Fleury's Algorithm:

To find Euler path or Euler circuit in a Connected Graph:

(I) Let G be an Eulerian connected graph with each vertex of even degree

Step1 Select any vertex u as the starting vertex.

Step2. Select an edge e = (u, v).
If there are many such edges, select one that is not a bridge. (ie. removing that edge does not make the graph disconnected).
Remove that edge.

Note: If e is a bridge (select only if there is no alternative) then remove the vertex u.

Now from vertex v proceed further.

Step3. Repeat step 2 until all the edges exhausted.

In this we get Euler circuit.
(In Euler circuit all vertices are of even degree).

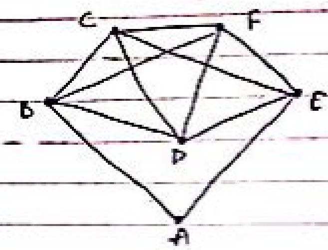
II. To find Euler path:
(We get Euler path if the Graph G has exactly two vertices of odd degree).

Step1. Select only those two vertices which have odd degree.

Step2. Same

Step3. Same.

Ex 1.



Sol^n. Firstly, check Degree.

deg (A) = 2   deg (D) = 4

deg (B) = 4   deg (E) = 4

deg (C) = 4   deg (F) = 4

Step 1. Start with vertex A.

Step 2. We can remove AB or AE

lets remove AB.

Then we can remove BC, BF or BD

Lets remove BC.

Now we can remove CF, CE or CD

Lets remove CF.

From F, we can remove FB, FD or FE

We will remove FE.

Now from E we can remove EC, ED, EA

We can't remove EA otherwise the graph will be disconnected

So, lets remove ED.

Now we remove DB

Then remove BF (only 1 option)

Then remove FD (")

Then remove DC (")

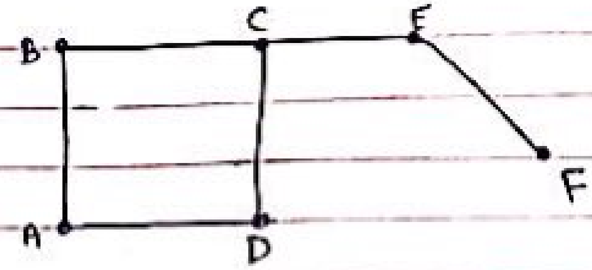
Then remove CE (")

Then remove EA (").

∴ Euler circuit :

A-B-C-F-E-D-B-F-D-C-E-A.

Ex 8



Sol-1 Firstly, Check Degree

deg (A) = 2 deg(D) = 2

deg (B) = 2 deg (E) = 2

deg (C) = 3 deg (F) = 1

Since two vertices C and F are of odd degree, we can find Euler path.

Step1. Start with vertex C or F.
lets start from C.

Step 2. We can remove CE, CD, or CB.
But CE is a bridge so we cant remove that.

Lets remove CB.

Then remove BA (only 1 option)

Then remove AD (only 1 option)

Then remove DC (only 1 option)

Then remove CE (only 1 option)

Then remove EF (only 1 option)

Now all the edges have been exhausted.
CB, BA, AD, DC, CE, EF.

Dijkstra's Algorithm

The algorithm relies on a series of iterations It involves assigning labels to vertices. Let G = (V, E) be a connected weighted graph. Let a and z be any two vertices where a be the starting point and z be the terminal point. Let L (v) denotes the labels at vertex v. At any point some vertices have temporary labels and the rest have permanent labels. It begins with by labelling starting vertex a, say, by 0 and other vertices with ∝. We next label all neighbours v of a by L (v), is the weight of the edge from a to v. Let u be the vertex among those v for which L (u) is minimum. Now find those neighbours of w of u and, for those w not already premanently labelled assign the label L (w) = L (u) + w (e), w (e) being the weight of the edge from u to w, while for those w already labelled L (w) change the label to L (u) + w (e) if this is smaller. Each iteration of the algorithm changes the status of one label from temporary to permanent, thus we may terminate the algorithm when z receives a permanent label.

Dijkstra's Algorithm

Input: A connected weighted graph G.

Output: L (z), the length of shortest distance from a to z

Step 1: Initially set the starting vertex a permanently with 0 i.e. L(a) = 0 and set L(v) = ∞ for all vertices $v \neq a$.

