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.

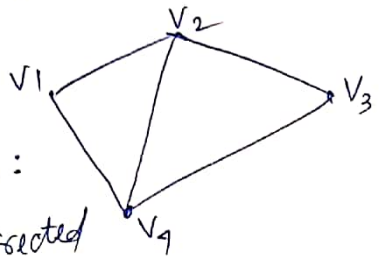
GRAPH TERMINOLOGIES

1) Graph $G=(V,E)$ consists of two sets

i) V: is the set of vertices

ii) E: is the set of edges.

Eg:

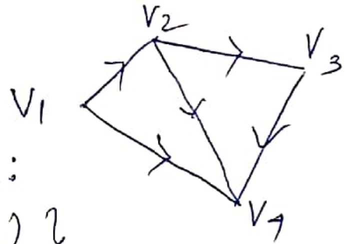


G1 : Undirected graph

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

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

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 pairs of vertices from V.



G2 :

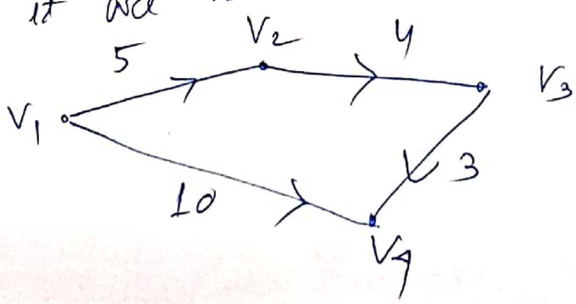
$V=\{V_1, V_2, V_3, V_4\}$

$E=\{(V_1,V_2), (V_2,V_3)(V_3,V_4), (V_1V_4), (V_2,V_4)\}$

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.



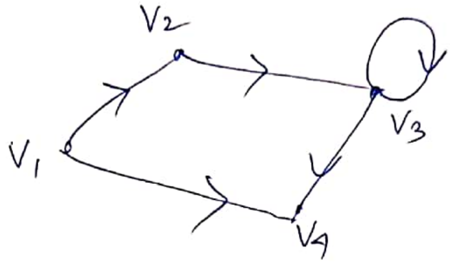
4) Adjacent Vertices:

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 ~~is~~ exist two or more edges between two vertices then those edges are parallel edges.

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

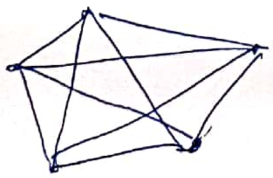
7) Simple Graph/ Diagraph

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 $V_i \xrightarrow{to} V_j$ 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.

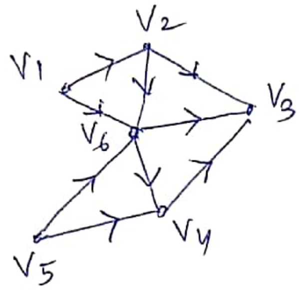
#1 MATRIX REPRESENTATION OF A GRAPH

— 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



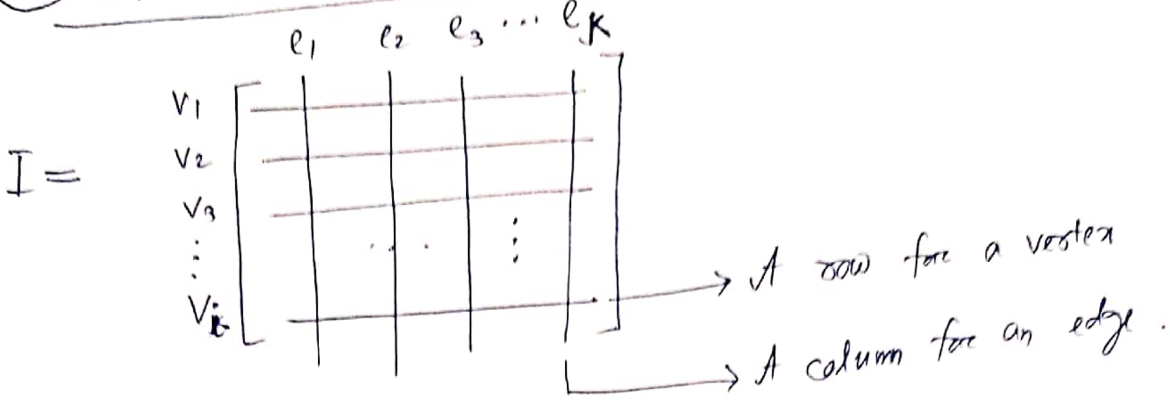
$$A = \begin{array}{c|cccccc} & V_1 & V_2 & V_3 & V_4 & V_5 & V_6 \\ \hline V_1 & 0 & 1 & 0 & 0 & 0 & 1 \\ V_2 & 0 & 0 & & & & \\ V_3 & & & & & & \\ V_4 & & & & & & \\ V_5 & & & & & & \\ V_6 & & & & & & \end{array}$$

Note: 1) Adjacency matrix does not 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:

