



Data Structure



Unit -V : Graphs

In
ONE SHOT



By. Dr. Kapil Kumar
M.Tech .(CSE). Ph.d (CSE)

V	<p>Graphs: Terminology used with Graph, Data Structure for Graph Representations: Adjacency Matrices, Adjacency List, Adjacency. Graph Traversal: Depth First Search and Breadth First Search, Connected Component, Spanning Trees, Minimum Cost Spanning Trees: Prims and Kruskal algorithm. Transitive Closure and Shortest Path algorithm: Warshal Algorithm and Dijikstra Algorithm.</p>	08
---	---	----

Topic 1 :- Graph Terminology & Graph representation - 2 marks

Topic 2 :- BFS & DFS : Graph Traversal

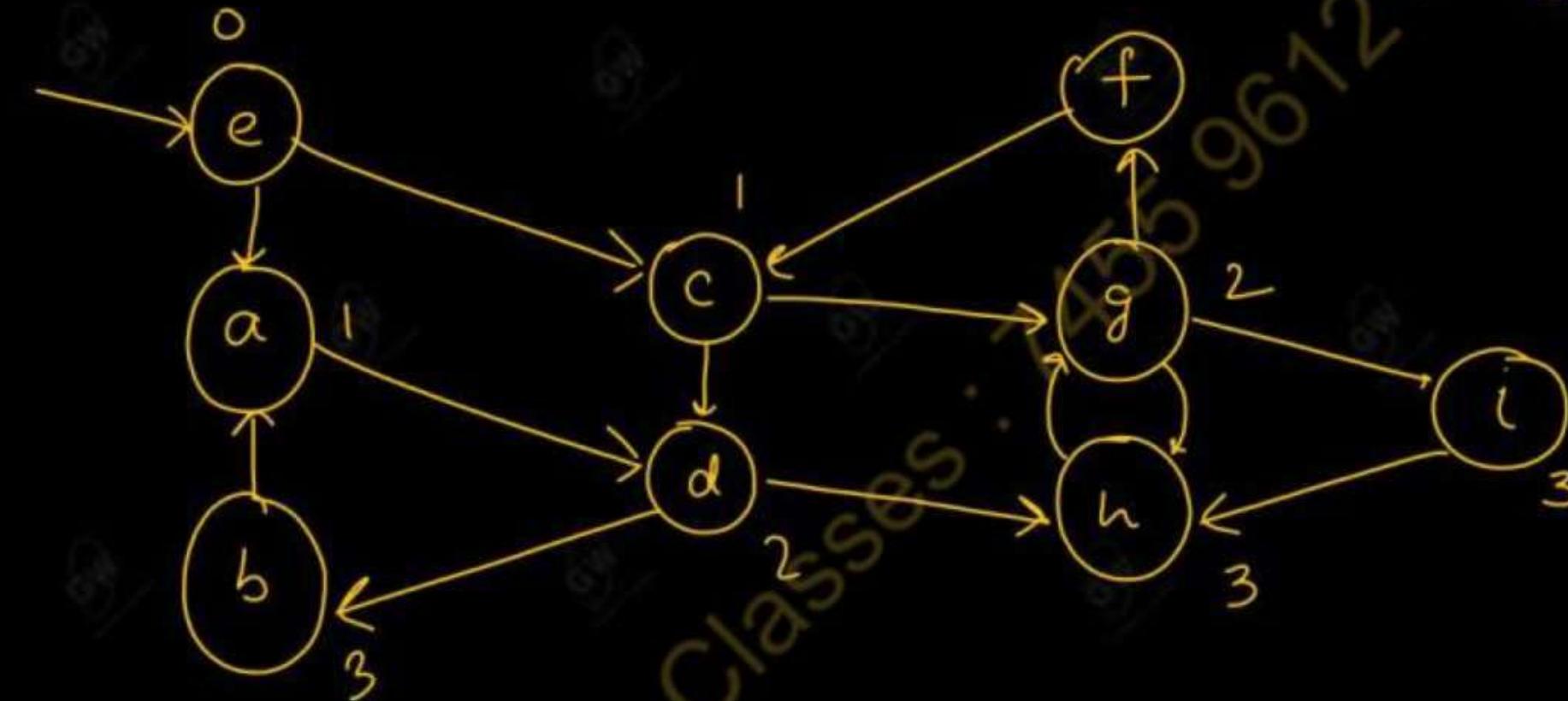
Topic 3 :- Prim's & Kruskal's algo., - minimum cost spanning

Topic 4 :- Shortest Path :- Dijkstra's Algo | Warshall's Algo

Topic 5 :- Transitive closure

Q.1 For the given Graph, give adjacency list, storage representation for adjacency list and edge list.

2014-15, 5 Marks



Q.2 Define connected and strongly connected graph.

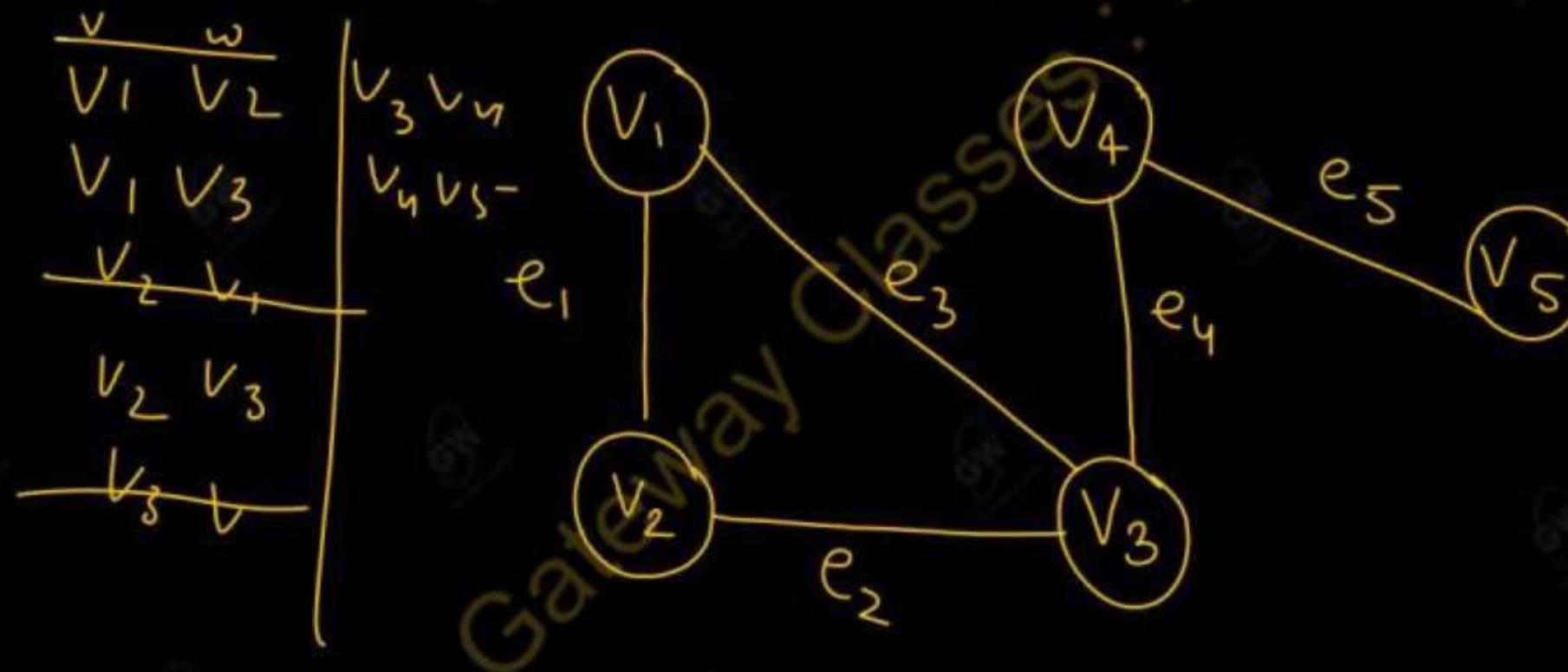
2015-16, 2 Marks

Graph Data Structure

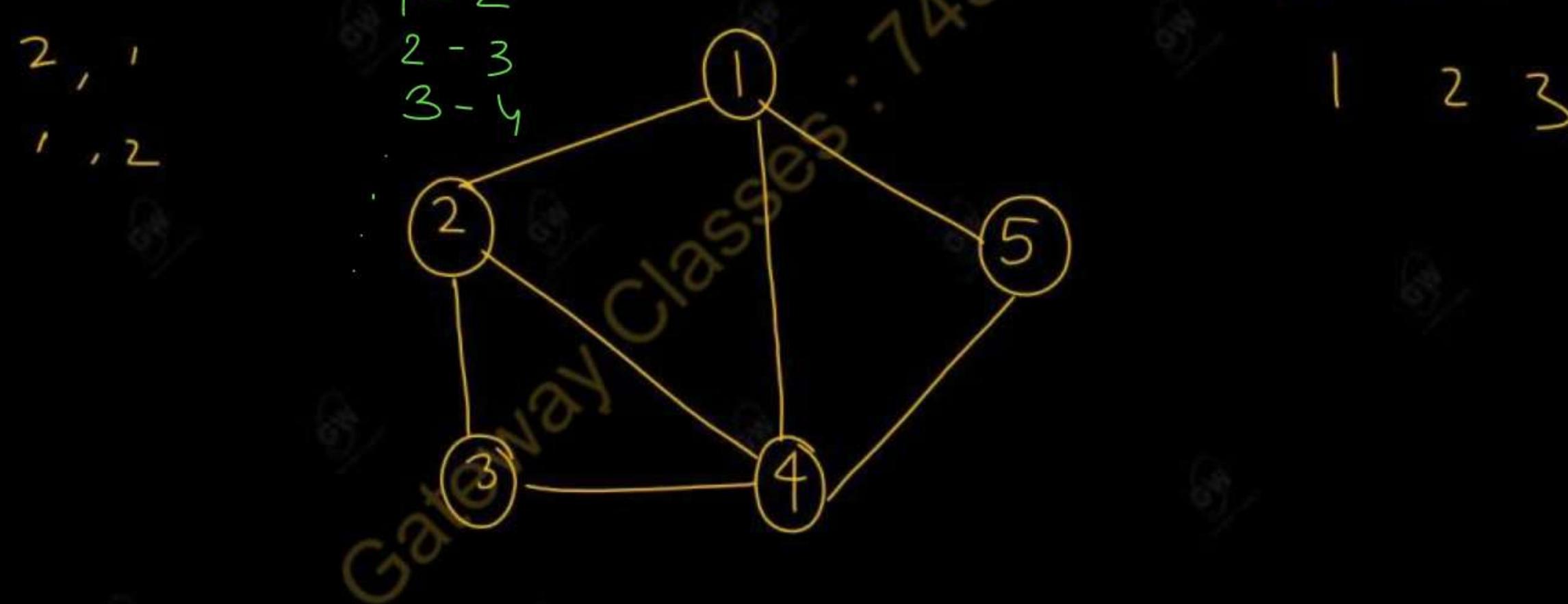
- A graph is another important non - linear data structure.
- We know that tree is also a non-linear data structure used to represent hierarchical relationship between parent and children, i.e. one parent and many children.
- In a graph, the relationship is from many parent to many children.
- Graphs are data structures which have wide-ranging applications in real life like airlines system, analysis of electrical circuits, source - destination networks, finding shortest routes, flow chart of a program, statistical analysis etc.
- They are also widely used in solving games and puzzles.

What is a Graph?

