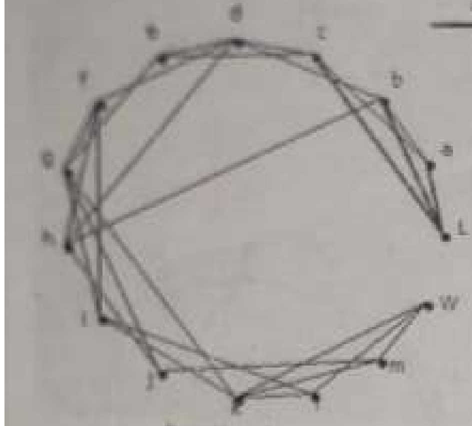
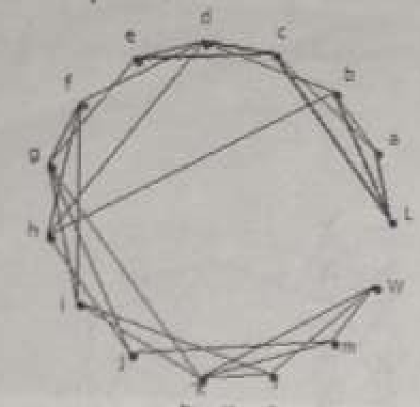
Alan Tucker, Graph Th & Combinatorics

Sandipan Dey

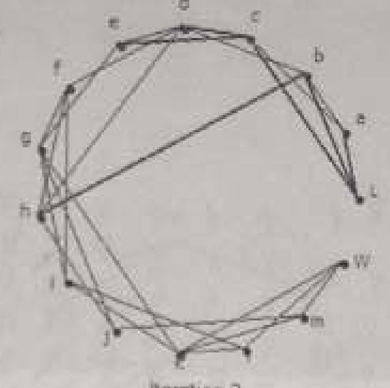
Math-685 HW 4.1 & 4.2



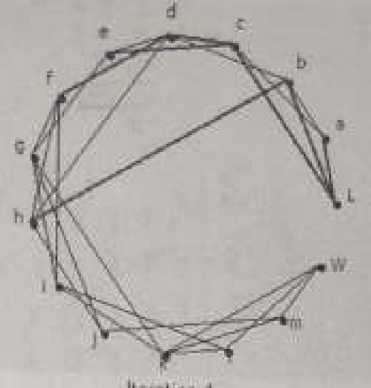
Iteration 1



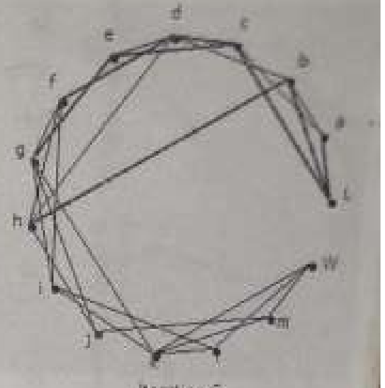
Iteration 2



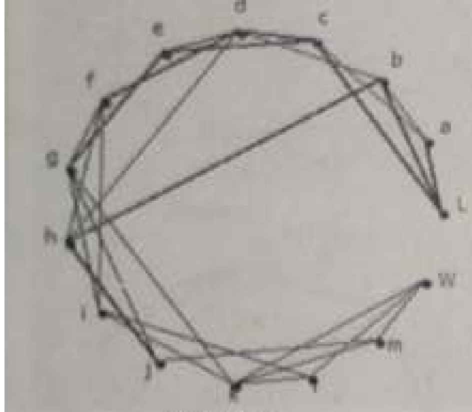
Iteration 3



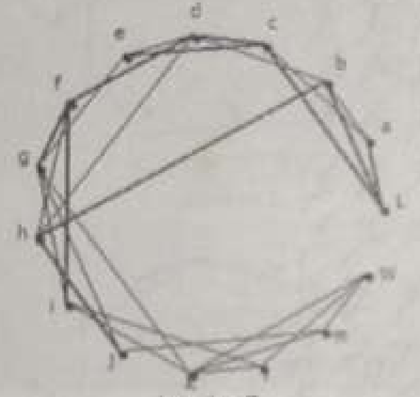
Iteration 4



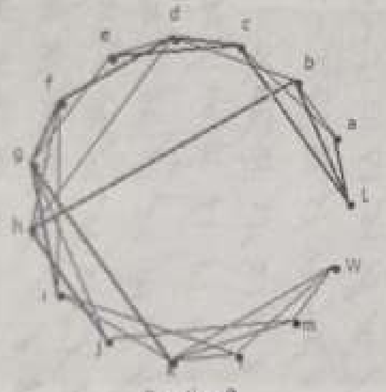
Iteration 5



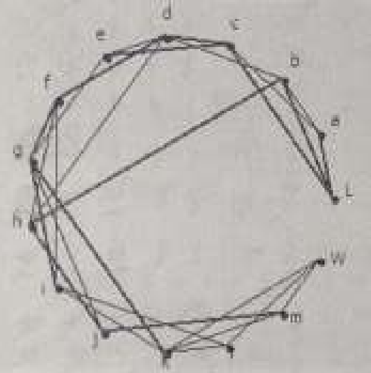
Iteration 6



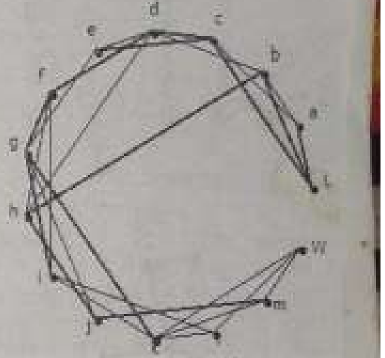
Iteration 7



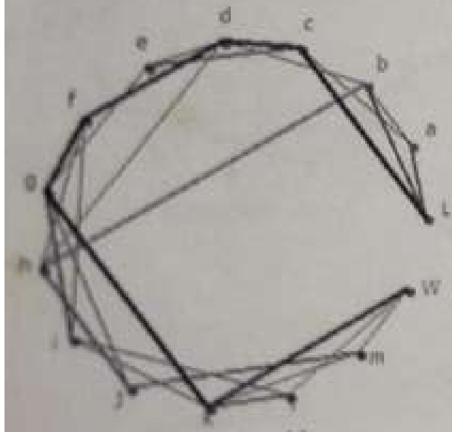
Iteration 8



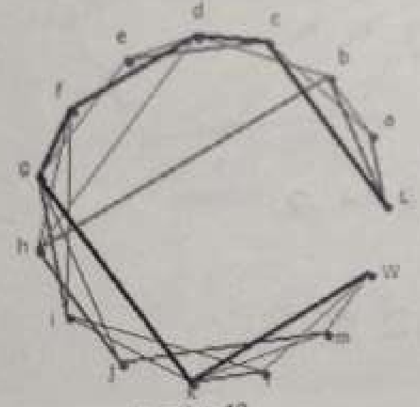
Iteration 9



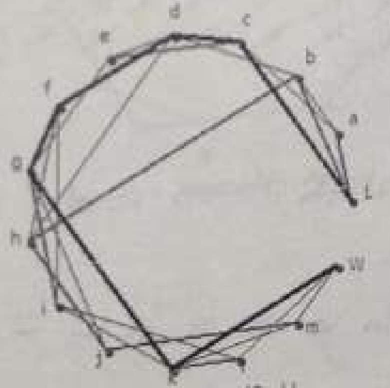
Iteration 10



Iteration 11



Iteration 12



Iteration 13, 14

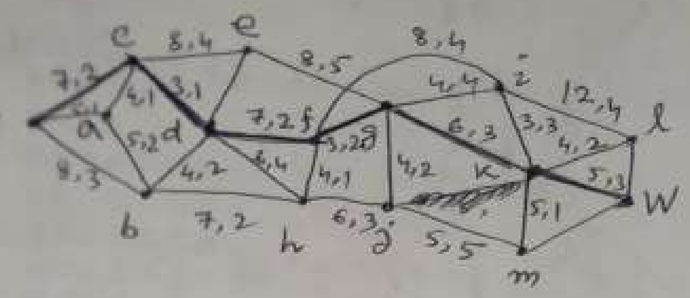
	Distance from L															
Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	0	5	8	7	?	?	?	?	?	?	?	?	?	?	?	L
1	0	5	8	7	?	?	?	?	?	?	?	?	?	?	?	a
2	0	5	8	7	10	15	?	?	?	?	?	?	?	?	?	c
3	0	5	8	7	10	15	?	?	15	?	?	?	?	?	?	b
4	0	5	8	7	10	15	17	?	15	?	?	?	?	?	?	d
5	0	5	8	7	10	15	17	23	15	?	?	?	?	?	?	e
6	0	5	8	7	10	15	17	23	15	?	21	?	?	?	?	h
7	0	5	8	7	10	15	17	20	15	25	21	?	?	?	?	f
8	0	5	8	7	10	15	17	20	15	24	21	26	?	?	?	g
9	0	5	8	7	10	15	17	20	15	24	21	26	?	26	?	j
10	0	5	8	7	10	15	17	20	15	24	21	26	36	26	?	i
11	0	5	8	7	10	15	17	20	15	24	21	26	30	26	31	[illegible]
12	0	5	8	7	10	15	17	20	15	24	21	26	30	26	31	[illegible]
13,14	0	5	8	7	10	15	17	20	15	24	21	26	30	26	31	[illegible]

20/20

L-c-d-f-g-K-W (Shortest path)

(? ≡ ∞)

(The graph drawn with the vertices on a circle)



Source = L

destination = W

As shown, ~~minimum~~ shortest distance from

L to W is 31 (L-c-d-f-g-k-W)

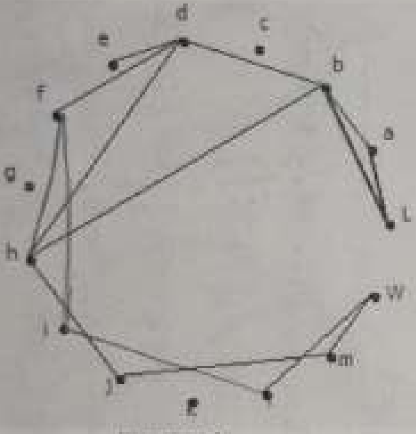
Distance from source to other nodes

iteration #	Next vertex chosen	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W
0	L	~~0~~	5	8	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	a	~~0~~	~~5~~	8	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	c	~~0~~	~~5~~	8	~~7~~	10	15	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	b	~~0~~	~~5~~	~~8~~	~~7~~	10	15	∞	∞	15	∞	∞	∞	∞	∞	∞
4	d	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	15	17	∞	15	∞	∞	∞	∞	∞	∞
5	e	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	17	23	15	∞	∞	∞	∞	∞	∞
6	h	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	17	23	~~15~~	∞	21	∞	∞	∞	∞
7	f	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	20	~~15~~	25	21	∞	∞	∞	∞
8	g	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	24	21	26	∞	∞	∞
9	j	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	24	~~21~~	26	∞	25	∞
10	i	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	~~24~~	~~21~~	26	36	25	∞
11	k	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	~~24~~	~~21~~	~~26~~	30	26	31
12	m	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	~~24~~	~~21~~	~~26~~	30	~~26~~	31
13	l	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	~~24~~	~~21~~	~~26~~	~~30~~	~~26~~	31
14	W	~~0~~	~~5~~	~~8~~	~~7~~	~~10~~	~~15~~	~~17~~	~~20~~	~~15~~	~~24~~	~~21~~	~~26~~	~~30~~	~~26~~	~~31~~

