Traversal method

BFS — Breadth first search

DFS — Depth first search.

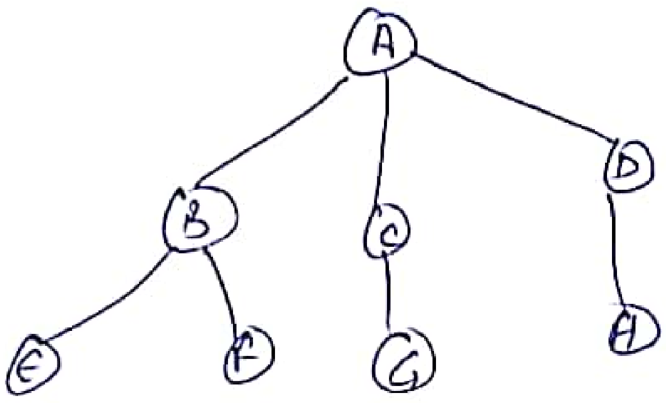
BFS

Algorithm

```
1.  (a) Visit the starting vertex and mark it
        as visited.
    (b) Display it.
    (c) Set a pointer to starting vertex.
2.  If ( current working vertex has adjacent
         unvisited adjacent vertex)
    {
         visited the adjacent ~~visitd~~ unvisited
         vertex and mark it visited. Insert
         it in a queue.
    }
    else
    {
       update the pointer to first element of Q and
       remove the first element from Q
    }
```

3. Repeat step 2 untill Q is empty.

Eg -



Queue.

Insert → H G F E D C B A → Remove

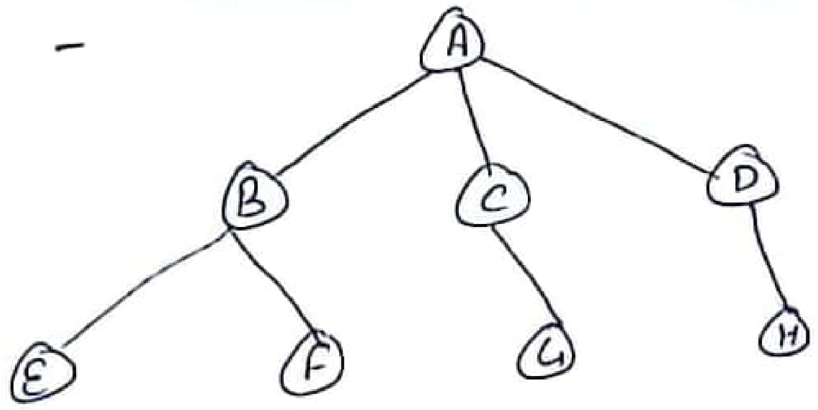
Display list - A, B C D E F G H

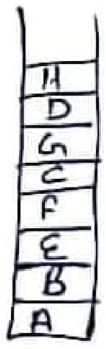
DFS

Algorithm

```
1.  Put the starting vertex into stack, mark
    it as visited and display it.
2.  If (stack [top] has adjacent unvisited vertex)
    {
        vist the vertex, mark it as visited and
        push it into the stack and display it
    }
    else
        pop top element from stack.
3.  Repeat step 2 untill stack is empty.
```

Eg -





Stack

Display list - A B E F C G D H

Topological Sort

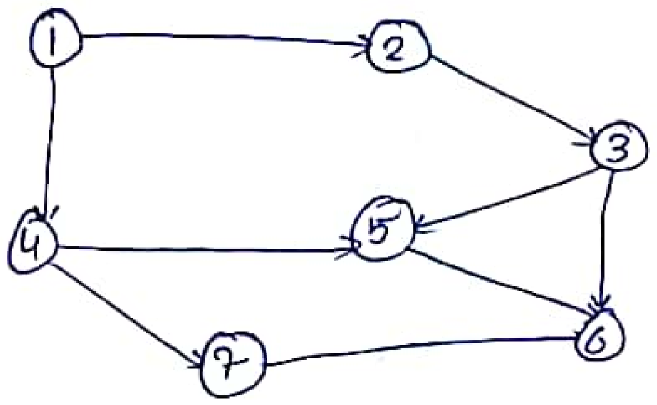
Directed acyclic graphs are used for topological sorts.