- A graph G consists of a set V of vertices (nodes) and a set E of edges (arcs).
- We write $\underline{G = (V, E)}$, V is a finite and non empty set of vertices.
- $V(G)$ read as V of G , is a set of vertices and $E(G)$ read as E of G , is a set of edges.
- An edge $e = \underline{(v, w)}$ is a pair of vertices v and w and is said to be incident with v and w .
- The following figure shows a sample graph:



- For this graph, set of vertices $V(G) = \{v_1, v_2, v_3, v_4, v_5\}$ and set of edges $E(G) = \{e_1, e_2, e_3, e_4, e_5\}$
- i.e., there are five vertices and five edges in the previous graph.
- A graph may also be pictorially represented as:

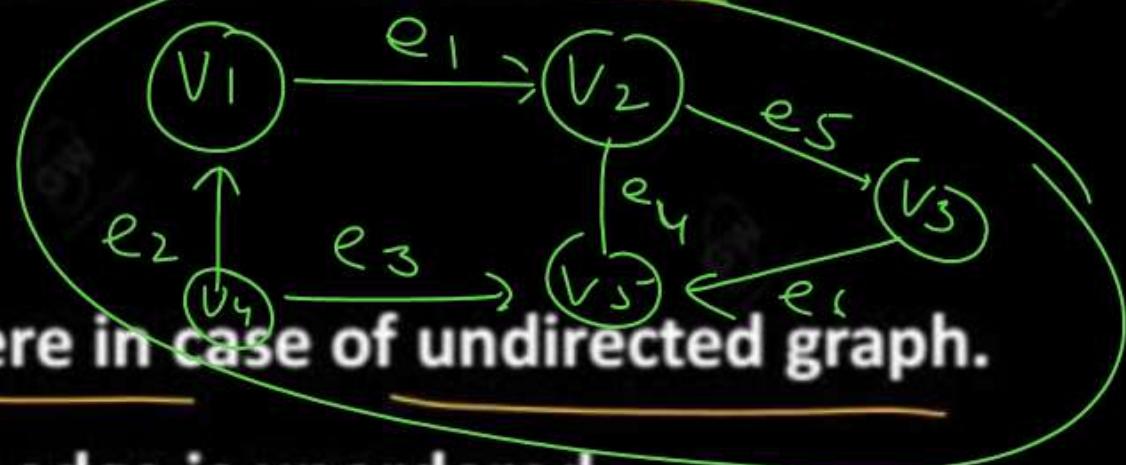


- We have numbered the nodes as 1, 2, 3, 4 and 5. Therefore

Set of vertices $V(G) = \{1, 2, 3, 4, 5\}$ and

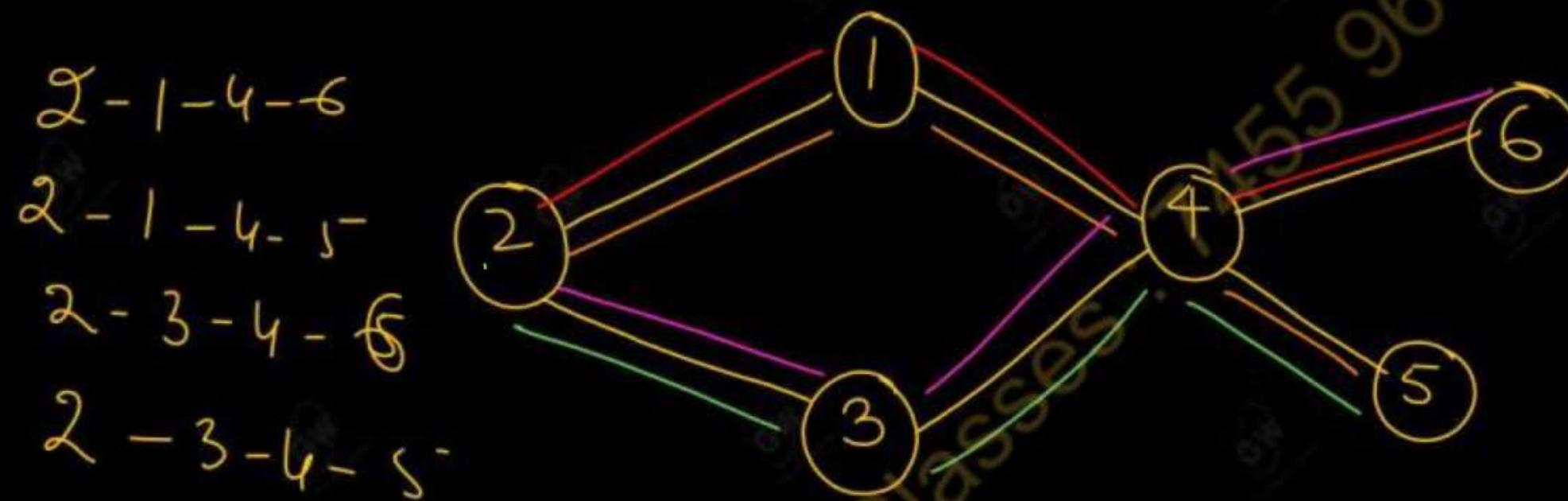
Set of edges $E(G) = \{(1, 2), (2, 4), (2, 3), (1, 4), (1, 5), (4, 5), (3, 4)\}$

- It is to be noted that the edge incident with node 1 and node 2 is written as (1, 2); it can also be written as (2, 1) instead of (1, 2).
- This is applicable for all other edges.
- We may say that ordering of vertices is not significant here in case of undirected graph.
- In an undirected graph, pair of vertices representing any edge is unordered.
- Thus (v, w) and (w, v) represent the same edge.
- In the directed graph each edge is an ordered pair of vertices i.e., each edge is represented by a directed pair.



BASIC TERMINOLOGY

- **Adjacent Vertices:-** Vertex v_1 is said to be adjacent to a vertex v_2 if there is an edge (v_1, v_2) or (v_2, v_1) . Consider the following graph: (Undirected graph)



Handwritten paths:

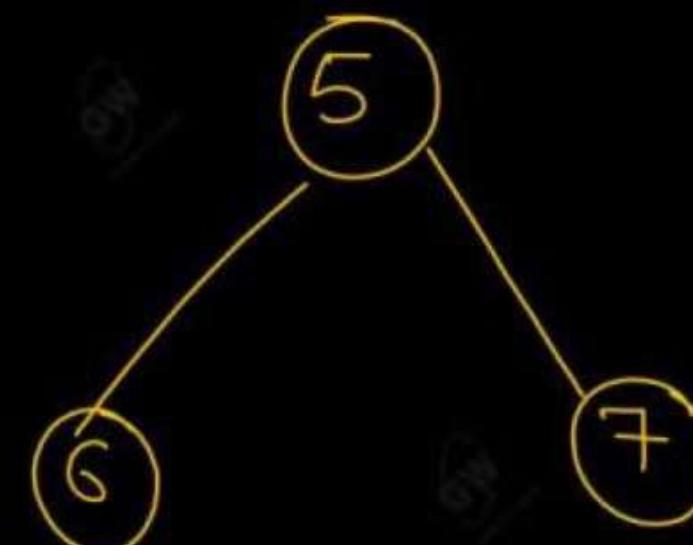
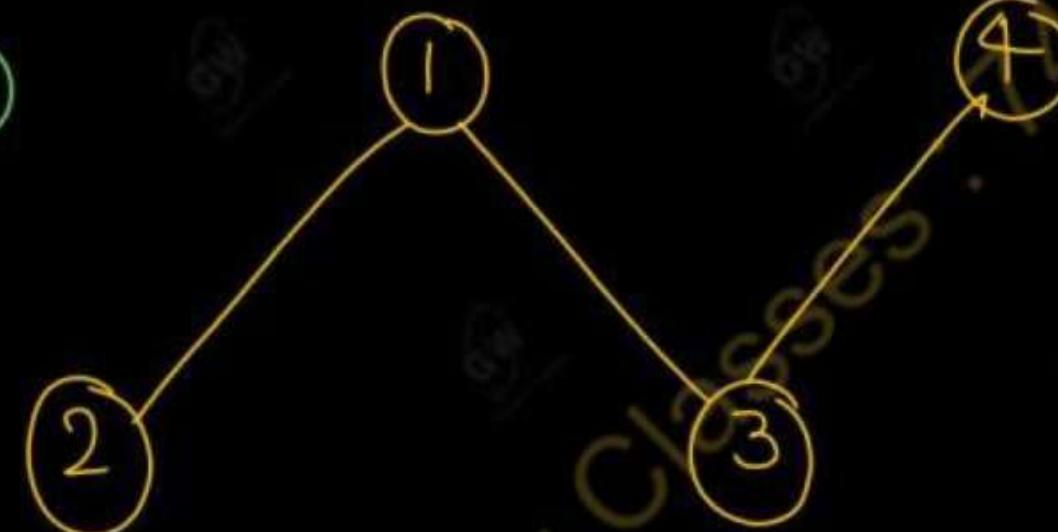
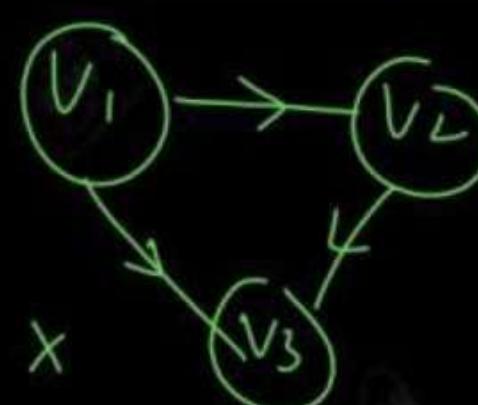
- 2 - 1 - 4 - 3 - 2
- 1 - 2 - 3 - 6 - 1

Vertices adjacent to node 2 are 1 and 3 and that to node 4 are 1, 3, 6 and 5.

- **Path:-** A path from vertex w is a sequence of vertices, each adjacent to the next. Consider the above example again 1, 2, 3 is a path and 1, 4 and 3 is also a path.

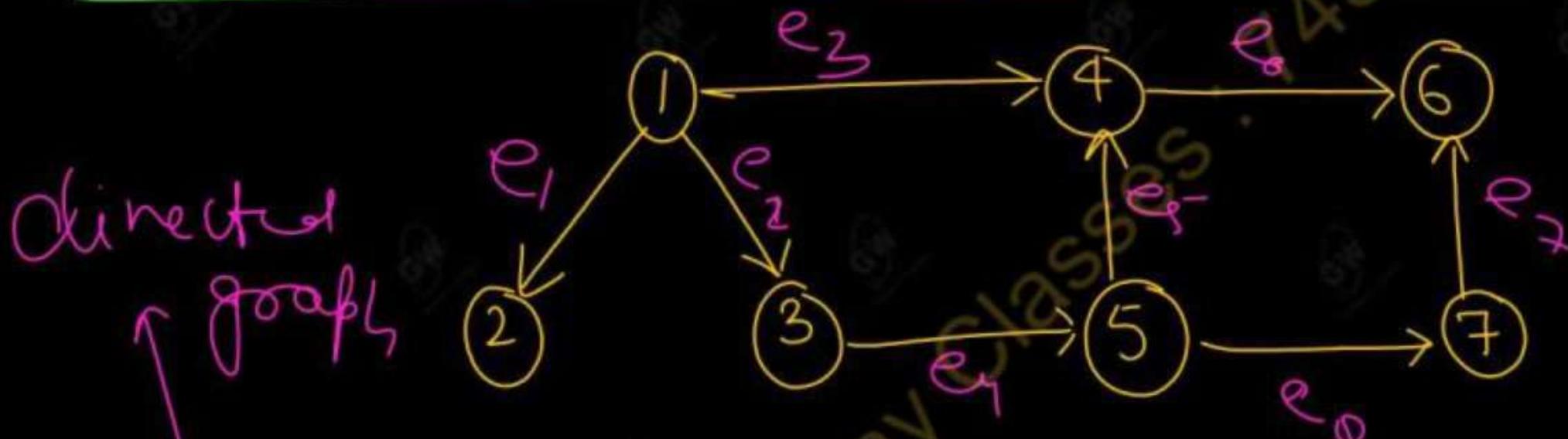
□ **Cycle**:- A cycle is a path in which first and last vertices are the same. In the above example $(1, 2, 3, 4, 1)$ is a cycle.

□ **Connected graph**:- A graph is called connected if there exists a path from any vertex to any other vertex. Consider the following unconnected graphs:



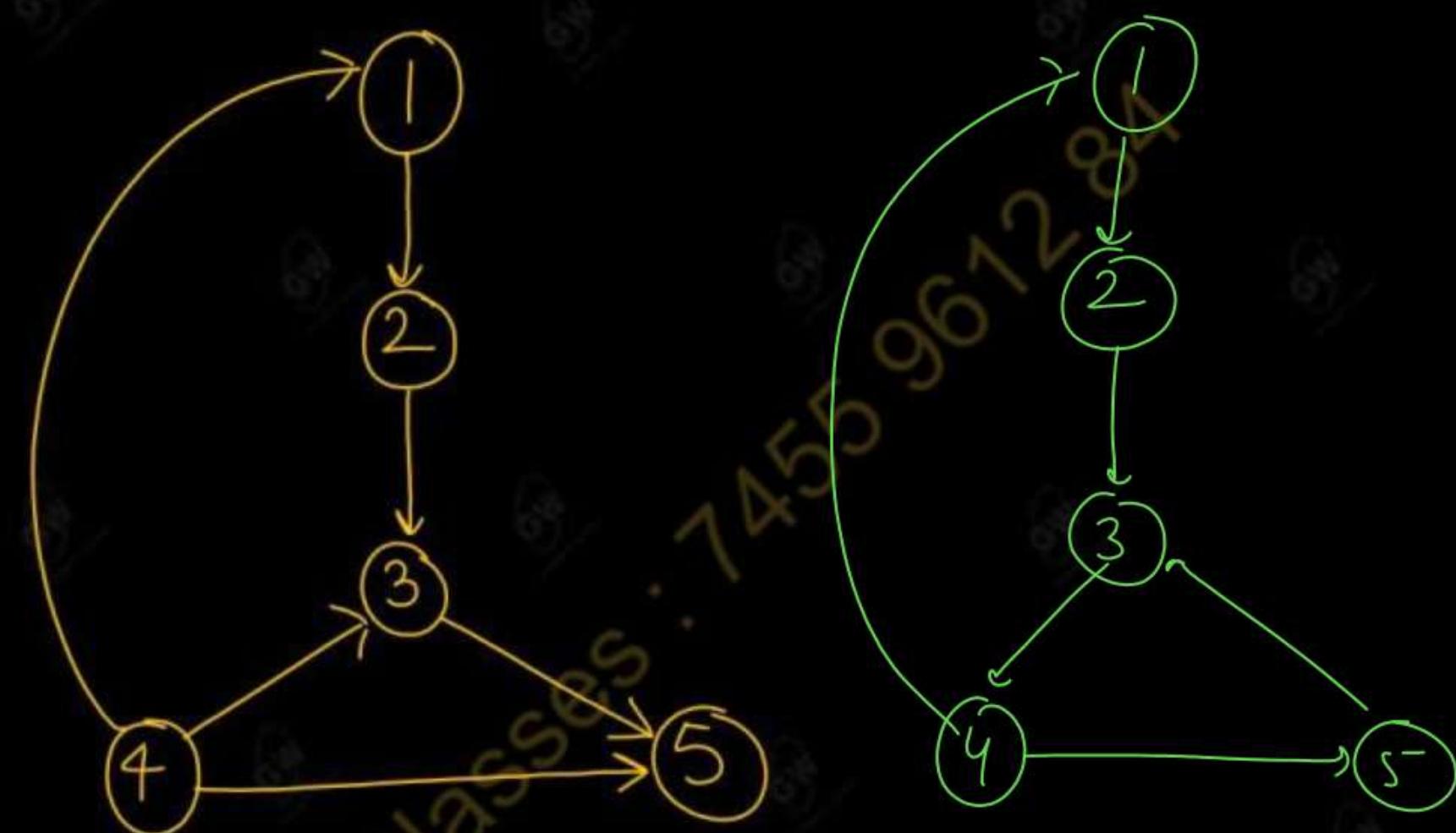
□ It is an unconnected graph. You may say that these are two graphs and not one. But it is one graph having two unconnected components. Since there are unconnected components, it is an unconnected graph.