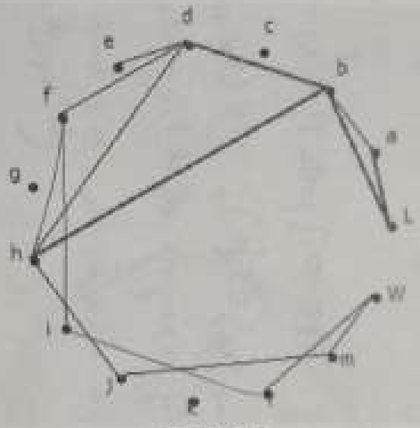
→ [d(a) + w(a,c) >
[d(a) + w(a,b)
nothing d

3. (b) In order to exclude c, g, k from the shortest path, we remove the edges incident on these vertices and run Dijkstra on the graph, to get the shortest path (quickest)

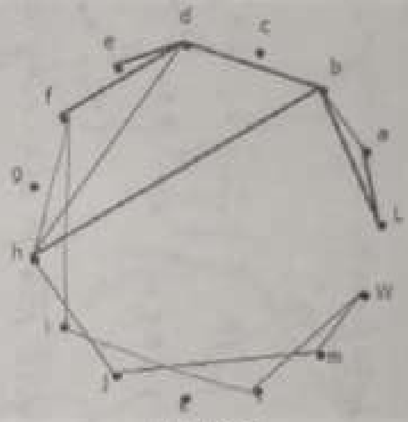
L-b-h-j-m-W as shown



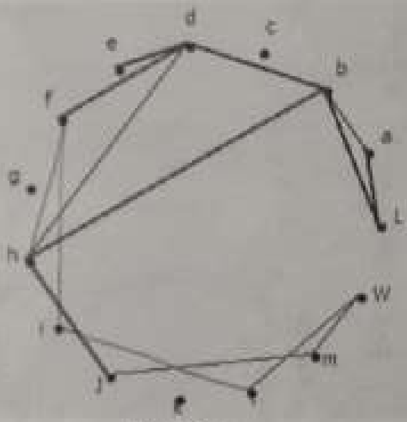
Iteration 1



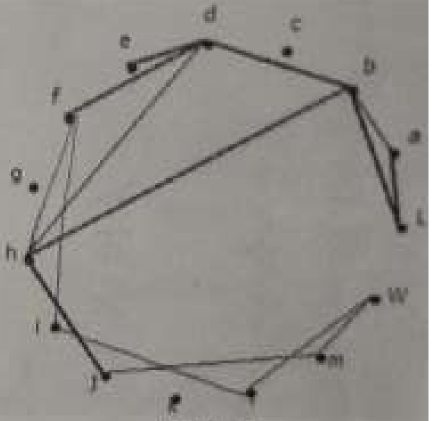
Iteration 2



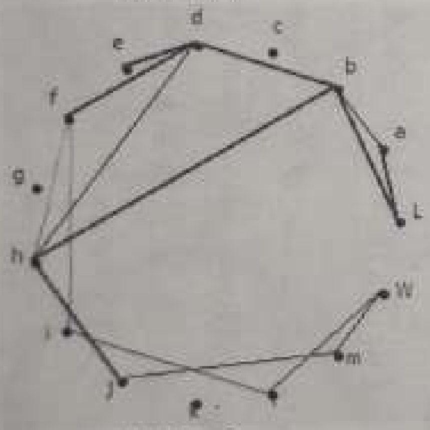
Iteration 3



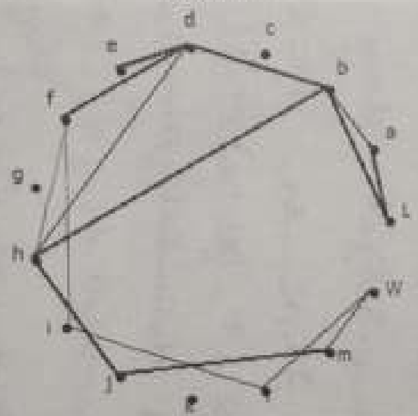
Iteration 4



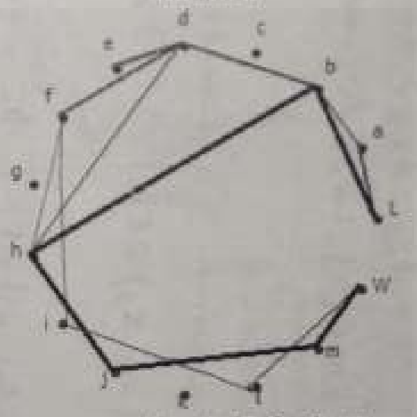
Iteration 5



Iteration 6



Iteration 7



Iteration 8, 9, 10, 11

	Distance from L															
Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	0	5	8	?	?	?	?	?	?	?	?	?	?	?	?	L
1	0	5	8	?	?	?	?	?	?	?	?	?	?	?	?	a
2	0	5	8	?	12	?	?	?	15	?	?	?	?	?	?	b
3	0	5	8	?	12	17	19	?	15	?	?	?	?	?	?	d
4	0	5	8	?	12	17	19	?	15	?	21	?	?	?	?	h
5	0	5	8	?	12	17	19	?	15	?	21	?	?	?	?	e
6	0	5	8	?	12	17	19	?	15	27	21	?	?	?	?	f
7	0	5	8	?	12	17	19	?	15	27	21	?	?	26	?	j
8	0	5	8	?	12	17	19	?	15	27	21	?	?	26	32	m
9,10, 11	0	5	8	?	12	17	19	?	15	27	21	?	37	26	32	i, W, l

4.1.

12. Let's prove the following modification of Floyd's algorithm comp the transitive closure in a directed graph G.

```
/* initialization of matrix D */
for i ← 1 to n do                /* n = |V(G)| */
    for j ← 1 to n do
        if ~~i=j OR~~ (i,j) ∈ E[G] then d_ij ← 1 else d_ij ← 0;
/* algorithm steps */
for k ← 1 to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if d_ik · d_kj > d_ij then d_ij ← 1;        ①
```

$\underbrace{(i,j) \in E[G]}$ there is an edge inbetween x_i & x_j in G directed from x_i to x_j

5/5

To prove: $\overrightarrow{(x_i, x_j)} \in E(G)$, i.e., $\exists$ directed path from x_i to x_j $\Leftrightarrow d_{ij} = 1$

Proof: $(\Rightarrow)$

Let's assume $\exists$ a directed path P from x_i to x_j, $x_i \overset{P}{\rightsquigarrow} x_j$, $\overset{P_r}{(x_i, x_j)} \in E[G]$

$\Rightarrow (x_i, x_j) \in E[G] \vee \left(\exists k, 1 \le k \le n \mid x_i \overset{P_1}{\rightsquigarrow} x_k \overset{P_2}{\rightsquigarrow} x_j\right.$, i.e., $\left.(x_i, \overset{P_1}{\rightarrow} x_k) \in E[G] \wedge (x_k, \overset{P_2}{\rightarrow} x_j) \in E[G]\right)$

if $(x_i, x_j) \in E[G]$, the modified Floyd already assigns d_{ij} to 1 in the initialization phase.

ow, $\exists k_1, k_2 \ldots k_r$, $1 \le k_i \le n \mid \exists$ path $P \equiv x_i - x_{k_1} - x_{k_2} - \cdots - x_{k_r} - x_j$ in G s.t., $(x_i, x_{k_1}) \in E[G]$, $(x_{k_1}, x_{k_2}) \in E[G]$, …, $(x_{k_r}, x_j) \in E[G]$.

s.t. the initialization already assigns $d_{ik_1} = d_{k_1k_2} = \cdots = d_{k_rj} = 1$ and

① assigns $d_{ik_2} \leftarrow 1$, since $d_{ik_1} = d_{k_1k_2} = 1$ and $d_{ik_2} = 0$ and $(d_{ik_1} \cdot d_{k_1k_2} > d_{ik_2})$ satisfied

Continuing this way $d_{ij} \leftarrow 1$.

$\therefore \overset{T}{(x_i, x_j)} \in E[G] \Rightarrow d_{ij} = 1$.

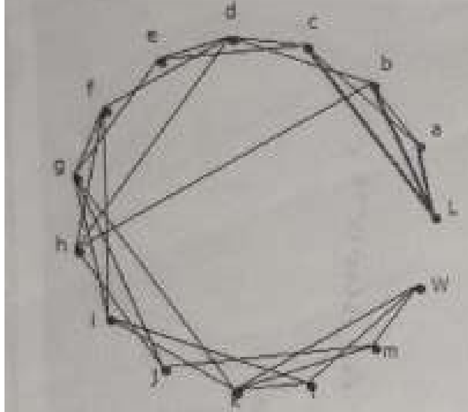
$d_{ij} = 1$ iff $d_{ik} \cdot d_{kj} = 1$ i.e. $d_{ij} = 1$ iff $(d_{ik} = 1 \wedge d_{kj} = 1)$

$\overrightarrow{(x_i, x_j)} \in E$ iff $\exists k \,/\, \overrightarrow{(x_i, x_k)} \in E \wedge \overrightarrow{(x_k, x_j)} \in E$

