DS 35

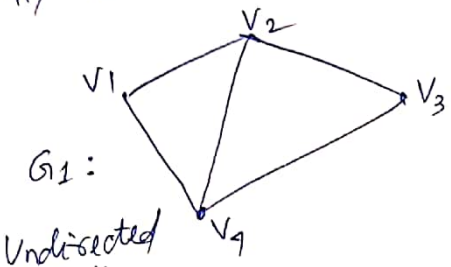
GRAPH :

* It is a non-linear data structure.
* In tree, there is a relationship: parent → child.
* In graph the relationship is less restricted.

GRAPH TERMINOLOGIES

1) Graph $G = (V, E)$ consists of two sets
   i) V: is the set of vertices
   ii) E: is the set of edges.

Eg:

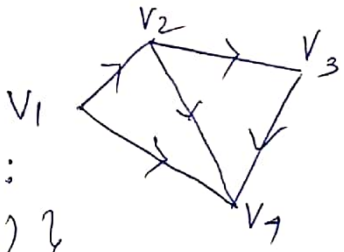


G_1: Undirected graph

$V = \{V_1, V_2, V_3, V_4\}$

$E = \{(V_1,V_2), (V_2,V_3), (V_3,V_4), (V_4,V_1), (V_2,V_4)\}$

2) Digraph : It is a directed graph s.t $G = \langle V, E \rangle$ where V is the set of all vertices and E is the set of ordered pair of vertices from V.



G_2:

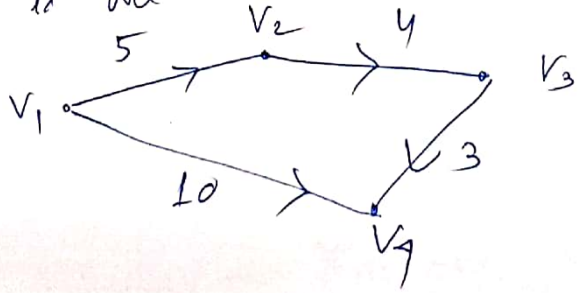
$V = \{V_1, V_2, V_3, V_4\}$

$E = \{(V_1,V_2), (V_2,V_3), (V_3,V_4), (V_1,V_4), (V_2,V_4)\}$

ie in a digraph $(V_i, V_j) \neq (V_j, V_i)$

3) WEIGHTED GRAPH :

A graph (or diagraph) is termed as weighted graph if all the edges in it are labeled with some weights.



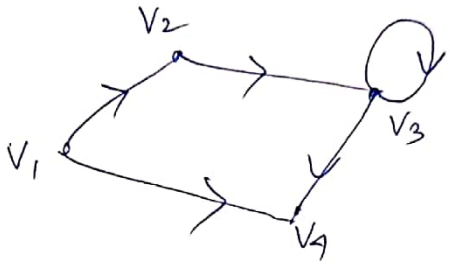
4) Adjacent Vertices:

Two vertices V_i and V_j are adjacent to each other if $\exists$ an edge

$e_{ij} : V_i \to V_j$ $\quad e_{ij} = (V_i, V_j)$

5) Self Loop:

If there is an edge whose start and end vertices are same that is (V_i, V_i) is an edge, then it is called a self loop (or simply a loop).



Here loop is at V_3.

6) Parallel Edges

If there exist two or more edge between two vertices then those edge are parallel edges.

Note: A graph which has either self loop or parallel edges or both is called a MULTIGRAPH.

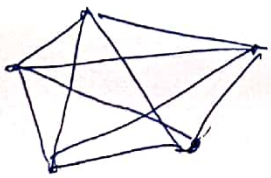
7) Simple Graph/Diagraph

A graph/diagraph having no self loop and parallel edges is called a simple graph/diagraph.

8) Complete Graph:

A graph/diagraph G is said to be complete if each vertex V_i is adjacent to every other vertex V_j in G.

i.e $\exists$ edges from any vertex to all vertices.



9) Acyclic Graph:

If a graph/diagraph doesnot have any cycle then it is called acyclic graph.

Eg: Tree.

10) Isolated Vertex

A vertex v_i of a graph G is said to be isolated if there is no edge which connects any vertex of G to v_i or vice-versa.

