Def^n

A graph G consist of two thing:

a) A set V of elements called nodes (or points or vertices)

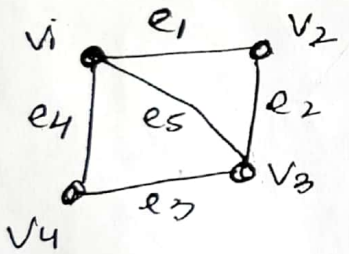
b) A set E of edges such that each edge e in E is identified with a unique (unordered) pair [u, v] of nodes in V, denoted by e=[u, v]

$G_1 = (V, E)$

$V(G_1)$ ⟶ set of vertices

$E(G_1)$ ⟶ set of edges

e.g.



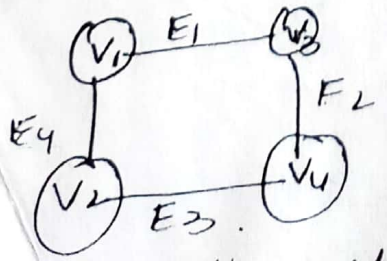
edges e_2

$E(G_1) = \{e_1, e_2, e_3, e_4, e_5\} \rightarrow 5$

$V(G_1) = \{v_1, v_2, v_3, v_4\} \rightarrow 4$

Undirected graph

If pair of vertices representing any edge is unordered i.e. as edge is represented by both (u, w) & (w, u)

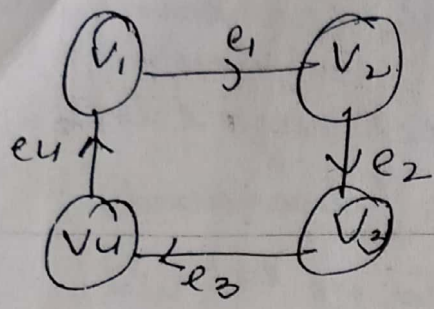


E_1 $[v_1, v_3]$ (v_3, v_1) are same.

NOTE: If graph shouldn't have multiple edges & loop. If any figure has multiple edges & loop then it is not graph.

Directed Graph

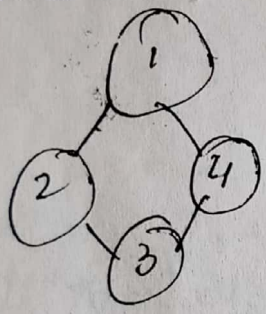
In which order is important. They are drawn with pointed arrows.



$e_1 = \{V_1, V_2\}$ — V_1: initial vertex; V_2: final vertex.

Adjacent Vertices:

→ Vertex V_1 is said to be adjacent to a vertex V_2 if there is a edge (V_1, V_2) or (V_2, V_1)



Vertex adjacent to node 2 = 3, 1

Path

→ A path P of length n from a node u to a node v is defined as a sequence of n+1 nodes.

→ A path from vertex u is a sequence of vertices, each adjacent to the next

Path = ⟨1, 2, 3⟩

Simple Path

→ The path P is simple if all the nodes are distinct, with the exception that $v_0 = v_n$

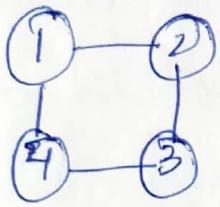
i.e. P is simple if nodes $v_0, v_1, \ldots v_{n-1}$ nodes are distinct.

→ when no node & no edge is repeated

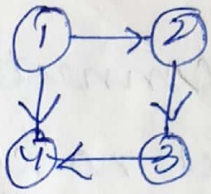
Cycle

A cycle is a path in which first & last vertic are the same.

e.g.



cycle → 1, 2, 3, 4, 1



NO cycle

Connected graph

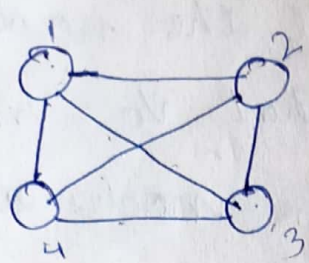
→ A graph is called connected if there exist a path from any vertex to any other vertex

Complete graph

A tree is said to be complete if every node u in G is adjacent to every other node v in G

i.e. A graph with n nodes will have $\frac{n(n-1)}{2}$ edges.

e.g.



nodes = 4

edges = $\frac{4(4-1)}{2}$

$= 12/2 = 6$

→ So complete graph is connected graph.

Tree:

→ Tree has n nodes, then it has n-1 edges

A graph is a tree if it follows:

2 properties.

(i) It is connected

(ii) There is no cycle in the graph.

Multigraph

A ~~graph is called as~~ multigraph if it satisfies one of the two properties:

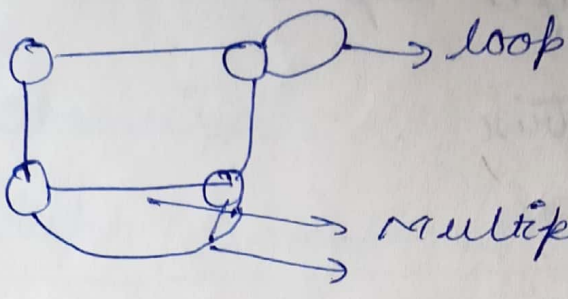
i) Multiple edges:

Distinct edges e & e' are multiple edges, if they connect the same end point, that is, $e = [u, v]$ & $e' = [u, v]$

ii) Loops: An edge e is called a loop if it has identical endpoint i.e. $e = [u, u]$

Multigraph is not graph as it doesn't allow above two properties.