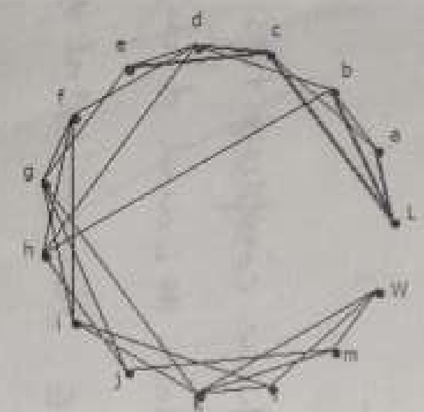
similar conditions

$(\Leftarrow)$ Let's assume the algorithm assign $d_{ij} \leftarrow 1 \Rightarrow$ it was assigned in the initialization phase (i.e. $(x_i, x_j) \in E[G]$) or in ①, which means $\exists k, 1 \le k \le n \mid d_{ik} \cdot d_{kj} > d_{ij}$, but this can only happen if $d_{ik} = d_{kj} = 1$ and $d_{ij} = 0$, i.e., (x_i, x_j) are non-adjacent but $\exists x_{k_1} \in V[G]$ s.t. $(x_i, x_{k_1}) \in E[G]$ and $\ell = 1..r$, $(x_{k_\ell}, x_{k_{\ell+1}}) \in E$ and $(x_{k_r}, x_j) \in E[G] \Leftrightarrow d_{ik_1} = d_{k_1k_2} = \cdots = d_{k_rj} = 1 \Rightarrow d_{ij} = 1$

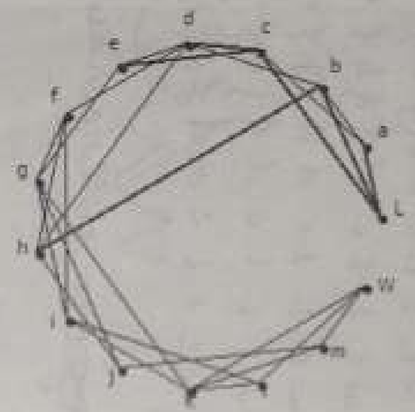
3.(c) As shown below, the path to success is L-c-d-f-g-K-W



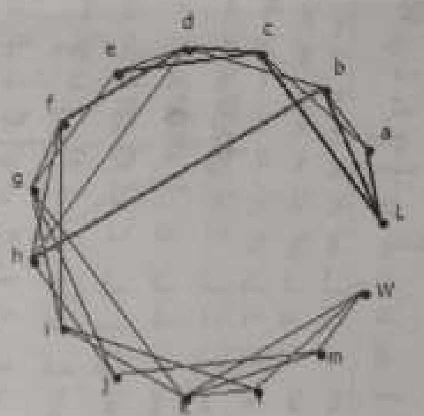
Iteration 1



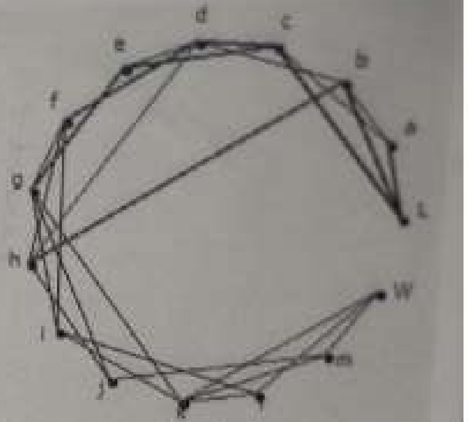
Iteration 2



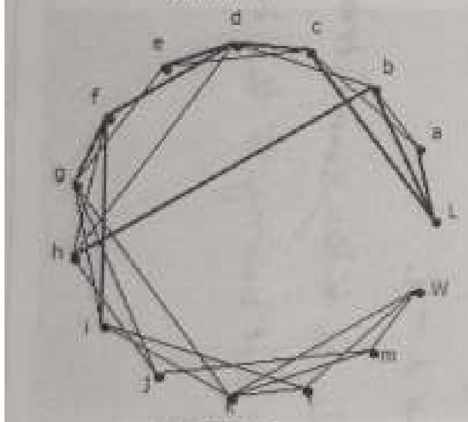
Iteration 3



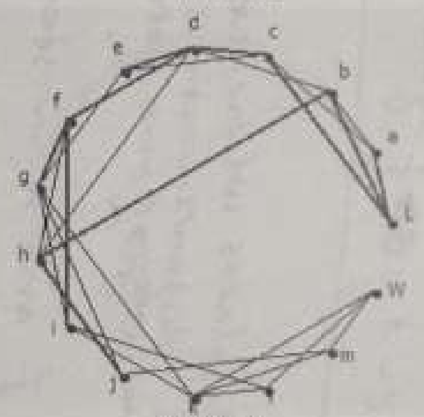
Iteration 4



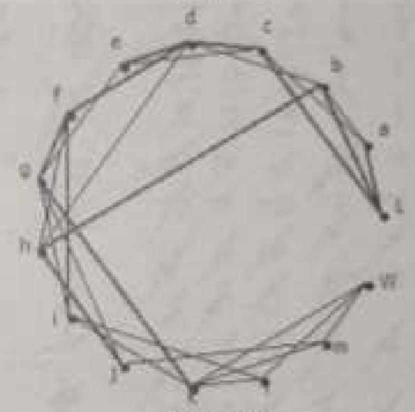
Iteration 5



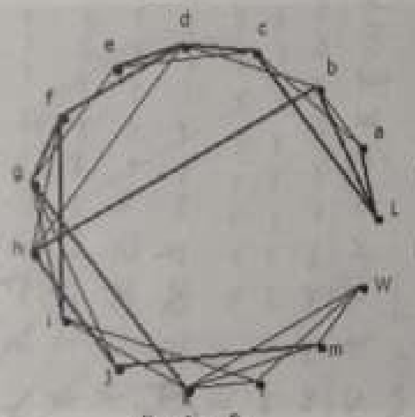
Iteration 6



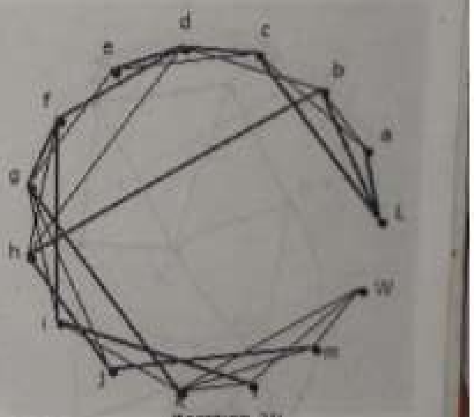
Iteration 7



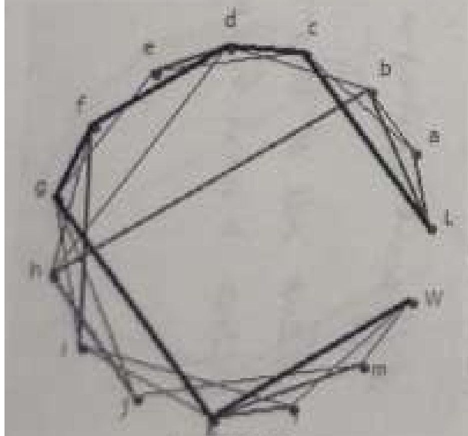
Iteration 8



Iteration 9



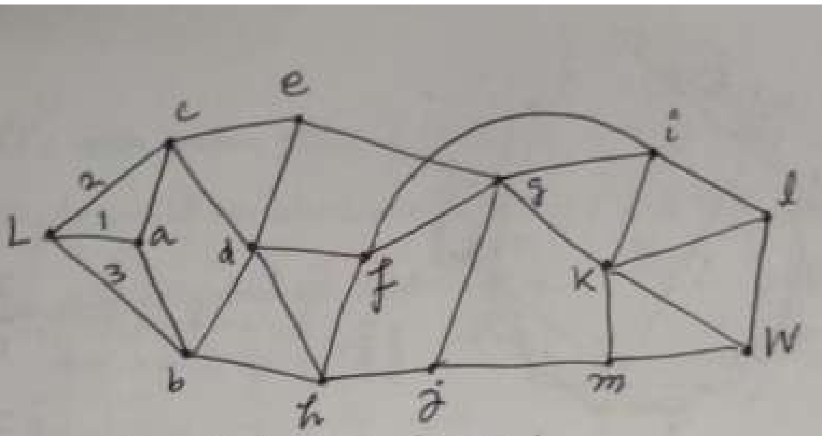
Iteration 10



Iteration 11

	Distance from L														
Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	0	1	3	2	?	?	?	?	?	?	?	?	?	?	?	L
1	0	1	3	2	?	?	?	?	?	?	?	?	?	?	?	a
2	0	1	3	2	3	6	?	?	?	?	?	?	?	?	?	c
3	0	1	3	2	3	6	7	?	5	?	?	?	?	?	?	b
4	0	1	3	2	3	4	5	?	5	?	?	?	?	?	?	d
5	0	1	3	2	3	4	5	9	5	?	?	?	?	?	?	e
6	0	1	3	2	3	4	5	7	5	9	?	?	?	?	?	f
7	0	1	3	2	3	4	5	7	5	9	8	?	?	?	?	h
8	0	1	3	2	3	4	5	7	5	9	8	10	?	?	?	g
9	0	1	3	2	3	4	5	7	5	9	8	10	?	13	?	j
10	0	1	3	2	3	4	5	7	5	9	8	10	13	13	?	i
11	0	1	3	2	3	4	5	7	5	9	8	10	12	11	13	k
12, 13, 14	0	1	3	2	3	4	5	7	5	9	8	10	12	11	13	m, l, W

3.(d)



iteration#	distance from L															parents														
	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W
0	~~0~~	1	3	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	L	L	L	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
1	~~0~~	~~1~~	3	2	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	L	L	L	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	~~0~~	~~1~~	3	~~2~~	3	6	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	L	L	L	c	c	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	~~0~~	~~1~~	3	~~2~~	~~3~~	4	5	∞	7	∞	∞	∞	∞	∞	∞	∞	L	L	L	c	d	d	∞	d	∞	∞	∞	∞	∞	∞
4	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	4	5	∞	5	∞	∞	∞	∞	∞	∞	∞	L	L	L	c	d	d	∞	b	∞	∞	∞	∞	∞	∞
5	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	5	9	5	∞	∞	∞	∞	∞	∞	∞	L	L	L	c	d	d	f	b	f	∞	∞	∞	∞	∞
6	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	~~5~~	7	5	9	∞	∞	∞	∞	∞	∞	L	L	L	c	d	d	f	b	f	i	∞	∞	∞	∞
7	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	~~5~~	7	~~5~~	9	8	∞	∞	∞	∞	∞	L	L	L	c	d	d	f	b	f	i	g	∞	∞	∞
8	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	~~5~~	~~7~~	~~5~~	9	~~8~~	10	∞	∞	∞	∞	L	L	L	c	d	d	f	b	f	i	g	∞	j	∞
9	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	~~5~~	~~7~~	~~5~~	9	~~8~~	10	∞	13	∞	∞	L	L	L	c	d	d	f	b	f	i	g	i	j	∞
10	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	~~5~~	~~7~~	~~5~~	~~9~~	~~8~~	10	13	13	∞	∞	L	L	L	c	d	d	f	b	f	i	g	K	K	K
11	~~0~~	~~1~~	~~3~~	~~2~~	~~3~~	~~4~~	~~5~~	~~7~~	~~5~~	~~9~~	~~8~~	~~10~~	12	11	13															

Hence the shortest path from L to W ~~changes to~~ still remains

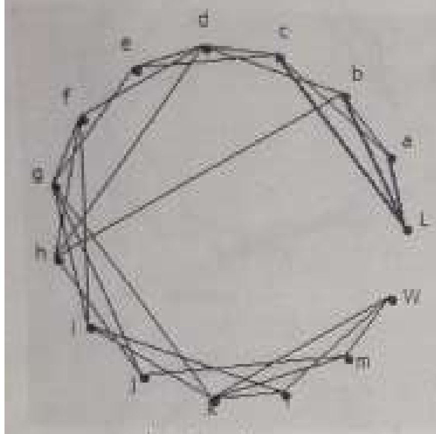
L-c-d- f-g-K-W

There are couple of places where we have ~~couple of~~ ties (2choices) for minimum ~~stage~~ distance vertices; one between ~~a~~ b and d in step3 the other between f and h in step 6. Hence, total # different choices = 2×2=4.