11) Degree of a Vertex

Degree of (v_i) = No. of edges connected with the vertex.

Indegree (v_i) = no. of edges incident into v_i (i.e ending at v_i)

Outdegree (v_i) = no. of edges emanating from v_i (Edges beginning at v_i)

12) Pendant Vertex

A vertex v_i is pendant if indegree $(v_i) = 1$, and outdegree $(v_i) = 0$.

Eg: Leafnode in a tree.

13) Connected Graph:

In a graph G, two vertices v_i and v_j are said to be connected if there is a path in G from v_i to v_j or $v_j \to v_i$

* A graph is said to be connected if there is at least one path for every pair of distinct vertices v_i, v_j in G.

* Strongly Connected: A graph/diagraph is strongly connected if $\exists$ path $v_i \to v_j$ and $v_j \to v_i$, $\forall$ $i = 1$ to n in G.

* Weakly Connected: If a diagraph is not strongly connected but underlying graph is connected $\Rightarrow$ the Graph is weakly connected.

Representation of Graphs:

The ADT to represent a graph are

1) Sequential or Matrix Representation
2) Linked List Representation.

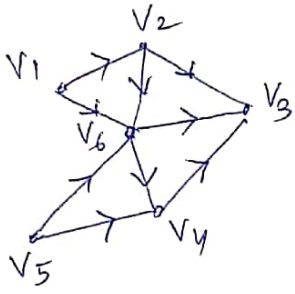
#1 MATRIX REPRESENTATION OF A GRAPH

— i) Adjacency Matrix
— ii) Incidence Matrix.

#i) Adjacency Matrix

$$A = [a_{ij}] = \begin{cases} 1, & \text{if } v_i \text{ is adjacent to } v_j \\ 0, & \text{if there is no edge between } v_i \text{ and } v_j \end{cases}$$

Eg: Satellite Network
Cellular Network
WSN
WiFi Network



$A =$

	V_1	V_2	V_3	V_4	V_5	V_6
V_1	0	1	0	0	0	1
V_2	0	0				
V_3						
V_4						
V_5						
V_6						

Note: 1) Adjacency matrix doesnot depend on the ordering of the vertices of a graph G. We can obtain same matrix by interchanging rows and columns.

2) If graph G is undirected then the adjacency matrix of G will be symmetric.

i.e $[a_{ij}] = [a_{ji}]$ for every i and j.

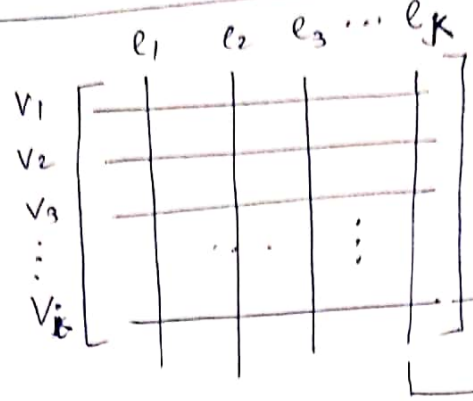
3) No. of 1's in an adjacency matrix = No. of edges in the graph.

4) This matrix is called a bit matrix or Boolean Matrix

DS
36

#ii INCIDENCE MATRIX:

$I =$



→ A row for a vertex

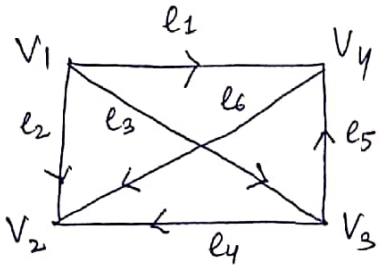
→ A column for an edge.

$$I[i,j] = \begin{cases} -1, \\ 0 \\ 1 \end{cases}$$

Imp

if $e_k : v_i \rightsquigarrow v_j \Rightarrow$ k the column has the value = 1, in ith row
= -1, in jth row
= 0, otherwise

Eg:

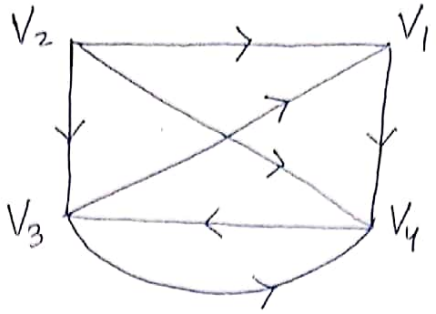


$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

$$I = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & -1 \\ 0 & 0 & -1 & 1 & 1 & 0 \\ -1 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \end{array}$$

Example:-

G:



The adjacency matrix A of graph G is:

$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Considers the powers $A, A^2, A^3, \ldots$ of the adjacency matrix A of G.

$$A^2 = A \cdot A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

$$A^3 = A^2 \cdot A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$A^4 = A^3 \cdot A = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Here $a_k(i,j) = (i,j)$th entry in the matrix A^k.

→ It gives the no. of paths of length "k" from node $v_i \longrightarrow v_j$

Let $B_k = A + A^2 + A^3 + \cdots + A^k$

$b(i,j) = m$. ie there are m number of paths of length "k" or less from node $v_i \xrightarrow{\text{to}} v_j$.

(iii) PATH MATRIX

Let G be a simple directed graph with n-nodes $v_1, v_2, v_3, \cdots v_n$. The path matrix or reachability matrix of graph G is the n-square matrix $P = p_{ij}$ is defined as

$$p_{ij} = \begin{cases} 1, & \text{if } \exists \text{ a path from } v_i \xrightarrow{\text{to}} v_j \\ 0, & \text{otherwise.} \end{cases}$$

Now from the above diagram, we have

$$B_4 = A + A^2 + A^3 + A^4$$

$$= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 1 \\ 2 & 0 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 2 & 3 \\ 5 & 0 & 6 & 8 \\ 3 & 0 & 3 & 5 \\ 2 & 0 & 3 & 3 \end{bmatrix}$$

$$\Rightarrow P = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

$\Rightarrow V_2$ is not reachable from any of the nodes.

SHORTEST PATH ALGORITHM

G : Directed graph with n-vertices, $v_1, v_2, \cdots, v_n$

Warshall's algorithm is more efficient than calculating the powers of the adjacency matrix A.

Let us define n-square logical matrices $P_0, P_1, P_2, \cdots, P_n$ as follows

Let $P_K[i][j] = (i,j)^{th}$ entry of the matrix P_K.

ie $P_K[i][j] = \begin{cases} 1, & \text{if there is a path from } v_i \to v_j \text{ which doesn't use any other nodes except possibly } v_1, v_2, \cdots, v_K \\ 0, & \text{otherwise} \end{cases}$

ie $P_0[i][j] = 1$, if there is an edge from $v_i \to v_j$ (direct).

$P_1[i][j] = 1$, there is a path from $v_i \to v_j$ which doesn't pass through any other node except node v_1.

$P_2[i][j] = 1$, if there is a path $v_i \xrightarrow{v_1, v_2} v_j$

$\vdots$

$P_K[i][j] = 1$ can occure only if one of the following two cases occures.

Case-i There is a path from node $v_i \to v_j$ which doesn't use any other nodes except possibly $v_1, v_2, v_3, \cdots, v_{K-1}$;

hence $P_{K-1}[i][j] = 1$

Case-ii There is a path from $v_i \to v_K$ and $v_K \to v_j$ where each path doesn't use any other nodes except possibly $v_1, v_2, \cdots, v_{K-1}$

i.e. $P_{K-1}[i][K] = 1$ & $P_{K-1}[K][j] = 1$

DS (37)

i.e Elements of matrix P_k can be computed as:

$$P_k[i][j] = P_{k-1}[i][j] \vee \left(P_{k-1}[i][k] \wedge P_{k-1}[k][j] \right)$$

$\wedge$ → logical AND

$\vee$ → Logical OR

WARSHALL'S ALGORITHM (TO FIND PATH MATRIX)

```
1) for i = 0 to n-1
2)     for j = 0 to n-1
3)         if (a[i][j] == 0)
4)             P[i][j] = 0;
5)         else
6)             P[i][j] = 1;
7) for k = 0 to n-1
8)     for i = 0 to n-1
9)         for j = 0 to n-1
10)            P[i][j] = P[i][j] V ( P[i][k] ∧ P[k][j] );
11) Exit
```

Initializing path matrix P.

