UNIT-4

Date - 10/10/19

GRAPH

A Graph consist of two sets.

(i) A non-empty set V whose elements are called vertices.

(ii) A set of E of edges such that $e \in E$ associated with ordered or unordered pair of element of V. The set $E(G)$ is called edges set of G.

The Graph with vertices V & edges E is written as $G(V, E)$ or $G = (V, E)$

* If an edge $e \in E$ is associated with an ordered or unordered pair (u, v) where $u, v \in V$. Then, E is said to connect u & v; u & v are called end point of edge e.

* An edge is said to be incident with the vertices it join.

* The edge e that join vertices u & v it said to be incident on each of its end point u & v. i.e., connected by edge. e in a graph is called adjacent vertix.

ISOLATED VERTEX -

In a graph vertix that ~~have~~ is not adjacent to any vertix is called isolated vertix.

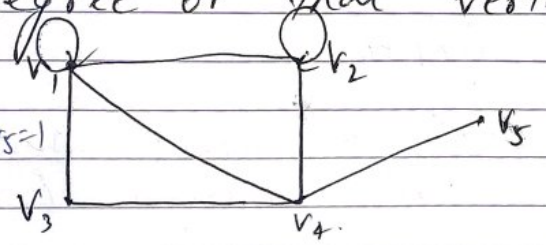
DEGREE OF VERTEX -

The degree of a vertix of an undirected graph is the number of edges incident with it

NOTE -

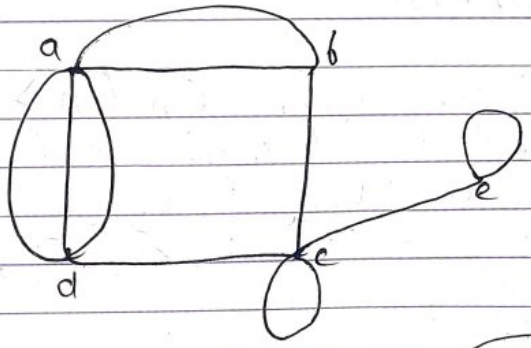
The loop at vertix contributes twice to the degree of that vertix.

ex -



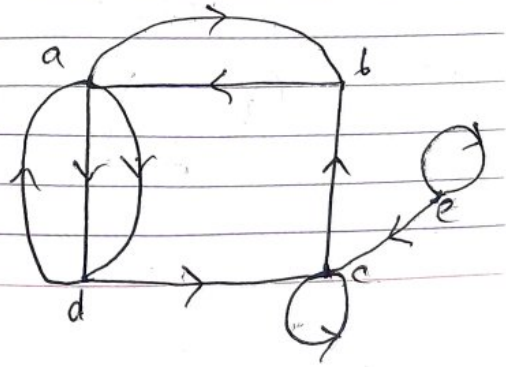
$v_1 \to 5$ | $V_3 = 2$ | $V_5 = 1$
$V_2 = 4$ | $V_4 = 4$

ex -



deg(a) = 5
deg(b) = 3
deg(c) = 5
deg(d) = 4
deg(e) = 3

Indeg(a) = 2, Outdeg(a) = 3
Indeg(b) = 2, Outdeg(b) = 1
Indeg(c) = 3, Outdeg(c) = 2
Indeg(d) = 2, Outdeg(d) = 2
Indeg(e) = 1, Outdeg(e) = 2



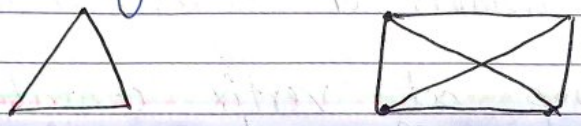
NULL GRAPH

Null graph is a graph which contain isolated vertix.

COMPLETE GRAPH

If all vertix are interconnected to all remaining vertix.

ex-

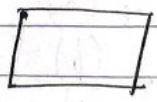
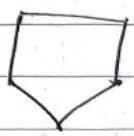


REGULAR GRAPH

If all vertix of graph have same degree then this graph is known as regular graph

CYCLE GRAPH

The Cycle graph C_n $(n \geq 3)$ of length n is a connected graph which consist of n vertices say $v_1, v_2, v_3, \ldots v_n$ & n edges $E_n = \{\{v_1, v_2\}, \{v_2, v_3\}, \ldots \{v_{n-1}, v_n\}, \{v_n, v_1\}\}$.

ex- C_3  , C_4  , C_5 

WALK

A walk in a graph G is a finite alternative sequence say $v_0 e_0, v_1 e_1, v_2 e_2, \ldots\ldots v_n e_n$ of vertices & edges of the graph such that edge e_i in the sequence joins vertices v_{i-1}, v_i where $1 \leq i \leq n$.

