* Suppose we have a Host A with IP address as IP_A and subnet mask of subnet where A belongs is S_A.

→ If host A wants to send packet to host B whose IP address is IP_B, then 'A' will do bitwise AND operation with S_A first and then it will do bitwise AND operation of IP_B with S_A in order to know wheather host 'B' belongs to same subnet or not.

→ If IP_A & IP_B belongs to same network, we must have:-

$$IP_A \text{ [AND operation] } S_A == IP_B \text{ [AND operation] } S_A \rightarrow \underline{\text{eq (1)}}$$

→ Ex-1) $IP_A = 200.1.2.134$ (Source)

$IP_B = 200.1.2.155$ (destination)

$S_A = 255.255.255.192$

Check if both 'A' & 'B' belongs to same subnet?

$x_4.x_3.x_2.11000000$

$x_4'.x_3.x_2'.10000110$

$200.1.2.128$

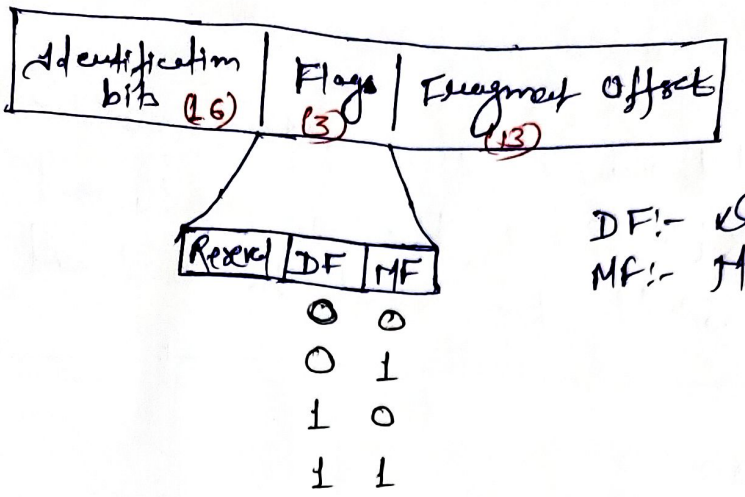
$x_4.x_3.x_2.11000000$

$x_4''.x_3''.x_2''.10011011$

$200.1.2.128$

→ Eq (1) satisfies, therefore both A & B are on same subnet.

Fragmentation



DF:- Don't Fragment (0/1)
MF:- More Fragment (0/1)

* IP layer (Network layer) follows's datagram service (connection-less), which means the datagram may follow any route to reach the destination. Therefore, whenever fragmentation is done, each time header is added to each frame.

Q.) A datagram of length 5000 Bytes with 20 Bytes of header in it reached a router. The router has to forward the datagram on the link where MTU is 700 bytes. How many fragments the router has to do? Determine the total length of each fragmented packet, the M-bit and the fragmentation offset.

Sol:- 5000 (20 + 4980), $\left\lceil \frac{4980}{680} \right\rceil \approx 8$ → ceiling value/upper value

	FP1 (680+20)	FP2 (680+20)	FP3 (680+20)	FP4 (680+20)	FP5 (680+20)	FP6 (680+20)	FP7 (680+20)	FP8 (220+20)
M-bit	1	1	1	1	1	1	1	0
Fragmentation offset	0	85	170	255	340	425	510	595

No. of hosts = 500

$n \geq 9$ ($2^n = 500$)

Network Mask = 255.255.254.0
(Address Mask)

171.43.16.37/23

171.43. 0 0 0 1 0 0 0 0 . 0 0 1 0 0 1 0 1

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 1 ⇒ 171.43.16.0 → network id

1 1 1 1 1 1 1 1 ⇒ 171.43.17.255 → Directed Broadcast addr.

⇒ 2 ip's reserved

⇒ 512-2 ⇒ 510 ip's available, 500 used, 10 remaining ips.