WARSHALL'S SHORTEST-PATH PROBLEM

Let G be a directed weighted graph with n nodes: $V_1, V_2, \ldots V_n$.

e : edge,

$w(e) \geq 0$: is the weight of an edge "e".

Weight of the graph G is $W = [w_{ij}]$

where $w_{ij} = \begin{cases} w(e), & \text{if there is an edge from } V_i \to V_j \\ 0, & \text{if there is no edge from } V_i \to V_j \end{cases}$

Let D : is the distance matrix representing shortest paths between the nodes.

$$D = [d_{ij}]$$

d_{ij} = length of a shortest path from $V_i \rightarrow V_j$

Now we can define sequence of matrices $D_0, D_1, D_2, D_3 \cdots D_n$ whose entries are defined as follows:

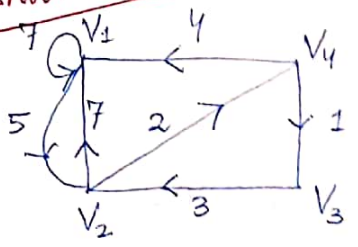
$$D_k[i][j] = \text{Min}\left(\text{Length of preceding path } V_i \rightarrow V_j, \text{ OR } \Sigma(\text{Length of preceding paths from } V_i \rightarrow V_k + V_k \rightarrow V_j)\right)$$

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

The initial matrix $D_0 = W$ (where each "0" $\leftarrow \infty$ (very large number))

The final matrix "D_n" will be the desired matrix D.

Example: (Warshall's Algorithm) / Floyd-Warshall Algorithm



$$W = \begin{bmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix}$$

$$D_0 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{bmatrix}$$

// No intermediate vertex between any two vertices

$$D_0 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{bmatrix} \qquad \Pi = \begin{bmatrix} V_1V_1 & V_1V_2 & - & - \\ V_2V_1 & - & - & V_2V_4 \\ - & V_3V_2 & - & - \\ V_4V_1 & - & V_4V_3 & - \end{bmatrix}$$

$D_1[2][3] = \min(D_0[2][3], D_0[2][1] + D_0[1][3]) = \min(\infty, 7+5) = 12$

$D_1[4][3] = \min(D_0[4][3], D_0[4][1] + D_0[1][3]) = \min(\infty, 4+5) = 9$

$$D_1 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & (12) & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & (9) & 1 & \infty \end{bmatrix} \qquad \Pi = \begin{bmatrix} V_1V_1 & V_1V_2 & - & - \\ V_2V_1 & V_2V_1V_2 & - & V_2V_4 \\ - & V_3V_2 & - & - \\ V_4V_1 & V_4V_1V_2 & V_4V_3 & - \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 7 & 5 & \infty & (7) \\ 7 & 12 & \infty & 2 \\ (10) & 3 & \infty & (5) \\ 4 & 9 & 1 & (11) \end{bmatrix} \qquad \Pi = \begin{bmatrix} V_1V_1 & V_1V_2 & - & V_1V_2V_4 \\ V_2V_1 & V_2V_1V_2 & - & V_2V_4 \\ V_3V_2V_1 & V_3V_2 & - & V_3V_2V_4 \\ V_4V_1 & V_4V_1V_2 & V_4V_3 & V_4V_2V_4 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 7 & 5 & \infty & 7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & (4) & 1 & (6) \end{bmatrix} \qquad \Pi = \begin{bmatrix} & & & \\ & & & \\ & & & \\ V_4V_3V_2 & & & V_4V_3V_2V_4 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 7 & 5 & (8) & 7 \\ (6) & (6) & (3) & 2 \\ (9) & 3 & (6) & 5 \\ 4 & 4 & 1 & 6 \end{bmatrix} \qquad \Pi = \begin{bmatrix} V_1V_1 & V_1V_2 & V_1V_2V_4V_3 & V_1V_2V_4 \\ V_2V_4V_1 & V_2V_4V_3V_2 & V_2V_4V_3 & V_2V_4 \\ V_3V_2V_4V_1 & V_3V_2 & V_3V_2V_4V_3 & V_3V_2V_4 \\ V_4V_1 & V_4V_3V_2 & V_4V_3 & V_4V_3V_2V_4 \end{bmatrix}$$