The end vertices v_0 & v_n are called end terminals of the walk whereas $v_1, v_2, \ldots\ldots v_{n-1}$ are called internal vertices of walk.

The number of edges in walk is called length of a walk.

* OPEN WALK

A walk is called open when its end terminals are distinct.

* CLOSED WALK

A walk is called closed when its end terminals are same.

NOTE- A walk may repeat both edge & vertices.

* TRAIL

A walk is called trail if all its edges are distinct but it can repeat the

Closed loop = Circuit

vertices.

A Trail is a open or closed depends on whether its end vertices are distinct or same.

* PATH -

A walk does not contain repeated vertix & edges is called Path.

SPANNING TREE -

A Spanning Tree is a subset of Graph G which has all the vertices covered with minimum possible no. of edges. Hence, a spanning tree does not have cycles & it cannot be disconnected.

Every connected & undirected graph G(V, E) has atleast one spanning tree.

A disconnected graph does not have any spanning tree as it cannot be spann to all its vertices

* PROPERTIES OF SPANNING TREE

(i) A graph G can have more than one spanning tree.

(ii) A complete undirected graph can have $(n^{(n-2)})$ spanning tree where n is no. of vertices.

(iii) All possible spanning tree of Graph G have same no. of V vertices & edges.

Spanning tree of a graph G is connected subgraph which implies minimally connected sub graph.
) Adding one edge to the spanning tree will create a circuit or closed loop it implies spanning tree is minimally acyclic subgraph.

MATHEMATICAL PROPERTIES OF S.T.

i) From complete graph by removing maximum $(e-n+1)$ edges we can construct a spanning tree. where e is no. of edges & n is no. of vertices

i) A Spanning tree has $(n-1)$ edges where n is no. of vertices.

APPLICATIONS OF S.T.

) Civil network planning.

i) Computer network routing protocol.

iii) Cluster analysis.

MINIMAL SPANNING TREE -

Inweighted graph, a minimal spanning tree is a spanning tree that have minimum weight, that all other spanning

(iv) Removing one edge from spanning tree will make graph disconnected it implies spanning tree minimally connected sub graph.

(v) Adding one edge to the spanning tree will create a circuit or closed loop it implies spanning tree is minimally acyclic subgraph.

* MATHEMATICAL PROPERTIES OF S.T.

(i) From complete graph by removing maximum $(e-n+1)$ edges we can construct a spanning tree. where e is no. of edges & n is no. of vertices

(ii) A Spanning tree has $(n-1)$ edges where n is no. of vertices.

* APPLICATIONS OF S.T.

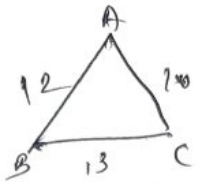
(i) Civil network planning.

(ii) Computer network routing protocol.

(iii) Cluster analysis.

MINIMAL SPANNING TREE -

Inweighted
~~Inwaited~~ graph, a minimal spanning tree is a spanning tree that have minimum weight, that all other spanning

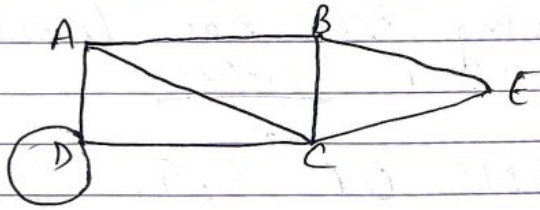


tree of same graph. In real world situation, this weight can be measured as distance, congestion, traffic load or any arbitrary value denoted to edge.

ADJACENCY MATRIX -

A connected graph G(V, E) can be represent into a 2-Dimensional adjacency matrix of size nxn where n is no. of vertices. By following value. A[i][j] = 1 if there exist an edge b/w from ith vertix to jth vertix. A[i][j] = 0. if there is no edge b/w ith & jth vertix.

Ex-



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

5x5

SPANNING TREE (ALGORITHM FOR CONSTRUCTING) USING BREADTH FOR SEARCH (BFS).

Step 1 - Choose (Arbitrary) a vertix & designate it as the root.

