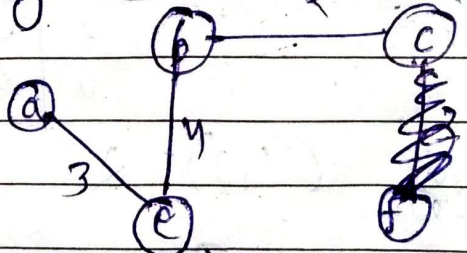
1) Select edge (a,e) = 3



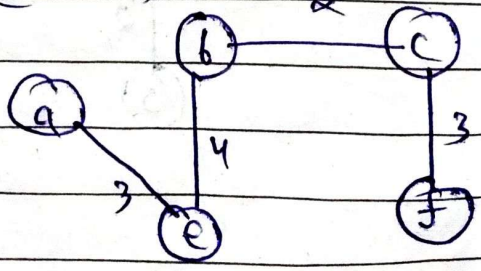
2) Select edge (e, b) = 4



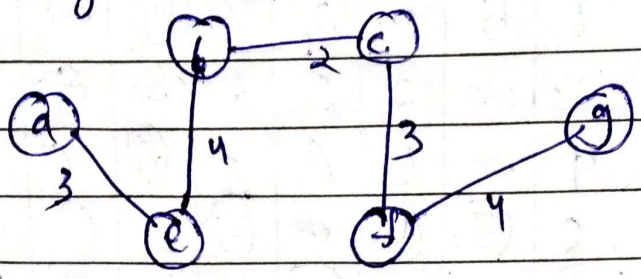
3) Select edge (b,c) = 2



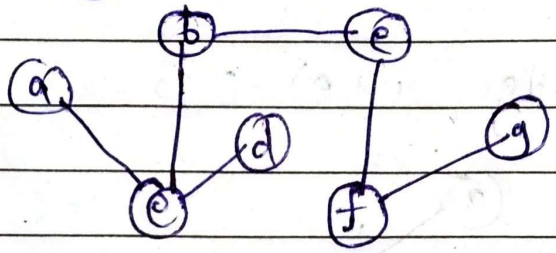
4) Select edge (c, f) = 3



5) Select edge (f,g) = 4

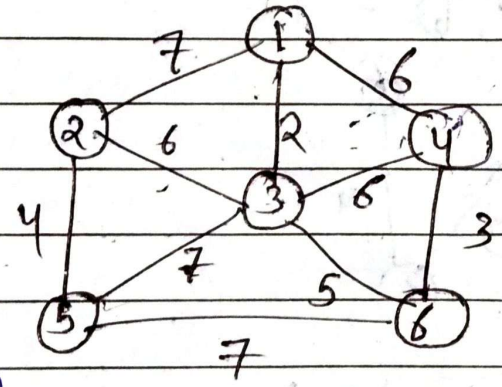


6) select edge (e,d) = 5



Total Cost = 2+3+3+4+4+5
= 21

Q3

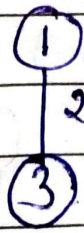


Sol:-

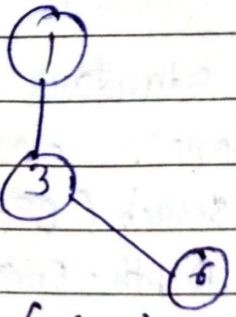
	1	2	3	4	5	6
1	0	7	(2)	6	∞	∞
2	7	0	6	∞	4	∞
3	(2)	6	0	6	7	(5)
4	6	∞	6	0	∞	3
5	∞	4	7	∞	0	7
6	∞	∞	(5)	3	7	0

6X6

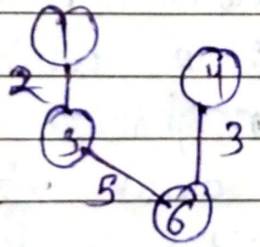
1) Select edge (1,3) = 2



2) select edge (3,6) = 5



3) select edge (6,4) = 3



4) select edge ( )

Imp

Graph Traversal :-

There are two standard ways that the graph traversal is done. one way is called a "Breath-First-Search (BFS)" and the other is called a "Depth-First-Search (DFS)"

(i) Breath-First-Search (BFS) - (FIFO) :-

Take an array Queue which will used to keep the unvisited neighbours of the node.
Take a boolean array 'visited' which will have value true if the node has been visited. And will have value false if the node has not be visited.
Initially Queue is empty and front = -1