1.2.3.4/23

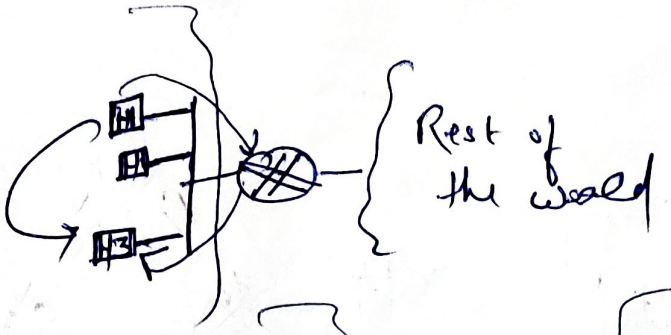
1.2. 0 0 0 0 0 0 1 1 . 0 0 0 0 0 1 0 0

0 . 0 0 0 0 0 0 0 0

⇒ 1.2.2.0

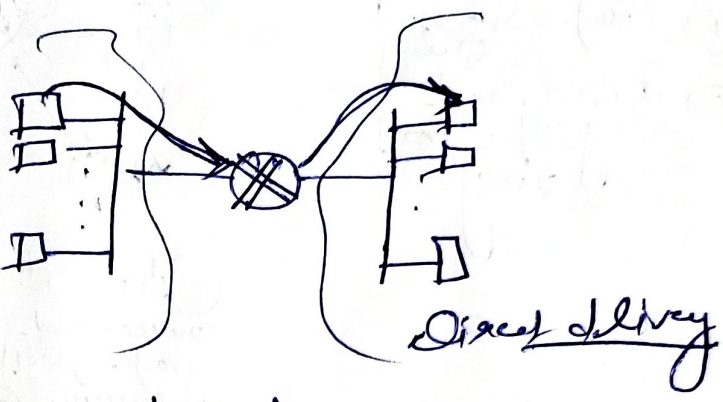
1 . 1 1 1 1 1 1 1 1 ⇒ 1.2.3.255

Delivery



(a) Direct Delivery

(b) Indirect Delivery

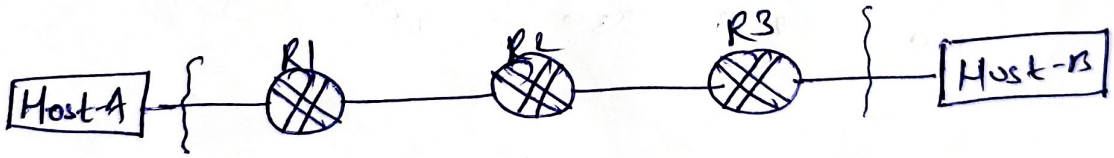


Forwarding: — Transferring a pkt from an i/p link interface to the appropriate o/p link interface.

Routing: — It is network-wide process which determines the end-to-end paths that pkt has to take from source to destination.

→ **Forwarding Techniques**

(a) Next-Hop Method verses Route Method



R-1 →

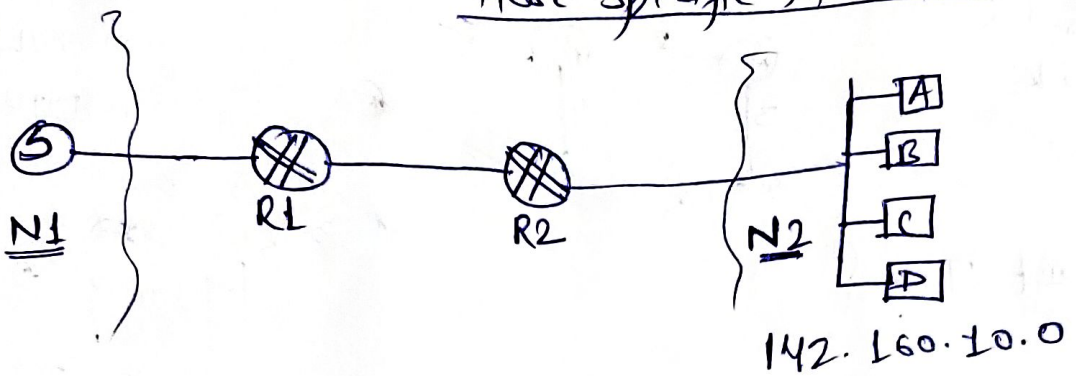
Destination	Route
Host-B	$R_2 \rightarrow R_3 \rightarrow$ Host B