□ So far we have talked about paths, cycles and connectivity of undirected graph. In a digraph the path is called a **directed path** and cycle is called as **directed cycle**.



□ There is no directed cycle in the above graph. You may verify the above statements. A **digraph is called strongly connected if there is a directed path from any vertex to any other vertex.**

□ Consider the following directed graph:

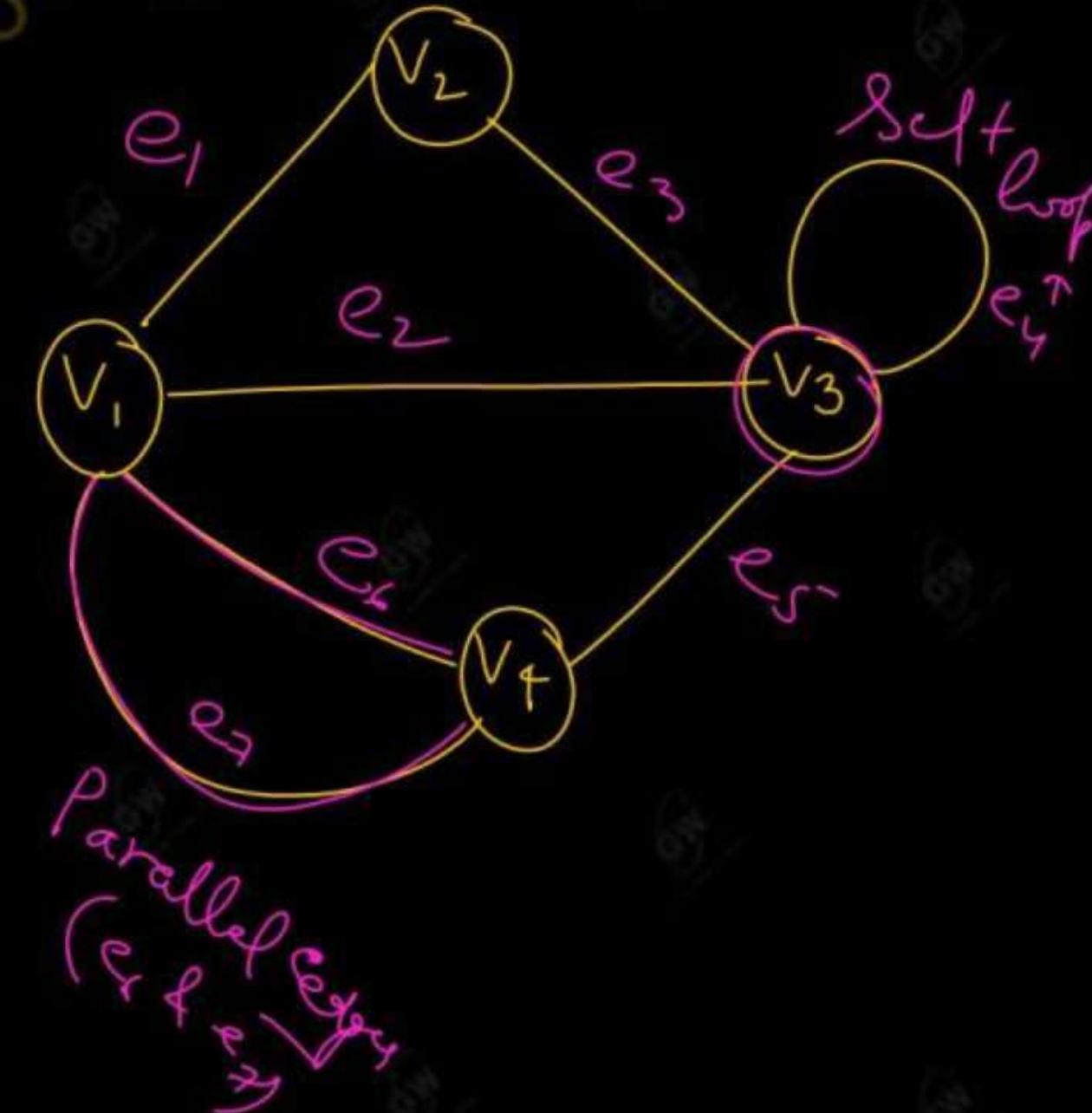
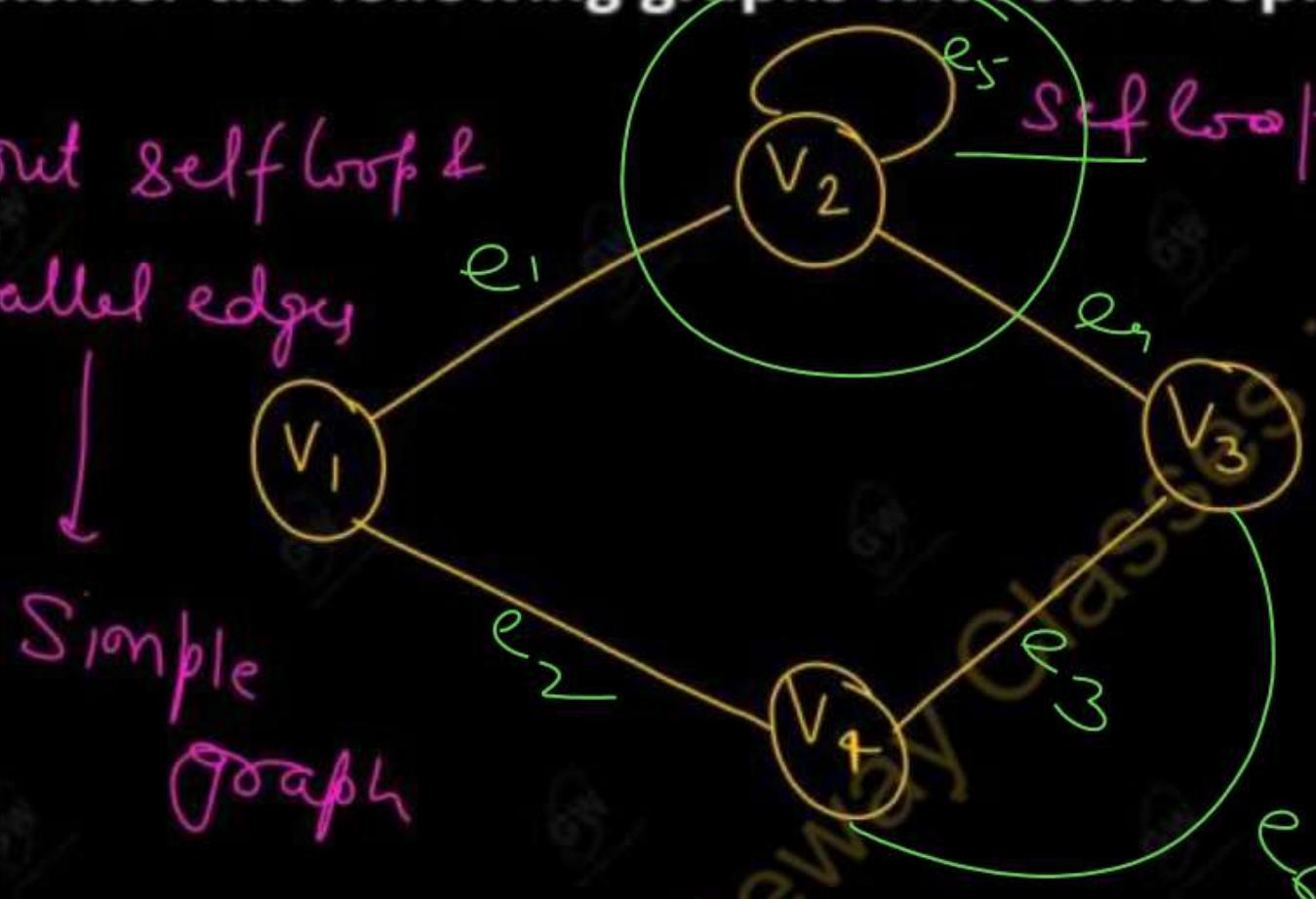


- There does not exist a directed path from vertex 1 to vertex 4, also from vertex 5 to any other vertices and so on.
- Therefore, it is a weakly connected graph. To make it strongly connected there are certain changes that are required given as in the next graph.

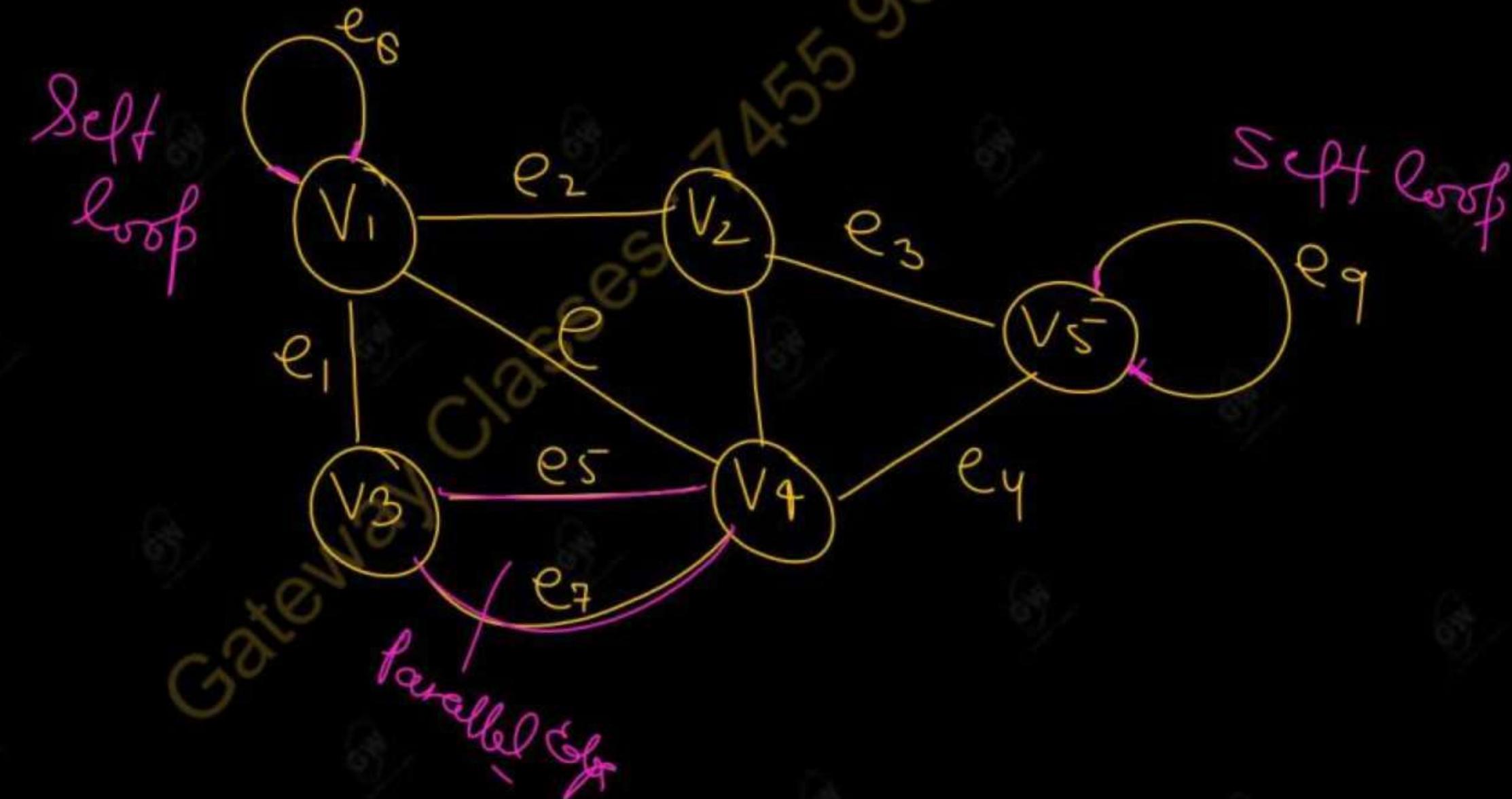
- If there is an edge whose starting and end vertices are same that is (v, v) is an edge then it is called a self loop or simply a loop.