(Ans)

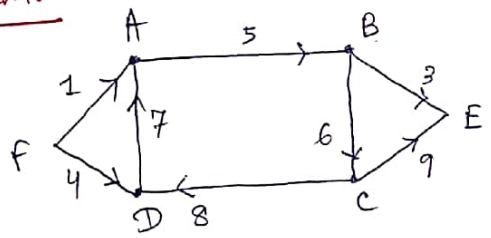
Warshall's All-Pairs Shortest Path Algorithm $\Theta(n^3)$.

Let W : Weight matrix.
G : graph
n : No. of nodes in the graph.

o/p: D : is a matrix to represent shortest path from $V_i \to V_j$

```
1. for i=0 ; i<= n-1; i++
2.     for j=0 to n-1
3.         if (W[i][j] == 0) then
4.             D[i][j] = ∞
5.         else
6.             D[i][j] = W[i][j]
7. for k=0 to n-1
8.     for i=0 to n-1
9.         for j=0 to n-1
10.            D[i][j] = Min(D[i][j], D[i][k] + D[k][j])
11. Exit.
```

Problem:



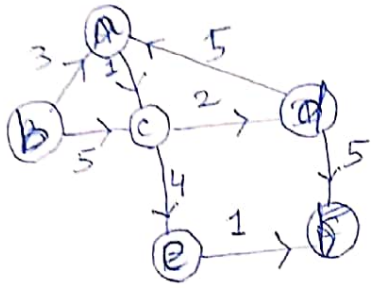
SINGLE - SOURCE SHORTEST PATH

DS 38

Background

1) In all-pairs shortest path problem there is a restriction of moving through certain vertices. Hence shortest path between any two pairs (via) restricted vertices may not be shortest. There may exist shorter path than this.

Example:



(#i) $b \rightsquigarrow f$ (path) : $b \to a \to c \to e \to f$ (weight)

$\Rightarrow$ weight (length) $(b \rightsquigarrow f) = 3 + 1 + 4 + 1 = 9$.

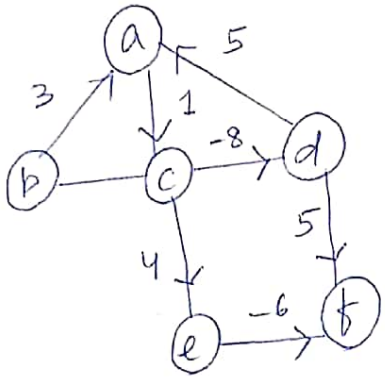
length $(b \rightsquigarrow f) = 4$ hops ✓

(#ii) $b \rightsquigarrow f$: $b \to c \to e \to f$

length $(b \rightsquigarrow f) = 3$ hops

2) When there exist negative edge weights this weighted edge for finding shortest path doesnot work properly.

Example:-



i) $d \rightsquigarrow f$: $d \to a \to c \to e \to f$

weight $(d \rightsquigarrow f) = 4$, length $(d \rightsquigarrow f) = 4$

weight $(d \to f) = 5$

length $(d \to f) = 1$ hop.

Note:- 1) Shortest path problem is not defined for graphs that contain negative cost cycles.

2) However $\exists$ soln for graphs that contain (both) positive and negative edge weights.

DIJKSTRA'S ALGORITHM (Single Source Shortest Path problem)

/* Non-negative weights */

Input:-

Output:-

For each vertex v, Dijkstra's algorithm keeps track of three pieces of information, K_u, d_u, and p_u.

K_u: False/True: shortest path to v is not known/known.

$K_u \leftarrow$ false $\forall\ v \in V$ initially.

d_u: Length of shortest path from $v \rightarrow$ Destination Vertex.