Front = -1

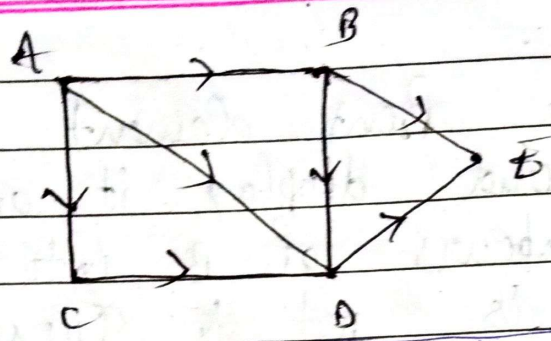
Rear = -1

Initially visited [i] = false, where i = 1 to n, n is total no. of nodes.

Algorithm :-

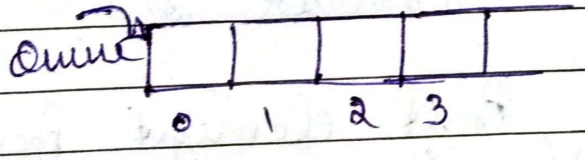
Step (1) Insert Starting node into the Queue
(2) Delete front element from the Queue and insert all its unvisited neighbours into the Queue at the end traverse them.
(3) Also make the value on visited array true for these nodes
(4) Repeat Step (2) until the Queue is empty.

Q.

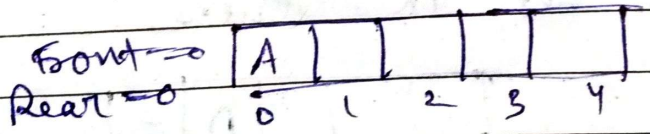


Sol:

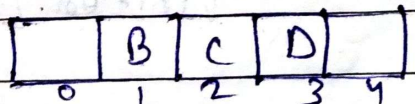
vertex	Adjacency list
A	B, C, D
B	D, E
C	D
D	E
E	-



1) Insert the starting node A into Queue



2) Delete the front element A from the Queue and display it and then insert all the neighbour of A into the Queue if it is not in Queue



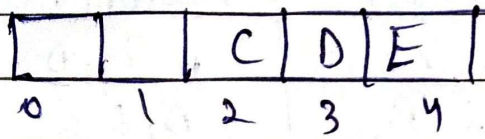
Front = 1

Rear = 3

Traversal node A

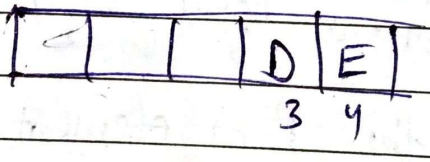
Visited [A] = True

(3) Delete the front element B from the Queue and display it and then add the neighbours of B into the Queue if it is not in Queue.



Front = 2
Rear = 4
visited [B] = true
Traversed node = A, B

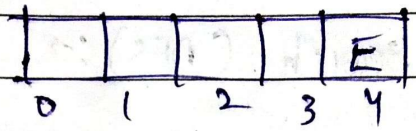
(4) Delete the front element from the Queue and display it and then insert all the neighbour of C into the Queue. If it is not in Queue



Front = 3
Rear = 4
Visited [C] = true

Traversal node = A, B, C

(5) Delete the front node from the Queue and display it and then insert all the neighbour of D into the Queue if it is not in Queue.



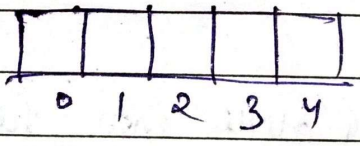
Front = 4

Rear = 4

Visited [D] = true

Traversal node = A, B, C, D,

(6) Delete the front node in the Queue and display it and then add all the neighbour of E into the Queue if it is not in Queue.



Front = -1

Rear = -1

Visited [E] = true

Traversal node = A, B, C, D, E

Date 16/11/22
Page No.

(2) Depth First Search (DFS) :-

→ DFS technique uses stack, take an array stack which will be used to keep the unvisited neighbours of the node. Take a another Boolean array (visited) which will have value true if the node has been visited and will have value false if the node has not been visited.

Initially, Stack is empty and Top = -1,
Initially, visited 'I' = false where, i = 1 to n and n is the total no. of node.

- Algorithm:-

(i) Push Starting node into the stack.

(ii) POP an element from the stack if it has not being traverse then traverse it if it has already being travel then just ignore if after traversing, make the value of visited array true, for this node.