Step 2 - Add all edges incident to this vertix, such that addition of edges doesnot produce cycle or circuit.

Step 3 - A new vertices added in previous

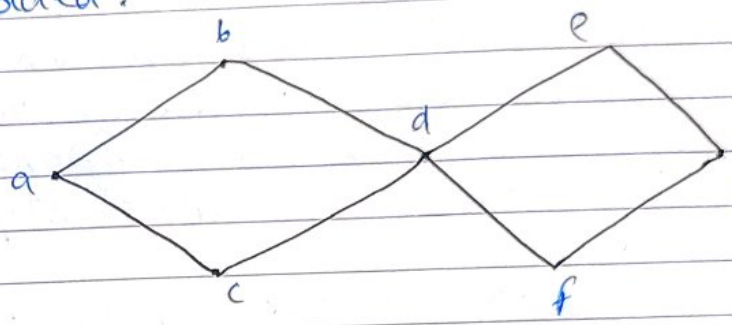
step become vertices at level 1 in th spanning tree, arbitrary order them

Step 4 - for each vertix of level 1 visit in order, add each edge incident to this vertix to the tree as long a it doesnot produce any cycle or circ

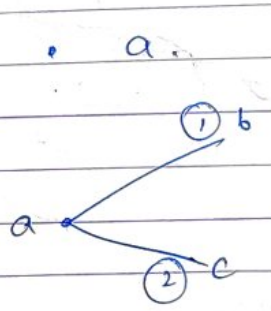
Step 5 - Arbitrary order the vertices at level 2.

Step 6 - Continue the same procedure until all the vertices in the tree have been added.

ex -

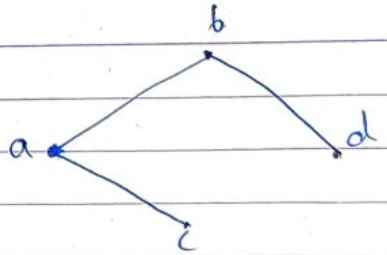


Step 1 -

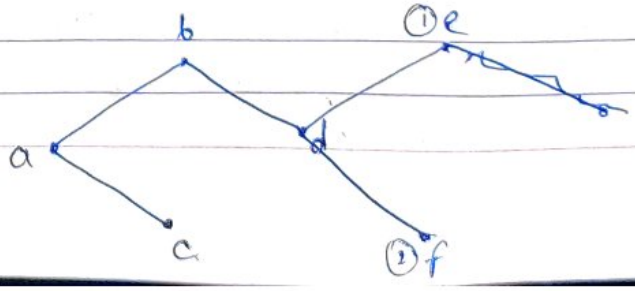


Step 2 -

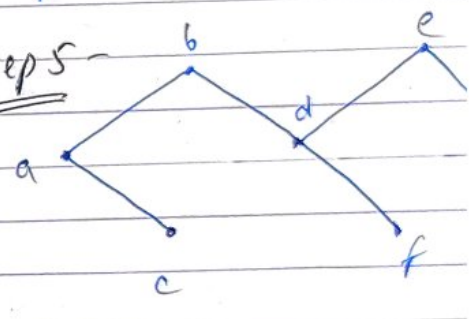
Step 3 -



Step 4 -



Step 5 -



DEPTH FIRST SEARCH ALGORITHM

Step 1- Arbitrary choose a vertix from a graph G & designate it as root.

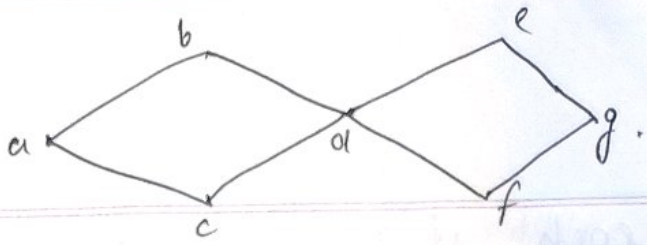
Step 2- form a path starting at this vertix by successively adding edges as long as possible where each new edge is incident with the last vertix in the path without producing any cycle.

Step 3- If the path goes through all the vertices of graph. The tree consisting of this path is a spanning tree. Otherwise go to the next step.

Step 4- Move back to the next to last vertix in the path & if possible, form a new path starting at this vertix passing through vertices that where not already visited.