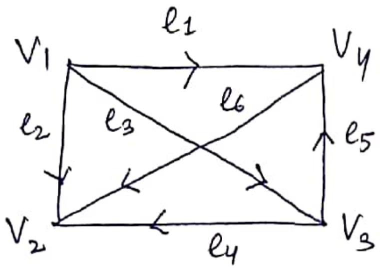


$$I[i,j] = \begin{cases} -1, \\ 0 \\ 1 \end{cases}$$

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:

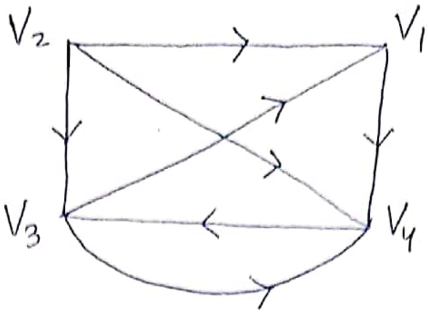


$$A = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

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

Example :-

G :



The adjacency matrix A of graph G is:

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

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

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

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

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

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 $B_K = A + A^2 + A^3 + \cdots + A^K$

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

(iii) PATH MATRIX

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

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

Now from the above diagram, we have

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

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

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

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

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

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.

i.e $P_K[i][j] = \begin{cases} 1, & \text{if there is a path from } V_i \to V_j \text{ which doesn't use any other nodes except possibly } V_1, V_2, \cdots, V_K \\ 0, & \text{otherwise} \end{cases}$

i.e $P_0[i][j] = 1$, 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 $V_i \xrightarrow{V_1 V_2} V_j$

⋮

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

$\boxed{\text{Case - i}}$ There is a path from node $V_i \to V_j$ which doesnot use any other nodes except possibly $V_1, V_2, V_3, \cdots, V_{K-1}$;

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

$\boxed{\text{Case - ii}}$ 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 $V_1, V_2, \cdots, V_{K-1}$

$\therefore P_{K-1}[i][K] = 1$ & $P_{K-1}[K][j] = 1$

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)

```
for i = 0 to n-1
    for j = 0 to n-1
        if (a[i][j] == 0)
            P[i][j] = 0;
        else
            P[i][j] = 1;
```

Initializing path matrix P.

```
for k = 0 to n-1
    for i = 0 to n-1
        for j = 0 to n-1
            P[i][j] = P[i][j] V (P[i][k] ∧ P[k][j]);
Exit
```

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 $D_0, D_1, D_2, D_3, \ldots, D_n$ whose entries are defined as follows:

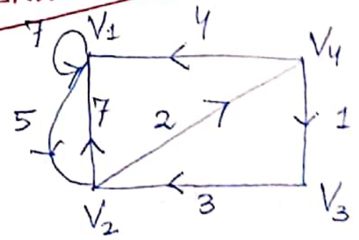
$$D_K[i][j] = \text{Min}\left(\text{Length of preceding path } V_i \rightarrow V_j, \text{ OR } \sum\left(\text{Length of preceding paths from } V_i \rightarrow V_K + V_K \rightarrow V_j\right)\right)$$

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

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

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

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

$D_1[4][2] = \min(D_0[4][2], D_0[4][1] + D_0[1][2]) = \min(\infty, 4+5) = 9$

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

$$D_2 = \begin{bmatrix} 7 & 5 & \infty & \cancel{9}\,7 \\ 7 & 12 & \infty & 2 \\ 10 & 3 & \infty & 5 \\ 4 & 9 & 1 & 11 \end{bmatrix} \qquad \Pi = \begin{bmatrix} V_1V_1 & V_1V_2 & - & V_1V_2V_4 \\ V_2V_1 & V_2V_1V_2 & - & V_2V_4 \\ V_3V_2V_1 & V_3V_2 & - & V_3V_2V_4 \\ V_4V_1 & V_4V_1V_2 & V_4V_3 & V_4V_1V_2V_4 \end{bmatrix}$$

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

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

(Ans)

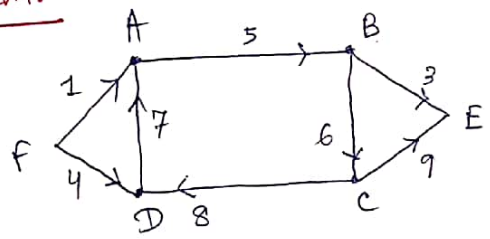
Warshall's All-Pairs Shortest Path Algorithm $\Theta(n^3)$.

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

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

1. for $i=0$; $i<=n-1$; $i++$
2. for $j=0$ to $n-1$
   if $(W[i][j]==0)$ then
3. $D[i][j]=\infty$
5. else $D[i][j]=W[i][j]$
6.
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

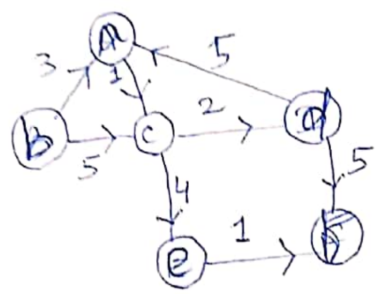
DS 38

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.