T = {vertices having temporary labels of G}

Step 2: Let u be a vertex in T for which L (u) is minimum and hence the permanent label of u.

Step 3: If $u = z$, stop.

Step 4: For every edge $e = (u, v)$, incident with u, if $v \in$ T, change L (v) to min (L (v), L (u) + w (e)).

Step 5: Change T to T − {u} and go to step 2.

Note 1. If the weight of the edge is not defined, then we assume the weight = 1.

2. There may exit more than one shortest path between two vertices in a graph.

3. The algorithm does not actually gives the shortest path, it gives only the shortest distance.

Example 36. Apply Dijkstra's algorithm to the graph given below and find the shortest path from a to f

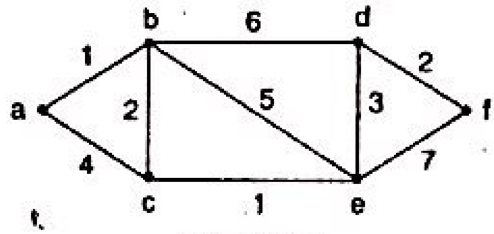


Fig. 13.49

Solution. The intial labelling is given by

Vertex V	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 two edges incident with a *i.e.* ab and ac where b and $C \in T$.

$$L(b) = \min\{\text{old } L(b), L(a) + w(ab)\}$$
$$= \min\{\infty, 0 + 1\} = 1$$
$$L(c) = \min\{\text{old } L(c), L(a) + w(ac)\}$$
$$= \min\{\infty, 0 + 4\} = 4$$

Hence minimum label is $L(b) = 1$

Vertex V	a	b	c	d	e	f
L (v)	0	1	4	∞	∞	∞
T	$\{$	$b,$	$c,$	$d,$	$e,$	$f\}$

Iteration 2: $u = b$, the permanent label of b is 1. T becomes $T - \{b\}$ there are three edges incident with b *i. e.* bc,bd and be where $c, d, e \in T$.

$$L(c) = \min\{\text{old } L(c), L(b) + w(bc)\}$$
$$= \min\{4, 1 + 2\} = 3$$
$$L(d) = \min\{\text{old } L(d), L(b) + w(bd)\}$$
$$= \min\{\infty, 1 + 6\} = 7$$
$$L(e) = \min\{\text{old } L(e), L(b) + w(be)\}$$
$$= \min\{\infty, 1 + 5\} = 6$$

Vertex V	a	b	c	d	e	f
L(v)	0	1	3	7	6	∞
T	$\{$		$c,$	$d,$	$e,$	$f\}$

Thus minimum label is $L(c) = 3$

Itreation 3 : $u = c$, the permanent label of c is 3, T becomes $T - \{c\}$. There is one edge incident with c *i.e.* ce where $e \in T$.

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

Thus minimum label is $L(e) = 4$

Vertex V	a	b	c	d	e	f
L(v)	0	1	3	7	4	∞
T	$\{$			$d,$	$e,$	$f\}$

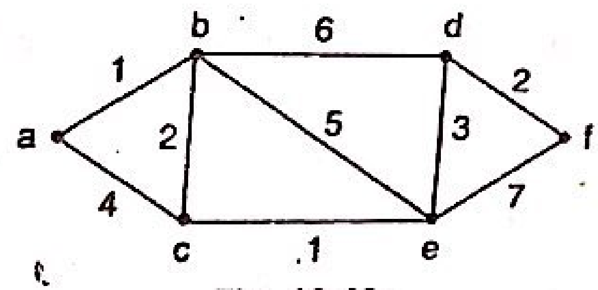


Fig. 13.49

Iteration 4 : $u = e$, the permanent label of e is 4, T becomes $T - \{e\}$. There are two edges incident with e *i.e.* ed and ef where $d, f \in T$

$$L(d) = \min\{\text{old } L(d), L(e) + w(ed)\}$$
$$= \min\{7, 4 + 3\} = 7$$
$$L(f) = \min\{\text{old } L(f), L(e) + w(ef)\}$$
$$= \min\{\alpha, 4 + 7\} = 11$$

Vertex V	a	b	c	d	e	f
L(v)	0	1	3	7	4	11
T	{			d,		f}

Thus minimum label is $L(d) = 7$.