Degree

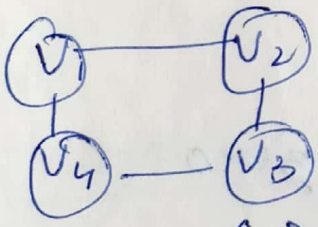


Degree

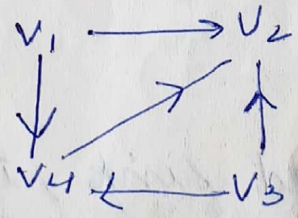
→ The no. of edges incident on a vertex determine its degree

if degree (u) = 0, then that vertex u does not belong to any edge, so, it is called isolated vertex.

In directed graph → The degree is term as indegree and outdegree



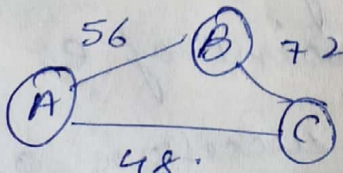
degree (V_1) = 2



for vertex V_4
indegree = 2
outdegree = 1

Weighted graph

→ A graph is said to be weight graph if every edge in the graph is assigned some weight or value



Graph representation

There are 2 graph representations.

(i) Adjacency Matrix

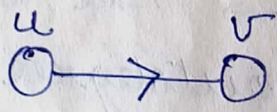
(ii) Adjacency list.

Arc

When G is directed graph with a directed edge $e = (u, v)$. Then e is also called arc

→ Source

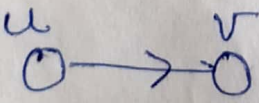
A node u is called source, if it has positive outdegree but zero indegree.



u → source

→ Sink

A node is called sink, if it has zero outdegree but a positive indegree.



v is sink

Note:

e begins at u & ends at v

u is origin or initial point of e & v is the destination or terminal point of e

u is predecessor of v, v is successor or neighbour of u

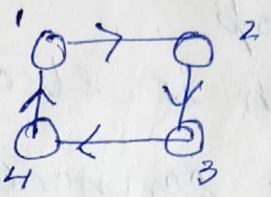
(iv) u is adjacent to v & v is adjacent to u.

Strongly connected.

Directed graph / digraph / graph

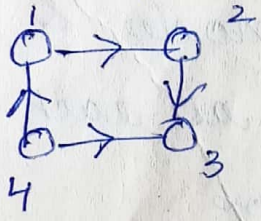
Strongly or connected directed graph

If for each pair u, v of nodes in G there is a path from u to v and there is also path from v to u. e.g.



Unilaterally connected

G is said to be unilaterally connected if for any pair u, v of nodes in G there is a path from u to v or a path from v to u



→ Parallel edges

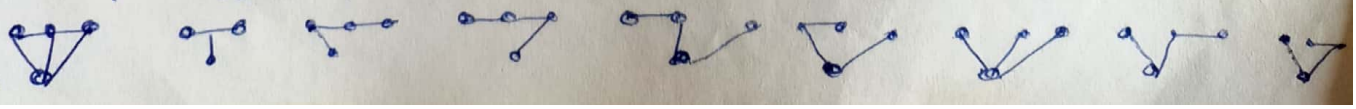
→ Multiple edges are known as parallel
→ when edges begin from same node & end at same node

→ Spanning tree of a graph

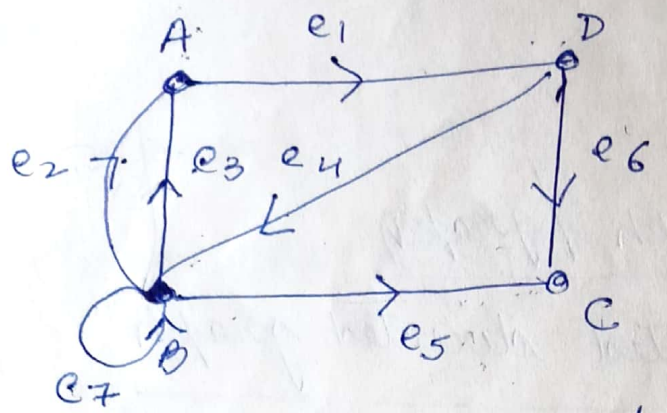
→ Spanning tree has n-1 edges, where n is no. of node in graph.

→ A tree T is called a spanning tree of a connected graph if T has same nodes as G & all edges of T contained among the edges of G.

e.g. G (graph) — Now T is



Ques:



(i) how many nodes & edges in above graph
(ii) find parallel edges
(iii) find loop
(iv) Is graph is strongly connected or unilaterally connected
(v) find the indegree & outdegree of node D.
(vi) find nodes which are source & sink.

Soln: (i) 4 nodes & 7 edges

(ii) e_3 & e_2 as each begin at B & end at A

(iii) e_7 is loop.

(iv) Graph is unilaterally connected & it is not strongly connected as there is no path from C.