- Consider the following graphs with self loop:

without self loop &
parallel edges



- If there are more than one edge between the same pair of vertices then they are known as Parallel edges.
- Consider the following graphs with self loop and parallel edges:



- A graph which has either a self loop or parallel edges or both is called a multi-graph.
- Previous graphs with self loop and parallel edges are multi-graphs.

Simple-graph:-

- A graph or directed graph which does not have self loop and parallel edges is called simple graph.

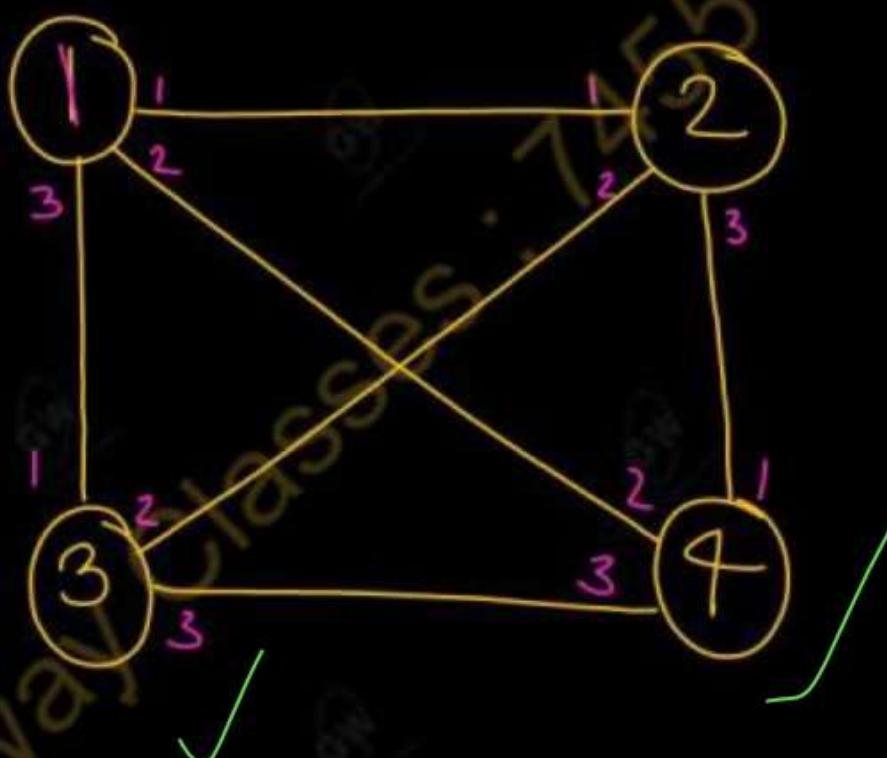
Regular Graph:- (Simple graph)

□ A graph is regular if every node is adjacent to the same number of nodes.

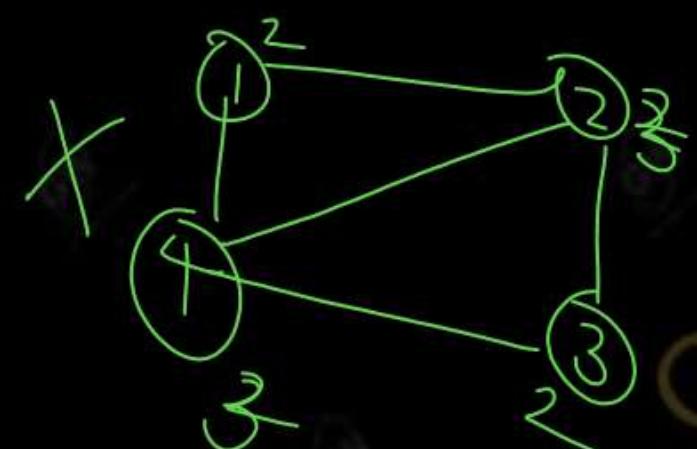
□ Consider the following regular graph:

□ In this graph every node has same degree.

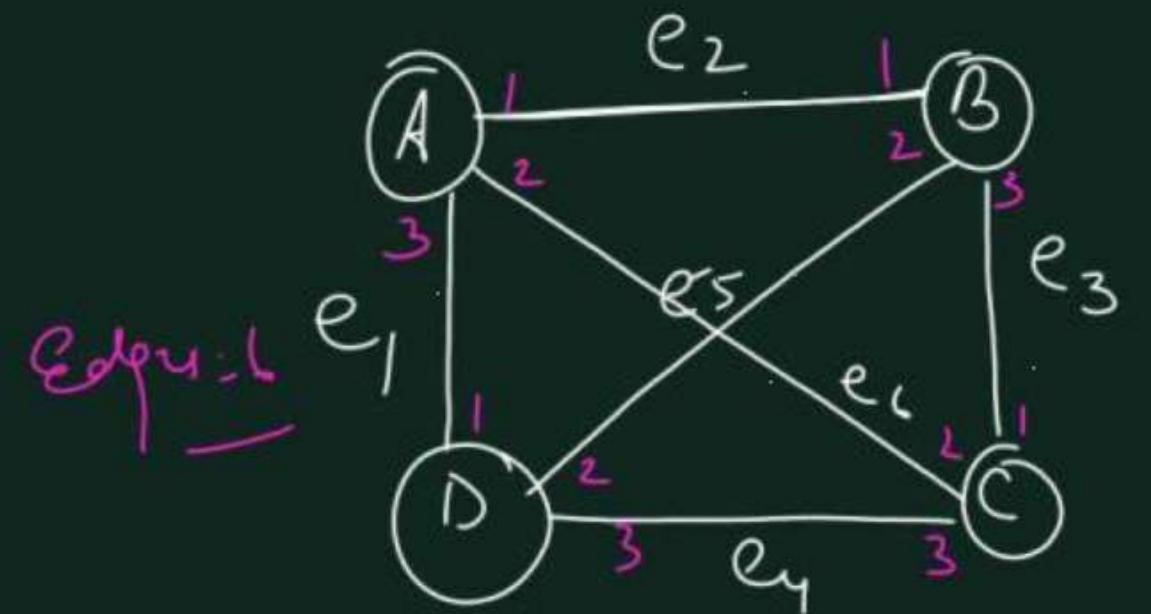
□ This is a simple graph not a multi graph.



$$\begin{aligned} \text{deg}(1) &= 3 & \checkmark \\ \text{deg}(2) &= 3 & \checkmark \\ \text{deg}(3) &= 3 & \checkmark \\ \text{deg}(4) &= 3 & \checkmark \end{aligned}$$



Consider the following graph -



✓ Regular graph

degree of every vertex = 3

no. of Edges = $\frac{n * d}{2}$

$$\begin{matrix} n=4 \\ d=3 \end{matrix}$$

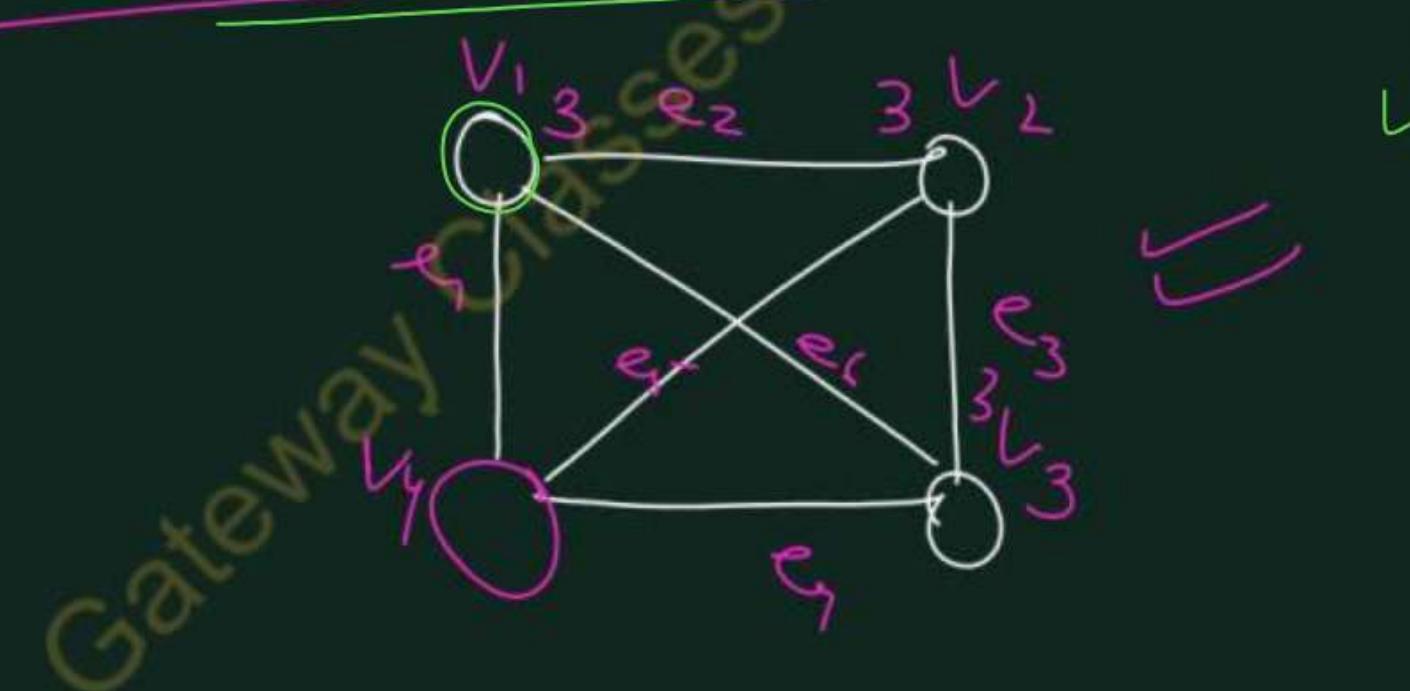
n = number of nodes or vertices

d = degree of a node

$$\text{no. of Edges} = \frac{4 \times 3}{2} = \frac{12}{2} = 6$$

Complete graph :- (Simple)

A simple graph is complete in which every pair of vertices are adjacent i.e. every vertex is connected to other vertex.



Regular + Complete

\Rightarrow In this graph every vertex is connected to other vertices so it is a complete graph.

\Rightarrow eq Degree of each node is same so it is a regular graph.

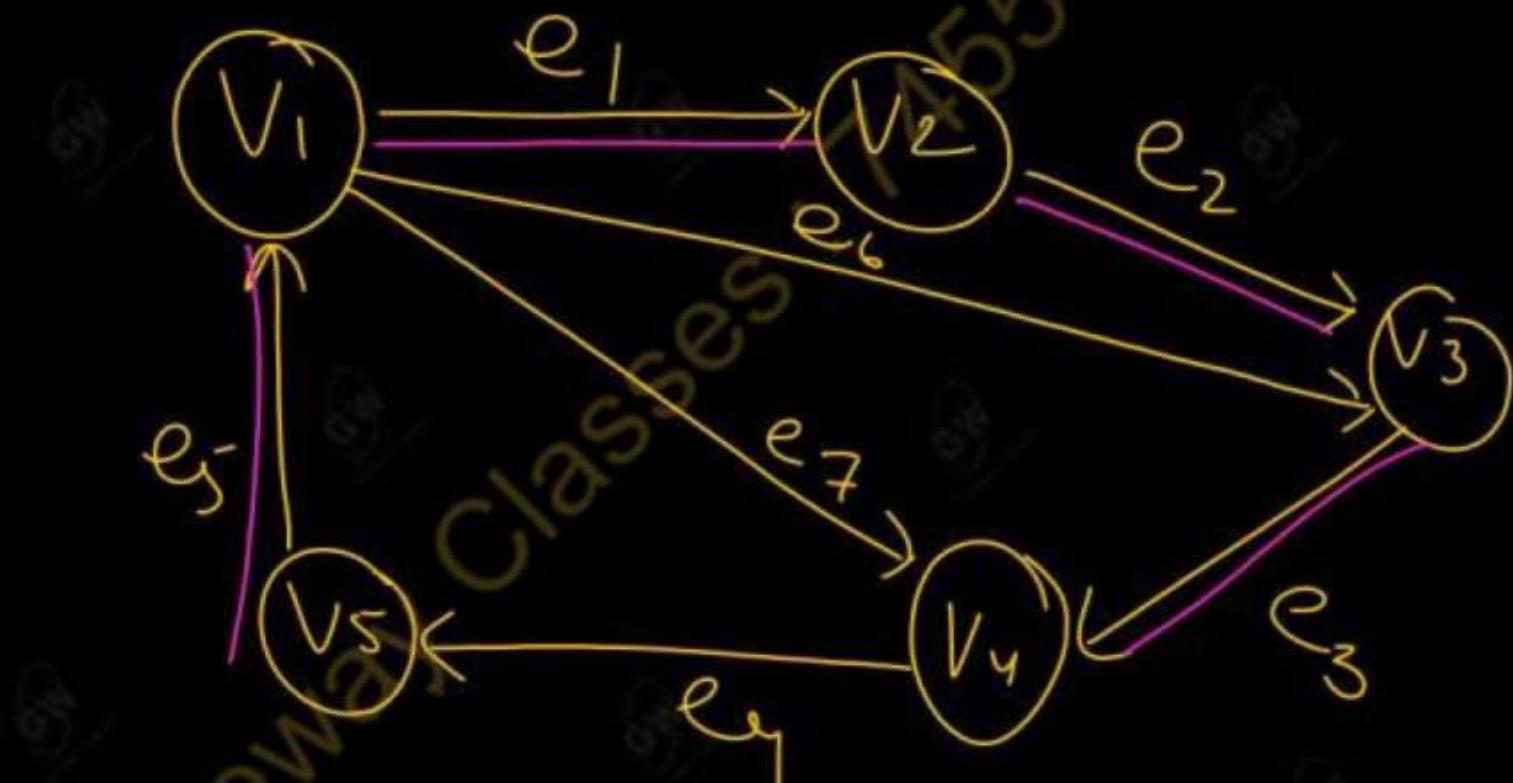
number of edges in complete graph =

$$= \frac{n(n-1)}{2}$$

$$= \frac{4 \times 3}{2} = \frac{12}{2} = 6$$

□ If a graph have cycle then it is called as cyclic graph.