Iteration 5 : $u = d$. The permanent label of d is 7. T becomes $T - \{d\}$. There is one edge incident with d *i.e.* df where $f \in T$

$$L(f) = \min\{\text{old } L(f), L(d) + w(df)\}$$
$$= \min\{11, 7 + 2\} = 9$$

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

The minimum label is $L(f) = 9$.

Since $u = f$, the only choice, iteration stops. Thus the shortest distance between a and f is 9.

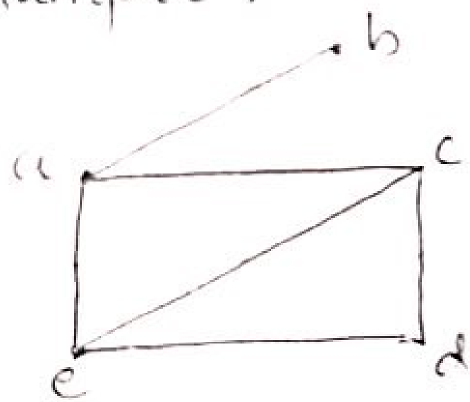
Representation of Graphs

Adjacency List :-

It is a way to represent a graph without multiple edges.
It specifies to each vertex of a graph.

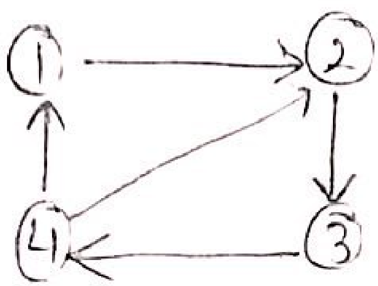
Example :

1



vertex		Adjacent vertices
a	→	b → c → e
b	→	a
c	→	a → d → e
d	→	c → e
e	→	a → c → d

2.



$\boxed{1} \rightarrow \boxed{2}$

$\boxed{2} \rightarrow \boxed{3}$

$\boxed{3} \rightarrow \boxed{4}$

$\boxed{4} \rightarrow \boxed{1} \rightarrow \boxed{2}$

BFS

Shortest Path in a Graph Without Weights

The length of a path in a graph without weights denotes the number of edges in a path and the shortest path is the path between two vertices in u and v that uses the least number of edges. One of the algorithms to find the shortest path between two vertices is known as **Breadth First Search** (B F S) algorithm . The algorithm is presented in terms of undirected graphs. Although the procedure works also for directed graphs. Let G be a graph and let s, t be two specified vertices of G. The general idea behind a breadth - fast search beginning at a starting vertex s is as follows. First we process the starting vertex s. Then we process all the neighbours of s. Then we process all the neighbours of neighbours of s and so on. Naturally we need to keep track of the neighbours of a vertex, and we need to guarantee that no vertex is processesd twice. The algorithm involves assigning labels to vertices.

Algorithm : The Breadth First Search Algorithm (BFS)

Step 1: Label vertex s with 0. set $i = 0$.

Step 2: Find all unlabelled vertices in G which are adjacent to vertices labelled i. If there are no such vertices then i is not connected to S.

If there are such vertices, label them $i + 1$.

Step 3: If t is labelled,go to step 4 . If not, increase i to $i + 1$ and go to step 2.

Step 4: The length of a shortest path from s to t is $i + 1$. Stop.

Once the length of the shortest path is found from the previous algorithm, we use the Backtracking algorithm to find the actual shortest path from s to t. This algorithm uses the label λ (v) which are generated in the BFS algorithm.

Algorithm: The back-tracking Algorithm for a Shortest Path

Step 1: Set $i = \lambda\ (t)$ and assign $v_i = t$

Step 2: Find a vertex u adjacent to v_i and with $\lambda(u) = i - 1$. Assign $v_{i-1} = u$.

Step 3: If $i = 1$ stop.

If not, decrease, i to $i - 1$ and go to step 2.

In general, there may be many shortest paths from s to t and the previous algorithm finds just one of them.

Example 35. Find the shortest path from vertex s to t and its length from the graph given below.