(v) indegree of D = 1
outdegree of D = 2

(vi) Sink is C as indegree(C) = 2
outdegree(C) = 0

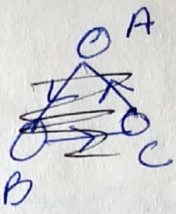
NO source

Diameter of graph

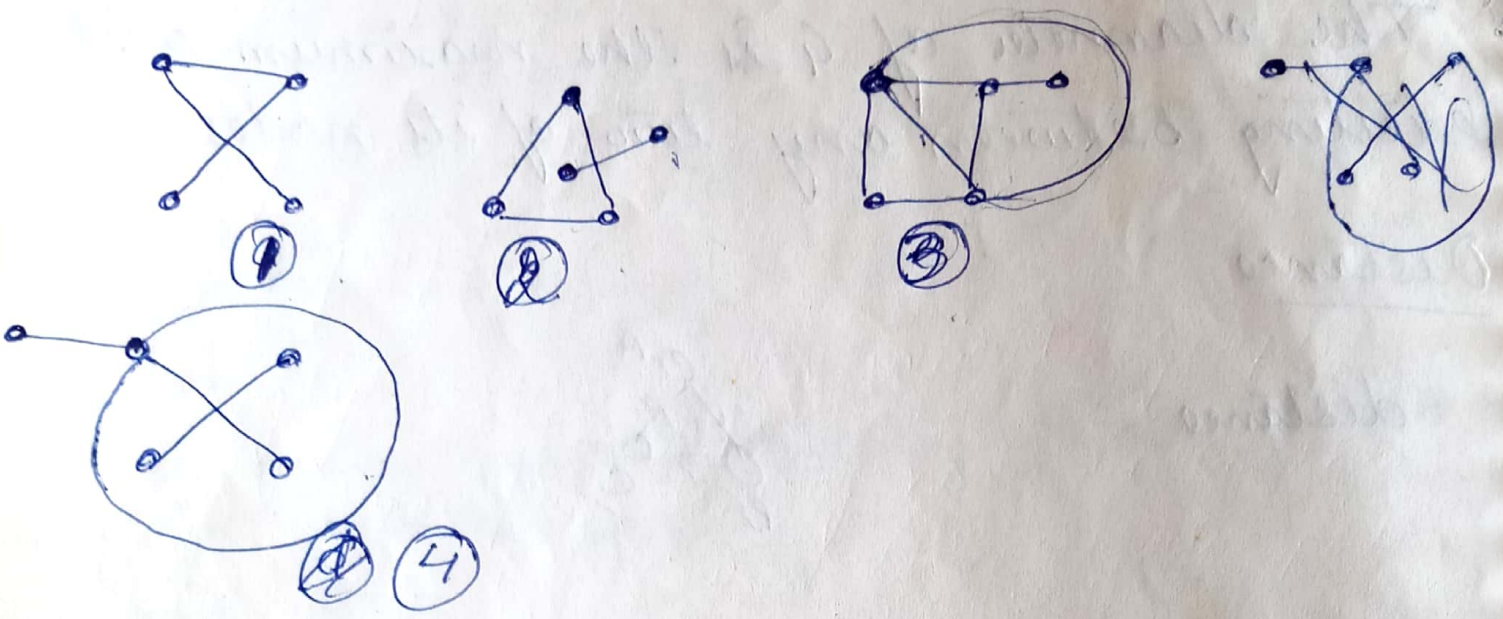
The diameter of G is the maximum distance existing between any two of its nodes

Distance

distance



Ques:



which of them are (a) connected (b) loop-free (i.e. without loops) (c) graphs

Soln: (i) 1, 3 are connected

(ii) 1, 2, 3 are loop-free

4 has loop

(iii) 1, 2 is a graph

where 3 & 4 are not graph because ~~they have~~ 3 has multiple edges & 4 has loop.

Representation of graph in memory

1) Sequential representation / adjacency matrix
2) Linked representation.

Adjacency Matrix

Suppose G is a simple directed graph with m nodes & suppose the nodes of G have been ordered & are called $v_1, v_2 \dots v_m$. Then the adjacency matrix $A = (a_{ij})$ of the graph G is the $m \times m$ matrix defined as follows:

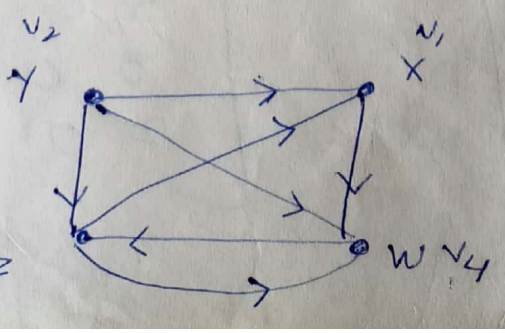
$$a_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is adjacent to } v_j, \text{ that is there is edge } (v_i, v_j) \\ 0 & \text{otherwise} \end{cases}$$

Such a matrix A, which contain entries of only 0 & 1 is called a bit matrix or a boolean matrix.

Eg: Consider a graph G, as shown in fig. Suppose the nodes are stored in memory in a linear array DATA as follows:

DATA : X, Y, Z, W

Ordering of node is as follows $v_1 = X$, $v_2 = Y$, $v_3 = Z$, $v_4 = W$.



Sol:

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

∴ In this no. of 1's in A is equal to no. of edges in G.