Route Method

Destination	Next-Hop
Host-B	R_2

Next-Hop method

⑤ Network-Specific Method Versus Host-Specific Method



R1

Destination	Next-hop
A	R2
B	R2
C	R2
D	R2

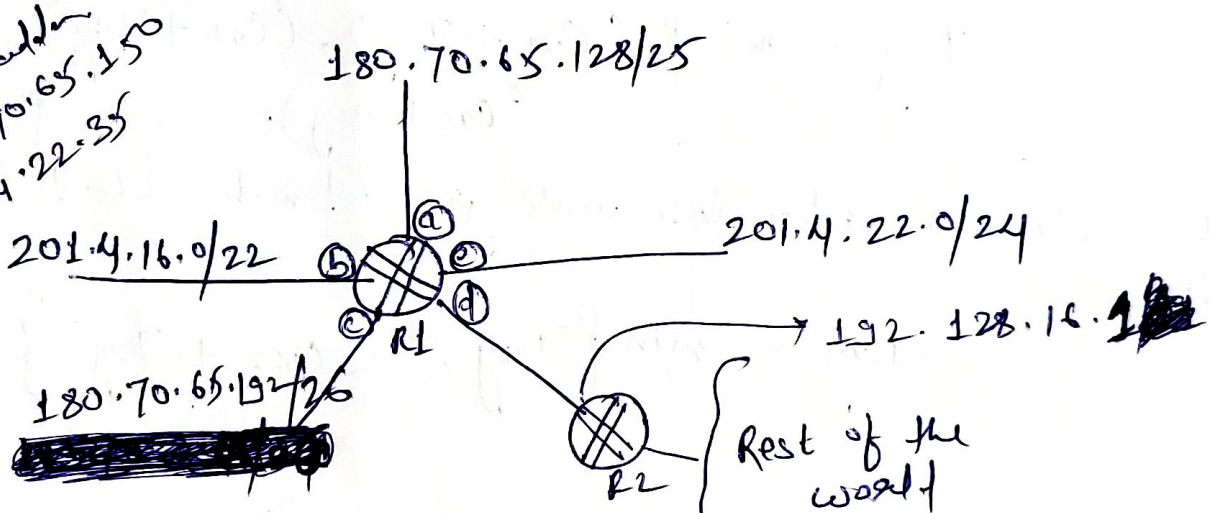
Host-specific Method

R1

Destination	Next-Hop
142.160.10.0	R2

Network-specific Method

chart with destination address
① 180.70.65.150
② 201.4.22.35

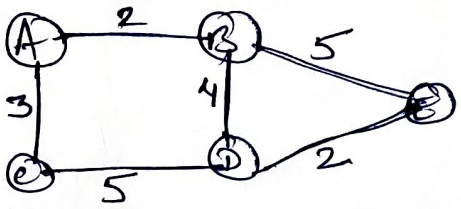


Mask	Network Address (Network Id)	Next-hop	Interface
/26	180.70.65.192	–	c
/25	180.70.65.128	–	a
/24	201.4.22.0	–	e
/22	201.4.16.0	–	b
[redacted]	[redacted]	R2 (192.128.16.1)	d
0.0.0.0 (Default)	0.0.0.0		

Longest Mask Matching

* Internet as a Graph

Node ⇒ Router
Edge ⇒ Links



3000 Bytes
555 MTU

528 2980 + 20
553 + 20

2980 / 532 ⇒ 6

532 100 (bytes)

→ Least-Cost Tree?

* Distance-Vector Routing

① Bellman-Ford Equation :- It is used to find the least cost between a source node, x, and a destination node, y, through some intermediary nodes (a, b, c ---), where,

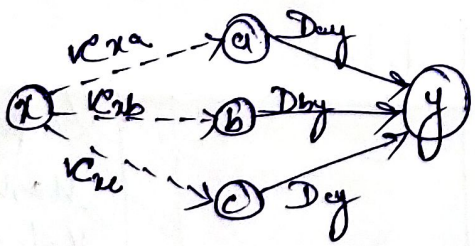
i) Costs b/w the source and the intermediary nodes are given.

ii) Costs b/w the intermediary nodes and the destination are given.

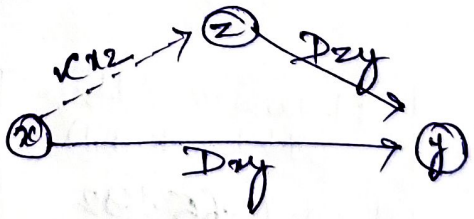
$$D_{xy} = \min\{(C_{xa} + D_{ay}), (C_{xb} + D_{by}), (C_{xc} + D_{cy}), \dots\} \rightarrow (1)$$

* After a while, an intermediate node gives information about D_{xy}. Then,

$$D_{xy} = \min\{D_{xy}, (C_{xz} + D_{zy})\} \rightarrow (2)$$



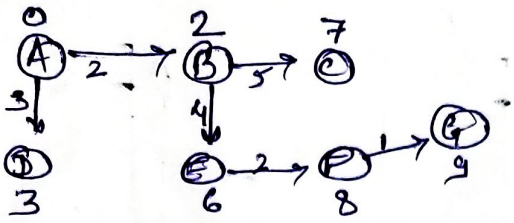
(1)