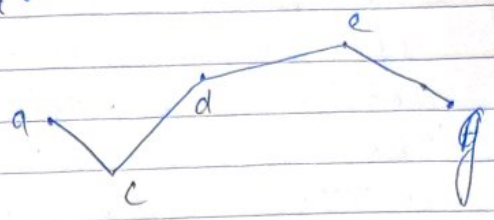
Step 5- If this cannot be done move back to another vertix in the path i.e., two vertix back in the path, repeat this procedure beginning at the last vertix visited.

Step 6- Moving back up the path one vertix at a time, forming new path that are as long as possible untill no more edges can be added.

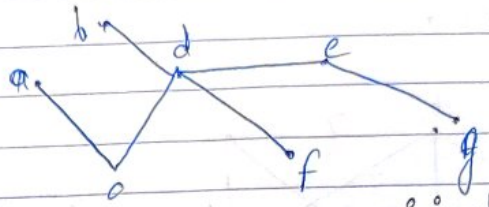


Step 1- a

Step 2-



Step 3-



Loop → within a vertix.

Circuit →

MST (MINIMUM SPANNING TREE).- (Based upon minimum weight).
→ or minimal

~~KRUSKAL's~~

KRUSKAL's ALGORITHM -

This Algorithm provides an acyclic graph T of a connected waited graph G which is minimum spanning tree of G.

I/P

O/P - MST T

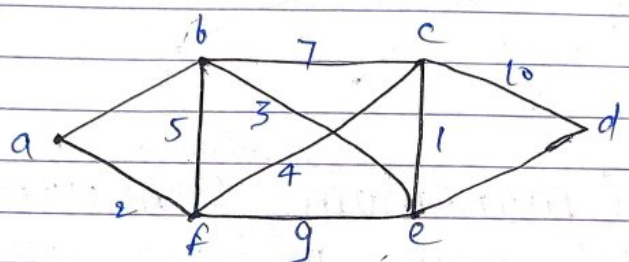
Step 1- List all the edges (which do not form a loop) of G in non-decreasing (increasing) order of their.

Step 2- Select an edge of minimal weight (if more than one edge of minimum weight, arbitrary choose one of them.

Step 3 - At each stage, select an edge of minimum weight from all the remaining edge of G. If doesnot form a circuit with the previously selected edge in T

Step 4 - Repeat the step 3 untill (n-1) edges are selected where n is no. of vertices.

Ex -

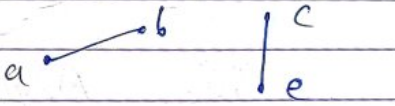


Step 1 -

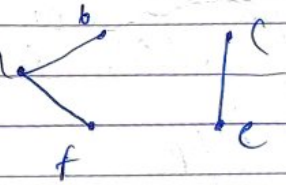
edges	ab	ce	af	be	cf	de	bf	bc	ef	ed
weight	1	1	2	3	4	4	5	7	9	10.

Step 2 - $T = \{ab\}$

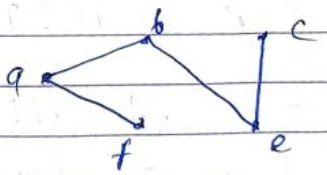
$T = \{(ab), (ce)\}$.



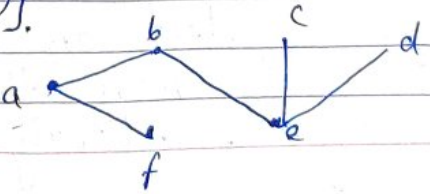
Step 3 - $T = \{(a,b), (c,e), (af)\}$



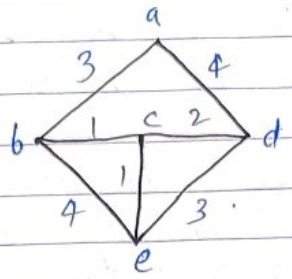
$T = \{(ab), (ce), (af), (be)\}$



$T = \{(ab), (ce), (af), (be), (de)\}$.



Q Draw the MST of the given graph with help of Kruskal's algorithm.



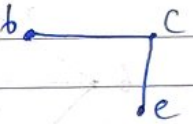
Answer Step 1:-

edge	ab	bc	ce	cd	de	ab	be	ad
weight	1	1	2	3	3	4	4	

Step 2-

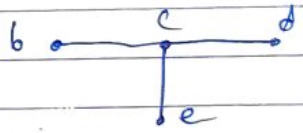
$T = \{bc\}$.