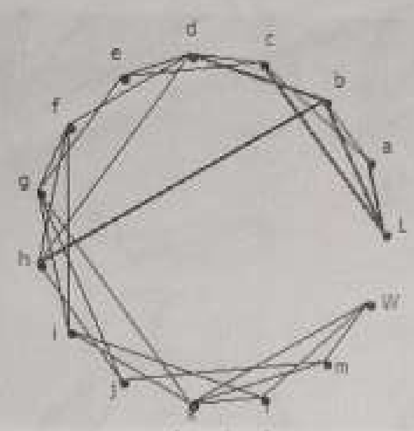
But all of them lead to the same tree as shown.

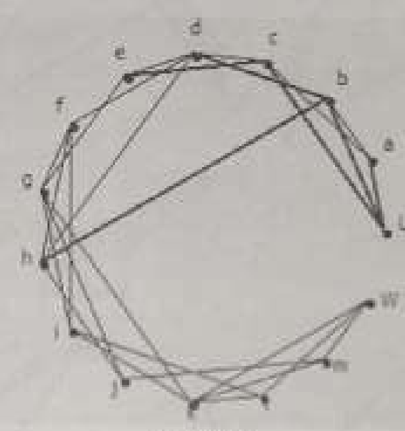
(since b and h does not beong to the tree)

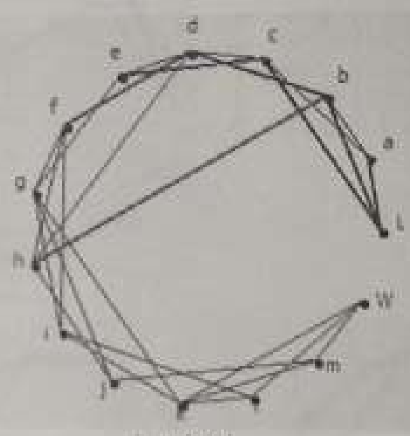
5.(a) Shortest path from L to W (as shown) is L-c-d-f-g-k-W

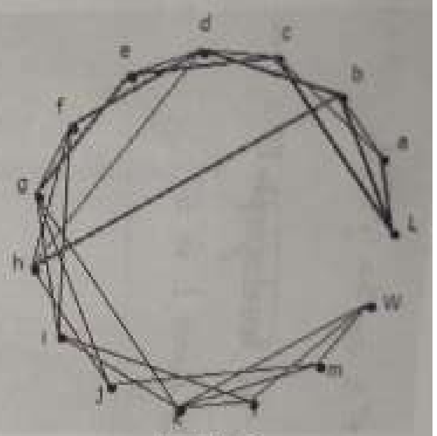


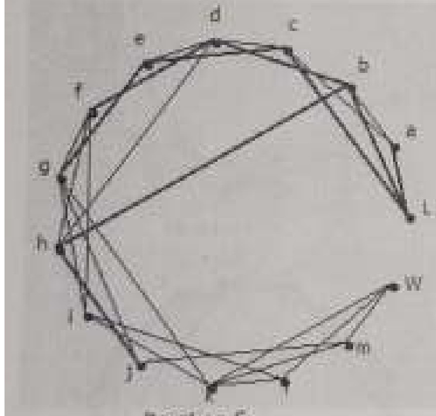
Iteration 1

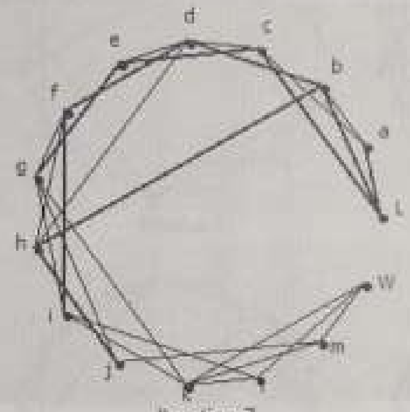

Iteration 2

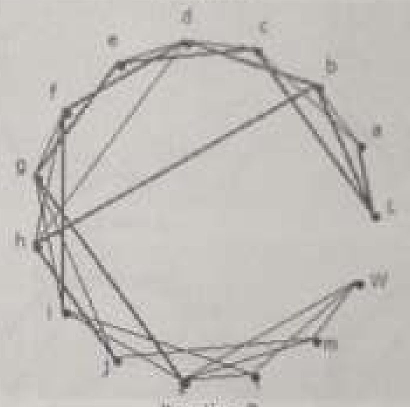

Iteration 3

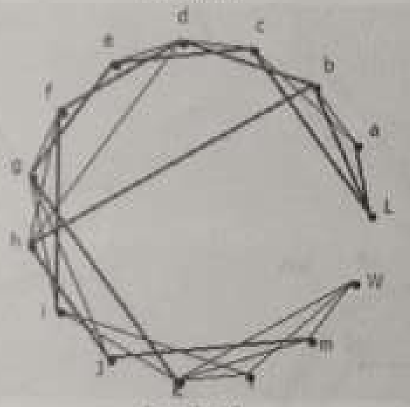

Iteration 4

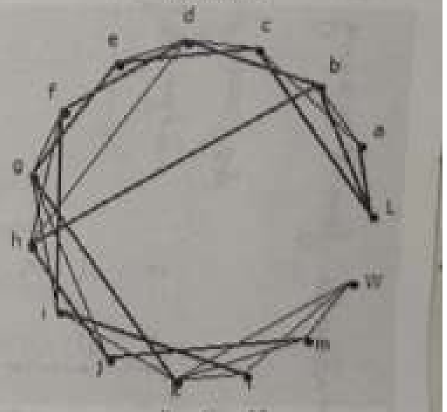

Iteration 5

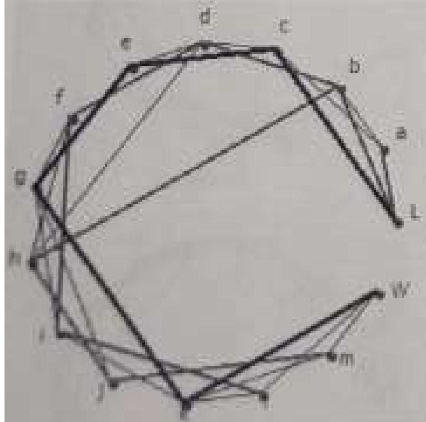

Iteration 6

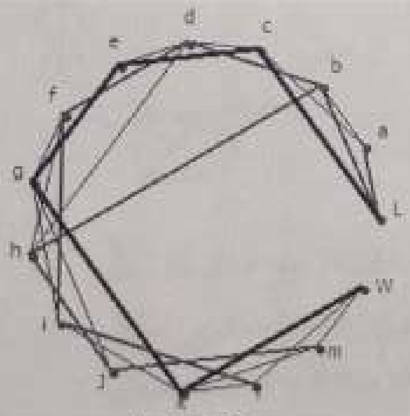

Iteration 7

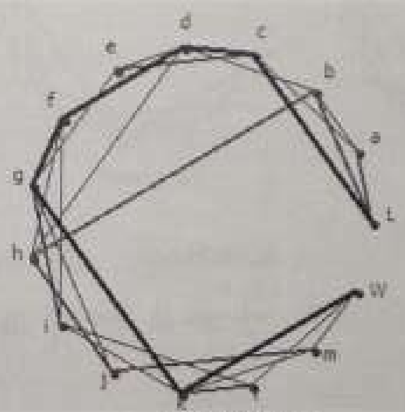

Iteration 8


Iteration 9


Iteration 10

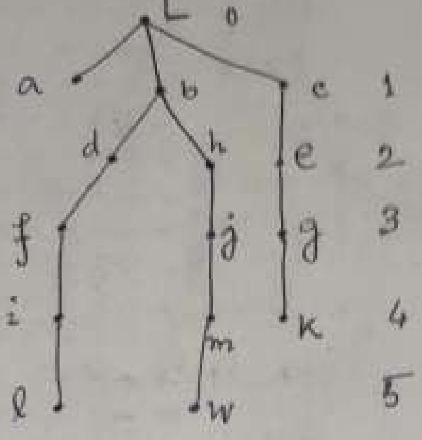

Iteration 11


Iteration 12


Iteration 13 / 14

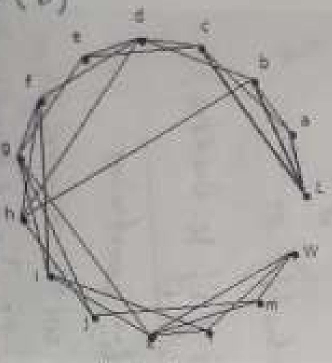
Iteration	Distance from L L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	L
1	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	a
2	0	1	1	1	2	?	?	?	2	?	?	?	?	?	?	b
3	0	1	1	1	2	2	?	?	2	?	?	?	?	?	?	c
4	0	1	1	1	2	2	3	?	2	?	?	?	?	?	?	d
5	0	1	1	1	2	2	3	3	2	?	?	?	?	?	?	e
6	0	1	1	1	2	2	3	3	2	?	3	?	?	?	?	h
7	0	1	1	1	2	2	3	3	2	4	3	?	?	?	?	f
8	0	1	1	1	2	2	3	3	2	4	3	4	?	?	?	g
9	0	1	1	1	2	2	3	3	2	4	3	4	?	4	?	j
10	0	1	1	1	2	2	3	3	2	4	3	4	5	4	?	i
11	0	1	1	1	2	2	3	3	2	4	3	4	5	4	5	k
12	0	1	1	1	2	2	3	3	2	4	3	4	5	4	5	m
13	0	1	1	1	2	2	3	3	2	4	3	4	5	4	5	l
14	0	1	1	1	2	2	3	3	2	4	3	4	5	4	5	W

~~Wh~~ This shortest path can also be found using BFS on the graph

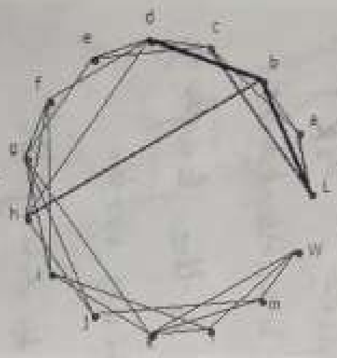


As seen from the ~~level graph~~ BFS tree distance of W from L is 5.