$A^1 \rightarrow$ no. of path of length 1 from v_i node to v_j node

$A^2 \rightarrow$ no. of path of length 2 " " " " ",

$A^3 \rightarrow$ no. of path of length 3 " " " ",

$$A^2 = \begin{array}{c} \\ v_2 \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{cccc} & & & v_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

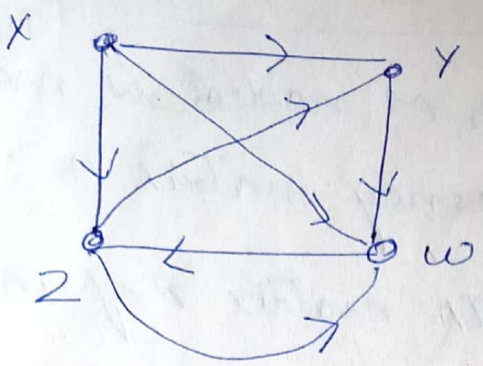
$$A^3 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

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

$$B = A^1 + A^2 + A^3 + A^4$$

$$P = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

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



$$A = \begin{array}{c} \\ X \\ Y \\ Z \\ W \end{array} \begin{array}{c} \begin{array}{cccc} X & Y & Z & W \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

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

$$A^4 = \begin{bmatrix} 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad B_4 = \begin{bmatrix} 0 & 5 & 6 & 8 \\ 0 & 1 & 2 & 3 \\ 0 & 3 & 3 & 5 \\ 0 & 2 & 3 & 5 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

Warshall's Algo: Shortest Paths

A directed graph G with M nodes is maintained in memory by its adjacency matrix A. This algo finds the (Boolean) path matrix P of the graph G.

(1) Repeat for $I, J = 1, 2, \ldots M$ [Initialize P]
  If $A[I, J] = 0$, then set $P_0[I, J] = 0$
  else set $P_0[I, J] = 1$
  end of loop.

(2) Repeat steps 3 & 4 for $K = 1, 2, \ldots M$ [Updated P]

(3) Repeat step 4 for $I = 1, 2, \ldots M$

(4) Repeat step for $J = 1, 2, \ldots M$

  set $P_K[I, J] = P_{K-1}[I, J] \vee (P_{K-1}[I, K] \wedge P_{K-1}[K, J])$

  [end of loop]

 [end of step 3 loop]

[end of step 2 loop]

(5) Exit.

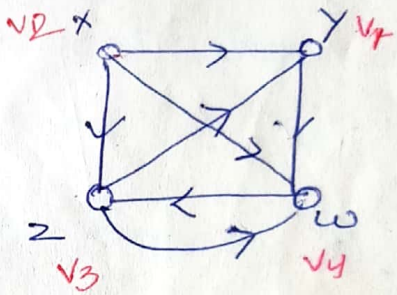
AND
$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
1×1

OR
$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 0 = 1$

$$P_1 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$P_3 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \qquad P_4 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$$P_0 = \begin{array}{c} \\ x \\ y \\ z \\ w \end{array} \begin{array}{c} \begin{array}{cccc} x & y & z & w \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$



$$P_1[1,1] = P_0[1,1] \underset{OR}{\vee} (P_0[1,1] \wedge P_0[1,1])$$

$$= 0 + (0 \times 0)$$

$$= 0$$

$$P_1[1,2] = P_0[1,2] + (P_0[1,1] \times P_0[1,2])$$

$$= 1 + 0 \times 1$$

$$= 1$$

$K = 1, 2, I = 1, J = 1$

$$P_1[1,1] = P_0[1,1] \vee (P_0[1,1] \wedge P_0[1,1])$$

$$= 0 \vee (0 \times 0)$$

$$= 0$$

Shortest Path algo → FLOYD WARSHALL

→ Suppose G is weighted, i.e. suppose each edge e in G is assigned a non-negative number w(e) called the weight or length of the edge e. Then G may be maintained in memory by its weight matrix $W = (w_{ij})$

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

Path matrix P → tells whether or not there are paths b/w the nodes.

$Q = q_{ij}$ → length of a shortest path from v_i to v_j

$Q_k[i,j]$ → the smaller of the lengths of the preceding path from v_i to v_j.

e.g.

Algo

(1) Repeat for $I, J = 1, 2 \dots M$: [Initializes Q]
$w[i, J] = 0$ then set $Q_0[i, J] = \infty$
else set $Q[i, J] = w[i, J]$
end of loop.

(2) Repeat step 3 & 4 for $K = 1, 2 \dots M$

(3) Repeat step 4 for $I = 1, 2 \dots M$

(4) Repeat for $J = 1, 2, \dots M$

set

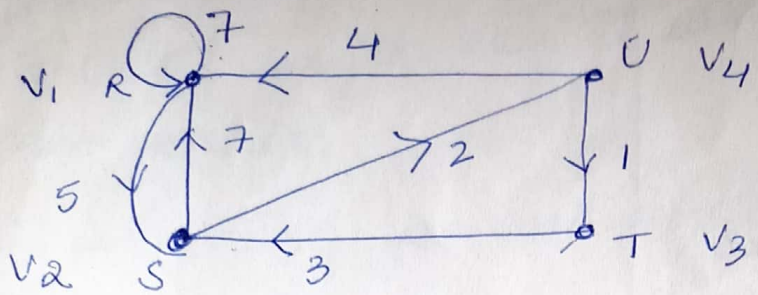
$$Q_K[i, j] = MIN(Q_{K-1}[i, j], Q_{K-1}[i, K] + Q_{K-1}[K, j])$$

end of loop.
end of step 3 loop
end of step 2 loop.