2) Example:



#i) $b \overset{path}{\rightsquigarrow} f : b \to a \to c \to e \to f$

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

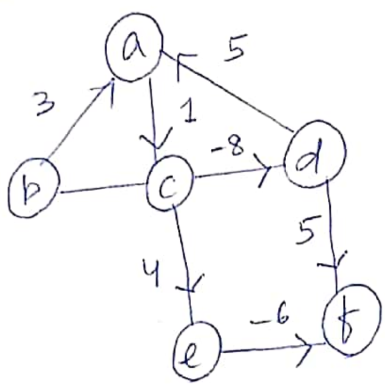
length $(b \rightsquigarrow f) = 4$ hops

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

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

3) 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 $\exists$ solu^n 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_v, d_v, and p_v.

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

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

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

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

$V_s \rightarrow$ source

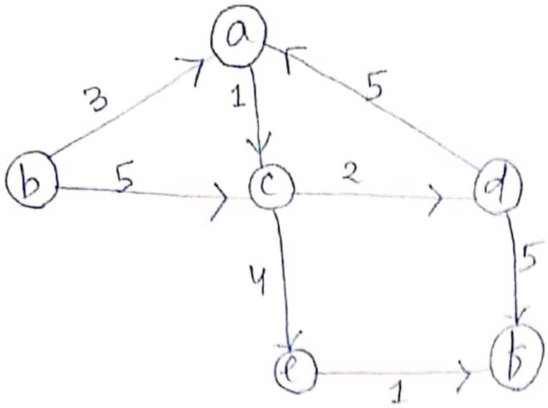
p_v : Predecessor of vertex v on the shortest path $V_s \rightarrow v$

i.e $\{V_s, \ldots, p_v, v\}$

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

Algorithm

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

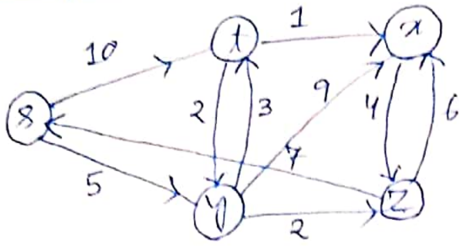
Pass-6

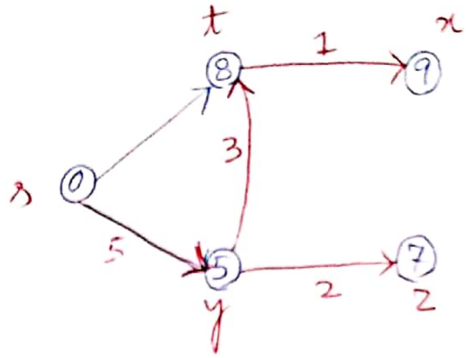
	a	b	c	d	e	f
Ku	T	T	T	T	T	T
du	3	0	4	6	8	9
pu	b	–	a	c	c	e

(Ans)

Scanned with CamScanner

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.

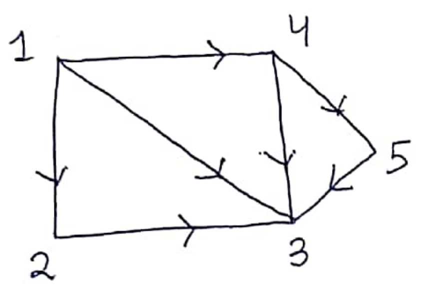
flag = 3 : (Processed State) : The node "n" has been processed.

PTO.

Algorithm (DFS)

1) Set $\boxed{flag = 1}$ for all nodes // Initialize all nodes to Ready State
2) Push (node 1 into the stack)
3) Set $\boxed{flag = 2 \text{ for node } 1}$ // 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.

Scanned with CamScanner

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) $x = Pop()$

$x = 2$

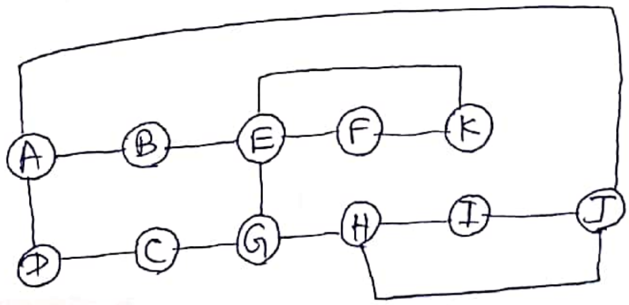
Processed | 1 | 4 | 5 | 3 | 2 |

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 "y = Adj (x)"

9) { if (flag == 1) then

10) { Insert (y)

11) flag = 2 for y

12) } }

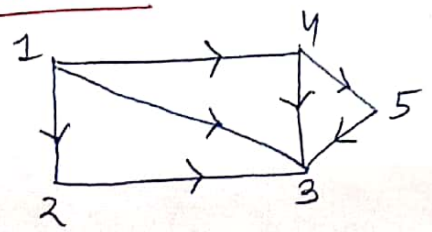
13) }

14) Exit

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

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

Problem:-



Scanned with CamScanner