Algorithm -

```
TopologicalSort(G)
        for each vertex $u \in V$
            in-degree[u] ← 0
        for each vertex $u \in V$
            for each $v \in Adj[u]$
                in-degree[v] ← in-degree[v] + 1
        Q ← $\phi$
```

```
for each vertex u ∈ V
        if in - degree [u] = 0
                ENQUEUE (Q, u)
while Q ≠ φ
        u ← DEQUEUE (Q)
        output u
        for each v ∈ Adj[u]
                in - degree [v] ← in - degree [v] - 1
                if in - degree [v] = 0
                        ENQUEUE (Q, v)
    if in - degree [v] ≠ 0
        report that there is a cycle
```

Eg -



Solution - 1, 2, 4, 3, 5, 7, 6

Vertex	Indegree							
1	0	~~ATS~~ added to sol	ATS	ATS	ATS	ATS	ATS	ATS
2	1	0	ATS	ATS		"	"	"
3	1	1	0	0		"	"	"
4	1	0	0	ATS		"	"	"
5	2	2	2	1	0		0	"
6	3	3	3	3	2	1	ATS	"
7	1	1	1	0	0	0		

Minimum Spanning Trees

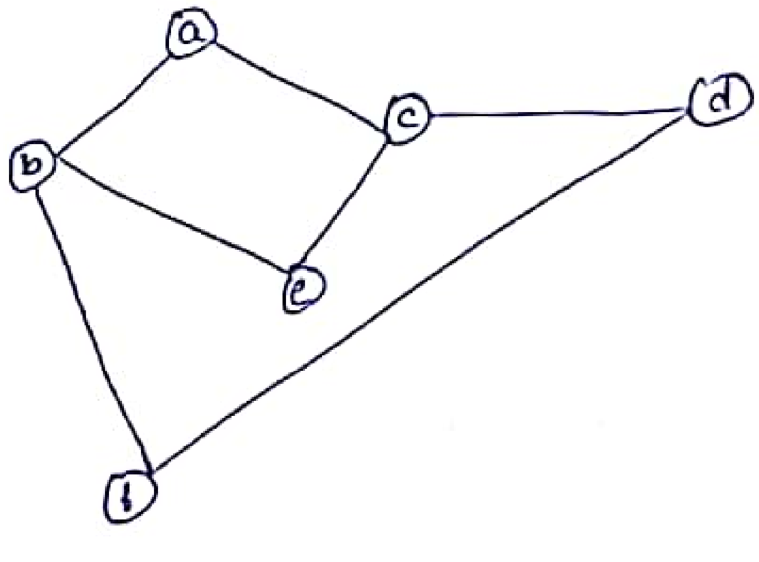
A minimum spanning tree is a spanning tree with weight less than or equal to weight of every other spanning tree.

All the vertex in a given graph are traversed.

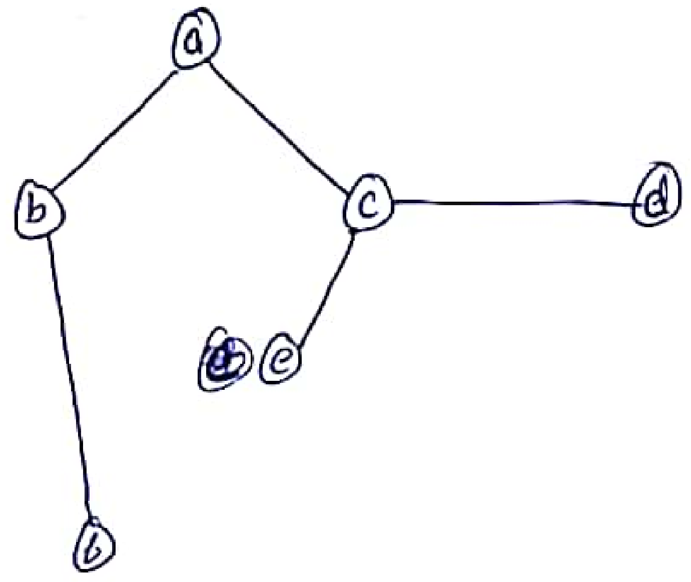
neglect all the edges that can form a cycle.

A tree formed by following the above condition is known as MST.