$T = \{(bc), (ce)\}$

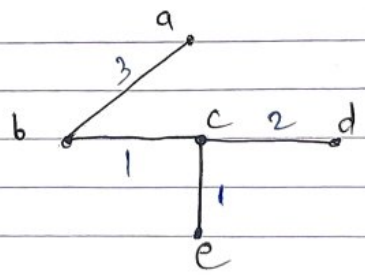


Step 3-

$T = \{(bc), (ce), (cd)\}$



$T = \{(bc), (ce), (cd), (ab)\}$



PRISM'S ALGORITHM -

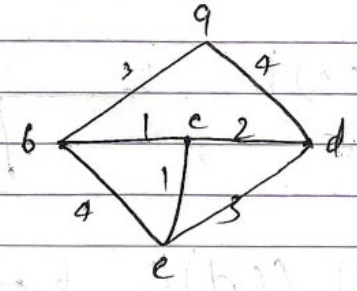
Theory as same as Kruskal's

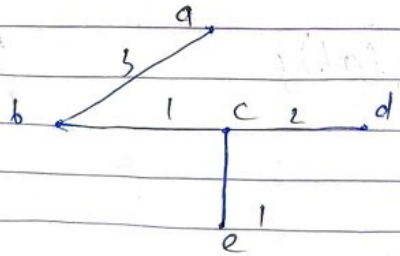
Step 1- Select any vertix in G among all the edges incident with selected vertix, choose an edge of minimum weight.

Step 2- At each stage choose an edge of smallest weight joining a vertix already included in T & a vertix not yet included, if it does not form a circuit with the edge in T.

Step 3- Repeat untill all the vertices of G are included in T.

Ex -





T {d, c, e, b, a}

DIGIKSTRA'S ALGORITHM -

By using digikstra's algorithm we find the length of shortest distance from source vertix to destination vertix in any connected, weighted graph.

Step 1 - Initially set the starting vertix a permanently with value 0. It implies $L(a) = 0$. & set $L(V) = \infty$ for all vertices $V \neq a$. with $T = \{$ All vertices $V \neq a$ having temporary label (name of vertex).

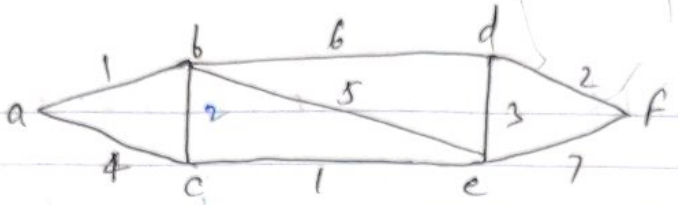
Step 2 - Let u be a vertix in T for which $L(u)$ is minimum & hence the permanent label of u.

Step 3 - If $u = Z$ where Z is a destination vertix. then stop. otherwise go to the next step.

Step 4 - for every edge e equals to (u,v) $e = (u,v)$ incident with u, & if $V \in T$ change $L(v)$ to $\min\{\text{old } L(v), L(u) + \underset{e}{w(u,v)}\}$.

Step 5 - change T to $T - \{u\}$. & Repeat from step 2.

ex-



Initial label is given by table.

Vertex	a	b	c	d	e	f
L(v)	0	∞	∞	∞	∞	∞
T	a	b	c	d	e	f

* Iteration 1 - $u = a$ has $L(u) = 0$. T becomes $T - \{a\}$. There are 2 edges incident with a i.e., ab & ac where $b \& c \in T$.

$L(b) = \min(\text{old}(L(b)), L(a) + w(a,b))$
$= \min(\infty, 0+1) = 1$

$L(c) = \min(\text{old}(L(c)), L(a) + w(a,c))$
$= \min(\infty, 0+4) = 4$

Since, $L(b) < L(c)$ minimum label is of b.
So, b vertex is choosen $L(b) = 1$.

Vertex	a	b	c	d	e	f
L(v)	0	1	4	∞	∞	∞
T		b	c	d	e	f

* Iteration 2 - $u = b$ has $L(u) = 1$. T becomes $T - \{b\}$. There are 3 edges incident with b i.e., bc, bd, be where $c, d, e \in T$.

$L(c) = \min(\text{old}(L(c)), L(b) + w(b,c))$
$= \min(4, 1+2) = 3$.