(5) Exit

EXAMPLE

Consider Weighted graph G.



$$W = \begin{array}{c} \\ V_1 \\ V_2 \\ V_3 \\ V_4 \end{array} \begin{array}{c} \begin{array}{cccc} V_1 & V_2 & V_3 & V_4 \end{array} \\ \begin{bmatrix} 7 & 5 & 0 & 0 \\ 7 & 0 & 0 & 2 \\ 0 & 3 & 0 & 0 \\ 4 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad Q_2 = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}$$

$$Q_0 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & \infty & 1 & \infty \end{bmatrix}$$

$$Q_1 = \begin{bmatrix} 7 & 5 & \infty & \infty \\ 7 & 12 & \infty & 2 \\ \infty & 3 & \infty & \infty \\ 4 & 9 & 1 & \infty \end{bmatrix}$$

$K=1, I=1, J=1$

$$Q_1[1,1] = \text{MIN}(Q_0[1,1], Q_0[1,1] + Q_0[1,1])$$
$$= \text{MIN}(7, 7+7)$$
$$= \text{MIN}(7, 14) = 7$$

→ Directed weighted graph
→ Based on Greedy approach and give optimal solution
→ G → graph
w → weight
→ $\pi[v]$ → parent
→ $d[s]$ → distance of source
→ Shortest path from one node to all other nodes in which weight are non-negative
→ Application
— Google Map

Dijkstra's algo

→ It is single source shortest path.

Dijkstra's algo solve the single-source shortest path problem on a weighted directed graph $G=(V,E)$ for the case in which all edges weight are non-negative

∴ $w(u,v) \geq 0$ for each edge $(u,v) \in E$

Dijkstra (G, w, s)

Initialize-single source (G, s)
$S \leftarrow \phi$
$Q \leftarrow V[G]$
while $Q \neq \phi$
do $u \leftarrow$ Extract_min (Q)
$S \leftarrow S \cup \{u\}$ // Highlight the path
for each vertex $v \in adj[u]$
do Relax (u, v, w)

Initialize-Single source (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)

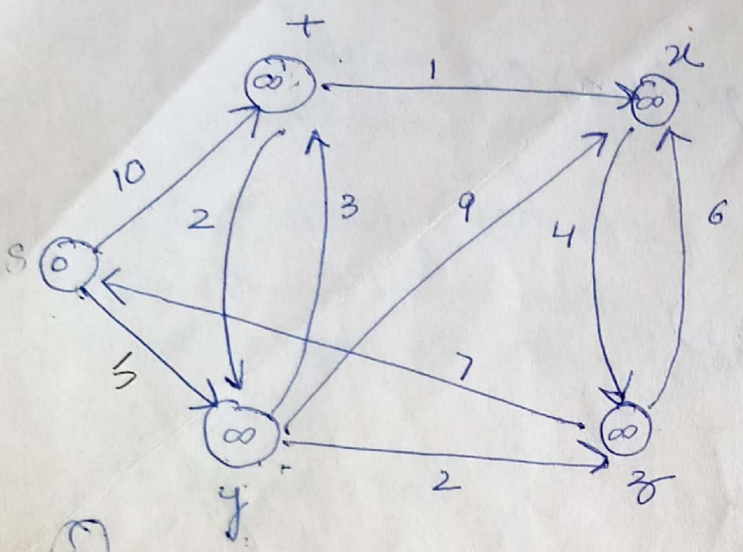
1. if $d[v] > d[u] + w(u,v)$
2. then $d[v] \leftarrow d[u] + w(u,v)$
3. $\pi[v] \leftarrow u$

where:

$\pi[v]$ → predecessor vertex, if vertex v not discovered then $\pi[v] = NIL$
OR PARENT OF v

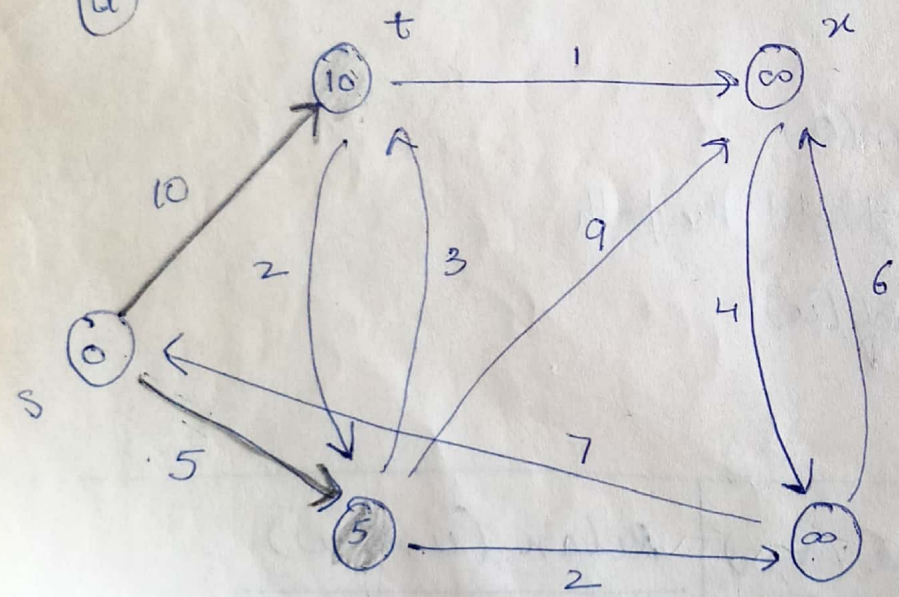
$d[u]$ → it store distance from source to u.