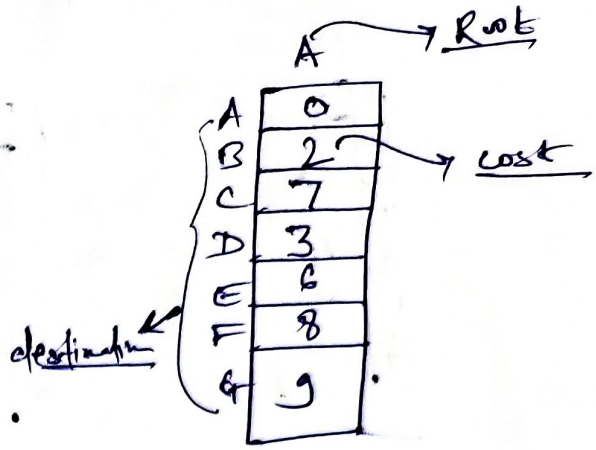


(2)

$$\boxed{d_x(y) = \min_v\{c(x,v) + d_v(y)\}}$$

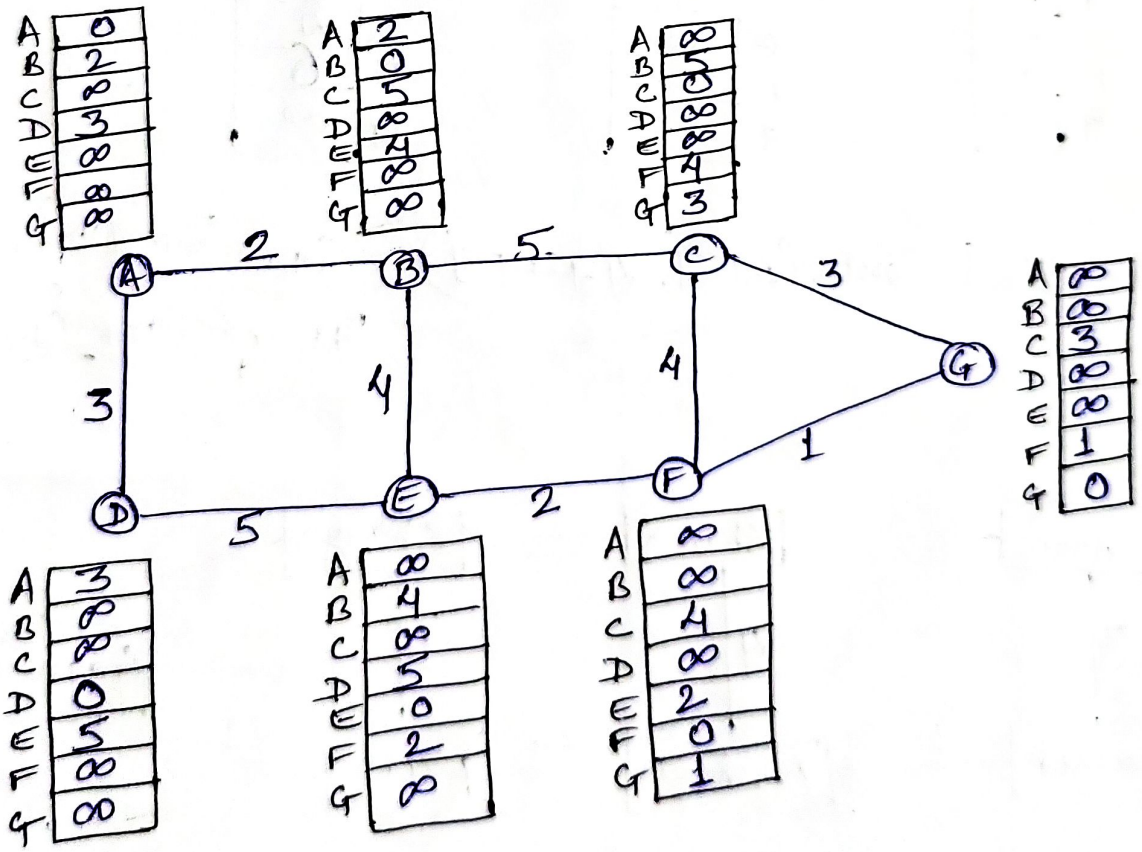
② Distance-Vector :- A one-dimensional array to represent the tree.





2.1) Distance vector doesn't give the path to the destinations, it only gives the least costs to the ~~destinations~~ other nodes in n/w.

2.2) Node sends some greeting messages out of its interfaces and discovers the identity of the immediate neighbors and distance b/w itself and each neighbor.