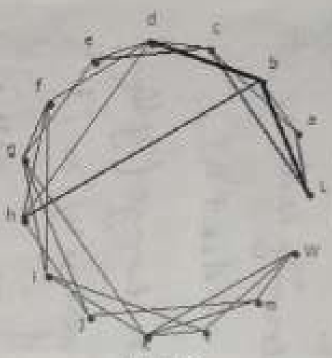
5.(b)



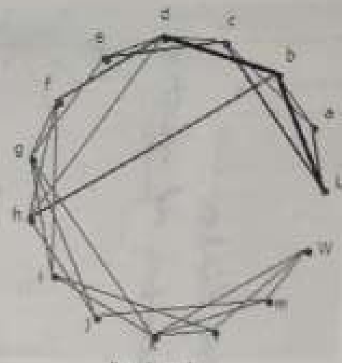
Iteration 1



Iteration 2



Iteration 3

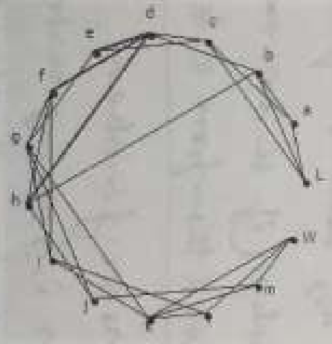


Iteration 4

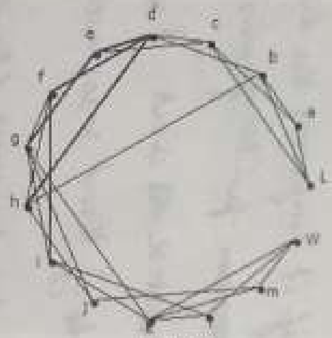
Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	L
1	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	a
2	0	1	1	1	2	?	?	?	2	?	?	?	?	?	?	b
3	0	1	1	1	2	2	?	?	2	?	?	?	?	?	?	c
4	0	1	1	1	2	2	3	?	2	?	?	?	?	?	?	d

Distance from L

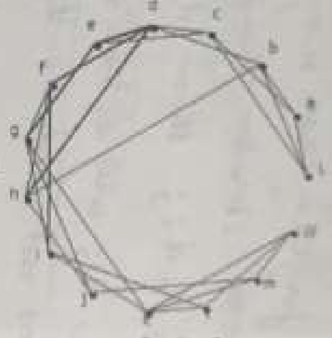
Shortest path from L to d



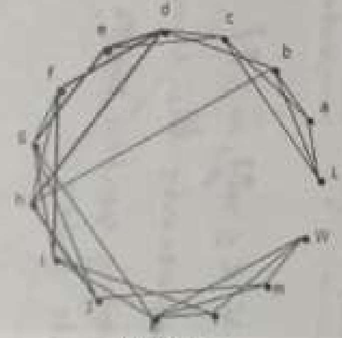
Iteration 1



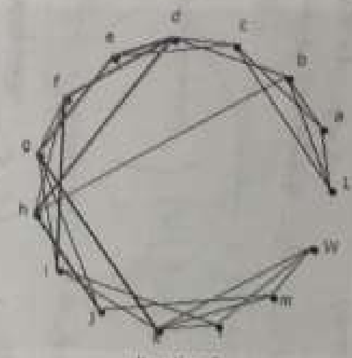
Iteration 2



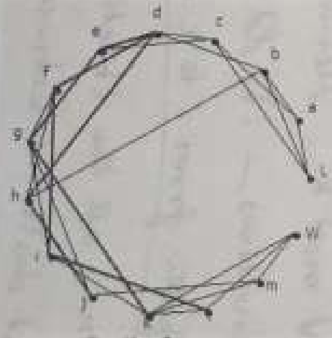
Iteration 3



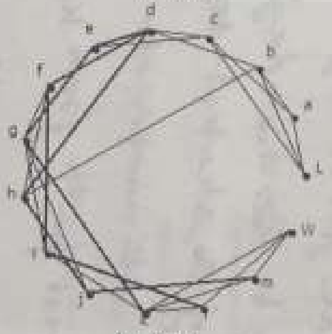
Iteration 4



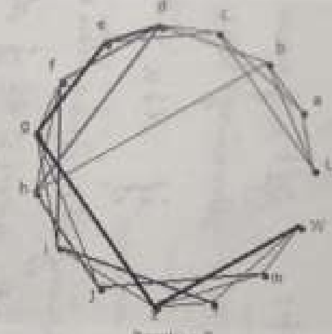
Iteration 5



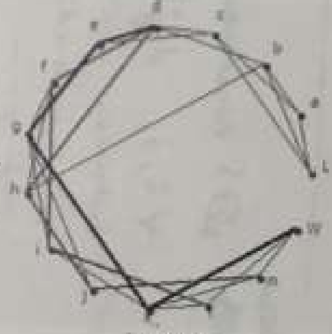
Iteration 6



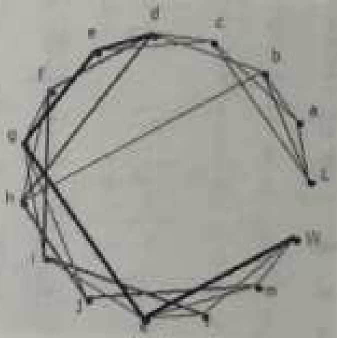
Iteration 7



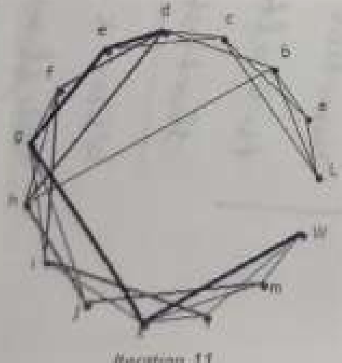
Iteration 8



Iteration 9



Iteration 10



Iteration 11

In order to include the vertex d,
the shortest path from L to W
can be written as:

$$SP_{L-d-W} = SP_{L-d} + SP_{d-W}$$

which is as shown:

L-b-ⓓ-e-g-K-W

SP_{L-d} ↑ SP_{d-W}

(~~merge~~ (merge two shortest paths)

Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	?	?	?	?	0	1	1	?	1	?	?	?	?	?	?	L
1	?	?	?	?	0	1	1	?	1	?	?	?	?	?	?	d
2	?	?	?	?	0	1	1	2	1	?	?	?	?	?	?	e
3	?	?	?	?	0	1	1	2	1	2	2	?	?	?	?	f
4	?	?	?	?	0	1	1	2	1	2	2	?	?	?	?	h
5	?	?	?	?	0	1	1	2	1	2	2	?	?	?	?	g
6	?	?	?	?	0	1	1	2	1	2	2	3	?	?	?	i
7	?	?	?	?	0	1	1	2	1	2	2	3	3	?	?	j
8	?	?	?	?	0	1	1	2	1	2	2	3	3	3	?	k
9	?	?	?	?	0	1	1	2	1	2	2	3	3	3	4	l
10	?	?	?	?	0	1	1	2	1	2	2	3	3	3	4	m
11	?	?	?	?	0	1	1	2	1	2	2	3	3	3	4	W

Distance from L

Shortest path from d to W

4.1.

8. **Proof** (Floyd's algorithm finds the shortest path between all pairs)

By induction on K (# of iterations)

After Kth iteration, for any two vertices i, j, d_{ij} ~~contains~~ represents the shortest distance between i, j using vertex with number at most K as an intermediate vertex

Base case: K=0, when the iterations have not started yet,

$$d_{ij} = \begin{cases} W(i,j), & \text{if } (i,j) \in E[G] \\ \infty, & \text{ow} \end{cases}$$ containing the length of the shortest path between vertices i and j using no (K=0) intermediate vertices. (i.e., ~~vertex with n~~ intermediate vertex number at most 0)

vertices are numbered from 1 to n
$1 \leq K \leq n$

Induction ~~Step~~ Hypothesis: Let's assume $\forall$ K ≤ m-1, m ∈ ℕ, $\forall$ i, j, d_{ij} represents the shortest distances between i, j using vertex with number at most K as intermediate vertex.

Induction Step: Let's prove for the case K = m.
During the K-th iteration, the following is done:
if $d_{iK} + d_{Kj} < d_{ij}$ then $d_{ij} \leftarrow d_{iK} + K_j$.

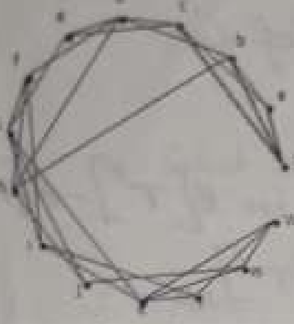
✓ 5/5

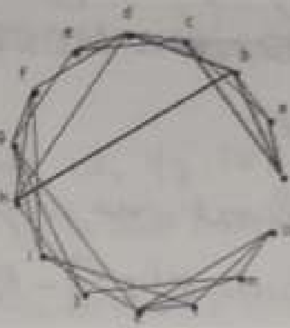
The loop invariant holds in each loop

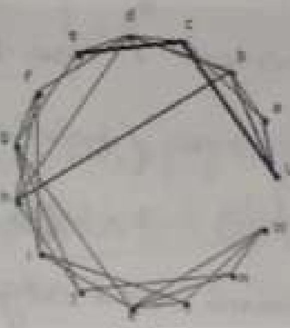
Hence, during the iteration K, d_{ij} will either not change (if the shortest path from i to j does not contain the vertex K), or it will change to $d_{iK} + d_{Kj}$ if it improves the distance. d_{iK} is the length of the shortest ~~distance~~ path from i to K that uses vertices with number at most K and d_{Kj} is the length of the shortest path from K to j that uses vertices with the number at most K. Hence d_{ij} becomes equal to one of the two alternatives — its old value (if K does not help) or the shortest path from i to K and then from K to j (if K helps)