Example:



Q: s, t, y, x, z

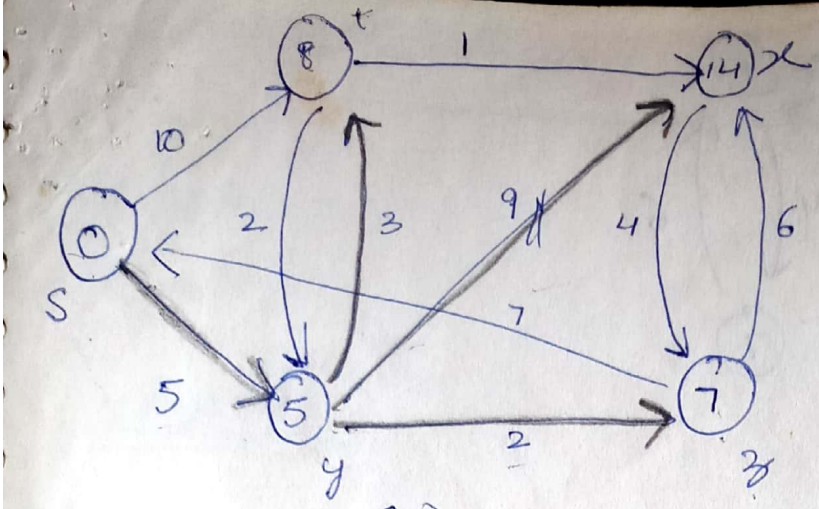
(a)



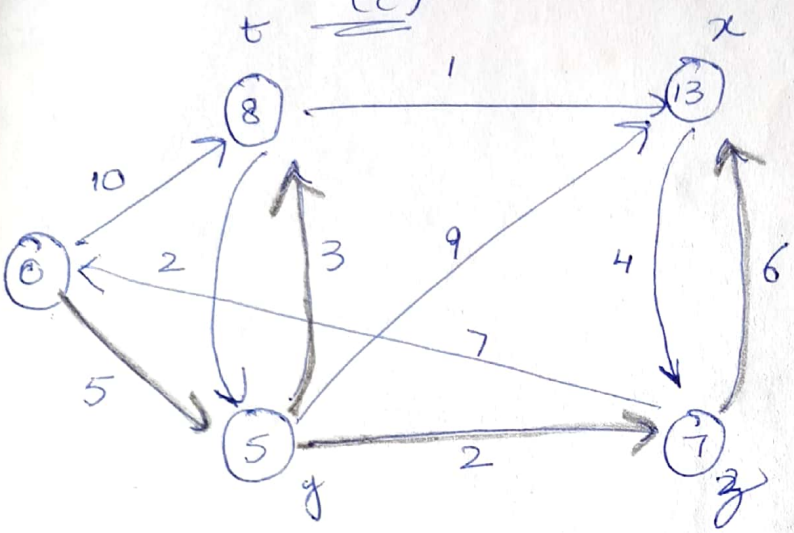
Q: t, y, x, z

Weight Matrix

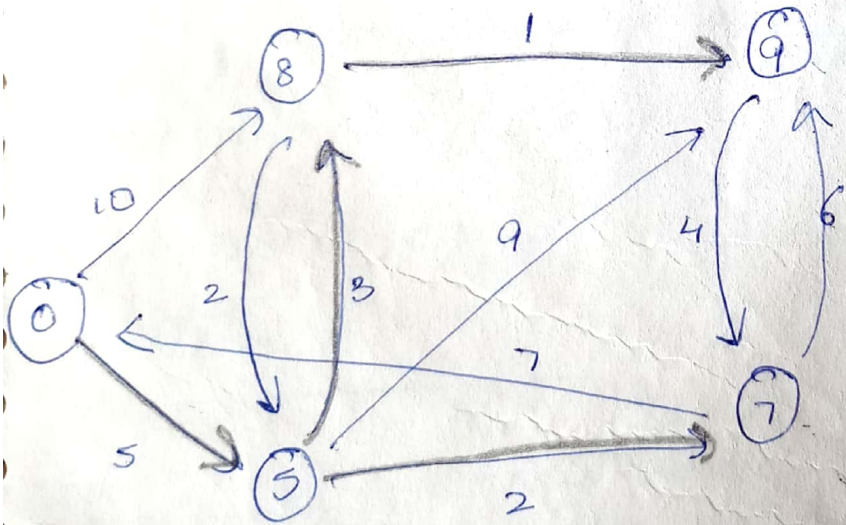
	s	t	y	x	z
s	0	∞	∞	∞	∞
s	0	10	5	∞	∞
	0				



(c)