① First Event: B receives a copy of A's vector

	OldB
A	2
B	0
C	5
D	∞
E	4
F	∞
G	∞

	A
A	0
B	2
C	∞
D	3
E	∞
F	∞
G	∞

⇒

	NewB
A	2
B	0
C	5
D	5
E	4
F	∞
G	∞

~~B[ ] = min (B[ ], 2 + A[ ])~~

② Second Event: B receives a copy of E's vector

	OldB
A	2
B	0
C	5
D	5
E	4
F	∞
G	∞

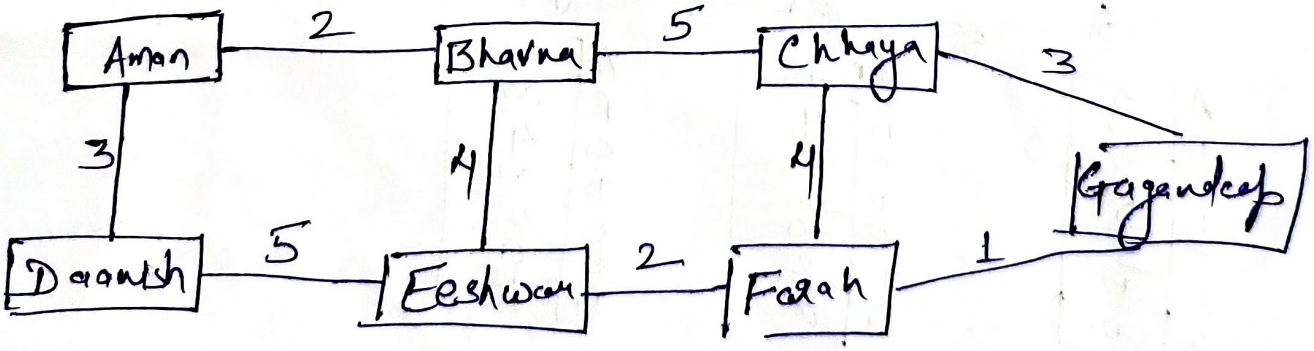
	E
A	∞
B	4
C	∞
D	5
E	0
F	2
G	∞

⇒

	New-B
A	2
B	0
C	5
D	5
E	4
F	6
G	∞

~~B[ ] = min (B[ ], 4 + E[ ])~~ ⇒ ~~Dba = min (Dba, (bet Dea)~~

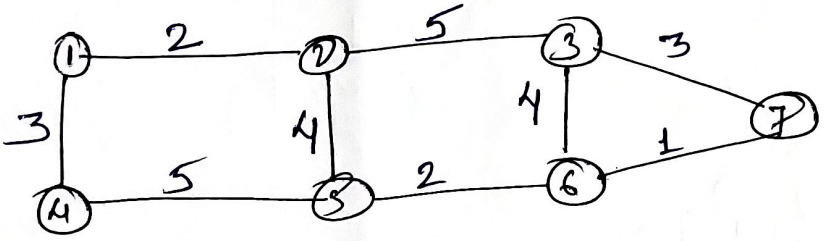
~~Dbb = min (Dbb, (bet Deb)~~



⇒ ① Distributed :- Receive information from the neighbours, performs a calculation, distribute the results of its calculation back to the neighbors.

② Iterative :- Process continues on until no more information is exchanged b/w neighbors.
(It is self terminating)

③ Asynchronous :- It doesn't require all of the nodes to operate in lockstep with eachother.

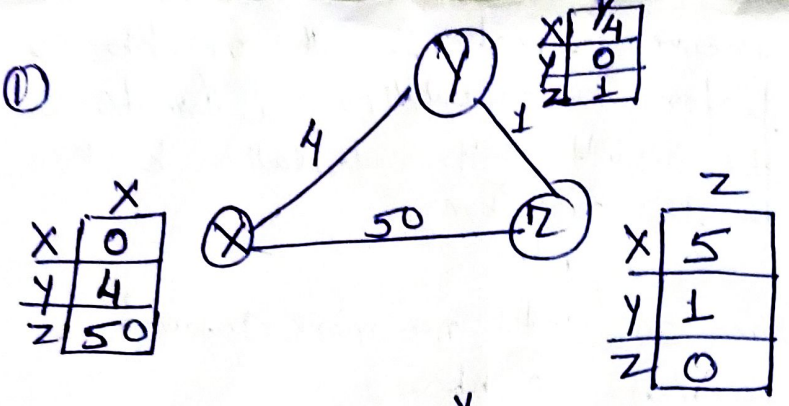


* Distributed Bellford-Ford Algorithm
OR
* Distance-Vector Routing

N1	N2	N3	N4	N5	N6
∞ (-1)	∞ (-1)	3 (7)	∞ (-1)	∞ (-1)	1 (7)
∞ (-1)	8 (3)	3 (7)	∞ (-1)	3 (6)	1 (7)
10 (2)	7 (5)	3 (7)	8 (5)	3 (6)	1 (7)
9 (2)	7 (5)	3 (7)	8 (5)	3 (6)	1 (7)
9 (2)	7 (5)	3 (7)	8 (5)	3 (6)	1 (7)