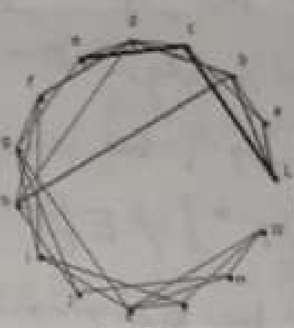
When K = n, all the iterations are finished, d_{ij} contains the shortest distance between i, j using vertex with number at most n as intermediate vertex (i.e. all vertices from 1 to n) and hence the shortest path indeed

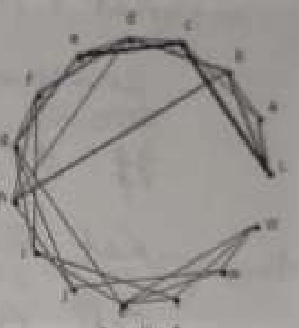
(c) Shortest path from L to W including both vertices e and m
= " " " L to e + Shortest path from e to m + edge (m, W)
as shown below: L-a-e — g — j - m — W


Iteration 1


Iteration 2


Iteration 3

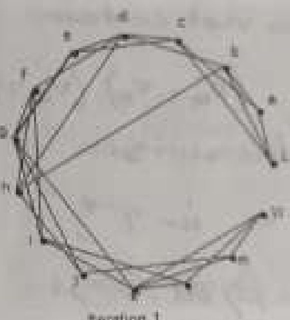

Iteration 4

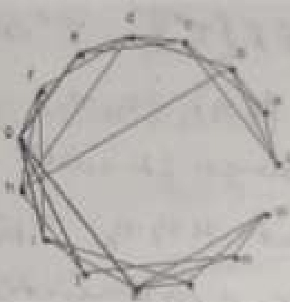

Iteration 5

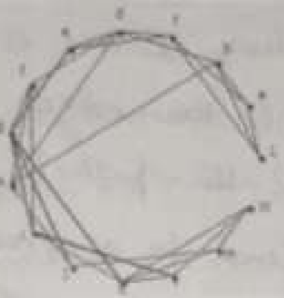
Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	L
1	0	1	1	1	?	?	?	?	?	?	?	?	?	?	?	a
2	0	1	1	1	2	?	?	?	2	?	?	?	?	?	?	b
3	0	1	1	1	2	2	?	?	2	?	?	?	?	?	?	c
4	0	1	1	1	2	2	2	?	2	?	?	?	?	?	?	d
5	0	1	1	1	2	2	2	2	2	?	?	?	?	?	?	e

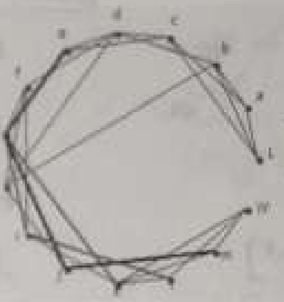
Distance from L

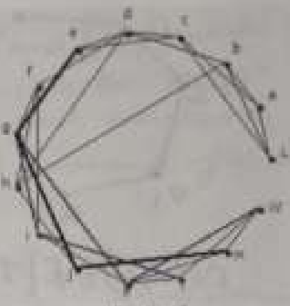
Shortest path from L to e

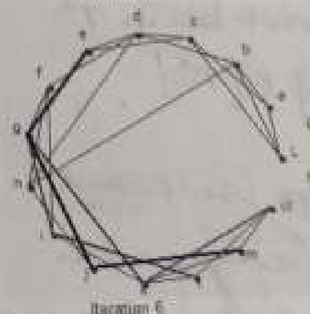

Iteration 1

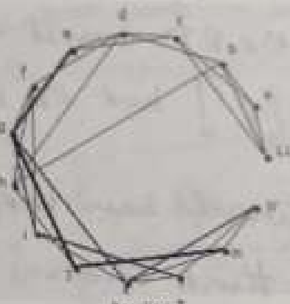

Iteration 2


Iteration 3


Iteration 4


Iteration 5


Iteration 6


Iteration 7

Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	?	?	?	?	?	0	?	1	?	?	?	?	?	?	?	L
1	?	?	?	?	?	0	?	1	?	?	?	?	?	?	?	e
2	?	?	?	?	?	0	?	1	?	2	2	2	?	?	?	g
3	?	?	?	?	?	0	?	1	?	2	2	2	3	?	?	i
4	?	?	?	?	?	0	?	1	?	2	2	2	3	3	?	j
5	?	?	?	?	?	0	?	1	?	2	2	2	3	3	3	k
6	?	?	?	?	?	0	?	1	?	2	2	2	3	3	3	l
7	?	?	?	?	?	0	?	1	?	2	2	2	3	3	3	m

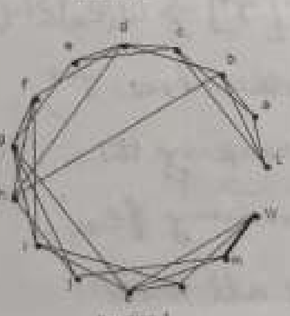
Distance from L

Shortest path from e to m

Iteration	L	a	b	c	d	e	f	g	h	i	j	k	l	m	W	Vertex picked
0	?	?	?	?	?	?	?	?	?	?	?	?	?	0	1	m
1	?	?	?	?	?	?	?	?	?	?	?	?	?	0	1	W

Distance from L

Shortest path from m to W


Iteration 1

4.2.
12. **Proof:** (Kruskal's algorithm gives an MST)

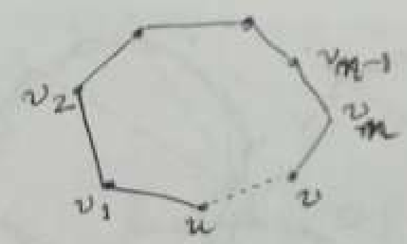
Let T^* be the spanning tree for the graph G generated by Kruskal's algorithm and T' be ~~the~~ a MCST of G.

If $T^* = T'$, we are done (trivial).

Let $E[T^*] \neq E[T'] \Rightarrow \exists$ at least one edge in $E[T^*] - E[$

Let e be the minimum cost edge in $E[T^*] - E[T']$, i.e., $e \in E[T^*]$ but $e \notin E[T']$, $e = (u, v)$.

Since T' is connected $\exists$ path P in between u, v in T'
$\overset{\text{|||}}{}$
Note $u\, v_1 v_2 \cdots v_m v$.



$e = (u, v) \in E[T^*]$
$\notin E[T']$.

Now, since $e \in E[T^*]$, T^* can not contain all the edges $(u, v_1), (v_1, v_2), \ldots, (v_{m-1}, v_m), (v_m, v)$ on the path P, since it will otherwise lead to a circuit $u v_1 v_2 \cdots v_m v u$ in T^*, which is acyclic, a contradiction $\Rightarrow$ at least one edge $e_K \in P$ must not be in T^* i.e., $\exists\, e_K \in E[T']$ but $e_K \notin E[T^*]$.

if $w(e_K) < w(e)$, then e_K would have been selected by the Kruskal's algorithm that chooses edges with minimum weights greedily. But $e_K \notin E[T^*] \Rightarrow w(e_K) \geq w(e)$

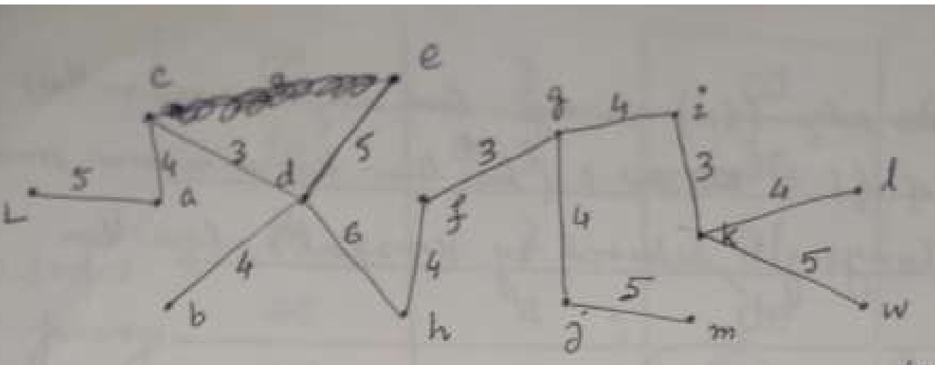
But then we can always construct another tree T'', where $T'' = E[T'] \cup \{e\} - \{e_K\}$, by breaking the circuit created by $E[T'] \cup \{e\}$ by removing e_K.

T'' is connected and acyclic and spans all the edges but cost of $T'' \leq$ cost T' since $w(e) \leq w(e_K)$

Repeat this process to correct all edges in $T^* - T'$ to eventually convert T' into T^* without increasing cost. Hence T^* is an MCST (proved)

2

(a)



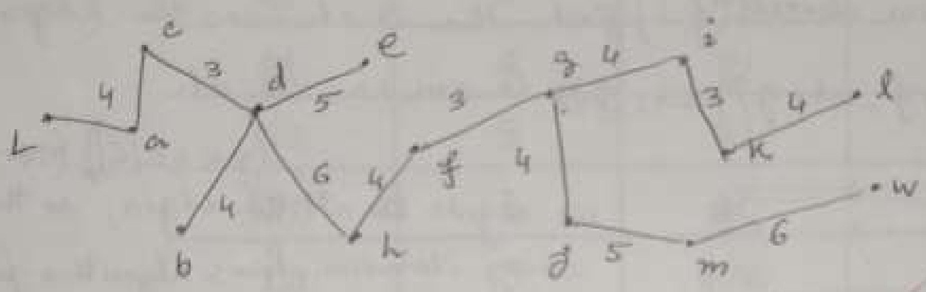
10/10

Select edges cd, fg, ik, ac, bd, hf, gi, gj, Kl, $\overset{de,}{\underset{h}{|}}$La, jm, kw, dh

in that order by Kruskal's algorithm

Hence, Min-cost = 5+4+3+4+5+6+4+3+4+5+4 $\overset{+3+}{\underset{h}{+}}$4 +5 = 59 units

(b)



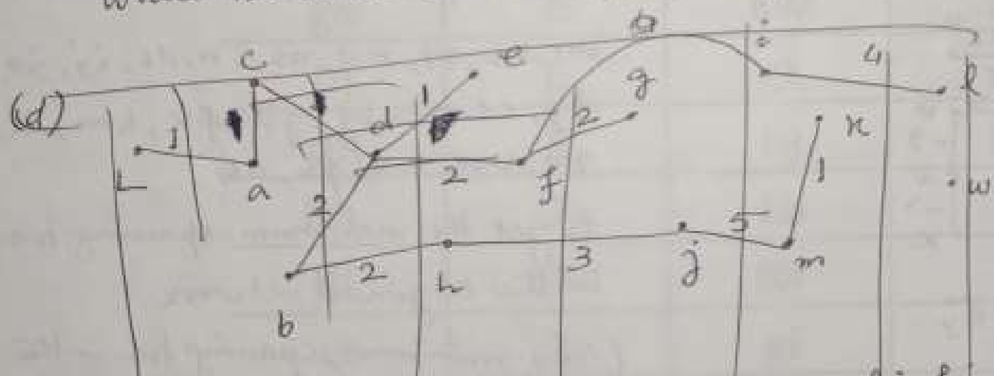
(c, e), (d, f), (k, w) are ruled out.