$L(d) = \min(\infty, 1+6) = 7$

$L(e) = \min(\infty, 1+5) = 6$

Since, $L(c) < L(e) < L(d)$. minimum label is of c. So, c vertex is choosen $L(c) = 3$.

Vertex	a	b	c	d	e	f
L(V)	0	1	3	7	6	∞
T			c	d	e	f

* Iteration 3 - $u = c$ has $L(u) = 3$. T becomes $T - \{c\}$. There are only one edge incident with e i.e., ce where $ce \in T$.

$L(e) = \min(old L(e)), L(c) + w(c, e)$
$= \min(6, 3+1) = 4$

Since, only one vertex is there. minimum label is of e. So, e vertex is choosen $L(e) = 4$

Vertex	a	b	c	d	e	f
L(V)	0	1	3	7	4	∞
T				d	e	f

* Iteration - 4 $u = e$ has $L(e) = 4$. T becomes $T - \{e\}$. There are two edges incident with e i.e., ed & ef where $d \& f \in T$.

$L(d) = \min(old(L(d)), L(e) + w(e, d))$
$= \min(7, 4+3) = 7$.
$L(f) = \min(\infty, 4+7) = 11$.

Since, $L(d) < L(f)$ minimum label is of d So, d vertex is choosen $L(d) = 7$.

Vertex	a	b	c	d	e	f
L(v)	0	1	3	7	4	18
T				d		f

* Iteration - 5 $u = d$. has $L(d) = 7$. T becomes $T - \{d\}$. There are only edge incident with d i.e., df where $f \in T$.

$L(f) = \min(\text{old}(L(f)), L(d) + w(d,f))$

$= \min(18, 7+2) = 9$.

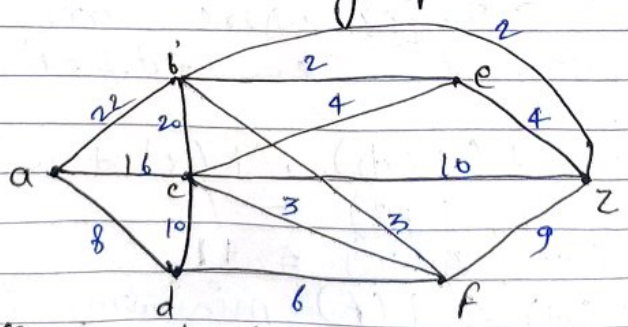
Since, only one vertex is there so, minimum label is of f, f vertex is choosen $L(f) = 9$

Vertex	a	b	c	d	e	f
L(v)	0	1	3	7	4	9
T						f

Min. Path for moving a to f.

$a \to b \to c \to e \to d \to f$

Q In below graph



find the shortest distance b/w a & z.

Using Digikstra's Rule -

Normally, it applies on directed graph.

FLOYD WARSHALL ALGORITHM.

* Floyd Warshall Algorithm is used for solving all pair shortest path problem which gives the shortest path between every pair of vertices in a given graph.
* Floyd Warshall algorithm is an example of dynamic programming.
* The main advantage of F.W.A. is it is very simple & easy to implement.

* ALGORITHM

STEP 1 - Remove all the self loop & parallel edges (keeping the edge with lowest weight from a graph.)

STEP 2 - Create VXV matrix where V is the vertex in a graph G. & value of matrix is given by

$$M[i][j] = \begin{cases} 0, & \text{when } i=j \text{ i.e., the diagonal elements.} \\ \infty, & \text{when } i \neq j. \text{ if there is no direct edge b/w } i^{th} \text{ \& } j^{th} \\ weight(i,j) & \text{if there exist a direct vertex to } i \text{ to } j. \end{cases}$$

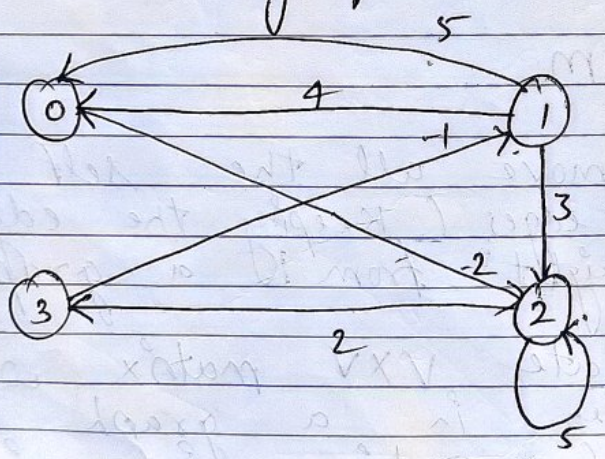
STEP-3

For each vertex K, where K is an intermediate vertex from i to j vertex. Calculate a distance matrix.