□ Consider the following cyclic graph:



- The number of edges incident on a vertex determine its degree.
- The degree of vertex u , is written as degree (u).
- If degree (u) = 0, this means that vertex u does not belong to any edge, then vertex u is called isolated vertex.
- In a Directed graph (or Digraph), we attach an indegree and an outdegree to each of the vertices.
- Consider the following graph:



$\text{deg}(1) = 1$
 $\text{deg}(2) = 2$
 $\text{deg}(3) = 3$
 $\text{deg}(4) = 2$
 $\text{deg}(5) = 2$

	In	Out
1	X	1
2	1	1
3	2	1
4	X	1
5	2	X

- The indegree of vertex (3) is 2 and outdegree is 1.

$$\text{Indeg}(v_1) = 0$$

$$\text{Outdeg}(v_1) = 2$$

$$\text{Indeg}(v_2) = 1$$

$$\text{Outdeg}(v_2) = 1$$

$$\text{Indeg}(v_3) = 1$$

$$\text{Outdeg}(v_3) = 1$$

$$\text{Indeg}(v_4) = 2$$

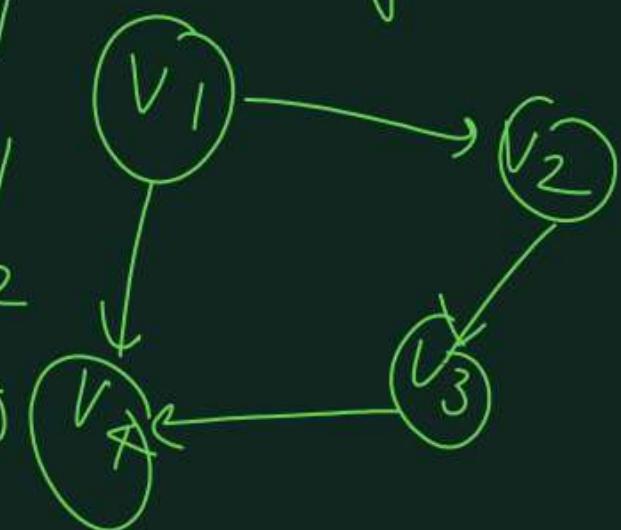
$$\text{Outdeg}(v_4) = 0$$

Directed graph

Digraph

Indegree

Outdegree



Undirected graph

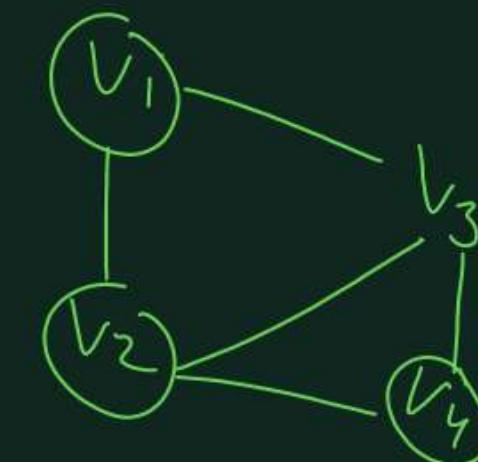
No concept of
In and Outdegree

$$\deg(v_1) = 2$$

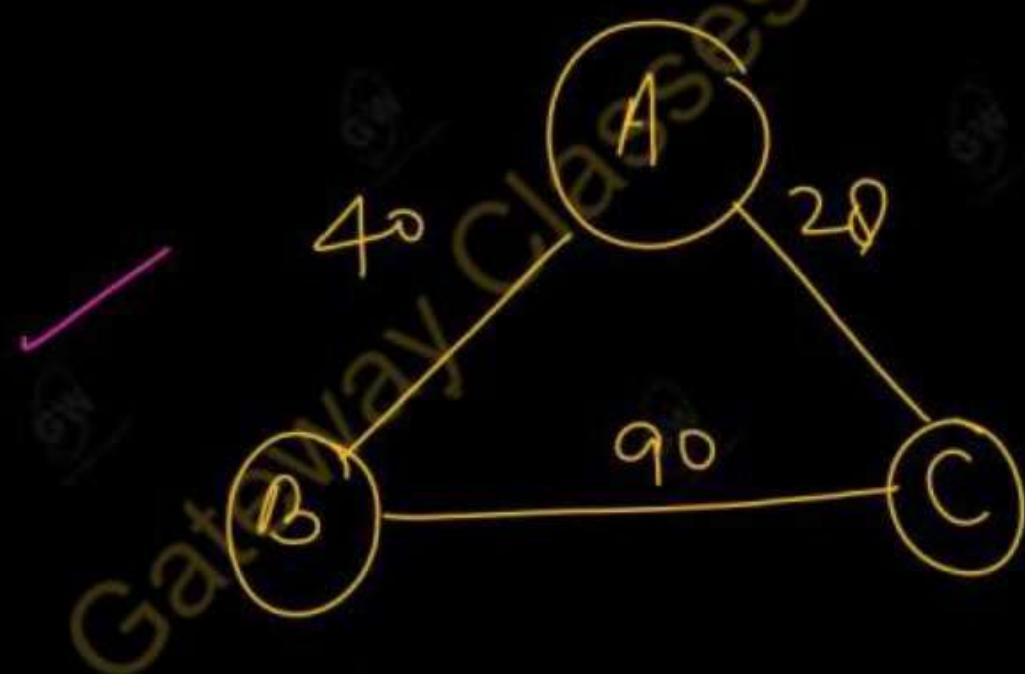
$$\deg(v_2) = 3$$

$$\deg(v_3) = 3$$

$$\deg(v_4) = 2$$



- A graph is said to be weighted graph if every edge in the graph is assigned some weight or value.
- The weight of the edge is a positive value that may be representing the cost of moving along the edge or distance between the vertices.
- Consider the following weighted graph -



REPRESENTATION OF GRAPH IN MEMORY

- Graph is a mathematical structure and finds its applications in many areas of interest in which problems need to be solved using computers.
- Thus, this mathematical structure must be represented in some kind of data structure. A graph can be represented in many ways.
- Some of these representations are as follows:

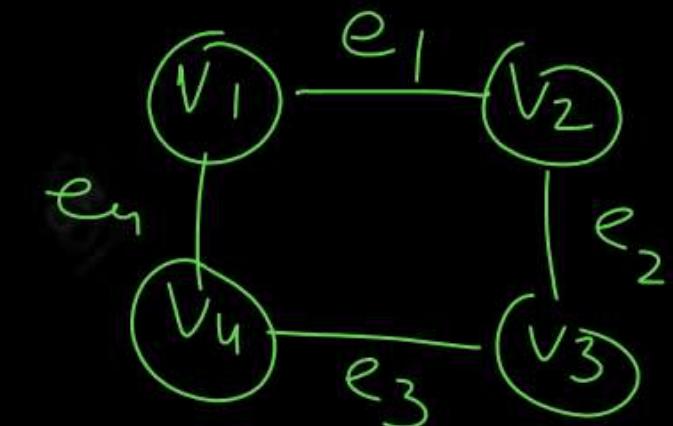
1. Set Representation of a graph

2. Sequential Representation of a graph

(a) Adjacency Matrix representation of a graph ($V - V$)

(b) Incidence Matrix representation of a graph ($V - c$)

3. Linked Representation of a graph



Set of Vertices, $V = \{v_1, v_2, v_3, v_4\}$
Set of Edges, $E = \{e_1, e_2, e_3, e_4\}$

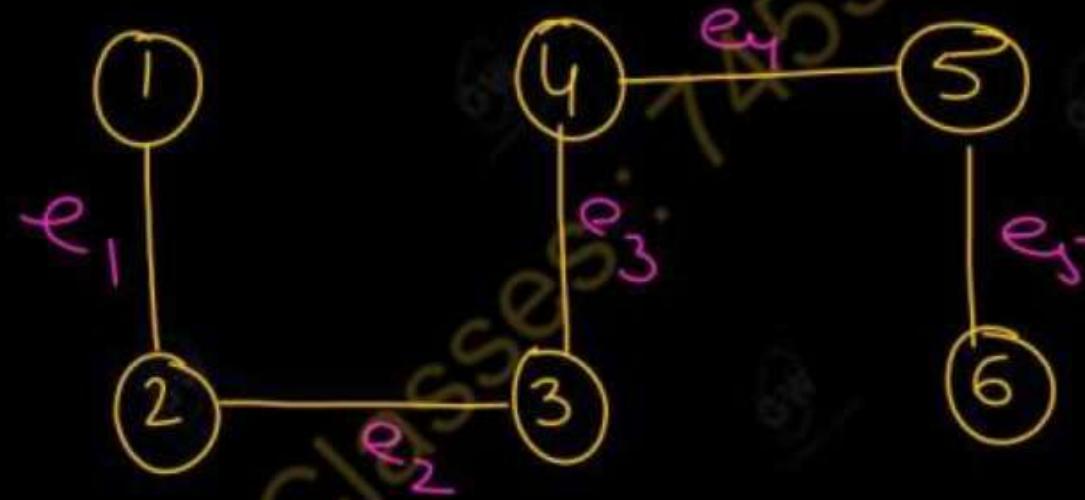
1. Set Representation of a Graph: -

In this representation, two sets are mentioned. They are

1. Set of vertices V and
2. Set of edges E

Consider the following graph:

(1,2)
(2,3)
(3,4)
(4,5)
(5,6)



$$E = \{ e_1, e_2, e_3, e_4, e_5 \}$$

This graph can be represented in set representation.

Therefore, Set of Vertices, $V = \{ 1, 2, 3, 4, 5, 6 \}$

And Set of Edges, $E = \{ (1,2), (2,3), (3,4), (4,5), (5,6) \}$

2. Sequential Representation of Graph in memory:-

□ A Graph can be represented in a matrix in sequential representation.

□ These are 3 most common matrices representations:-

(i) Adjacency Matrix Representation

(ii) Incidence Matrix Representation

(iii) Matrix Representation of a Weighted Graph

(i) **Adjacency Matrix representation of a Graph:-**

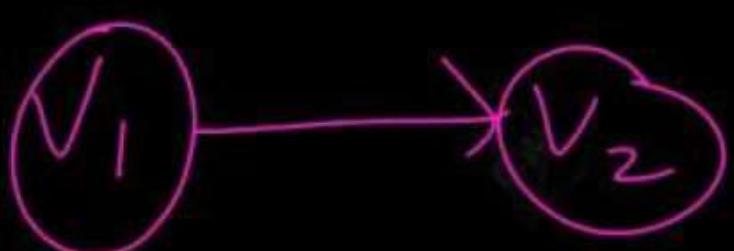
□ Adjacency matrix is the matrix, which keeps the information of adjacency nodes or vertices.

□ In other words, we can say that the matrix contains the information that whether this vertex is adjacent to any other vertex or not.

- The adjacency matrix is also called a **Bit matrix or Boolean Matrix** because the entries are either 0 or 1.
- The adjacency matrix A for a graph $G = (V, E)$ with n vertices, is an $n \times n$ matrix of bits, such that -

$A_{ij} = 1$, if there is an edge from V_i to V_j

and $A_{ij} = 0$, if there is no such edge.