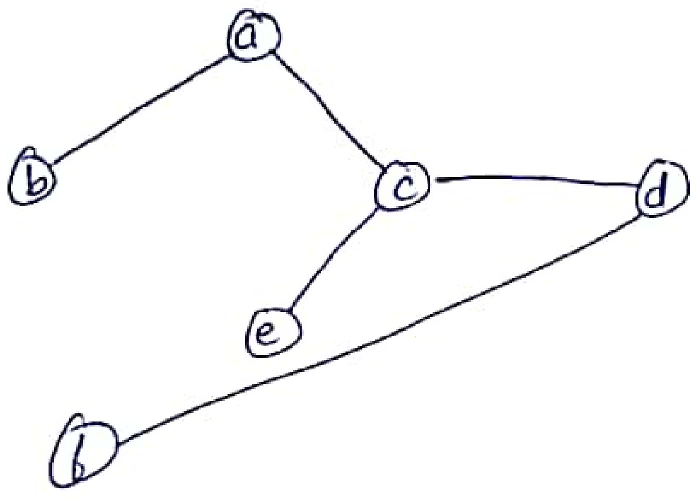
Eg -



1.



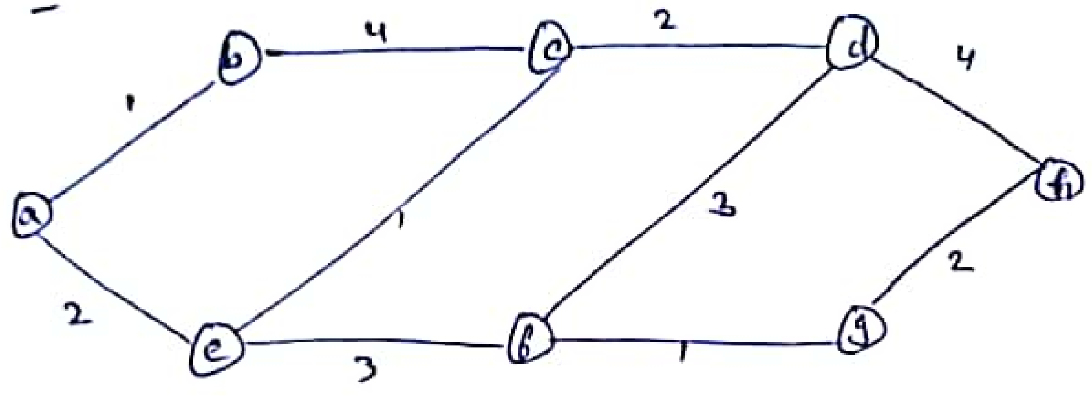
2.



MST (Kruskal's algo)

```
    A ← $\phi$
    for each vertex $v \in V[G]$
        makeset [v]
    sort the edges into increasing order of weight
    for each edge $(u,v) \in E$
    if findset [u] $\neq$ findset [v]
        $A \leftarrow A \cup \{u,v\}$
    union (u,v)
    return A
```

Eg -



edge	weight
(a,b)	1
(f,g)	1
(e,c)	1
(c,d)	2
(g,h)	2
(a,e)	2
(e,f)	3
(d,f)	3
(b,c)	4
(d,h)	4