(c, e) and (d, f) were not inside the MCST constructed. We choose (m, w) instead of (k, w)

Hence, Min-cost = 59 − c(k, w) + c(m, w)
= 59 − 5 + 6 = 60 units.

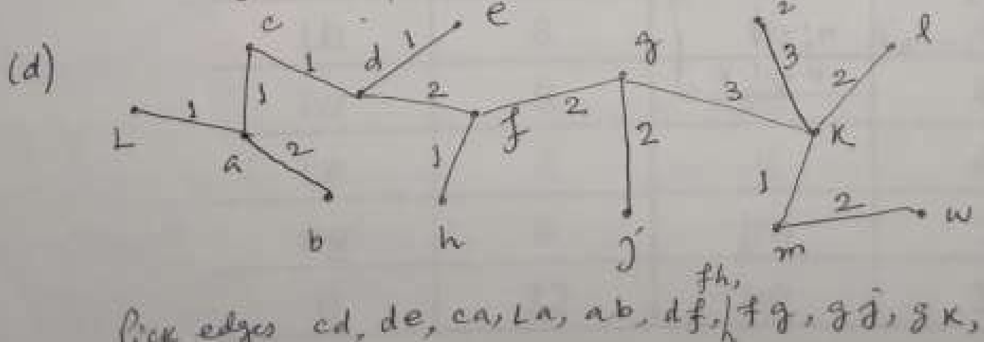
(c) Such two non-adjacent vertices are c, f.
The cheapest path in between them is c-d-f costing 3+7=10 units
while the Kruskal's MCST picks c-d-h-f with cost 3+6+4=13 units.

(d)



~~Pick edges cd, ac, de, La, bd, bh, df, hj, fi, il, km, fg~~

(d)

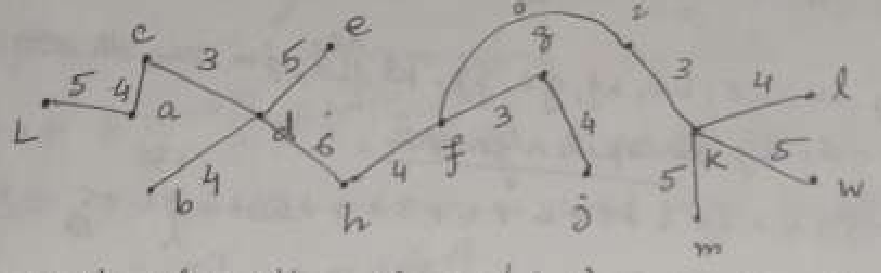


MCST cost (minimizing # trees cut down)
= 1+1+2+1+1+2+1+2+
2+3+1+3+2+2=24

According to Prim's algo

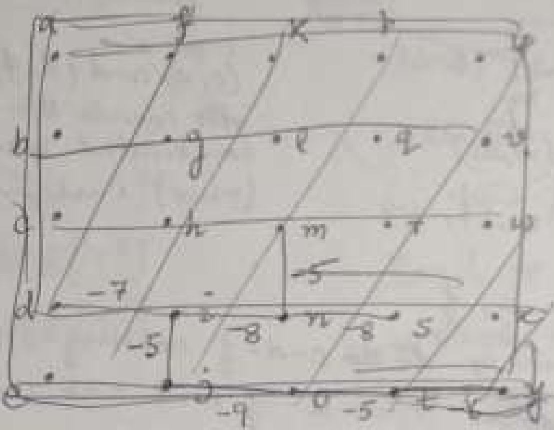
Pick edges cd, de, ca, La, ab, df, $\overset{fh,}{\underset{h}{}}$fg, gj, gK, Km, Kl, ik, mw edges.

4. We need to choose the edge (f,i) always. One way to ensure this is to change the cost of fi to 0 (less $\overset{\text{OR}}{}$ than the minimum available cost) so that it always gets selected by Kruskal's algorithm.



Kruskal's algorithm chooses (f,i) first, then it chooses the edges fg, iK, ac, hf, bd, gj, Kl, de, Km, La, Kw, dh

5.



$\curvearrowright (\forall e \in E[G]), (e \leftarrow -e)$

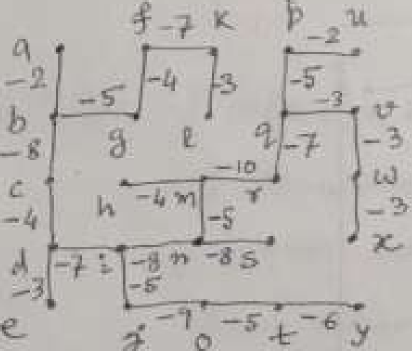
We negate all the edges, so that at every iteration. Prim's algorithm picks up an edge which is originally maximum (since $\min_i\{n_i\} = \max_i\{-n_i\}$, $n_i \in N$) edges will be picked by Prim's algorithm in the following order:

~~ie, ot, ty, ij, in, ns, di, mn,~~

mr, rq, qp, mn, ns, in, di, ij, jo, ot, ty, cd, bc, bg, gf, fK, hm, Kl, de, qv, vw, wx, pu, ab.

to get the maximum spaning tree in the original network

(i.e, minimum spaning tree in the converted network)



$$\boxed{\begin{aligned} &\max\{n_1, n_2, \ldots, n_k\} \\ &= \min\{-n_1, -n_2, \ldots, -n_k\}, \quad n_i \in N \\ &\qquad\qquad\qquad\qquad \forall i = 1..k \end{aligned}}$$

Edges sorted
w.r.t. freeways

(Used in
Prim's &
Kruspal's
algorithm)

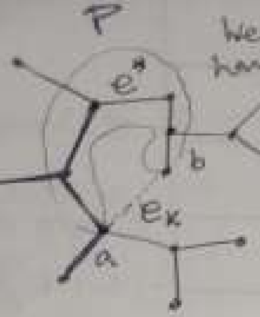
cd	3
fg	3
ik	3
ac	4
hf	4
bd	4
gj	4
kl	4
gi	4
de	5
km	5
La	5
ab	5
kw	5
jm	5
mw	6
gk	6
hj	6
dh	6
bh	7
df	7
Lc	7
fi	8
Lb	8
lw	8
ce	8
eg	8
il	12

cd	1
ac	1
hf	1
de	1
km	1
La	1
fg	2
bd	2
gj	2
kl	2
ab	2
mw	2
bh	2
df	2
Lc	2
ik	3
kw	3
gk	3
hj	3
Lb	3
lw	3
gi	4
dh	4
fi	4
ce	4
il	4
jm	5
eg	5

Edges sorted
w.r.t. #trees
cut

(used in
Prim's &
Kruspal's
algorithm)

10.



We have, $e_K \in \underset{\substack{\text{Prim}\\ \text{Chosen}\\ \text{Tree}}}{T^*} - \underset{\text{MST}}{T'}$

$e^* \in \underset{\text{MST}}{T'} - T^*$

<u>To prove:</u> $(T' - \{e^*\}) \cup \{e_K\}$ is still connected and circuit-free

$e^* \neq e_K$

<u>Proof:</u> Consider $T' - \{e^*\}$. Since T' is a tree and hence it has a unique path P from a to b and it's minimally connected, removal of $e^* \in P$ from T' will create exactly two connected components C_1 & C_2, s.t. $a \in C_1$ and $b \in C_2$. All vertices inside C_1 are reachable from one other, since if not, let's assume to the contrary $u, v \in C_1$ and u is not reachable from v $\Rightarrow$ even after adding e^* back to $T' - \{e^*\}$, u will NOT be reachable from v, (since at least 2 different edges in between C_1 & C_2 are needed to complete a path between u and v, but there is one edge, namely e^*) leading to a contradiction, since T' is connected. Similarly C_2 is also a connected component but ∄ a path between $u \in C_1$ and $v \in C_2$. Now, adding $e_K = (a, b)$ to $T' - \{e^*\} = C_1 \cup C_2$, adds an edge in between ~~this~~ these components, now any $u \in C_1$ ~~and~~ will be connected to any $v \in C_2$ via path $\underbrace{u \overset{P_1}{\rightsquigarrow} a}_{C_1 \text{ is connected}} \xrightarrow{e_K} \underbrace{b \overset{P_2}{\rightsquigarrow} v}_{C_2 \text{ is connected}}$ $\Rightarrow \forall u, v \in C_1 \cup C_2 \cup \{e_K\}$, ∃ a path $P = P_1 \cup \{e_K\} \cup P_2$ in between them

$\Rightarrow (T^* - \{e^*\}) \cup \{e_K\} = C_1 \cup C_2 \cup \{e_K\}$ is <u>connected</u>.

Also, since C_1, C_2 are subgraphs of the tree T', C_1, C_2 must be <u>circuit free</u> (ow T' is not circuit-free, a contradiction). Also, adding $e_K = (a, b)$ to $C_1 \cup C_2$ creates exactly one unique path from any $u \in C_1$ and $v \in C_2$, since these components were disconnected earlier. Hence adding e_K can't create a circuit (ow, ~~added~~ removing e_K would not have created components C_1, C_2).

$\Leftarrow$ $C_1 \cup C_2 \cup \{e_K\}$ is circuit free

Do the following steps before running the Kruskal's algorithm to find the minimum spanning tree on graph $G(V, E)$

Let the selected (prescribed) edge be $e = (u, v) \in E[G]$, with $u, v \in V[G]$.

① Find the ~~m~~ edge e_{min} with minimum cost in the graph G, by scanning through all the edges in the set $E[G]$.