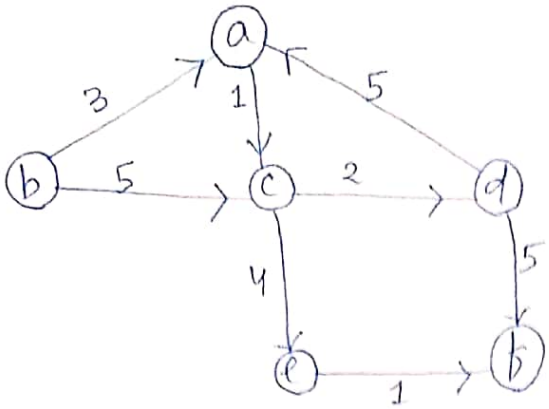
$d_u \leftarrow \infty$, $\forall\ v \in V$ s.t $v \neq V_s$ where $d_u = 0$
↳ source

p_u: Predecessor of vertex v on the shortest path $V_s \rightarrow v$
ie $\{V_s, \ldots, p_u, v\}$

Initially p_u is unknown for all $v \in V$.

Algorithm

1. for $i = 1$ to $|V|$
   Initialize K_u = false $\forall\ v \in V$
2. Select a vertex v having the smallest tentative distance (d_u)
3. Set K_u = true
4. For each vertex w adjacent to v for which $K_w \neq$ true
5. if $(d_w > d_u + c(v,w))$
6. then $d_w = d_u + c(v,w)$
7. set $p_w = v$
8. Exit



Initial

	a	b	c	d	e	f
Ku	F	F	F	F	F	F
du	∞	0	∞	∞	∞	∞
Pu	-	-	-	-	-	-

Pass-1

	a	b	c	d	e	f
Ku	F	T	F	F	F	F
du	3	0	5	∞	∞	∞
Pu	b	-	b	-	-	-

Pass-2

	a	b	c	d	e	f
Ku	T	T	F	F	F	F
du	3	0	4	∞	∞	∞
Pu	b	-	a	-	-	-

Pass-3

	a	b	c	d	e	f
Ku	T	T	T	F	F	F
du	3	0	4	6	8	∞
Pu	b	-	a	c	c	-

Pass-4

	a	b	c	d	e	f
Ku	T	T	T	T	F	F
du	3	0	4	6	8	11
Pu	b	-	a	c	c	d

Pass-5

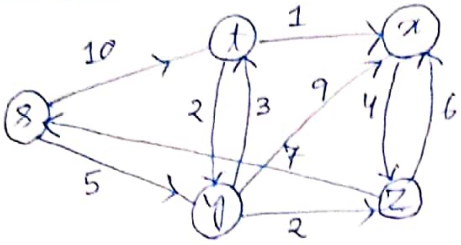
	a	b	c	d	e	f
Ku	T	T	T	T	T	T
du	3	0	4	6	8	9
Pu	b	-	a	c	c	e

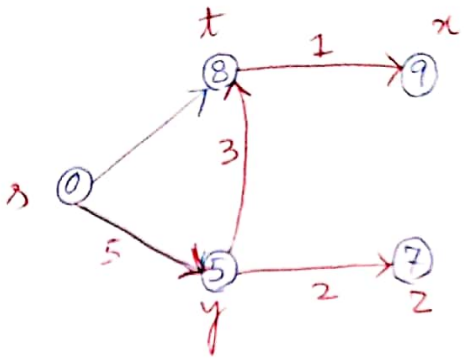
Pass-6

	a	b	c	d	e	f
Ku	T	T	T	T	T	T
du	3	0	4	6	8	9
Pu	b	-	a	c	c	e

(Ans)

Example :





DS
40

TRAVERSING A GRAPH :

Objective: Systematically examine the nodes and edges of a graph G.

Methods:- Two standard methods are:

— 1) Depth First Search (DFS)
   ↳ It uses STACK to hold nodes for future processing

— 2) Breadth First Search (BFS)
   ↳ It uses QUEUE as an auxiliary structure to hold nodes for future processing

Assumptions

Each node "n" of G will be in one of the 3-states, that are

flag = 1 : (Ready State) : The initial state of the node "n".

flag = 2 : (Waiting State) : The node "n" is in STACK/QUEUE waiting to be processed.