{a}{b}{c}{d}{e}{f}{g}{h}

$\underbrace{\{a,b\}\ \{c}_{\text{in diff sets}},e\}$ {d} {f,g} {h}

{a,b} {c,e,d} {f,g,h}

{a,b,c,e,d} {f,g,h}

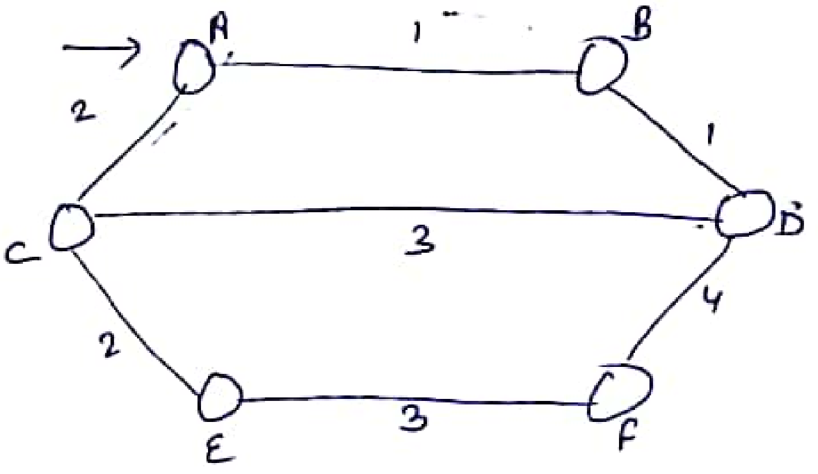
{a,b,c,e,d,f,g,h}

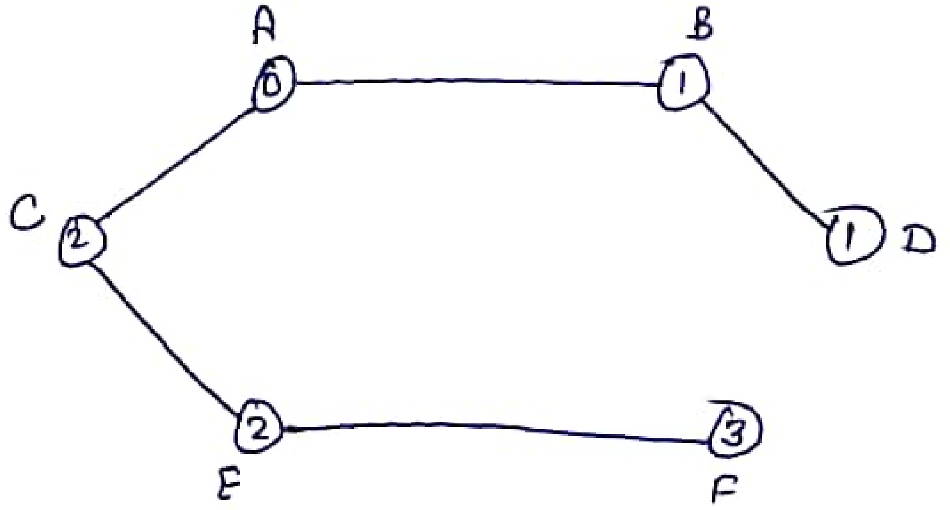
MST (Prim's algo)

```
MST_PRIM (G,w,r)
  for each u ∈ V[G]
    do key[u] ← ∞
       π[u] ← NIL
         key[r] ← 0
         Q ← V[G]
         while Q ≠ φ
       do u ← Extract_min (Q)
       for each v ∈ adj[u]
       ~~do if v ∈ Q and w(u,v) < key~~
```

```
do if $v \in Q$ and $w(u,v) < key[v]$ $w$
    $\pi[v] \leftarrow u$
    $key[v] \leftarrow w(u,v)$
```