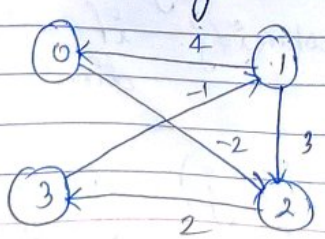
```
for(K=0 to K<=V-1)
    for( i=0 to i<=V-1)
        for (j=0 to V-1)
            if((d[i][K] + d[K][j]) < d[i][j])
                d[i][j] = d[i][K] + a[K][j]
```

Q Consider a graph G.

=



Step 1- Remove all self loop & parallel edges - the graph will be -



$$D = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{bmatrix} \end{array}$$

Weight Matrix (Vertex X Vertex) of a Graph

$D_0 =$

	0	1	2	3
0	0	∞	-2	∞
1	4	0	2	∞
2	∞	∞	0	2
3	∞	-1	∞	0

$D_1 =$

	0	1	2	3
0	0	∞	-2	∞
1	4	0	2	∞
2	∞	∞	0	2
3	3	-1	1	0

$D_2 =$

	0	1	2	3
0	0	∞	-2	0
1	4	0	2	4
2	∞	∞	0	2
3	3	-1	1	0

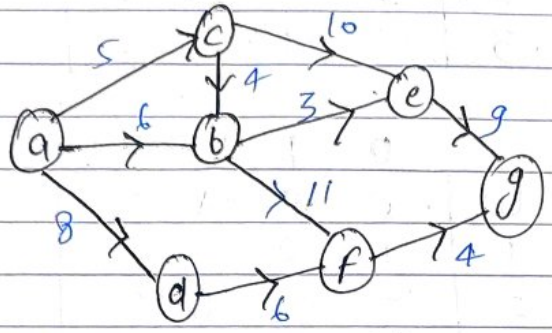
$D_3 =$

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

Above matrix is required matrix.

Sorting RMQ CMQ, Postfix,

Q Find the minimum distance b/w every pair of vertices of following graph.



$D =$

	a	b	c	d	e	f	g
a	0	6	5	8	∞	∞	∞
b	∞	0	∞	∞	3	11	∞
c	∞	4	0	∞	10	∞	∞
d	∞	∞	∞	0	∞	6	∞
e	∞	∞	∞	∞	0	∞	9
f	∞	∞	∞	∞	∞	0	4
g	∞	∞	∞	∞	∞	∞	0

$D_a =$

	a	b	c	d	e	f	g
a	0	6	5	8	∞	∞	∞
b	∞	0	∞	∞	3	11	∞
c	∞	4	0	∞	10	∞	∞
d	∞	∞	∞	0	∞	6	∞
e	∞	∞	∞	∞	0	∞	9
f	∞	∞	∞	∞	∞	0	4
g	∞	∞	∞	∞	∞	∞	0

3, 4 or 5

$D_b =$

	a	b	c	d	e	f	g
a	0	6	5	8	9	17	∞
b	∞	0	∞	∞	3	11	∞
c	∞	4	0	∞	7	15	∞
d	∞	∞	∞	0	∞	6	∞
e	∞	∞	∞	∞	0	∞	9
f	∞	∞	∞	∞	∞	0	4
g	∞	∞	∞	∞	∞	∞	0

$D_c =$

	a	b	c	d	e	f	g
a	0	6	5	8	9	17	∞
b	∞	0	∞	∞	3	11	∞
c	∞	4	0	∞	7	∞	∞
d	∞	∞	∞	0	∞	6	∞
e	∞	∞	∞	∞	0	∞	9
f	∞	∞	∞	∞	∞	0	4
g	∞	∞	∞	∞	∞	∞	0

$D_d =$

	a	b	c	d	e	f	g
a	0	6	5	8	9	14	∞
b	∞	0	∞	∞	3	11	∞
c	∞	4	0	∞	10	∞	∞
d	∞	∞	∞	0	∞	6	∞
e	∞	∞	∞	∞	0	∞	9
f	∞	∞	∞	∞	∞	0	4
g	∞	∞	∞	∞	∞	∞	0