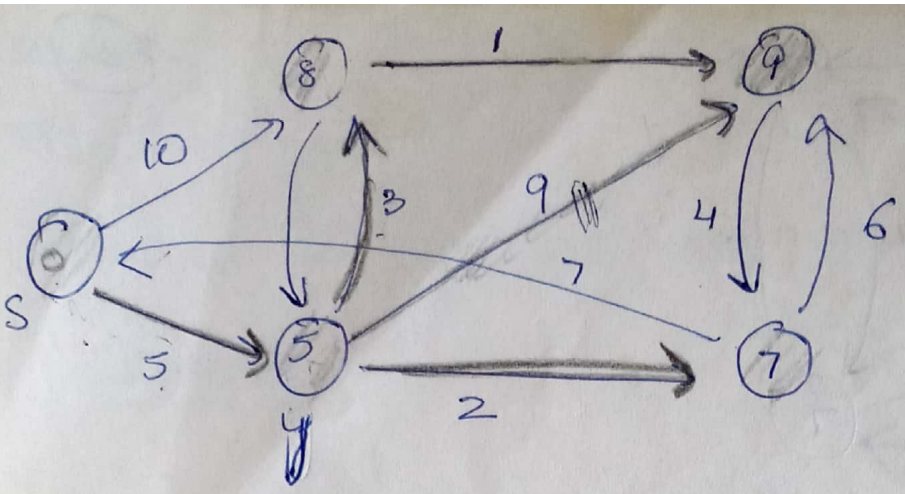


d



e

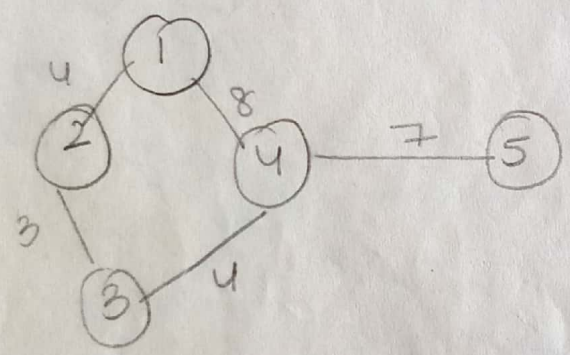
	s	t	x	y	z
s	0	10	∞	5	∞
y					



(y)

~~Order of~~ $O(V^2)$.

Example



VISITED	1	2	3	4	5
	∞	∞	∞	∞	∞
{1}	0	4	∞	8	∞
{1,2}	0		7		

Minimum Spanning tree

1) KRUSKAL
2) PRIMS.

KRUSKAL

Algo

1) $A \leftarrow \phi$
2) for each vertex $v \in V[G]$
3) do Make-Set(v)
4) sort the edges of E in ~~dec~~ increasing order by weight w
5) for each edge $(u, v) \in E$, taken in increasing order by weight
6) do if find_set(u) $\neq$ find_set(v)
7) then $A \leftarrow A \cup \{(u, v)\}$
8) UNION(u, v)
9) return A.

Make_Set(x) → creates a new set whose only member (& thus representative) is x. Since the sets are disjoint we require that x not already be in some other set.

Union(x, y) → unites the dynamic sets that contain x & y, say S_x & S_y into a new set that is the union of these two sets. i.e. $S_x \cup S_y$.

find_set(x) → returns a pointer to the representa. of the (unique) set containing x.

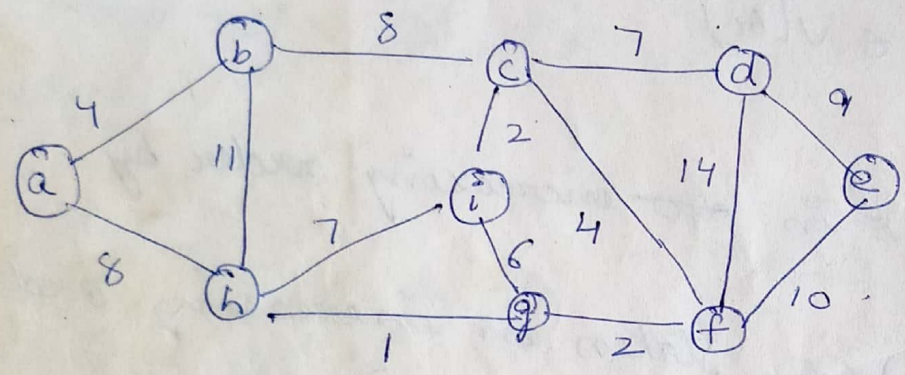
Complexity →

→ $O(E \lg V)$ [when $E < V$]

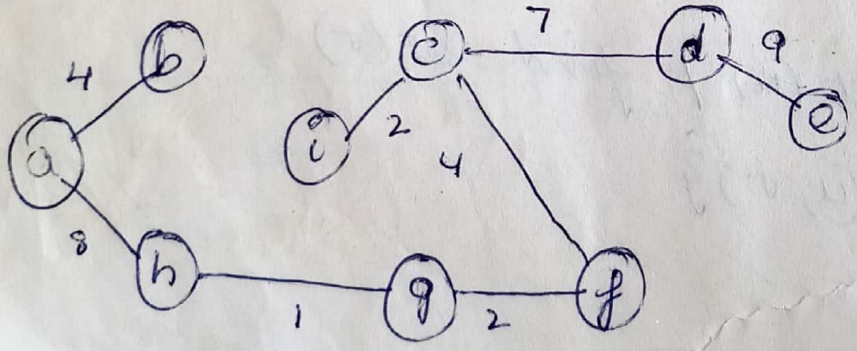
$O(E \lg E)$ [when $E = V$].