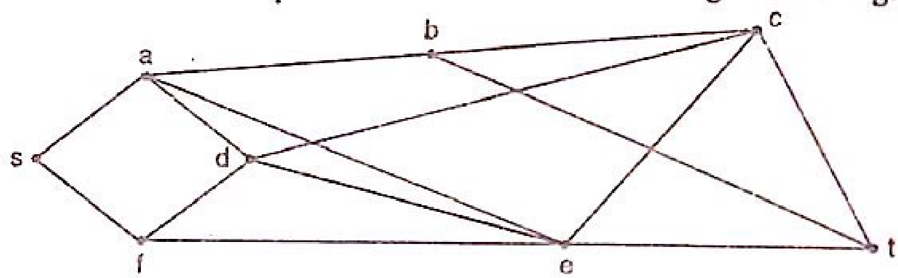


Fig. 13.47

Solution : Using BFS, label s as 0. Then a and f adjacent vertices of s are labelled $0 + 1 = 1$. Then b, d, e (adjacent vertices of a and f) are labelled $1 + 1 = 2$. Then c and t are labelled $2 + 1 = 3$. Since t is labelled 3, the length of a shortest path from s to t is 3.

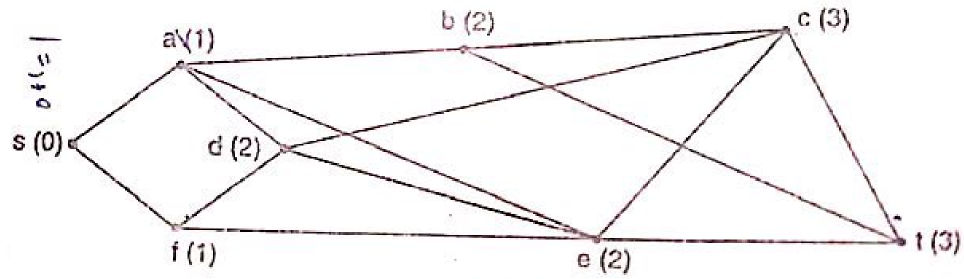
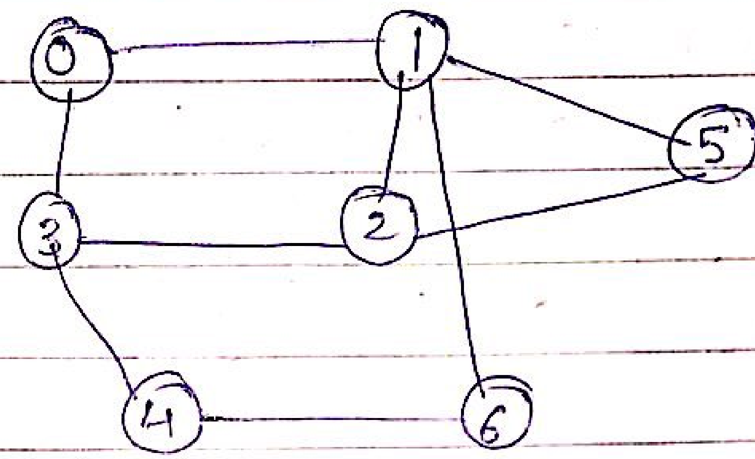


Fig. 13.48

Now, using second algorithm, since $\lambda(t) = 3$. We start with $i = 3$ and $v_i = t$.
We choose e (or b) adjacent to $v_3 = t$, with $\lambda(e) = 2$, and assign $v_2 = e$.
Next, we choose f adjacent to $v_2 = e$ with $\lambda(f) = 1$ and assign $v_1 = f$
Finally, we take s adjacent to f with $\lambda(s) = 0$ and assign $v_0 = s$.
This gives the shortest path $v_0, v_1, v_2, v_3 = sfet$ from s to t.

Note : There could be several shortest path from s to t, to be precise, there are 3 *i.e., saet, sabt* and *sfet.*

(Undirected)
Example



(FIFO) Queue

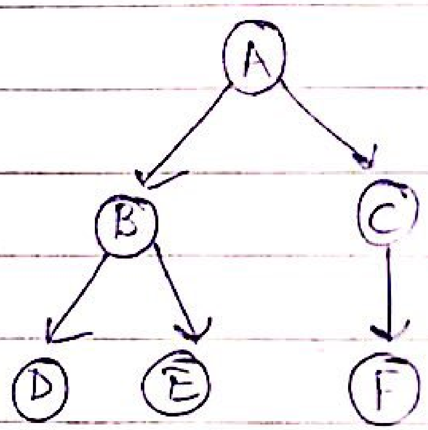
0	1	3	2	5	6	4

Result 0 1 3 2 5 6 4

BFS in directed Graphs:

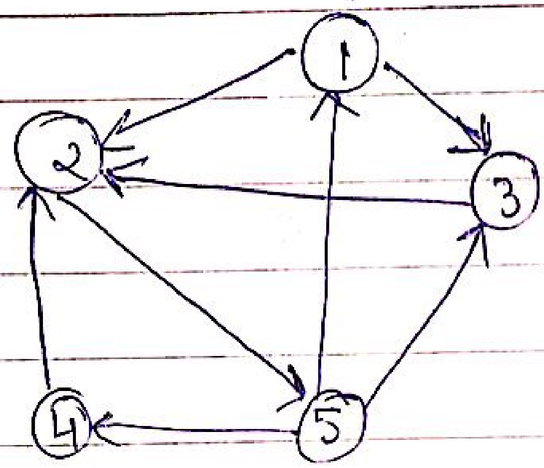
It starts at ~~the~~ some arbitrary node of a graph and explores the neighbour nodes first, before moving to the next level neighbours.

Example:

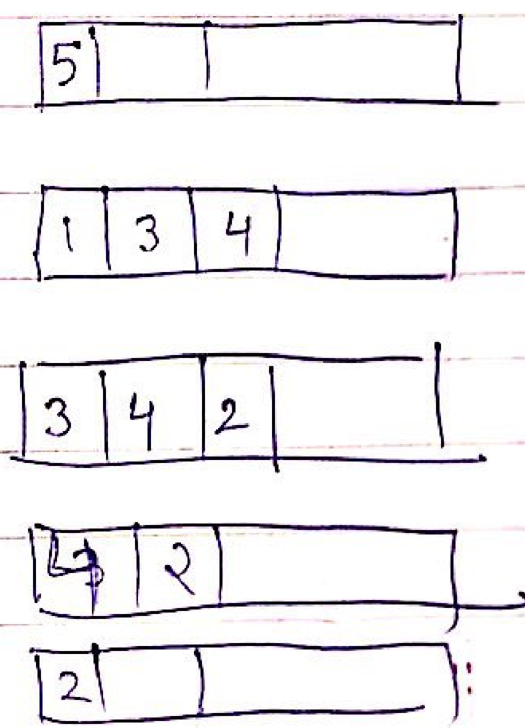


A B C D F E

BFS (Directed Graph)



lets Start with 5.



5 1 3 4 2.

DFS in Directed Graphs:-

DFS is a systematic way of visiting the nodes of either a directed or an undirected graph.

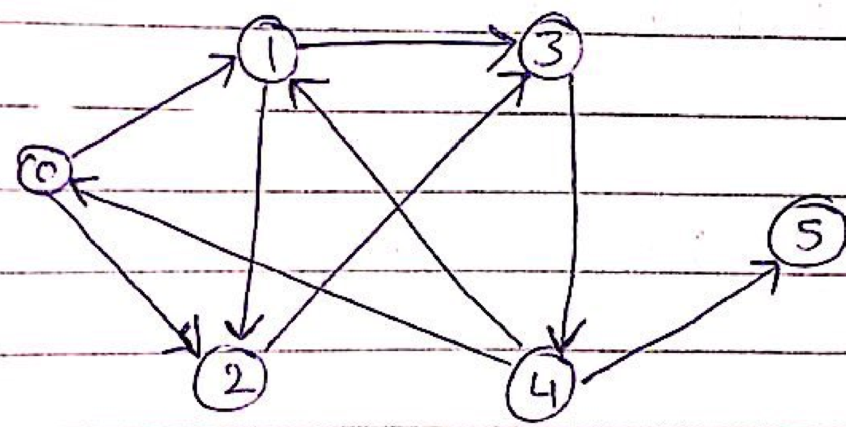
DFS visits the vertices of a graph in the following manner:

1. It selects a starting vertex v.
2. Then it chooses an incident edge (v,w) and searches recursively deeper in the graph whenever possible.
3. It does not examine all the incident edges one by one at the same time, but on the other hand, it goes deeper in the graph till no other such path exists.
4. When all edges incident to v have been explored, the algorithm backtracks to explore edges leaving the vertex from which v was discovered.

NOTE:- 1. We can pick any arbitrary node to start with.
2. We don't want to repeat 'visited' nodes.