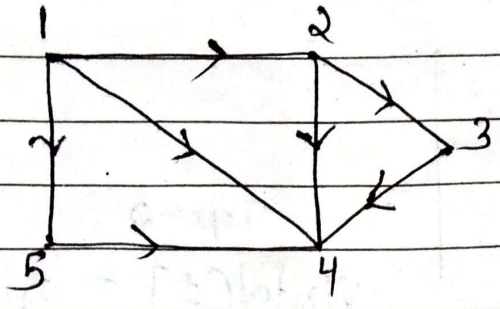
(iii) Now push all the unvisited adjacent nodes of the Pop element on stack.

(iv) Push the element even if it is not already on the stack.

(v) Repeat steps ② & ③ until Stack is empty.

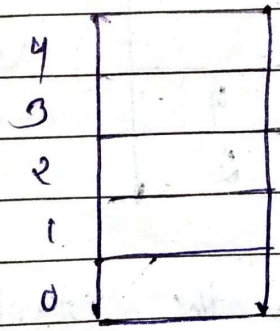
* pop = [Top = Top - 1]
* push = [Top = Top + 1]

Q. →

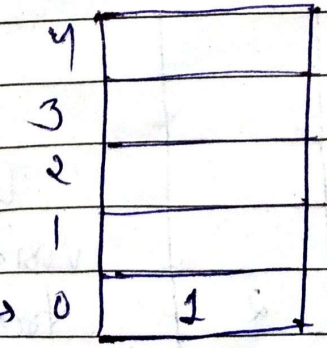


Vertex	Adjacency list
1	2, 4, 5
2	3, 4
3	4
4	—
5	4

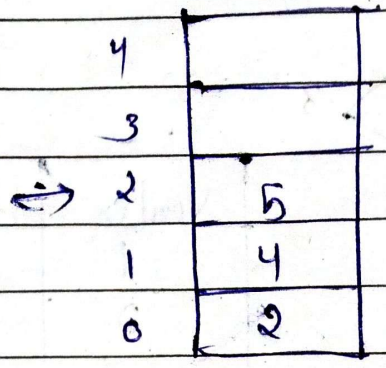
Initially, Stack is empty i.e. Top = -1 and visited [i] = false



(i) Push the starting node '1' into stack

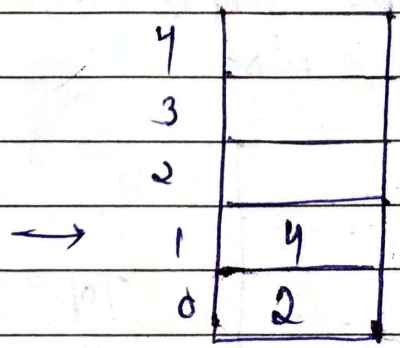


(ii) POP node 1 from the stack and traverse it. Now, push all adjacent node of the popped element on stack.



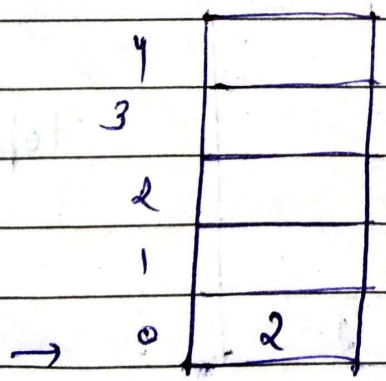
Top = 2
Visited[1] = true
traversed node = 1

(iii) POP node 5 from the stack and traverse it, Now push all the adjacent node of the popped element in stack.



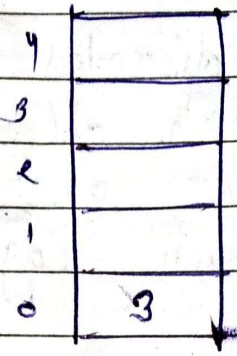
Top = 1
Visited [5] = true
traversed node = 1, 5

(iv) Pop node 4 from the stack and traverse it, Now push all the adjacent node ~~from~~ the popped element in Stack



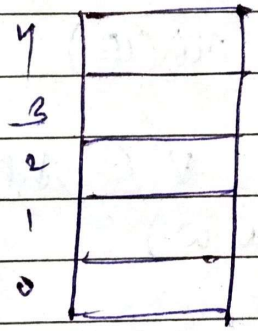
Top = 0
Visited [4] = true
traverse = 1, 5, 4

(v) pop node 2 from the stack and traverse it, pushed all the adjacent element (node) of the popped element in stack.