$D_e =$

	a	b	c	d	e	f	g
a	0	6	5	8	9	14	18
b	∞	0	∞	∞	3	11	12
c	∞	4	0	∞	10	∞	19
d	∞	∞	∞	0	∞	6	∞
e	∞	∞	∞	∞	0	∞	9
f	∞	∞	∞	∞	∞	0	4
g	∞	∞	∞	∞	∞	∞	0

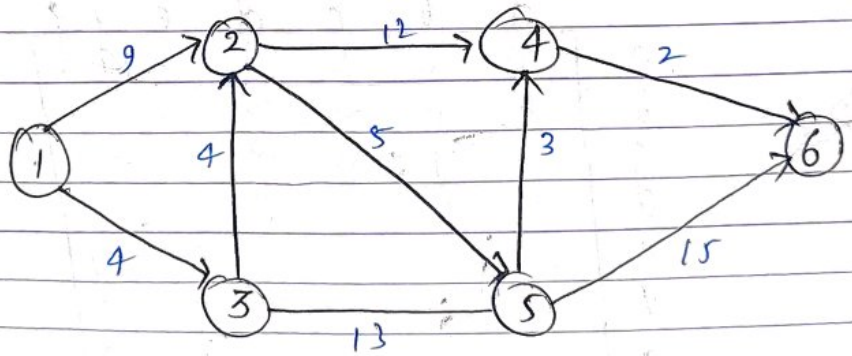
P_f =

	a	b	c	d	e	f	g
a	0					14	
b						11	
c						∞	
d						6	
e						∞	
f	∞	∞	∞	∞	∞	0	4
g						∞	

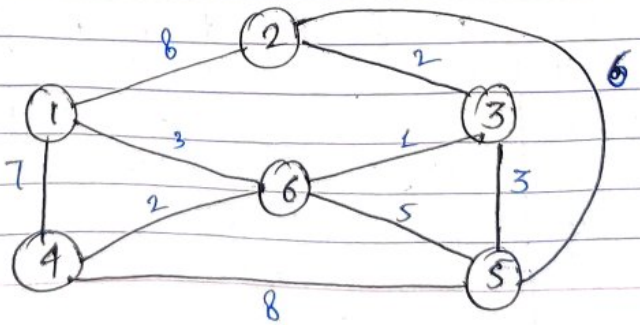
P_g =

	a	b	c	d	e	f	g
a							
b							
c							
d							
e							
f							
g							

Q 1 –



Q 2 –



Solution - 2

$D =$

	1	2	3	4	5	6
1	0	8	∞	7	∞	~~3~~
2	8	0	2	∞	~~6~~	∞
3	∞	2	0	∞	~~3~~	~~1~~
4	7	∞	∞	0	8	2
5	∞	~~6~~	3	8	0	5
6	3	∞	1	2	5	0

$D_1 =$

	1	2	3	4	5	6
1	0	8	∞	7	∞	3
2	8	0	2	15	6	11
3	∞	2	0	∞	3	1
4	7	15	∞	0	8	2
5	∞	6	3	8	0	5
6	3	11	1	2	5	0

$D_2 =$

	1	2	3	4	5	6
1	0	8	10	7	14	3
2	8	0	2	15	6	11
3	10	2	0	17	3	1
4	7	15	17	0	8	2
5	14	6	3	8	0	5
6	3	11	1	2	5	0

$D_3 =$

	1	2	3	4	5	6
1	0	8	10	7	13	3
2	8	0	2	15	5	3
3	10	2	0	17	3	1
4	7	15	17	0	8	2
5	~~19~~	5	3	8	0	~~4~~
6	3	3	1	2	4	0

$D_4 =$

	1	2	3	4	5	6
1	0	8	10	7	13	3
2	8	0	2	15	5	3
3	10	2	0	17	3	1
4	7	15	17	0	8	2
5	13	5	3	8	0	4
6	3	3	1	2	4	0

$D_5 =$

	1	2	3	4	5	6
1	0	8	10	7	13	3
2	8	0	2	13	5	3
3	10	2	0	11	3	1
4	7	13	11	0	8	2
5	13	5	3	8	0	4
6	3	3	1	2	4	0

$D_6 =$

	1	2	3	4	5	6
1	0	6	4	5	7	3
2	6	0	2	5	5	3
3	4	2	0	3	3	1
4	5	5	3	0	6	2
5	7	5	3	6	0	4
6	3	3	1	2	4	0