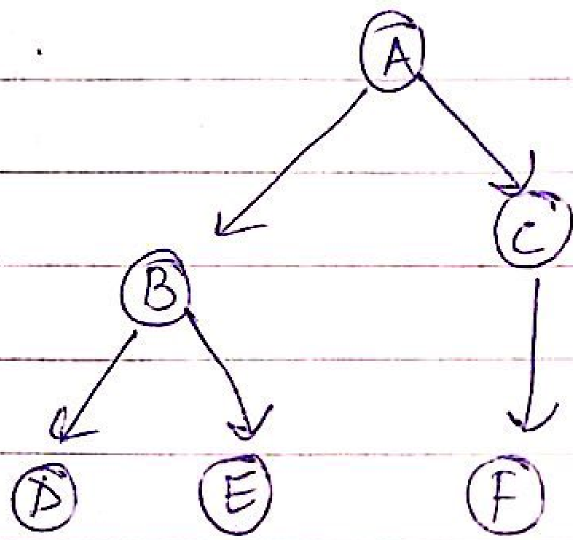
Scanned with CamScanner

Example:



0 1 3 4 5 2.

Example:



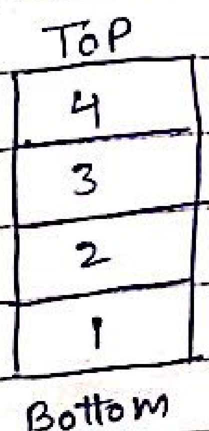
A B D E C F

Scanned with CamScanner

DFS :-

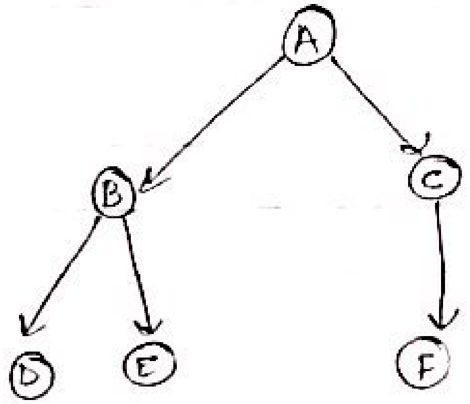
Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO (Last in first out) or FILO (First In Last Out).

Example: Plates stacked over one another in the canteen. The plate which is at the top is the first one to be removed i.e. the plate which has been placed at the bottommost position remain in the stack for the longest period of time. It follows LIFO or FILO.



Push - Add item to top
Pop - Remove item from top

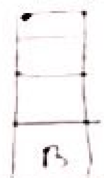
Example:



Soln. (1) Start initially with the vertex. A.
Push A in the stack.

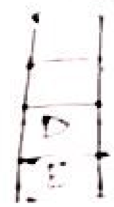
(2) Now see the neighbours of A i.e B or C
let us choose B.
A will be popped out and Push B in stack

Path – A



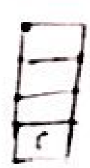
(3) Look for the neighbours of B i.e D and E
Pop B out, and push D and E

Path – AB



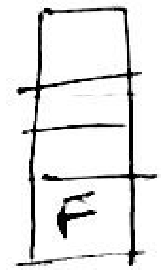
(4) Now D will be popped out first then E
Path – AB D E.

(5) Now Backtrack to A and see neighbours of A which are still not visited. i.e C
Push C in the stack



(6) Now C will popped out. and F will be pushed in.

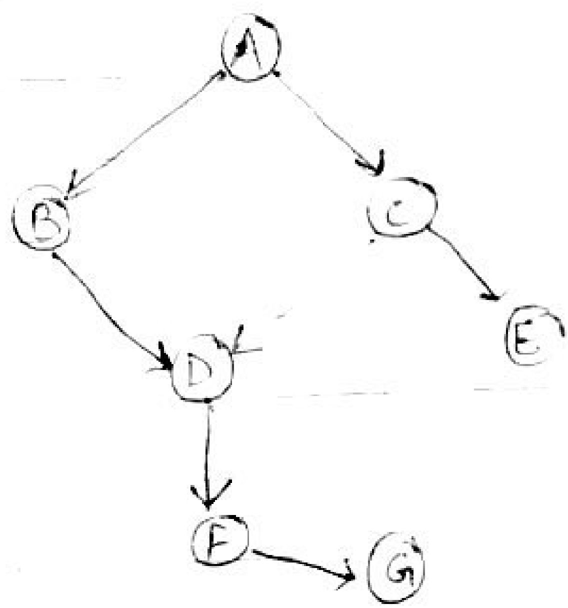
Path - ABDEC



(7) F will be popped out.