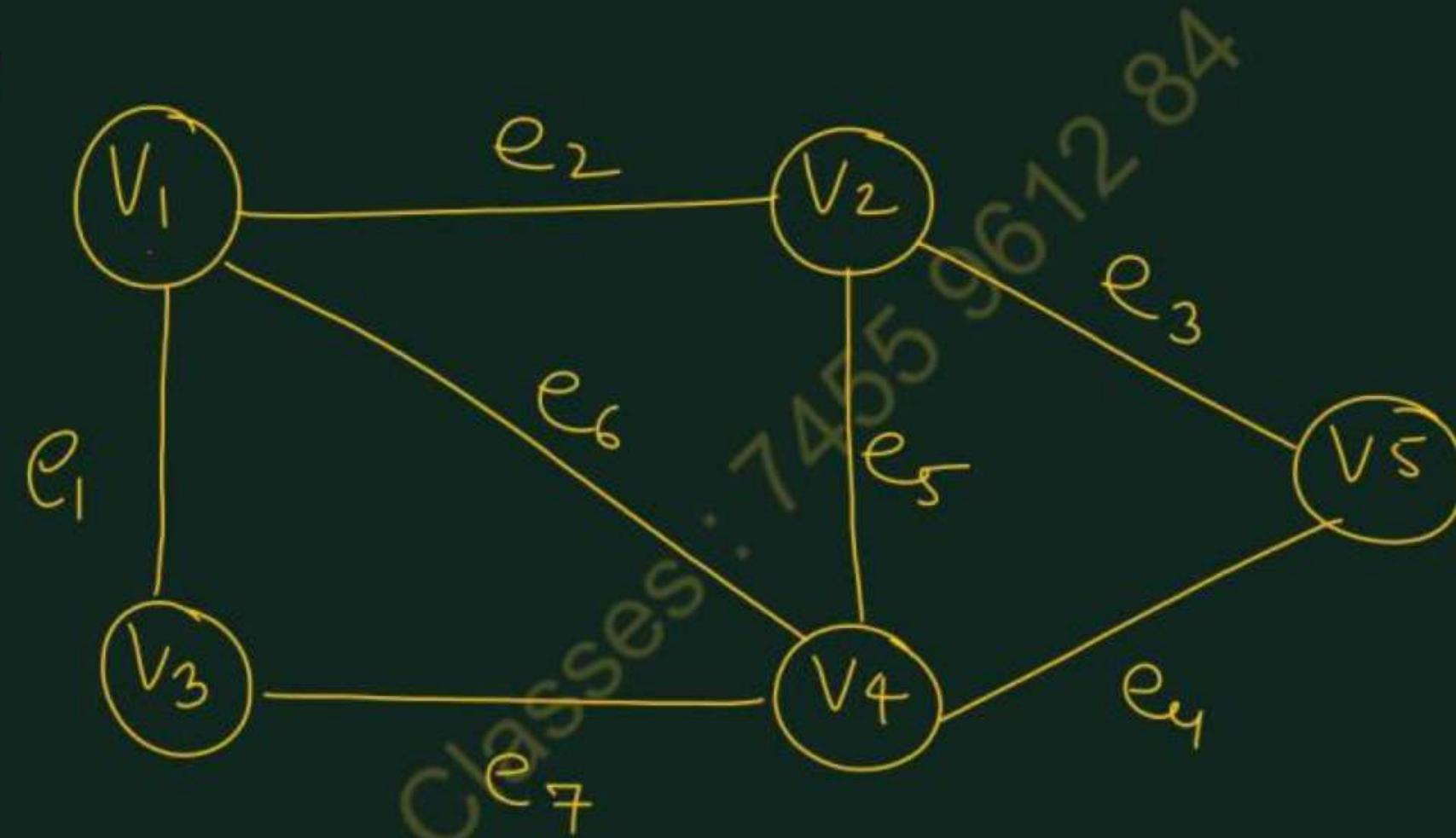


We can also write it as -

$$A(i, j) = \begin{cases} 1 & \text{if and only if edge } (V_i, V_j) \text{ is in EG.} \\ 0 & \text{otherwise} \end{cases}$$

Consider the following undirected graph in the next slide:

	v_1	v_2	v_3	v_4	v_5	
v_1	0	1	1	1	0	
v_2	1	0	1	1	1	
v_3	1	0	0	1	0	
v_4	0	1	1	0	1	
v_5	0	1	0	1	0	



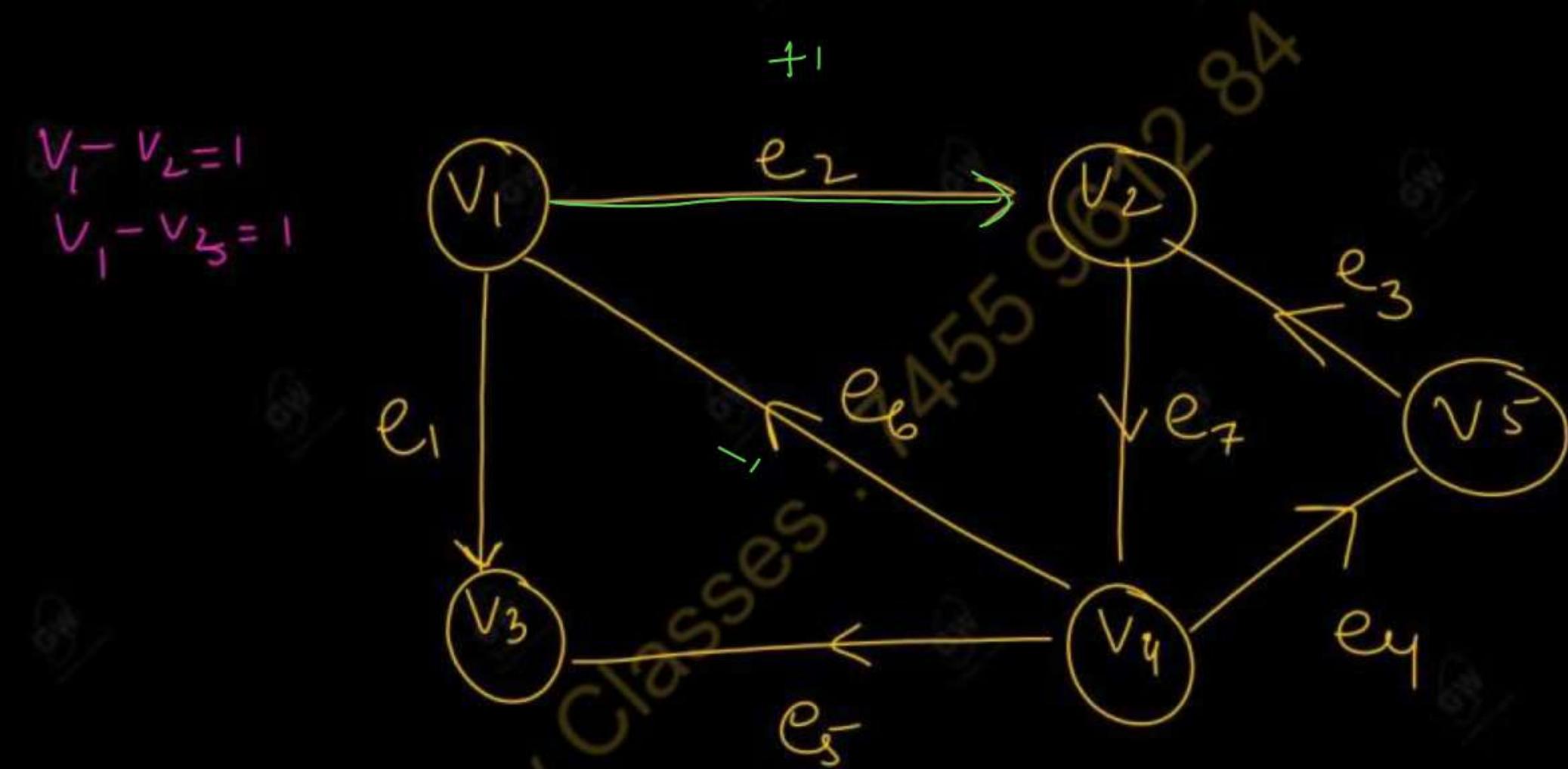
UNDIRECTED GRAPH

□ The adjacency matrix representation for the previous undirected graph is as follows:

	v_1	v_2	v_3	v_4	v_5
v_1	0 ✓	1	1	1	0
v_2	1 ✓	0	0	1 ✓	1 ✓
v_3	1 ✓	0	0	1 ✓	0
v_4	1	1	1	0	1
v_5	0	1	0	1	0

ADJACENCY
MATRIX

□ Consider the following directed graph:-



DIRECTED RAPH

□ The adjacency matrix of the previous directed graph is as follows:

	v ₁	v ₂	v ₃	v ₄	v ₅
v ₁	0	1 ✓	1 ✓	0	0
v ₂	0	0	0	1 ✓	0
v ₃	0	1	0	0	0
v ₄	1 ✓	0	1 ✓	0	1 ✓
v ₅	0	1	0	0	0

Adjacency
matrix

- An another representation in terms of indegree (-1) and outdegree (+1) can also be considered as:

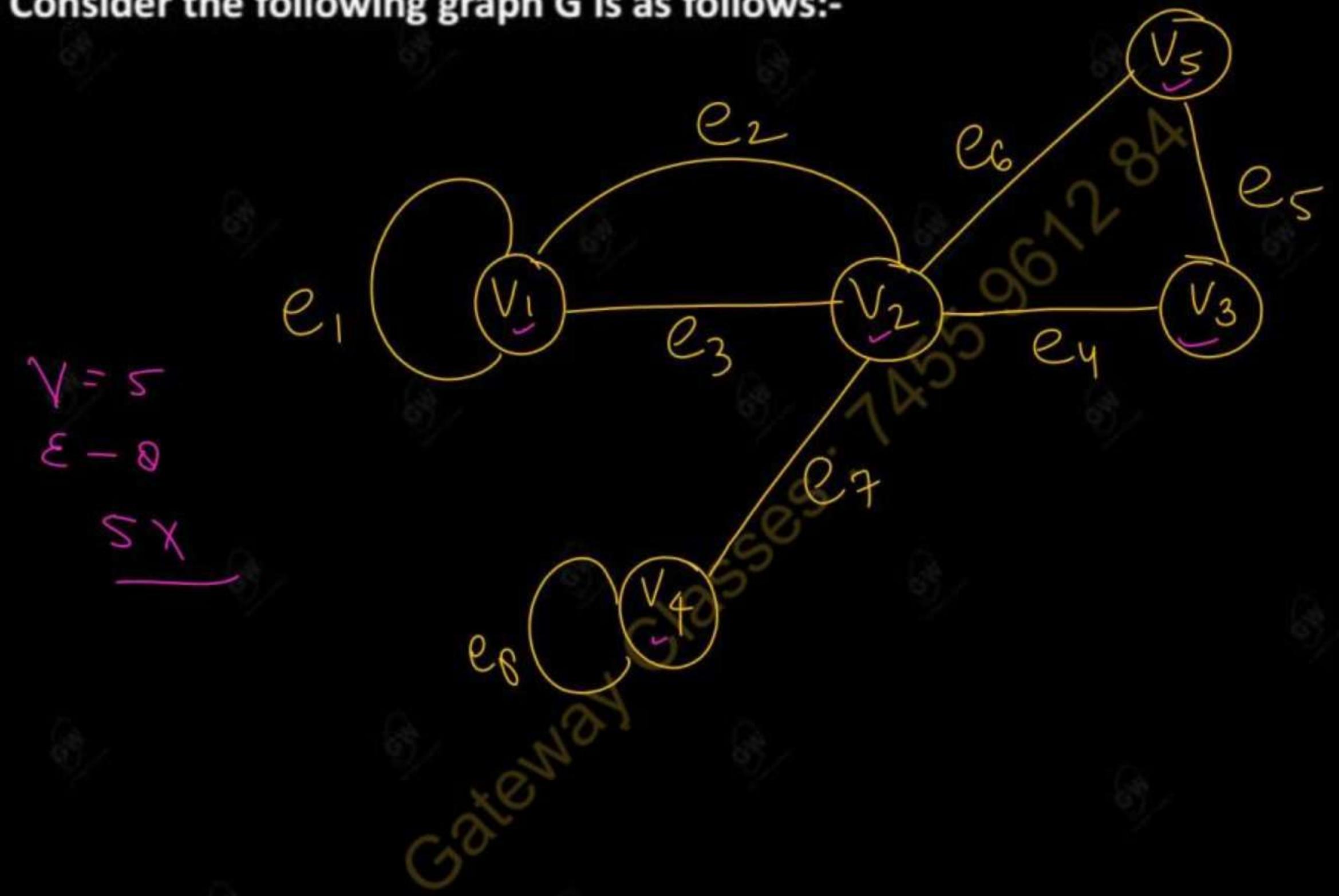
	v_1	v_2	v_3	v_4	v_5
v_1	0	<u>+1</u> ^{out}	+1	<u>-1</u> ^{indegree}	0
v_2	<u>-1</u>	0	0	+1	-1
v_3	-1	0	0	-1	0
v_4	+1	-1	+1	0	+1
v_5	0	+1	0	<u>-1</u>	0

(ii) Incidence Matrix Representation of a graph in memory:-

Let G be a graph with n vertices and e edges then the incidence matrix is a matrix of order $n \times e$, whose n rows corresponds to the n vertices and e columns corresponds to the e edges as follows:-

$$\begin{array}{l} \cancel{n \times n} \\ \cancel{5 \times 5} \\ V = 5 \\ E = 7 \\ 5 \times 7 \end{array} \quad a_{ij} = \begin{cases} 1 & \text{if the } j^{\text{th}} \text{ edge } e_j \text{ is incident on } i^{\text{th}} \text{ vertex } v_i \\ 0 & \text{otherwise} \end{cases}$$

□ Consider the following graph G is as follows:-



The incidence matrix for the previous graph is :