E → Edges
V → vertex

EXAMPLE :



(i)



PRIMS

PRIMS (G, w, r)

1. for each $u \in V[G]$
2. do Key[u] ← ∞
3. $\pi[u]$ ← NIL
4. Key[r] ← 0
5. ~~while Q ≠ ∅~~
5. Q ← V[G]
6. while Q ≠ ∅
7. do u ← Extract_min(Q)
8. for each $v \in adj[u]$
9. do if $v \in Q$ & $w(u,v) <$ Key[v]
10. then $\pi[v] \leftarrow u$
11. Key[v] ← w(u, v).

complexity → $O(E \lg V)$

OR

$(V \lg V + E \lg V)$]

→ Prims is fastest

TRAVERSING A GRAPH

Traversing of graph can be done:

(i) Breadth first search → using queue

(ii) Depth first search → using stack

→ During execution of algo, each node N of G will be in one of three status of N as follows:

STATUS = 1: (Ready state) The initial state of the node N

STATUS = 2: (waiting state) The node in stack or queue waiting ~~for~~ to be processed.

STATUS = 3: (Processed state) The node N has been processed.

(1) BFS (Breadth first search)

→ In this first starting node processed then neighbours of that node is examined, then examine neighbours of neighbours of that ~~of~~ starting node.

→ In this Queue is used

Algo executes BFS.

(1) Initialize all node to the ready state (status = 1)

(2) Put the starting node A in Queue & change its status to the waiting state (status = 2)

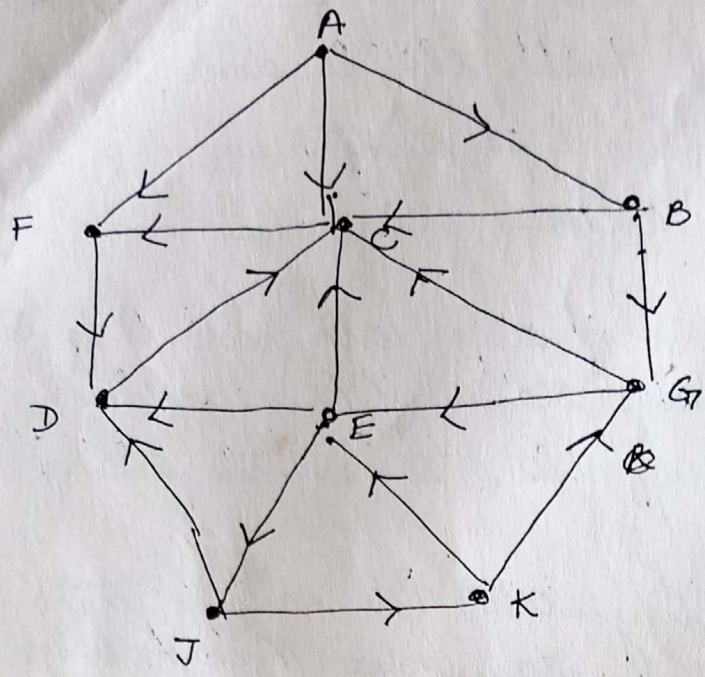
(3) Repeat step 4 and 5 until Queue is empty.

(4) Remove the front node N of Queue. Process N and change the status of N to the processed state (STATUS = 3)

(5) Add to the rear of Queue all the neighbours of N that are in the steady state (STATUS = 1) & change their status to the waiting state (STATUS = 2)

[End of step 3 loop]

Ques: Find path P from A to J



ADJACENCY LIST

A : F, C, B
B : G, C
C : F
D : C
E : D, C, J
F : D
G : C, E
J : D, K
K : E, G

Q

STATUS			
A	1	2	
B	1	2	3
C	1	2	3
D	1	2	
E	1	2	
F	1	2	3
G	1	2	
J	1	2	
K			

(i) Q : A
O : ϕ

F = 0
R = 0

(ii) Q : A, F, C, B
O : ϕ, A, A, A

F = 1
R = 3

(iii) Q : A, F, C, B, D
O : ϕ, A, A, A, F

F = 2
R = 4

(iv) QUEUE : A, F, C, B, D
ORIGIN : ϕ, A, A, A, F

F = 3
R = 4

(v) QUEUE : A, F, C, B, D, G
ORIGIN : ϕ, A, A, A, F, B

F = 4
R = 5

(vi) QUEUE : A, F, C, B, D, G F=5
ORIGIN : ϕ, A, A, A, F, B R=5

(vii) QUEUE : A, F, C, B, D, G, E F=6
ORIGIN : ϕ, A, A, A, F, B, G R=6

(viii) QUEUE : A, F, C, B, D, G, E, J F=7
ORIGIN : ϕ, A, A, A, F, B, G, E R=7

~~J ← G ← B ← A~~

J ← E ← G ← B ← A

(5)

path - your ...

DES

In this stack is used

Algorithm

Initialize all nodes to the ready state (status = 1)
Push the starting node A onto STACK & change its status to the waiting state (status = 2)
Repeat step 4 & 5 until stack empty.
Pop the top Node N of stack. Process N & change its status to the processed state (STATUS = 3)
Push onto STACK all the neighbours of N that are still in the ready state (STATUS = 1) & change their status to the waiting state (STATUS = 2)
[End of step 3 loop]
Exit.

Find reachable nodes from J

STACK : J

PRINT : J STACK : D, K
PRINT : K STACK : D, E, G
PRINT : G STACK : D, E, C
PRINT : C STACK : D, E, F
PRINT : F STACK : D, E, E
PRINT : E STACK : D
PRINT : D STACK :

∴ Reachable nodes from J are
J, K, G, C, F, E, D

(1)