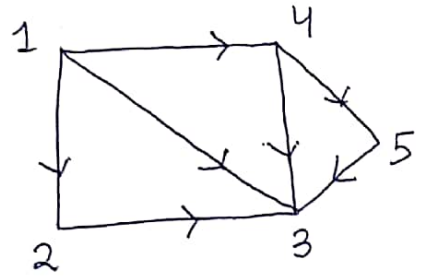
flag = 3 : (Processed State) : The node "n" has been processed.

PTO.

Algorithm (DFS)

1) Set flag = 1 for all nodes // Initialize all nodes to Ready State
2) Push (node 1 into the stack)
3) Set flag = 2 for node 1 // change the status of node 1 to Waiting state
4) while (stack != NULL)
5) x = Pop()
6) Process (x)
7) set flag = 3 for node x // change status to process state
8) for all y = Adj(x)
9) if (flag(y) == 1) then
10) Push(y) into the stack
11) flag = 2 for y.
12) Exit

Example (DFS):



Node	Adjacent List
1	2, 3, 4
2	3
3	
4	3, 5
5	3

Ans:-

1) Ready state | 1 | 1 | 1 | 1 | 1 | Status.
2) Push(1) onto the stack | 2 | 1 | 1 | 1 | 1 | Status.

3) $x = Pop()$ status | 3 | 1 | 1 | 1 | 1 |
$x = 1$

4) Push(2), Push(3), Push(4) // All adjacent of 1.

5) Stack | 4 | 3 | 2 |

| 3 | 2 | 2 | 2 | 1 | Status

Processed: | 1 |

6) $x = Pop()$
$x = 4$
Process (4):
Processed | 1 | 4 |
Status | 3 | 2 | 2 | 3 | 1 |

7) Push Adj(4) i.e Push(5) into stack. | 5 | 3 | 2 |,
Status. | 3 | 2 | 2 | 3 | 2 |

8) $x = Pop()$
$x = 5$
Processed | 1 | 4 | 5 |

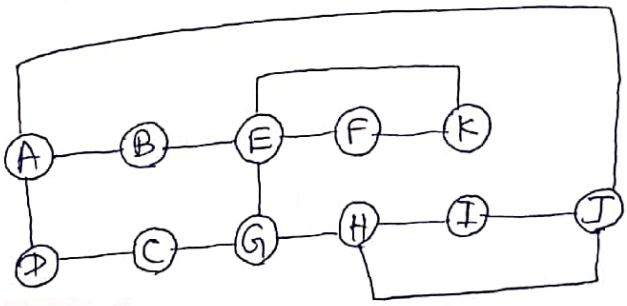
9) $x = Pop()$
$x = 3$
| 1 | 4 | 5 | 3 |

10) $x = Pop()$
$x = 2$
Processed | 1 | 4 | 5 | 3 | 2 |

Now stack is Empty.
∴ DFS is completed

The nodes are processed in the order 1, 4, 5, 3, 2. (Ans)

Problem:-



Algorithm (BFS) : Queue

1) Set flag = 1 for all nodes

1	1	1	...	1

Status: Ready

2) Insert (node 1) into the queue

1	

3) Flag = 2 for node 1

2	1	1	...	1

Status.

W ↑ Ready.

4) while (queue != Empty)

5) { x = Delete(); // Remove front node from queue

6) Process (x)

Queue

7) Flag = 3 for x

3	1	1	1	...	1

Status.

Processed: | 1 |

8) for all "Adj (x)"

9) { if (flag == 1) then

10) {

Insert (y)

11) flag = 2 for y

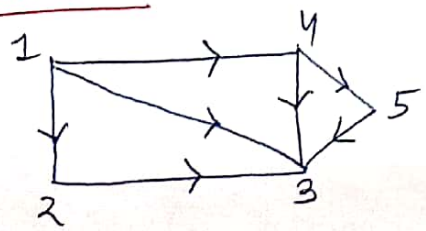
12) }

13) }

14) Exit

Node	Adjacent List
1	2, 3, 4
2	3
3	
4	3, 5
5	5

Problem:-



BFS: Traversal: 1, 2, 3, 4, 5