* All sources, single destination

①

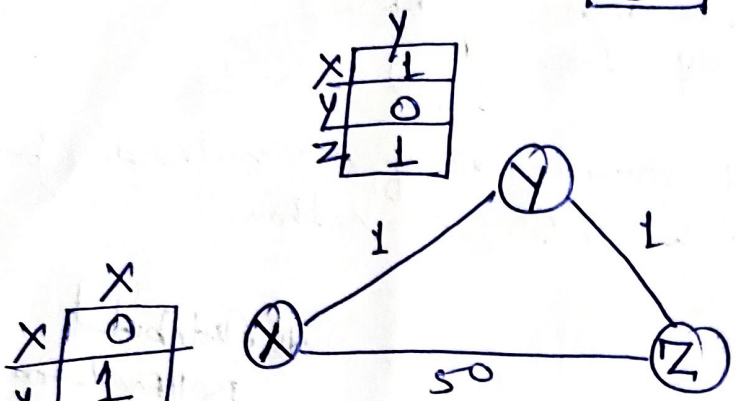


X	
X	0
Y	4
Z	50

Y	
X	4
Y	0
Z	1

Z	
X	5
Y	1
Z	0

②



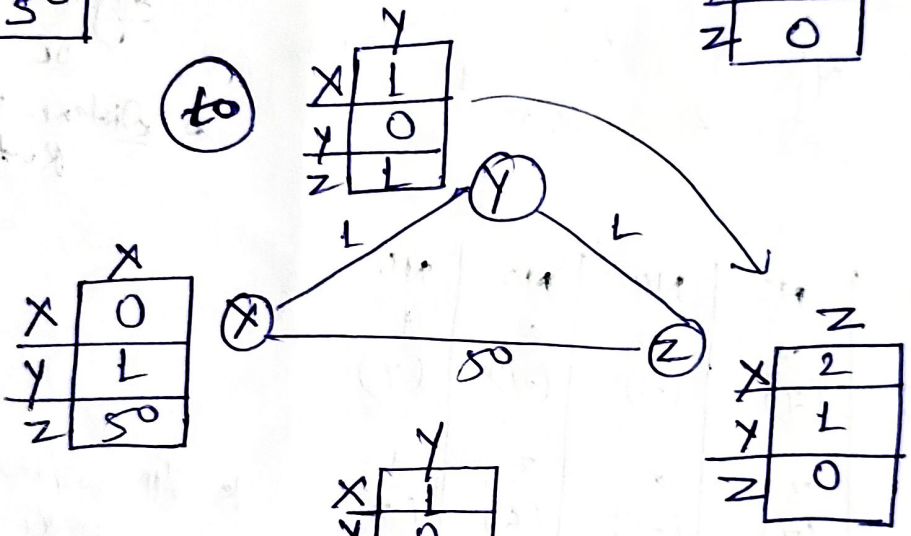
to ⇒ 'Y' detects link cost change, updates its DV & send to its neighbours.