Path - A B L E C F.

Example:



Path - A B D F G C E

FLOYD- WARSHALL ALGORITHM:-

Floyd Warshall algorithm is a shortest-path algorithm. for graphs.

This algorithm computes the shortest distances between every pair of vertices in the input graph.

It can be used for finding shortest paths in a weighted graph with positive or negative edge weights. (but with no negative cycles).

Note: A negative cycle is a cycle whose edges sum to a negative value.

Formula :-

$d_{ij}^{(k)}$ = weight of shortest path from vertex i to j for which all intermediate node (vertices) are in $\{1, 2, --- k\}$.

$$d_{ij}^{(k)} = \begin{cases} w_{ij} , & \text{if } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right), & \text{if } k \geq 1 \end{cases}$$

We find $V \times V$ matrix.

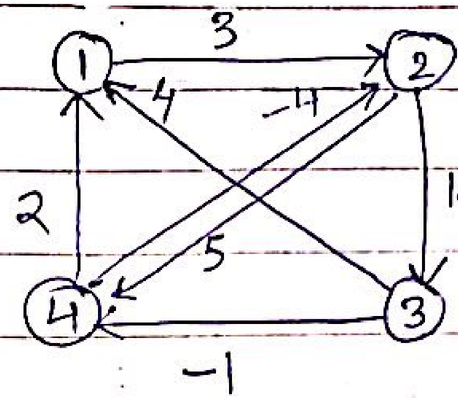
Note : Order of the matrix is given by the numbers of vertices).

$$D^{(m)} = d_{ij}^{(m)}$$

and $$d_{ij}^{(0)} = \begin{cases} 0 , & \text{if } i = j \\ w_{ij} , & \text{if } i \neq j \\ \infty , & \text{otherwise} \end{cases}$$

Here w_{ij} is the weight of the edge from vertex i to j

Example:



I $\quad D^0 =$

	1	2	3	4
1	0	3	∞	∞
2	∞	0	12	5
3	4	∞	0	-1
4	2	-4	∞	0

for $k=0$ $\quad d_{ij}^{(0)} = \{ \; w_{ij}$ if $k=0$

$$d_{ij}^{0} = \begin{cases} 0 & \text{if } i=j \\ w_{ij} & \text{if } i \neq j \\ \infty & \text{otherwise} \end{cases}$$

$\therefore\ d_{11}^{(0)} = 0$ { ∵ since there is no edge from ① to ① hence

Similarly $d_{22}^{(0)} = 0$ " " weight = 0 }.

$d_{33}^{(0)} = 0$

$d_{44}^{(0)} = 0$

$d_{12}^{(0)} = w_{12} = 3$ { weight from vertex ① to ② }.

$d_{13}^{(0)} = \infty$ { No edge from ① to ③ vertex }

$d_{14}^{(0)} = \infty$ { No edge from ① to ④ vertex }.

II for $k = 1$.

D^{1} =

	1	2	3	4
1	0	3	∞	∞
2	∞	0	12	5
3	4	7	0	-1
4	2	-4	∞	0

$d_{11}^{(1)} = \min \{ d_{11}^{(0)}, d_{11}^{(0)} + d_{11}^{(0)} \}$

$= \min \{ 0, 0 \}$

$= 0$

$\therefore\ d_{22}^{(1)} = 0,\ d_{33}^{(1)} = 0,\ d_{44}^{(1)} = 0.$

Now $d_{12}^{(1)} = \min \{ d_{12}^{(0)}, d_{11}^{(0)} + d_{12}^{(0)} \}$

$= \min \{ 3, 0 + 3 \}$

$= 3$

For $k=2$

III Similarly with find. D^2.

$$D^2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 15 & 8 \\ 2 & \infty & 0 & 12 & 5 \\ 3 & 4 & 7 & 0 & -1 \\ 4 & 2 & -4 & 8 & 0 \end{array}$$