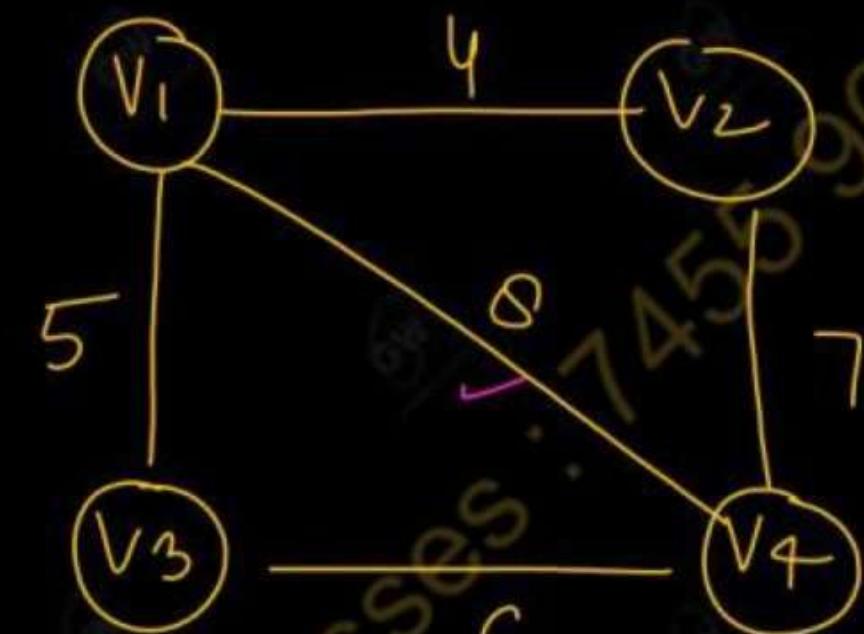
Incidence Matrix $I_{5 \times 8}$ of a Graph G.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
v_1	1 ✓	1 ✓	1 ✓	0	0	0	0	0
v_2	0	1 -	1 -	1 ✓	0	1 ✓	1 -	0
v_3	0	0	0	1	1	0	0	0
v_4	0	0	0	0	0	0	1	1
v_5	0	0	0	0	1 ✓	1 ✓	0	0

5×8

(iii) Matrix Representation of a Weighted Graph:-

- If the graph is weighted , then corresponding weights are entered in the matrix.
- Consider the following weighted graph:

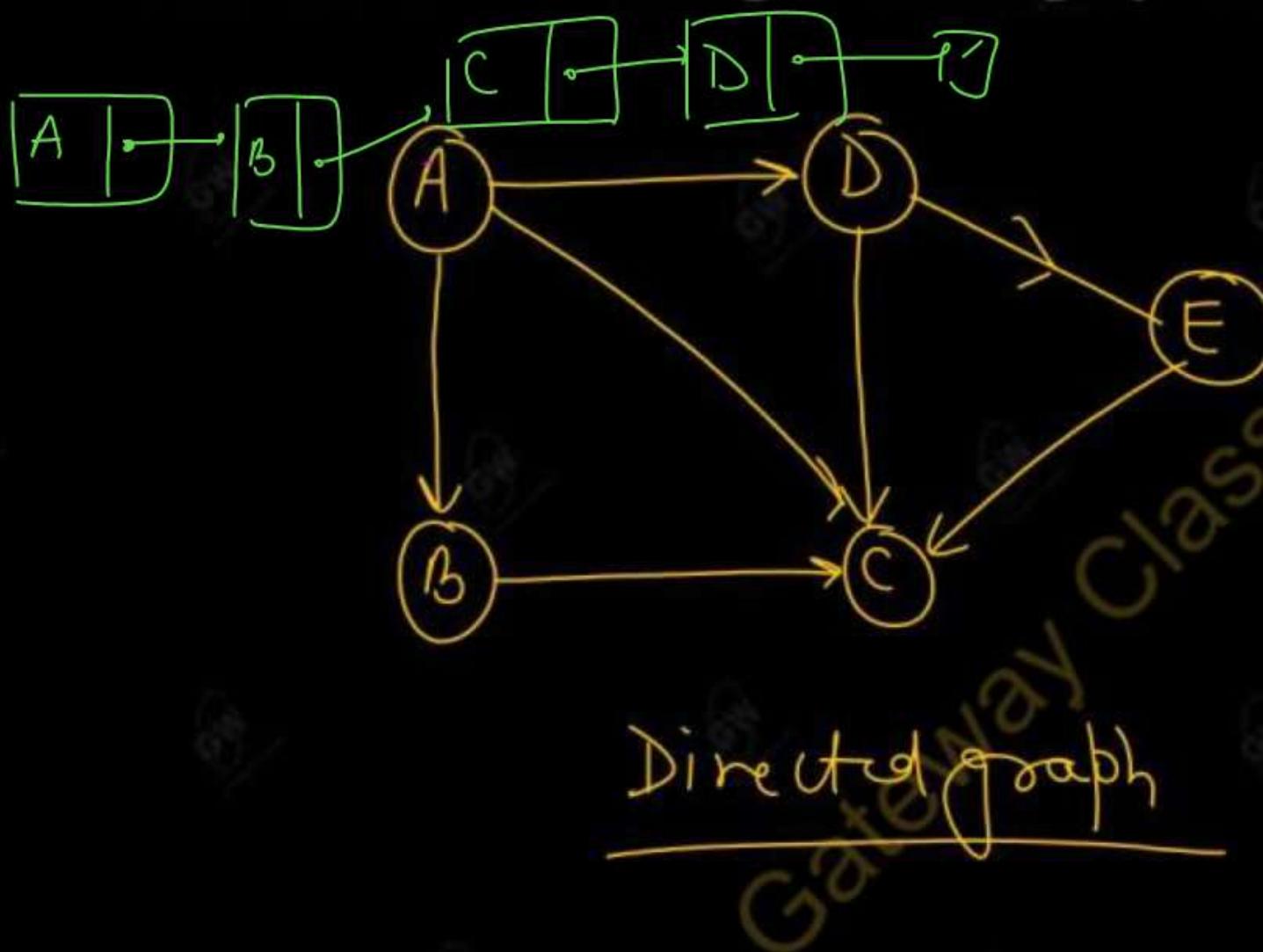


- The matrix representation of above weighted graph is as follows:-

	v_1	v_2	v_3	v_4
v_1	0	4	5	0
v_2	4	0	0	7
v_3	5	0	0	6
v_4	0	7	6	0

2. Linked Representation of a Graph in Memory:-

- In this representation, first, adjacency list of a graph is created, which is basically a list of all neighbors.
- Consider the following directed graph and its adjacency list:



Node	Adjacency
A	B C D
B	C
C	-
D	C, E
E	C

Adjacency list

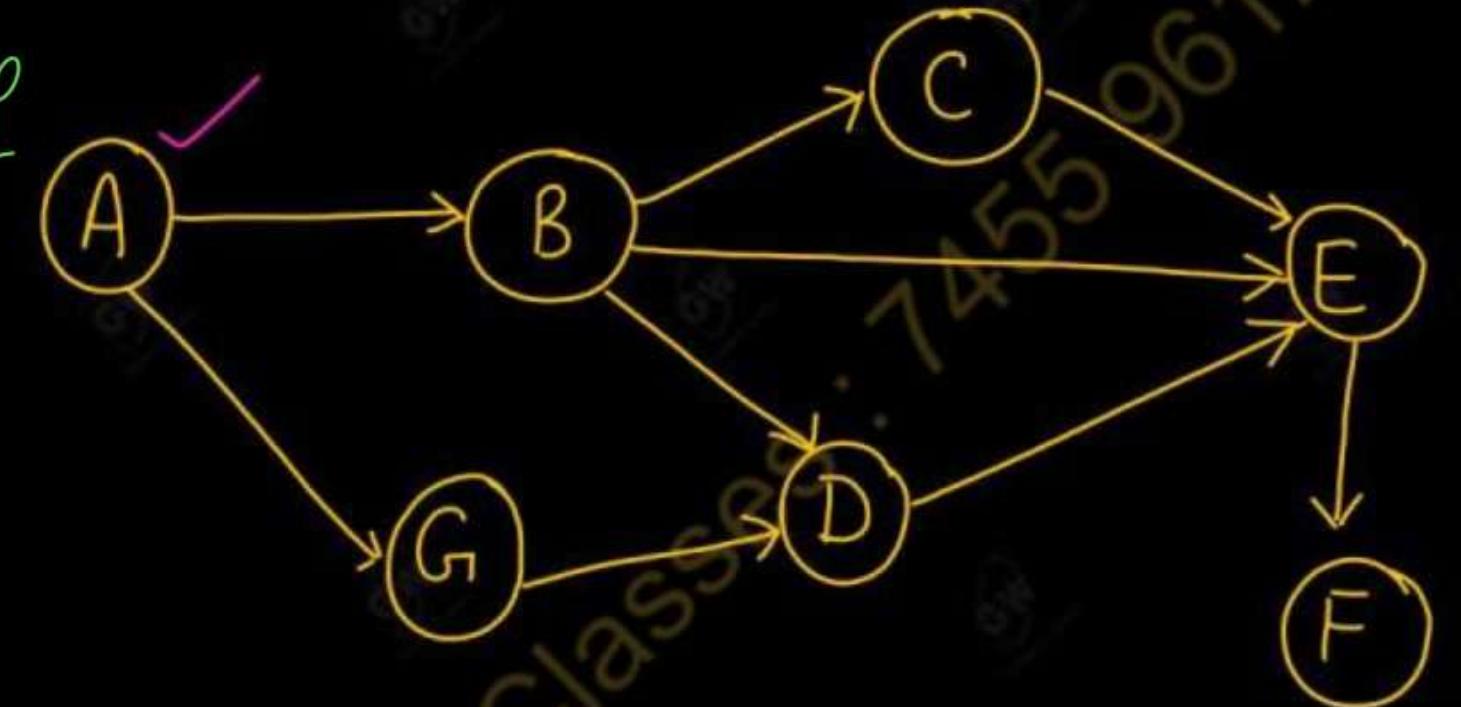
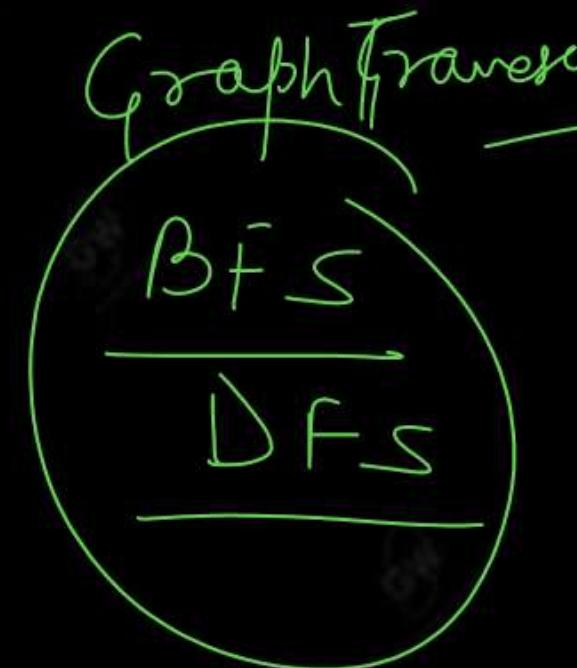
let us see a schematic diagram of linked representation
of previous graph:-

The linked representation will contain two lists , a node list **NODE**
and an edge list **EDGE**.

stack

BFS \Rightarrow Queue

Q.3 Discuss the data structure used in DFS. Write an algorithm for DFS, Traverse the given graph starting from node A using DFS.



2017-18, 7 Marks

Q.4 Differentiate between DFS and BFS. Draw the breadth first tree for the above graph.

BFS

given



Q.5 Write an algorithm for Breadth First Search (BFS) traversal of a graph.

2021-22, 10 Marks

2022-23, 2 Marks

Graph Traversal

- A graph traversal means visiting all the nodes of the graph.
- Graph traversal may be needed in many of application areas and there may be many methods for visiting the vertices of the graph.
- There are two graph traversal methods.
 - (i) Breadth First Traversal (BFS)
 - (ii) Depth First Traversal (DFS)
- The BFS will use a queue data structure to hold nodes for future processing, and DFS will use a stack data structure.

Algorithm : BREADTH FIRST SEARCH (BFS) (G, A)

This algorithm executes a breadth-first search on a graph G beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
2. Put the starting node A in QUEUE and change its status to the waiting state (STATUS=2).
3. Repeat Steps 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 neighbors of N that are in the steady state (STATUS=1), and change their status to the waiting state (STATUS = 2).
[End of Step 3 loop.]
6. Exit