X	
X	0
Y	1
Z	50

Y	
X	1
Y	0
Z	1

Z	
X	5
Y	1
Z	0

③



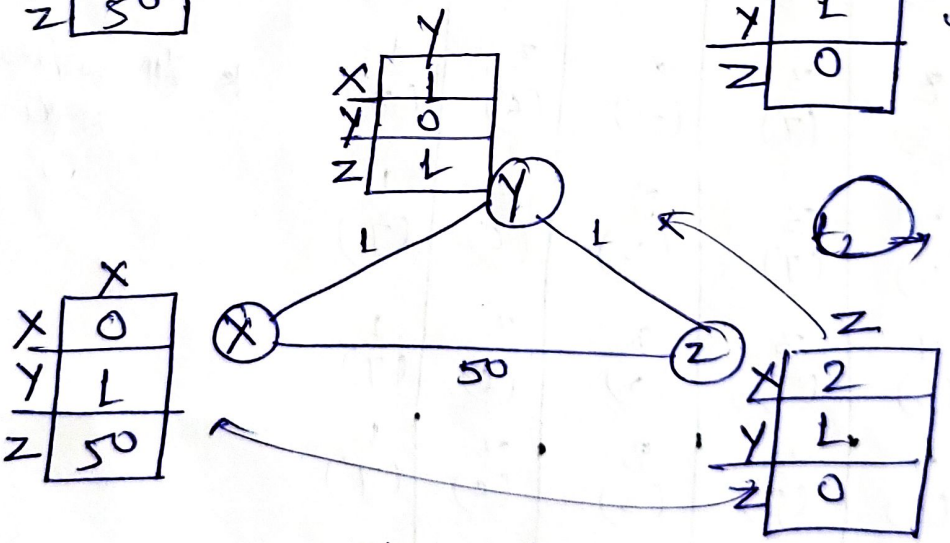
X	
X	0
Y	1
Z	50

Y	
X	1
Y	0
Z	1

Z	
X	2
Y	1
Z	0

t1 → 'Z' updates its DV & sends to its neighbours.

④



X	
X	0
Y	1
Z	50

Y	
X	1
Y	0
Z	1

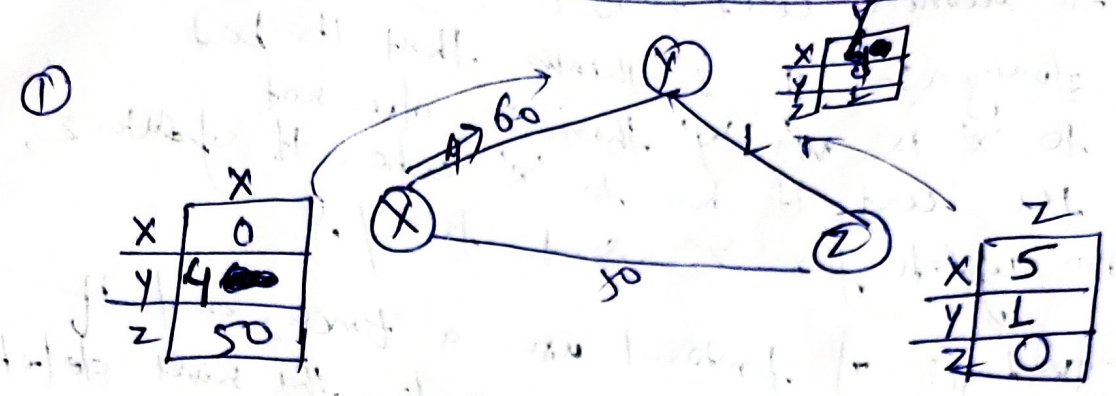
Z	
X	2
Y	1
Z	0

t2 → 'Y' receives 'Z' updates 'Y' least cost do not change

⇒ No change
Algorithm reaches a constant state!

* Good News Travel Faster!

Bad news travels slowly

① 

'to' $D_y(x) = \min\{ c(y,x) + D_x(x), c(y,z) + D_z(x) \}$

$= \min(60+0, 1+5) = 6.$

⇒

Y	
X	6
Y	0
Z	1

② 't1' → 'y' X for updt to 'z'.

After 't' $D_z(x) = \min\{ c(z,y) + D_y(x), c(z,x) + D_x(x) \}$

$= \min(1+6, 50+0) = 7$

Z	
X	7
Y	1
Z	0

③ t_2 → 'z' informs it new distance vector

& so on.

→ This is called Routing Loop

→ Count-to-Infinity problem

→ System is unstable.

* Solution to remove "Count to Infinity" problem:-

① Split Horizon:- If Z thinks that its best route to 'X' is via 'Y' then Z does not send the cost it has to 'X' when it updates Z, i.e. no update for 'Z' sent to Y.

Disadvantage: A protocol uses a timer, and if there is no news about a route, the node deletes the route from its table.

② Split Horizon with Poisoned Reverse :- If 'Z' thinks that its best route to 'X' is via 'Y', then 'Z' advertises its cost to 'X' as ∞ when it sends its DV to 'Y'. So, 'Y' will not route to 'X' via 'Z'.

Link → Active Interface.

