Shortest Path Algorithm:

→ Determines the minimum cost of the path from source to destination.

→ Cost of the path $v_1 v_2 \cdots v_n$ is

$$\sum_{i=1}^{N-1} c_{i,i+1}$$

→ This is referred as weighted path length.

→ Unweighted path length → No of edges on the path namely N-1.

→ 2 types

① Single source shortest path
Unweighted shortest path
Dijkstra's Algorithm

② All pair shortest path
Warshall Algorithm
Floyd's Algorithm.

Unweighted shortest path:

∀ i/p Graph $G = (V, E)$ and distinguished vertex 's'.

→ applied to both weighted & Unweighted Graph!

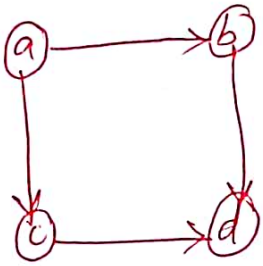
→ 3 pieces of infrn is maintained

Known

Distance vertex (dv) – Initially all vertices are '∝'

Path (Pv)

Eg.



step 1: Initialize table

V	Known	dv	Pv
a	0	0	0
b	0	∝	0
c	0	∝	0
d	0	∝	0

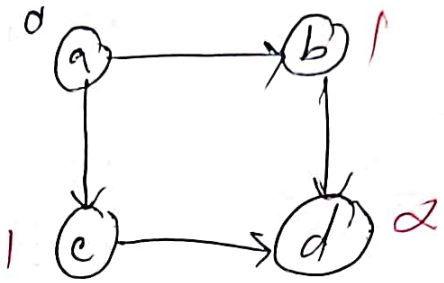
source vertex 'a' is initially assigned a path length '0'.

step 2:

Update adjacent vertices of 'a'

adjacent of 'a' is b, c

a is dequeue			
V	Known	dv	Pv
a	1	0	0
b	0	1	a
c	0	1	a
d	0	0	0
Enqueue : b, c			

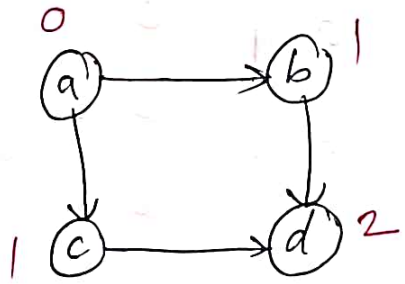


Step 3:

Next visit node 'b' & enqueue the unvisited adjacent of 'b'

adjacent of b is 'd'

b is dequeued			
V	Known	dv	Pv
a	1	0	0
b	1	1	a
c	0	1	a
d	0	2	b
Enqueue c, d			



Step 4: Visit Node 'c'

c is dequeued			
V	Known	dv	Pv
a	1	0	0
b	1	1	a
c	1	1	a
d	0	2	b

Enqueue 'd'

step 5: D is dequeued Queue becomes empty.

V	known	d_v	p_v
a	1	0	0
b	1	1	a
c	1	1	a
d	1	2	b

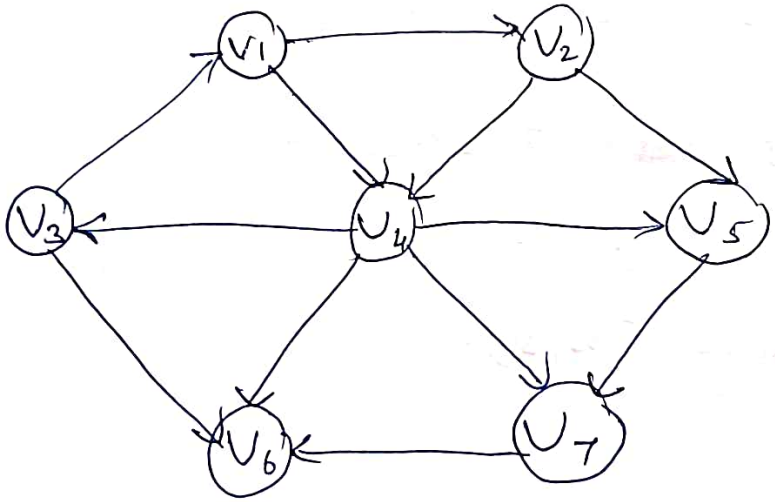
Shortest path from source vertex 'a' is

a → b is 1

a → c is 1

a → d is 2.