- Top=0
Visited[2] = true
traverse = 1, 5, 4, 2

(vi) Pop the node 3 from the stack and traverse it, Pushed all the adjacent node of the popped element of stack.



Top = -1, visited[3] = true
traverse = 1, 5, 4, 2, 3

Since, the stack is empty so we will stopped our process.

☆ Shortest path Algorithm :-

(i) Single Source Shortest path Algorithm:
(a) Dijkastra's Algorithm :-

☆ Single Source shortest path Algorithm :-
Dijkastra :- Dijkastra's Algorithm name after its inventor Dutch Computer Scientist Edsger Dijkastra's. it's a greede algorithm that solve the Single Source Shortest path problem

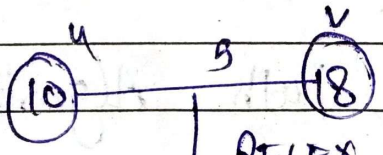
on a weighted directed graph $G=(V,E)$ for the case in which all Edge weights are non-negative. we assume that weight of $(u,v) \geq 0$ for each edges $(u,v) \in E(G)$.

- Algorithm:-

(i) Initialize - single - source (G,s)
(ii) $S \leftarrow \phi$
(iii) $Q \leftarrow V[G]$
(iv) while $Q \neq \phi$
(v) do $u \leftarrow$ Extract_Min(Q)
(vi) $S \leftarrow S \cup \{u\}$
(vii) for each vertex $v \in Adj[u]$
(viii) do RELAX (u,v,w)

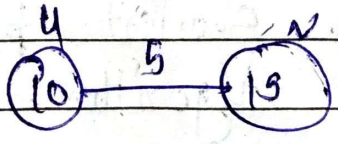
RELAX (u,v,w)
(i) if $d[v] > d[u] + w(u,v)$
(ii) then $d[v] \leftarrow d[u] + w(u,v)$
(iii) $\pi[v] \leftarrow u$

exp:

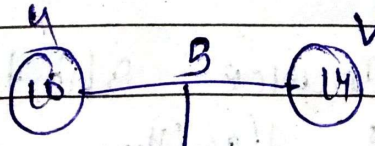


RELEX (u,v,w)

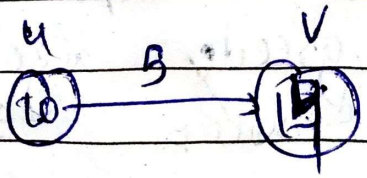
$d[v] > d[u]$



exp:



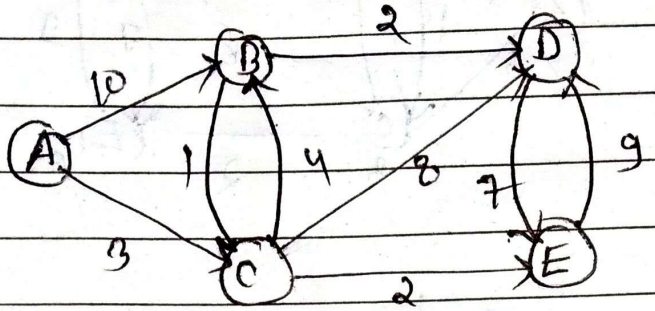
RELEX (u,v,w)