IV for $K=3$

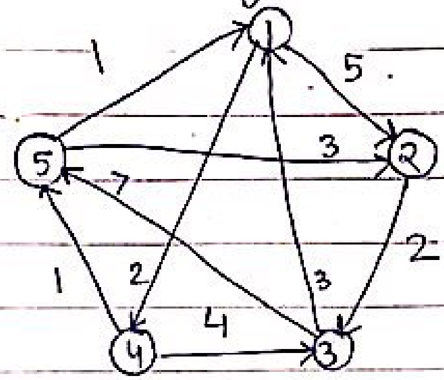
$$D^3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 15 & 8 \\ 2 & 16 & 0 & 12 & 5 \\ 3 & 4 & 7 & 0 & -1 \\ 4 & 2 & -4 & 8 & 0 \end{array}$$

V for $k=4$.

$$D^4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 15 & 8 \\ 2 & 7 & 0 & 12 & 5 \\ 3 & 1 & -5 & 0 & -1 \\ 4 & 2 & -4 & 8 & 0 \end{array}$$

Hence, we have find the shortest path between each pair of vertices.

Example: Floyd Warshall's Algorithm



Step 1.

$$D^{(0)} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 5 & \infty & 2 & \infty \\ \infty & 0 & 2 & \infty & \infty \\ 3 & \infty & 0 & \infty & 7 \\ \infty & \infty & 4 & 0 & 1 \\ 1 & 3 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

For $K=0$ $\quad d_{ij}^{(0)} = \{ w_{ij}, \text{ if } K=0 \}$

& $\quad d_{ij}^{(0)} = \begin{cases} 0, & \text{if } i=j \\ w_{ij}, & \text{if } i \neq j \\ \infty, & \text{otherwise} \end{cases}$

$\therefore \quad d_{11}^{(0)} = 0$

$d_{22}^{(0)} = d_{33}^{(0)} = d_{44}^{(0)} = d_{55}^{(0)} = 0.$

$d_{12}^{(0)} = 5.$, $d_{13}^{(0)} = \infty$ (as there is no edge from 1 to 3)

$d_{14}^{(0)} = 2$,

$d_{23}^{(0)} = 2$,

etc.

Scanned with CamScanner

Step 2.

$$
\begin{array}{c|ccccc}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
D^{(1)} = 1 & 0 & 5 & \infty & 2 & \infty \\
2 & \infty & 0 & 2 & \infty & \infty \\
3 & 3 & 8 & 0 & 5 & 7 \\
4 & \infty & \infty & 4 & 0 & 1 \\
5 & 1 & 3 & \infty & 3 & 0
\end{array}
$$

for $k=1$:

$$d_{ij}^{(k)} = \min\{(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})\}$$

$$d_{11}^{(1)} = \min\{d_{11}^{(0)}, d_{11}^{(0)} + d_{11}^{(0)}\}$$
$$= \min\{0, 0\}$$
$$= 0.$$

$$d_{12}^{(1)} = \min\{d_{12}^{(0)}, d_{11}^{(0)} + d_{12}^{(0)}\}$$
$$= \min\{5, 0+5\}$$
$$= 5.$$

$$d_{13}^{(1)} = \min\{d_{13}^{(0)}, d_{11}^{(0)} + d_{13}^{(0)}\}$$
$$= \min\{\infty, 0+\infty\}$$
$$= \infty$$ and so on.

Step 3.
$K=2$

$$
D^{(2)} = \begin{bmatrix}
0 & 5 & 7 & 2 & \infty \\
\infty & 0 & 2 & \infty & \infty \\
3 & 8 & 0 & 5 & 7 \\
\infty & \infty & 4 & 0 & 1 \\
1 & 3 & 5 & 3 & 0
\end{bmatrix}
$$

Step 4. $k = 3$.

$$D^{(3)} = \begin{bmatrix} 0 & 5 & 7 & 2 & 14 \\ 5 & 0 & 2 & 7 & 9 \\ 3 & 8 & 0 & 5 & 7 \\ 7 & 12 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$

Step 5. $K = 4$.

$$D^{(4)} = \begin{bmatrix} 0 & 5 & 6 & 2 & 3 \\ 5 & 0 & 2 & 7 & 8 \\ 3 & 8 & 0 & 5 & 6 \\ 7 & 12 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$

Step 6. $K = 5$.

$$D^{(5)} = \begin{bmatrix} 0 & 5 & 6 & 2 & 3 \\ 5 & 0 & 2 & 7 & 8 \\ 3 & 8 & 0 & 5 & 6 \\ 2 & 4 & 4 & 0 & 1 \\ 1 & 3 & 5 & 3 & 0 \end{bmatrix}$$