Eg:-



Routine :

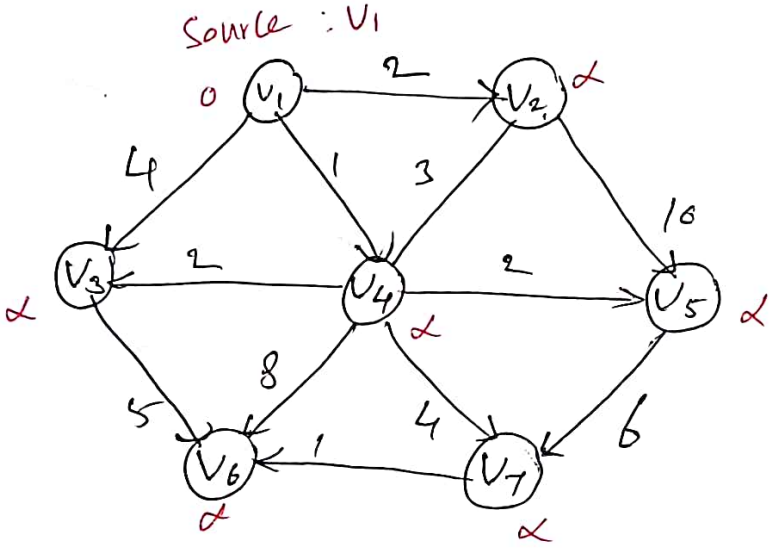
```
Void unweighted (Table T)
{
    Queue Q;
    Vertex v, w;
    Q = Create Queue (Num Vertex);
    Make empty (Q);
    Enqueue (s, Q);
    while (! Is empty (Q))
    {
        v = dequeue (Q);
        T[v].Known = True;
        for each w adjacent to v
        if (T[w].dist == Infinity)
        {
            T[w].dist = T[v].dist + 1;
            T[w].path = v;
            Enqueue (w, Q);
        }
    }
    dispose queue (Q);
}
```

Dijkstra's Algorithm

General method to solve single source shortest path is known as Dijkstra's alg.

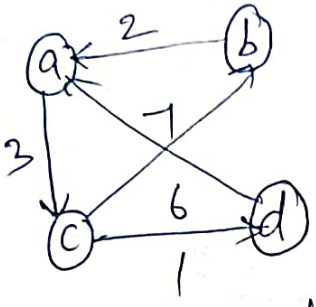
check - $d_w = d_v + c_{vw}$.

Eg:



Floyd's Algorithm:

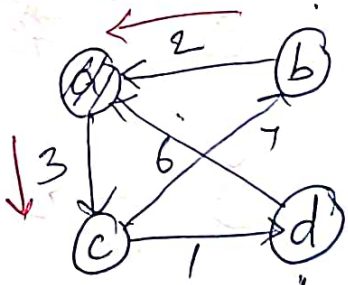
Eg:-1



$D^{[0]}$ =

	a	b	c	d
a	0	∞	3	∞
b	2	0	∞	∞
c	∞	7	0	1
d	6	∞	∞	0

$D^{[1]}$ - 1 intermediate node will be there

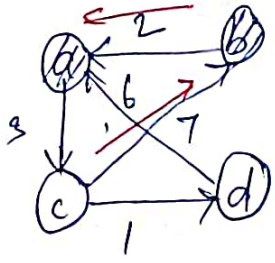


d → a → c = 9

b → a → c = 5

	a	b	c	d
a	0	∞	3	∞
b	2	0	[5]	∞
c	∞	7	0	1
d	6	∞	[9]	0

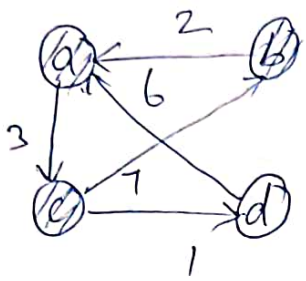
$D^{[2]}$ - 2 intermediate node 'a, b'



c → b → a = 9

	a	b	c	d
a	0	∞	3	∞
b	2	0	5	∞
c	[9]	7	0	1
d	6	∞	9	0

$D^{[3]}$ - 3 intermediate node: [a, b, c]



	a	b	c	d
a	0	[10]	3	[4]
b	2	0	5	[6]
c	9	7	0	1
d	6	[16]	9	0

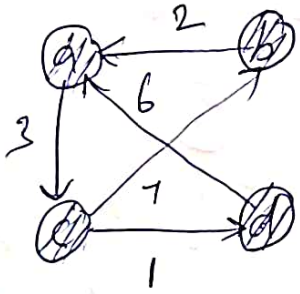
a → c → b = 10

a → c → d = 4

b → a → c → d = 6

d →(6) a →(3) c →(7) b = 16

$D^{[4]}$ - 4 intermediate node [a, b, c, d]



c → d → a = 7

	a	b	c	d
a	0	10	3	4
b	2	0	5	6
c	7	(9)	7	0	1
d	6	16	9	0