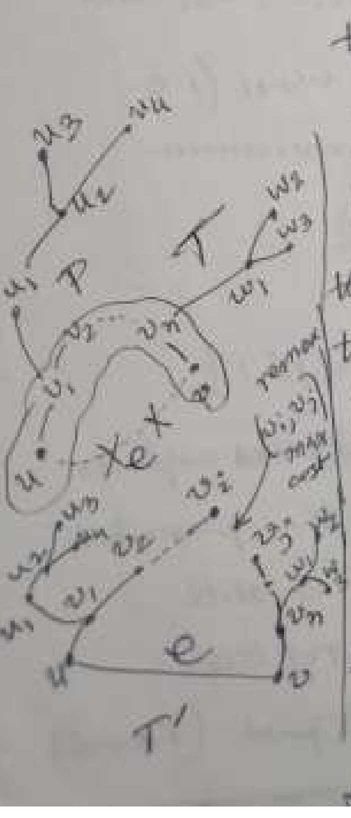
$$w_{min} = w(e_{min}) = \min\{ \underset{\substack{\uparrow \\ \text{weight/or cost}}}{w(u,v)} \mid (u,v) \in E[G] \}$$

② If $e_{min} = e$ do nothing (if it's the unique minimum)

else ~~do~~ Change the weight (cost) of the ~~selected~~ prescribed edge e ~~to~~ to an weight (strictly) less than e_{min}.
e.g. $w(e) \leftarrow w(e_{min}) - 1$, i.e., $w(e) \leftarrow w_{min} - 1$ and obtain the graph $G'(V, E)$ modified

③ Now run Kruskal's algorithm of G'

Proof If $e_{min} = e$ (i.e., the prescribed edge is the (unique) minimum cost edge ~~[illegible] if the Kruskal~~ in the graph G, in which case Kruskal always picks up that edge first) or if ~~the [illegible]~~ Kruskal's algorithm already has picked up the edge e in the MST on the original graph, there is nothing to prove. (trivial)



If running Kruskal's algorithm on the original graph G the prescribed edge e was NOT selected in the MST, that's the only non-trivial case to prove.

Let $e = (u, v)$ and Kruskal's algorithm on the modified graph has chosen e in the new MST T', while the old graph resulted in the tree T upon running Kruskal's on it and $e \notin T$ but $e' \in T$!

Since T is a spanning tree (containing all vertices) $u, v \in T$ and $\exists$ path between u and v in T (since connected). Let this path be $P \equiv u - v_1 - v_2 - \cdots - v_k - v$ and

$u \overset{P}{\rightsquigarrow} v$ T

e not present in T

notices there exist <u>exactly one</u> path between u & v in T. (~~[illegible]~~ since acycli

All the other vertices outside this path in T must be unaffected by the modification, i.e., the edges selected to cover V−P in T must be exactly identical to those in T′ (because of minimality) ⇒

<u>Only edges</u> that can be <u>different</u> in <u>T and T′</u> will be the edges <u>belonging to the path P.</u> (note ~~[illegible]~~ $(u,v) \notin P$)

Let's order the edges in P in ascending order of the weights (or costs):

~~$w(u,v_1) \le w(u,v_2) \le \cdots \le w(u,v_n)$~~ $w_1 \le w_2 \le \cdots \le w_{n+1}$ where w_i's are nothing but a permutation of $w(u,v_1), w(v_1,v_2), \ldots, w(v_n,v)$, the ~~weights wij~~ weights (costs) of edges on P.

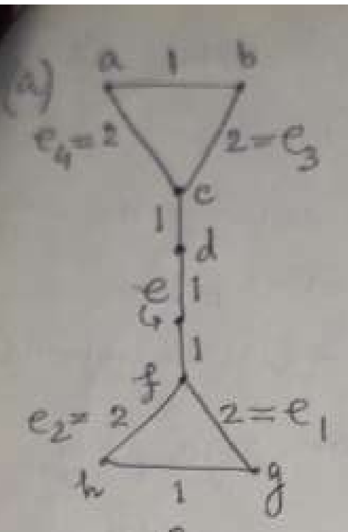
Now in the modified graph, the algorithm must choose ~~[illegible]~~ the edge (u,v), since by construction, $w(u,v) < w_1 \le w_2 \le \cdots \le w_{n+1}$. At the same time the algorithm can not choose all the $n+2$ edges ~~for n+2 vertex~~ to span $n+2$ vertices $u, v_1, v_2, \ldots, v_n, v$ (otherwise it will be a circuit). Hence in the modified graph the algorithm must drop an edge. Since the algorithm greedily chooses the edges, it will choose all the edges corresponding to the weights $w(u,v), w_1, w_2, \ldots, w_n$ and ~~drop~~ will NOT choose the edge corresponding to w_{n+1} (i.e., the edge ~~with m~~ belonging to path $u \overset{P}{\leadsto} v$ with maximum cost) and ~~T′ = T − {(u,v)} ∪ [illegible] (v_1,v_2)~~

~~T′ = T ∪~~ $T' = T \cup \{(u,v)\} - \{(v_1,v_2)\}$,

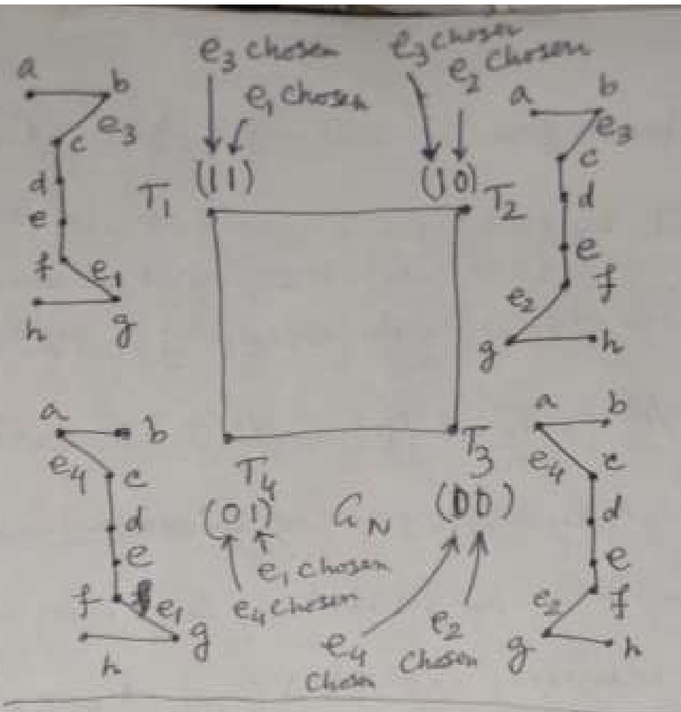
where $(v_1,v_2) = \underset{i,j}{\arg\max} \{(v_i,v_j) \mid (v_i,v_j) \in P\}$.

~~By minim~~ Since T is a MCST and T′ only ~~[illegible]~~ replaces ~~the total cost of~~ the maximum cost edge on P with the prescribed edge (u,v) keeping all other edges unaltered, T′ is still MCST of the modified tree and hence hence MCST of the original tree forcing the prescribed edge to be its part. (Proved)

(a)



G



$T_1 = T_2 - ~~\beta~~ e_2 + e_1$

$T_2 = T_3 - e_4 + e_3$

$T_3 = T_4 - e_1 + e_2$

$T_4 = T_1 - e_3 + e_4$

$T_1 = T_3 - e_2 - e_4 + e_1 + e_3$

$T_2 = T_4 - e_1 - e_4 + e_2 + e_3$

T_i can be coded as $b_1 \mid b_0$

b_1: 1 if e_3 chosen, 0 " e_4 "

b_0: 1 if e_1 chosen, 0 " e_2 "

exactly ~~at least~~ one of (e_3, e_4)
and " " " " (e_1, e_2)
must be chosen by any MST

(b) $|T_1 \cap T_2| = n-k-1$, if they differ by k edges.

(e.g. if $k=0$, $|T_1 \cap T_2| = n-1$, $T_1 = T_2$ with $|T_1| = |T_2| = n-1$ ~~etc~~
$k=1$, $|T_1 \cap T_2| = n-2$, T_1, T_2 differ by 1 edge. etc)

Proof by induction on k

Base case: $k=1$ (assuming $T_1 \neq T_2$).

T_1 & T_2 differ by exactly one edge, hence by definition of G_N, T_1 ~~T_2 will be~~ $v_1 \equiv T_1 \in V(G_N)$, $v_2 \equiv T_2 \in V(G_N)$, will be adjacent, i.e., $(v_1, v_2) \in E[G_N]$

$\Rightarrow \exists$ path of length $k=1$ in between v_1 and v_2.

Induction Hypotheses: Let's assume $\forall k \leq m$, if T_1 & T_2 differ by exactly k edges, $\exists$ path of length ~~m~~ k in between them. $(m \in \mathbb{N})$

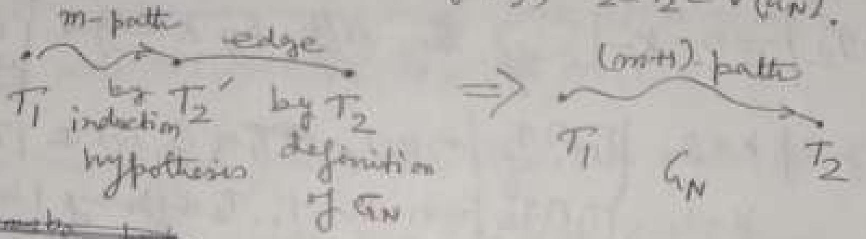
Induction Step: Let's prove for $k = m+1$.

T_1 & T_2 differs in $m+1$ edges, i.e., $\exists\, E_1 = \{e_{11}, e_{12}, \dots, e_{1m+1}\} \subseteq T_1$
and $E_2 = \{e_{21}, e_{22}, \dots, e_{2m+1}\} \subseteq T_2$
s.t. $E_1 \cap E_2 = \emptyset$ and $T_1 - E_1 = T_2 - E_2$.

choose an edge $e_{1s} \in E_1$ and corresponding edge $e_{2s} \in E_2$, where $1 \leq s \leq m+1$.

~~$1 \le s \le m+1$~~. Create tree $T_2' = T_2 - e_{2s} + e_{1s}$

T_2' is an MST by construction, which differs in exactly m edges with T_1. (since T_1 is MST and it ~~also~~ has chosen e_{1s}) By induction hypothesis, $\exists$ path of m edges in between T_1 and T_2'. Also, T_2' differs with T_2 in exactly one edge (by construction) and by ~~induction hypothesis, $\exists$ an edges~~ definition T_2' and T_2 must be adjacent in G_N, i.e., $\exists$ an edge (path of length 1) in between $v_2' \equiv T_2' \in V(G_N)$, $v_2 \equiv T_2 \in V(G_N)$.



$\Rightarrow \exists$ an $(m+1)$ path in G_N in between the vertex $v_1 \equiv T_1 \in V(G_N)$ and $v_2 \equiv T_2 \in V(G_N)$

$\Rightarrow \forall k \in \mathbb{N}$, if T_1, T_2 differ by k ~~vertices~~ edges, $\exists$ a path of length k in between the vertices representing T_1, T_2 in G_N.

(Proved)