Eg, -

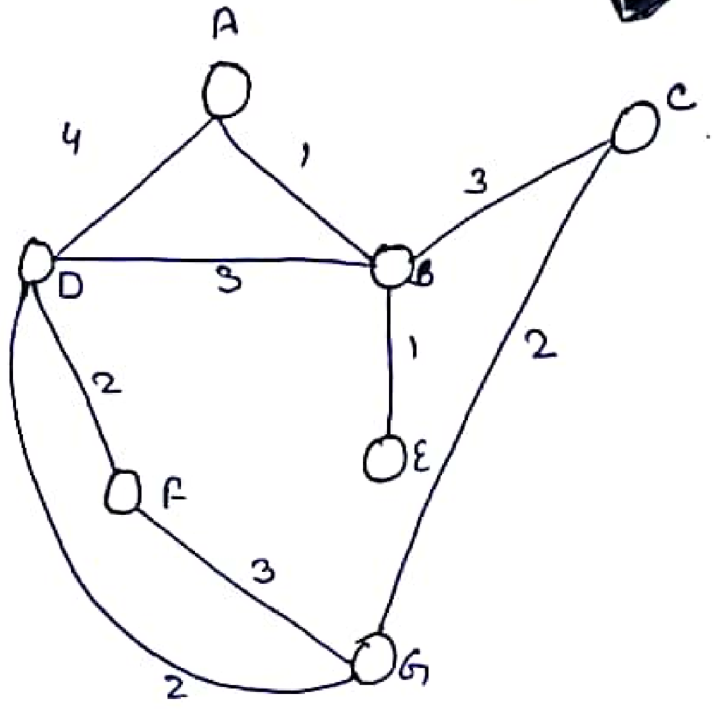


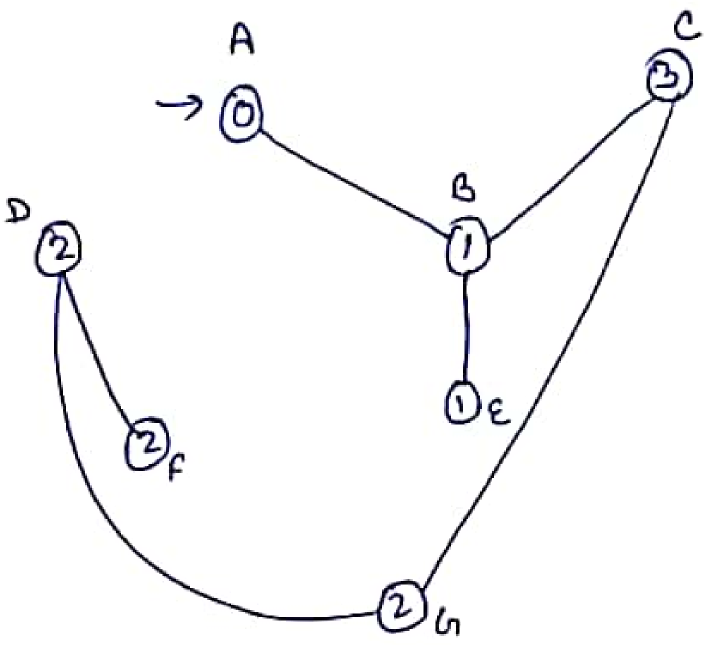


Scanned by CamScanner

Eg2

MST
(Prim's)





Relaxation

The single source shortest paths algorithm are based on a technique known as relaxation, a method that repeatedly decrease an upper bound on the actual shortest path weight of each vertex untill the upper bound equals the shortest path weights.

The process of relaxing an edge (u, v) consists of testing whether we can improve the ~~shortest~~ shortest path to v found so far by going through u.

```
Relax (u, v, w)
    if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$
             $\pi[v] \leftarrow u$
```

Dijkstra's Algorithm

Dijkstra's algorithm is a greedy algorithm that solves the ~~shortest~~ single-source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights.

Algorithm

~~DIJKSTRA (G, w, s)~~

~~Initialize-Single-Source (G, s)~~

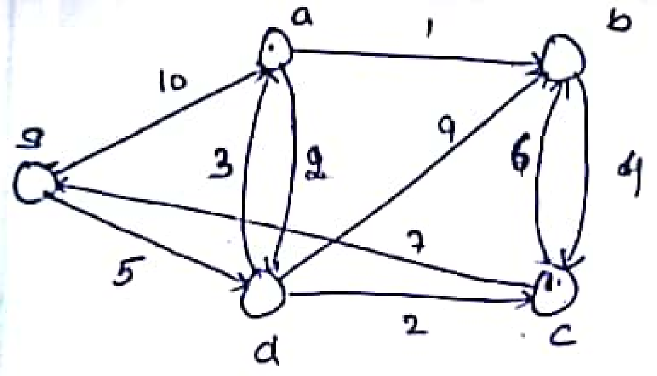
~~S←~~

```
Initialize_Single_Source(G, s)
    for each vertex v ∈ V[G]
        do d[v] ← ∞
           π[v] ← NIL
    d[s] ← 0

Relax (u, v, w)
    if d[v] > d[u] + w[u, v]
        then d[v] ← d[u] + w(u, v)
             π[v] ← u

Dijkstra (G, w, s)
    Initialize_Single_Source (G, s)
    S ← φ
    Q ← V[G]
    while Q ≠ φ
        do u ← Extract_Min(Q)
           S ← S ∪ {u}
           for each vertex v ∈ Adj[u]
           do Relax (u, v, w)
```

Eg –



	s	a	b	c	d
d[v]	∞	∞	∞	∞	∞
π[v]	-	-	-	-	-

s	a	b	c	d
0	10	∞	∞	∞
-	s	-	-	-

u = s
v = a
$\omega(u,v) = 10$
$\infty > 0 + 10$

d[v]	0	10	∞	∞	5
π[v]	-	s	-	-	s

0	10	∞	27	5
-	s	-	d	s

d[v]	0	10	914	~~9~~7	5
π[v]	-	s	d	d	s

0	8	14	7	5
-	d	d	d	s

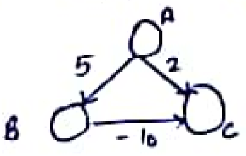
d[v]	0	8	13	7	5
π[v]	-	d	c	d	s

0	8	9	7	5
-	d	a	d	s

~~s(0)~~
~~[illegible])~~

s(0)
a(8) ← d ← s
b(9) ← a ← d ← s
c(7) ← d ← s
d(5) ← s

negative edges are not allowed because of negative edges we can not reach the proper solution.