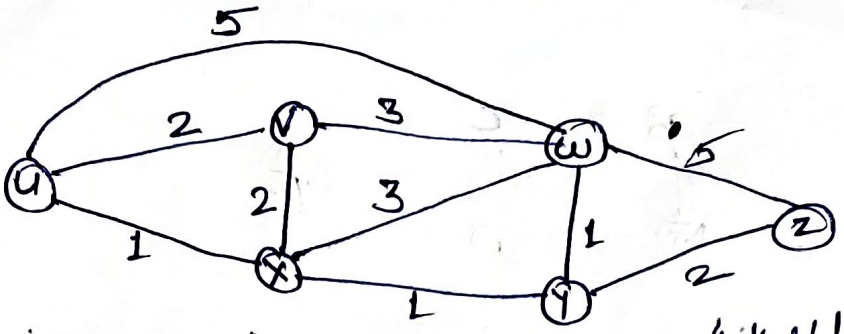
Link-state Routing Algorithm (Dijkstra's Algorithm)

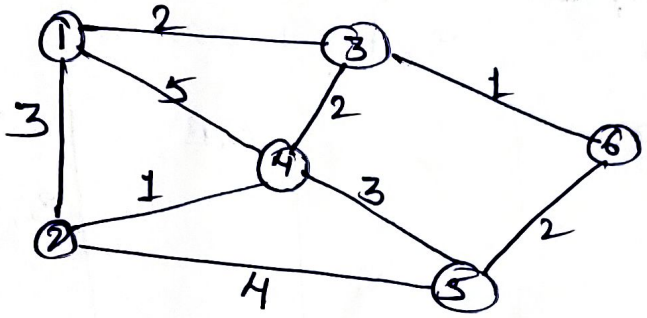
① Each node has complete map of the network called Link-state database (LSDB) → This is achieved by using Link-State Packet (LSP). It has identity of the node & cost of the link.

② All nodes have same info using "Link State Broadcast". (Flooding)

	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

LSDB of the ex. n/w present in all the nodes after Flooding





* Destination = 6

Link-State Routing has 2 phases
① Reliable Flooding
② Route Calculation (uses Dijkstra's algorithm)

Features:
① Link state packet
② Link State Database
③ Shortest path First Algo (Dijkstra's Algo)
④ Routing Table

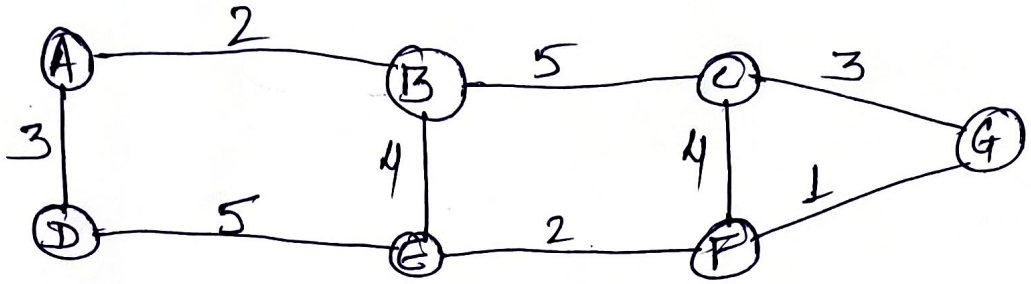
D1	D2	D3	D4	D5
∞ (-1)	∞ (-1)	1 (6)	∞ (-1)	2 (6)
3 (1-3-6)	6 (2-5-6)	1 (6)	3 (4-3-6)	2 (6)
3 (1-3-6)	4 (2-4-3-6)	1 (6)	3 (4-3-6)	2 (6)
3 (1-3-6)	4 (2-4-3-6)	1 (6)	3 (4-3-6)	2 (6)

$c(x,y)$:→ Link cost from 'x' to 'y'

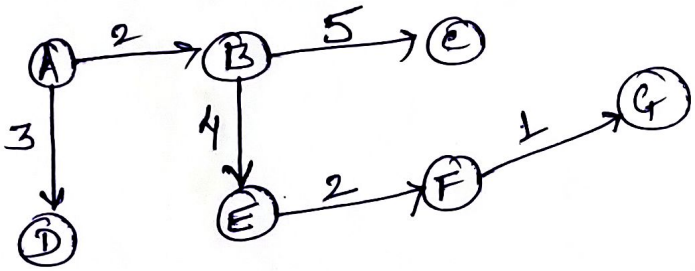
$D(v)$:→ current value of cost of path from source to destination v.

$p(v)$:→ predecessor node along path from source to v.

N :→ set of nodes whose least cost path from source is known.



N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)	D(G),p(G)
A	2,A	∞	3,A	∞	∞	∞
AB	2,A	7,B	3,A	6,B	∞	∞
ABD	2,A	7,B	3,A	6,B	∞	∞
ABDE	2,A	7,B	3,A	6,B	8,E	∞
ABDEC	2,A	7,B	3,A	6,B	8,E	10,C
ABDECF	2,A	7,B	3,A	6,B	8,E	9,F
ABDECFG	—	—	—	—	—	—



RIP (Routing Information Protocol) ⇒ Distance Vector.

OSPF (Open Shortest Path First) ⇒ Dijkstra's algorithm