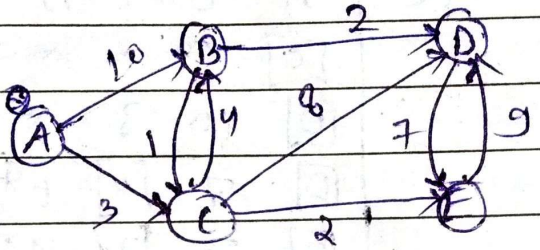
=

Q→ Consider A as a source vertex

A - B
B - C
C - D
~~D - E~~

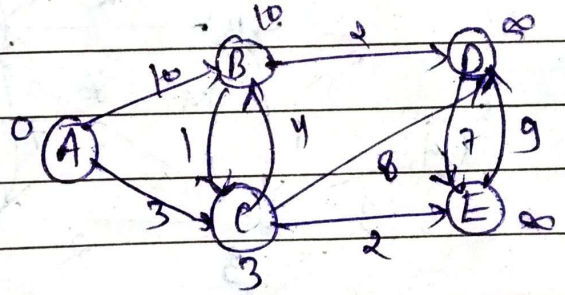


Sol→ Initialize



Q	A	B	C	D	E
	[0]	∞	∞	∞	∞

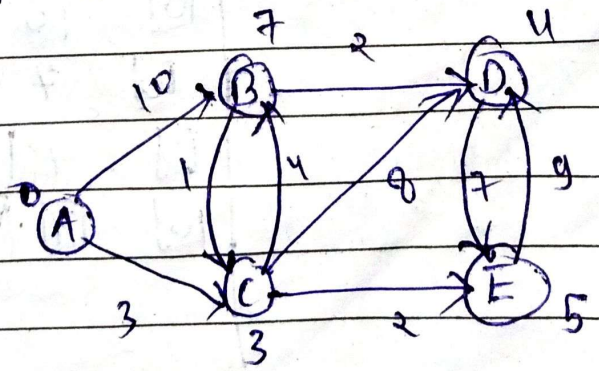
Step 1) "A" ← Extract-Min(Q)



S = { A }

Q	A	B	C	D	E
	[0]	∞	∞	∞	∞
	[0]	10	3	∞	∞

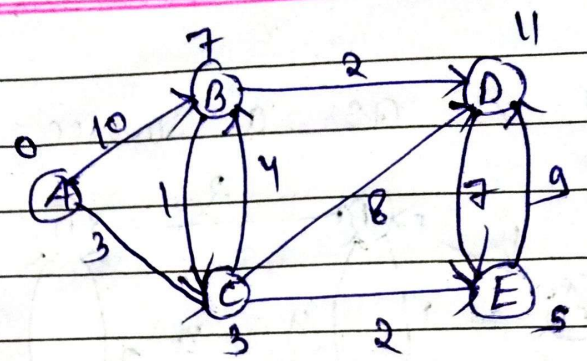
Step 2)



S = { A, C }

Q	A	B	C	D	E
	[0]	∞	∞	∞	∞
	[0]	10	3	∞	∞
	[0]	7	[3]	11	5

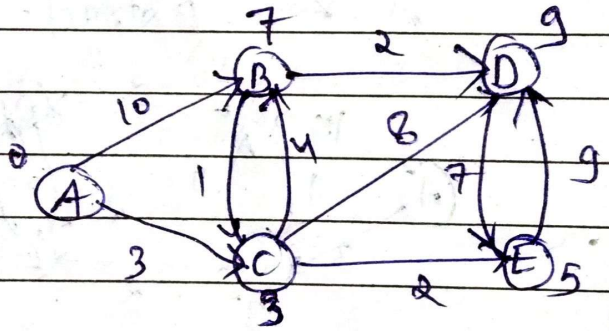
Step 3)



$S = \{A, C, E\}$

Q	A	B	C	D	E
	[0]	∞	∞	∞	∞
	[0]	10	3	∞	∞
	[0]	7	[3]	11	[5]
	[0]	7	[3]	11	[5]

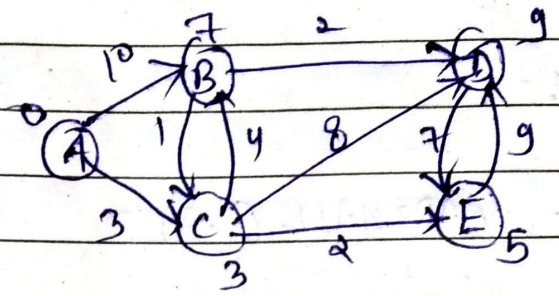
Step 4)



$S = \{A, C, E, B\}$

Q	A	B	C	D	E
	[0]	∞	∞	∞	∞
	[0]	10	3	∞	∞
	[0]	7	[3]	11	5
	[0]	7	[3]	11	[5]
	[0]	[7]	[3]	9	[5]

Step 5)



~~S = { A, B, C, D,~~

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

Q	A	B	C	D	E
	0	∞	∞	∞	∞
	0	10	3	∞	∞
	0	7	3	11	5
	0	7	3	11	5
	0	7	3	9	5
	0	7	3	9	5

☆ All pair Shortest path Algorithm :-

1. ~~Floyd~~ Floyd warshall Algorithm :-

It is used for finding the Shortest path b/w every pair of vertices of a graph.
The Algorithm works for both directed and undirected graph.

Let $D^{(k)}[i,j]$ or $D^{k}i,j$ denotes the weight of shortest path from v_i to v_j using $v_1 v_2 v_3 \ldots v_k$ As intermidiate vertices. $D^{(0)}$ is computed as weight matrix.