Bellman - Ford algorithm

Bellman ford algorithm finds all shortest path lengths from a source $s \in V$ to all $v \in V$ or determines that a negative - weight cycle exist. It overcome the drawback of Dijkstra algorithm.

Algorithm -

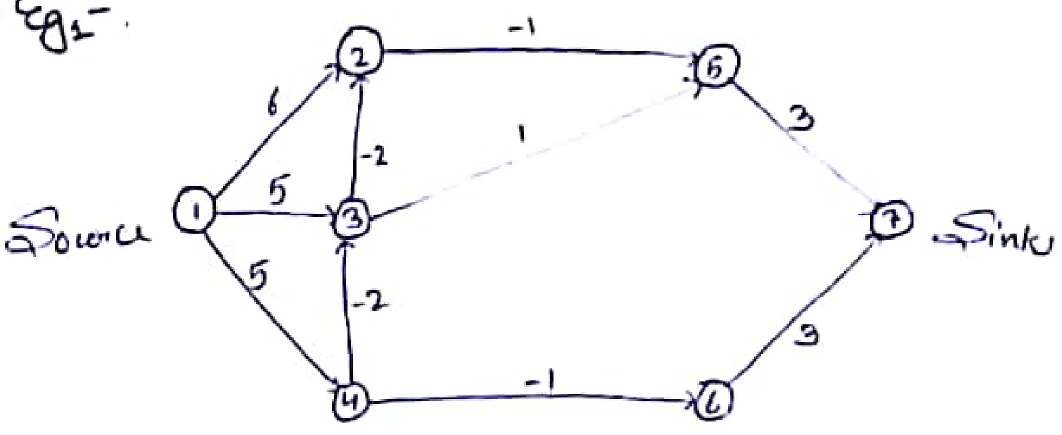
```
InitializeSingleSource(G, s)
    for each vertex $v \in V[G]$
        do $d[v] \leftarrow \infty$
        $\pi[v] \leftarrow NIL$
    $d[s] \leftarrow 0$

Relax (u, v, w)
    if $d[v] > d[u] + w(u,v)$
        then $d[v] = d[u] + w(u,v)$
        $\pi[v] \leftarrow u$

Bellman - Ford (G, w, s)
    InitialiseSingleSource (G, s)
    for $i \leftarrow 1$ to $|V[G]| - 1$
        do for each edge $(u,v) \in E[G]$
            do Relax (u, v, w)
    for each edge $(u,v) \in E[G]$
        do if $d[v] > d[u] + w(u,v)$
        return false
    return true
```

Scanned by CamScanner

Eg:-



(1,2)

(1,3)

(1,4)

(2,5)

(3,2)

(3,5)

(4,3)

(4,6)

(5,7)

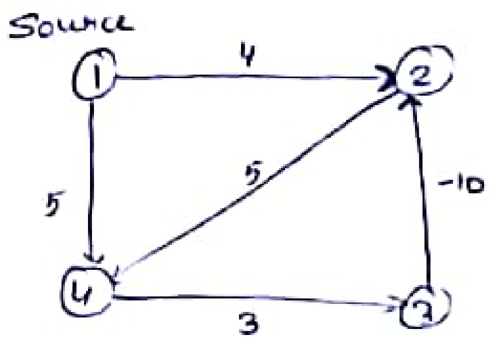
(6,7)

↑

this order can be changed.

	1	2	3	4	5	6	7
	0	∞	∞	∞	∞	∞	∞
(i)	0	~~6~~ 3	~~5~~ 3	5	5	4	~~8~~ 7
(ii)	0	~~3~~ 1	3	5	2	4	5
(iii)	0	1	3	5	0	4	3

Eg₂ -



(1,4)
(1,2)
(2,4)
(3,2)
(4,3)

	1	2	3	4
	0	∞	∞	∞
i	0	4 -10	8	5
ii	0	-10	-2	~~5~~ -5
iii	0	-10 -12	-2	-5

If relax travel are more than or equal to |V| - 1, then there ~~exist~~ a exist a -ve weight cycle in the graph.

Floyd Warshall Algorithm

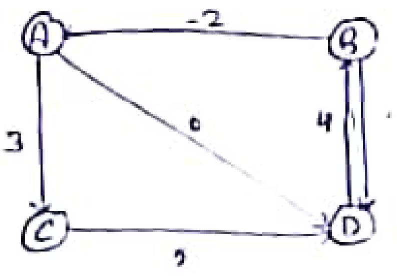
Floyd Warshall Algorithm is also known as Roy Floyd algorithm. It is a graph analysis algorithm for finding shortest paths in a weighted, directed graph. A single execution of the algorithm will find the shortest path in a ~~weighted~~, between all pairs of vertices.

Negative weights may be present but not negative weight cycle.

Algorithm -

```
FLOYD_WARSHALL (W)
1   n ← rows[W]
2   $D^{(0)} ← W$
3   for k = 1 to n
4       do for i ← 1 to n
5           do for j ← 1 to n
6               do $d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
7   return $D^{(n)}$
```

Eg -



$$A^0 = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & \infty & 3 & 0 \\ B & -2 & 0 & \infty & 1 \\ C & \infty & \infty & 0 & 0 \\ D & \infty & 4 & \infty & 0 \end{array}$$

The row and column are corresponding to intermediate vertex remains same.

$$A^k[i,j] = \text{Min}\left(A^{k-1}[i,j],\ A^{k-1}[i,k] + A^{k-1}[k,j]\right)$$

Using above formula we will calculate

A^1, A^2, A^3, A^4

$$A^1 = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & \infty & 3 & 0 \\ B & -2 & 0 & 1 & -2 \\ C & \infty & \infty & 0 & 5 \\ D & 4\infty & 4 & \infty & 0 \end{array}$$

$$A^2 = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & \infty & 3 & 0 \\ B & -2 & 0 & 1 & -2 \\ C & \infty & \infty & 0 & 5 \\ D & 2 & 4 & 5 & 0 \end{array}$$

$$A^3 = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & \infty & 3 & 0 \\ B & -2 & 0 & 1 & -2 \\ C & \infty & \infty & 0 & 5 \\ D & 2 & 4 & 5 & 0 \end{array}$$

$$A^4 = \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 4 & 3 & 0 \\ B & -2 & 0 & 1 & -2 \\ C & 7 & 9 & 0 & 5 \\ D & 2 & 4 & 5 & 0 \end{array}$$

The Floyd warshall algorithm consider the intermediate vertext of a shortest path. In above example, intermediate vertices are A, B, C, D

	BFS	DFS
1	BFS stands for Breadth first search	DFS stands for depth first search.
2	BFS can be done with the help of queue	DFS can be done with the help of stack.
3	BFS is slower than DFS	DFS is more faster than BFS
4	BFS require more memory consumption	DFS require less memory consumption
5	BFS is ~~[illegible]~~ useful in find shortest path	DFS is not useful in find shortest path.
6	This algorithm works in single stage. visited Vertices are removed from queue and then displayed at once.	This algorithm works in two stages. ~~[illegible]~~ Visited vertices are pushed onto stack and ~~where~~ later when there is no vertex left they are ~~[illegible]~~ popped from stack.
7	Structure of tree is wide and short	Narrow & long
8	Application → Spanning Tree	Application – Cycle Detection

#

	Prim's	~~Kruskal's~~ Kruskal's
1	~~Select the shortest edge~~ Select any vertex	Select the shortest edge in a network.
2	Select the shortest edge connected to the vertex	Select the next shortest edge which does not create a cycle

3	~~arcbest~~ Select shortest edge ~~a~~ connected to any vertex ~~[illegible]~~ already connected and repeat untill all vertices have been connected.	repeat step 2 untill all Vertices have been connected.
4	Prim's alg always stays as a tree	Kruskal's begins with forest and merge into tree.
5	Complexity is $O(N \log N)$ Search the least weight edge for every vertex.	Complexity is $O(N \log N)$ Comparison sord for edges
6	Running Time = $O(m \log n)$ m = edges n = nodes/vertex	Running time = $O(m + \log n)$ m = edges n = nodes/vertex