BFS

Consider the following graph G:

$$E = \{$$

$$\{(A, F), (F, D)\}$$

$$\{(A, B), (G, C)\}$$

$$\{(A, C), (G, E)\}$$

$$\{(B, C), (J, D)\}$$

$$\{(B, G), (J, K)\}$$

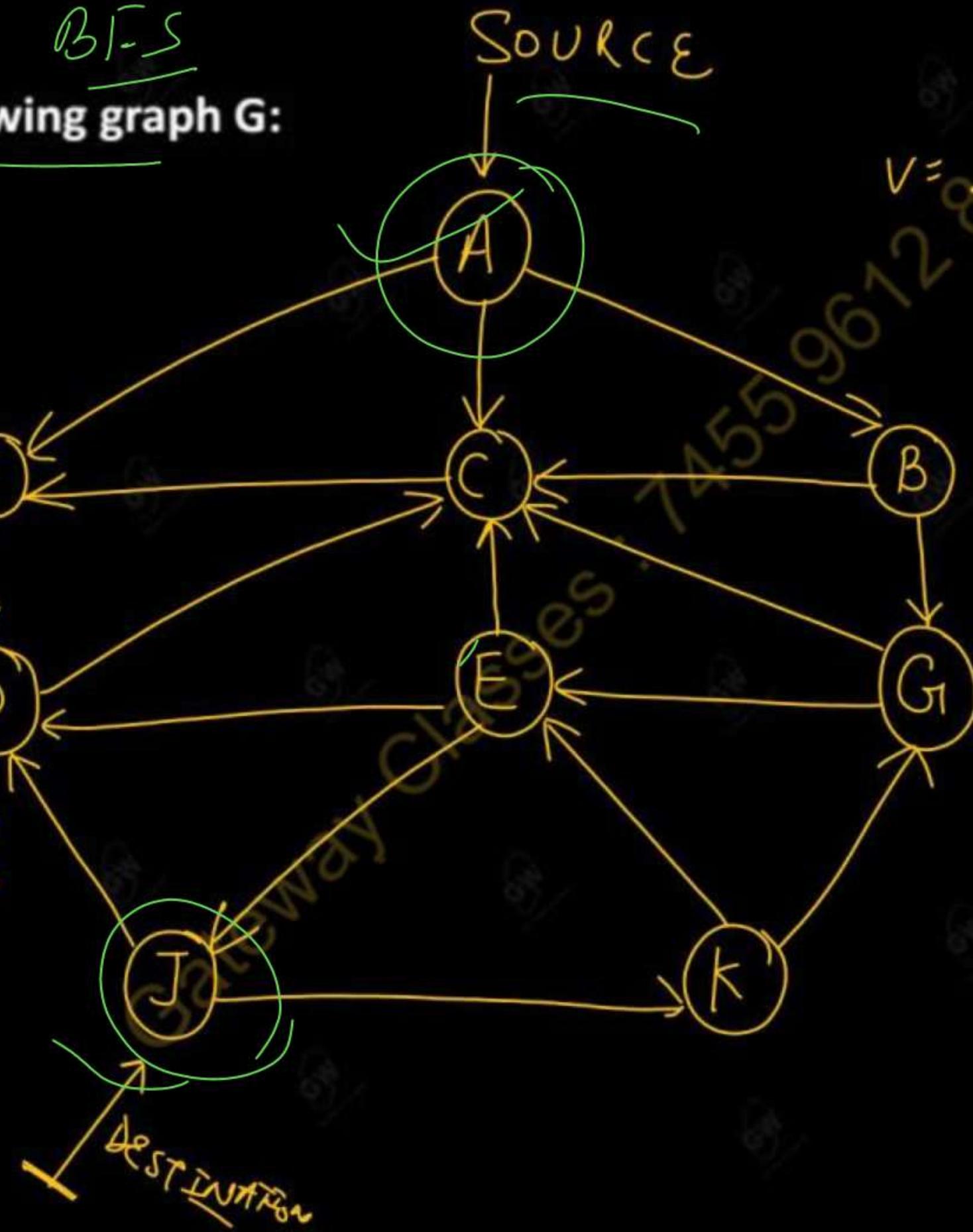
$$\rightarrow (C, F)$$

$$\rightarrow (D, C), (K, E)$$

$$\{(E, C), (K, G)\}$$

$$\{(E, D)\}$$

$$(E, J)\}$$



$$V = \{A, B, C, D, E, F, G, J, K\}$$

DIRECTED
GRAPH

The adjacency lists of the graph is as follows:

ADJACENCY LIST		
A	F, C, B	✓
B	G, C	✓
C	F	✓
D	C	✓
E	D, C, J	✓
F	D	✓
G	C, E	✓
I	D, K	✓
K	E, G	✓

- Suppose G represents the daily flights between cities of some airline, and suppose we want to fly from city A to city J with the minimum number of stops.
- In other words, we want the minimum path P from A to J (where each edge has length 1).

- The minimum path P can be found by using a breadth-first search beginning at city A and ending when J is encountered.

BFS

- During the execution of the search, we will also keep track of the origin of each edge ORIGIN by using an array ORIG together with the array QUEUE.
- The steps of our search follow.

- (a) Initially, add A to QUEUE and add NULL to ORIG as follows:

FRONT = 1

REAR = 1

QUEUE : A
ORIG : \emptyset

- (b) Remove the front element A from QUEUE by setting FRONT = FRONT + 1, and add to QUEUE the neighbors of A as follows:

FRONT = 2

REAR = 4

QUEUE	1	2	3	4
FRONT				
REAR				

QUEUE	A, F, C, B
ORIG	\emptyset, A, A, A

- Note that the origin A of each of the three edges is added to ORIG.

(c) Remove the front element F from QUEUE by setting FRONT = FRONT + 1, and add to QUEUE the neighbors of F as follows:

FRONT = 3

REAR = 5

FRONT	1	2	3	4	5	REAR
QUEUE : A, F, C, B, D						
ORIG : Ø, A, A, A, F						

(d) Remove the front element C from QUEUE, and add to QUEUE the neighbors of C

(which are in the ready state) as follows:

FRONT = 4

REAR = 5

FRONT	1	2	3	4	5	REAR
QUEUE : A, F, C, B, D						
ORIG : Ø, A, A, A, F						

□ Note that the neighbor F of C is not added to QUEUE, since F is not in the ready state
(because F has already been added to QUEUE).

(e) Remove the front element B from QUEUE, and add to QUEUE the neighbors of B (the ones in the ready state) as follows:

FRONT = 5

REAR = 6

	2	3	4	5	6	
QUEUE	A	F	C	B	D	G
ORIG	Ø	A	A	A	F	B

Note that only G is added to QUEUE, since the other neighbor, C is not in the ready state.

(f) Remove the front element D from QUEUE, and add to QUEUE the neighbors of D (the ones in the ready state) as follows:

FRONT = 6

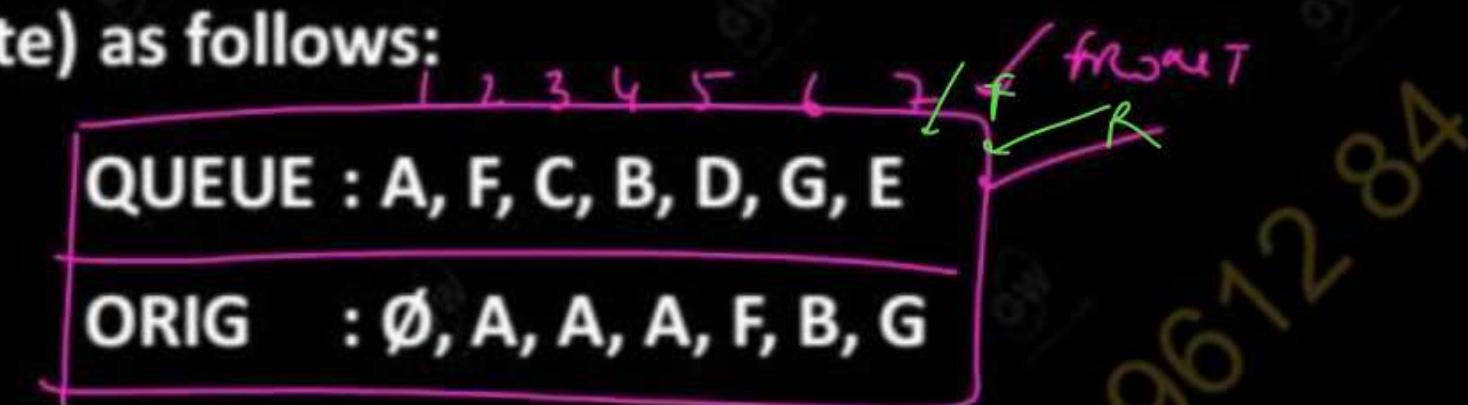
REAR = 6

	1	2	3	4	5	6	
QUEUE	A	F	C	B	D	G	
ORIG	Ø	A	A	A	F	B	

(g) Remove the front element G from QUEUE and add to QUEUE the neighbors of G (the ones in the ready state) as follows:

FRONT=7

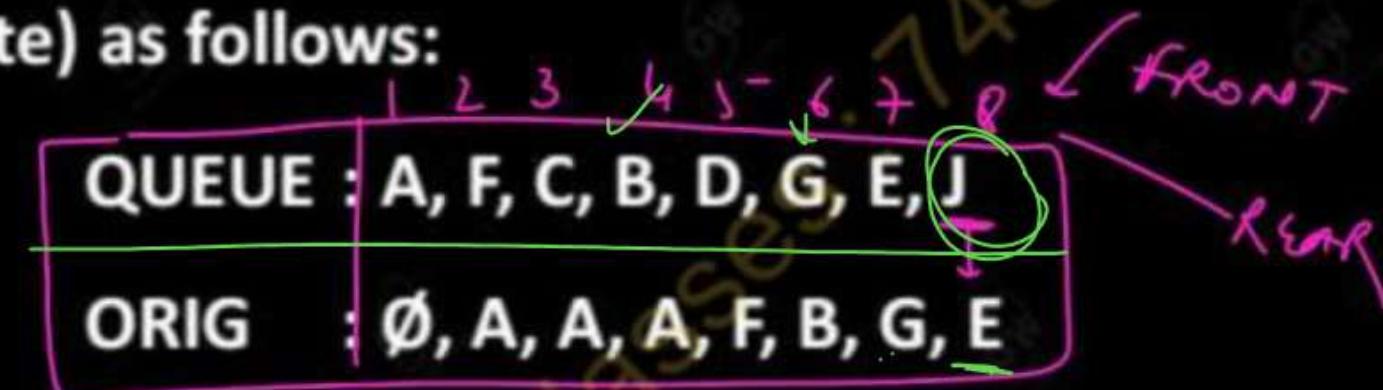
REAR = 7



(h) Remove the front element E from QUEUE and add to QUEUE the neighbors of E (the ones in the ready state) as follows:

FRONT = 8

REAR = 8



We stop as soon as J is added to QUEUE, since J is our final destination.

We now backtrack from J, using the array ORIG to find the path P. Thus

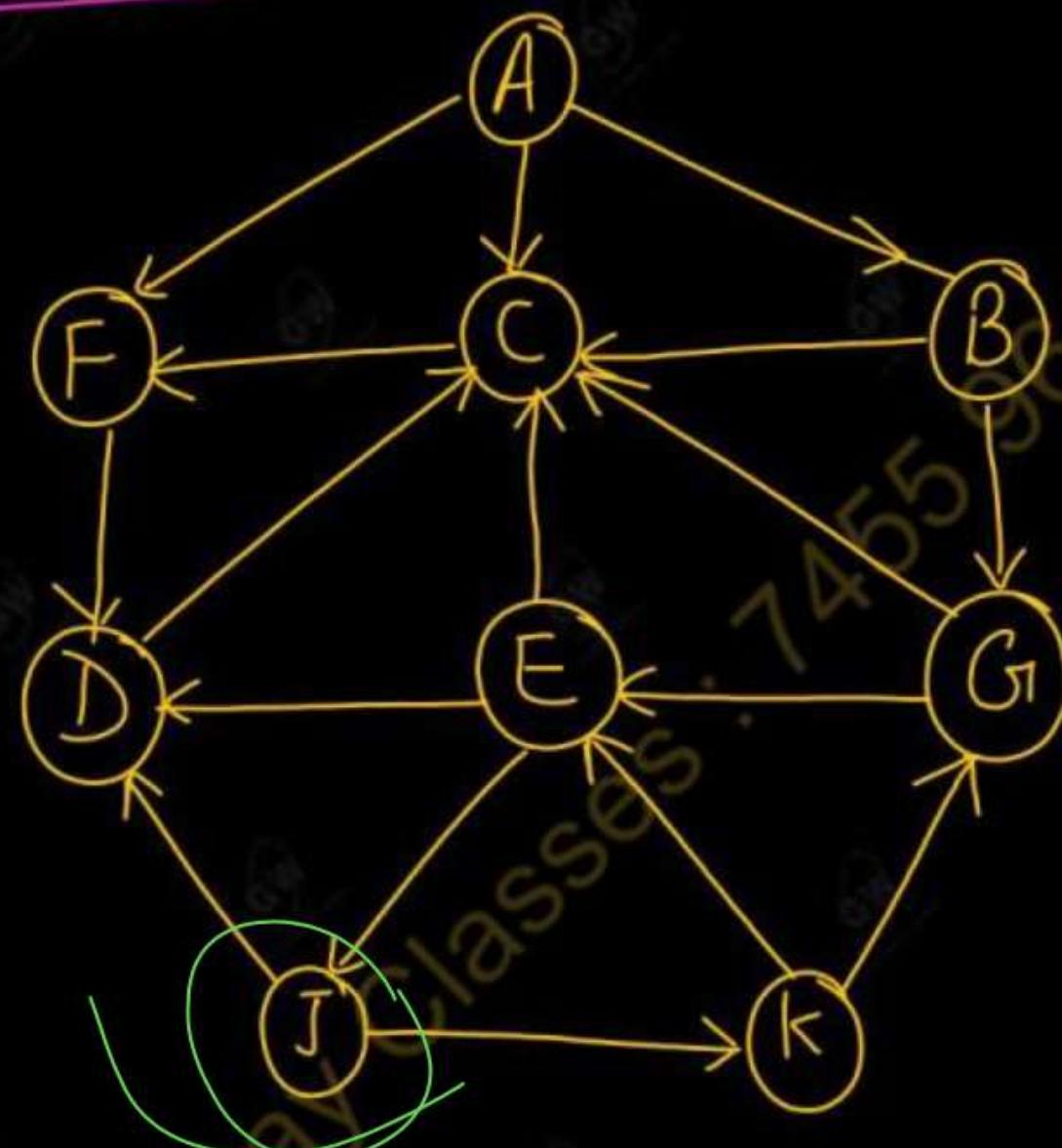
~~J ← E ← G ← B ← A~~

is the required path P.

FS

This algorithm executes a depth-first search on a graph G beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
 2. Push the starting node A onto STACK and change its status to the waiting state (STATUS=2).
 3. Repeat Steps 4 and 5 until STACK is empty.
4. Pop the top node N of STACK. Process N and change its status to the processed state
(STATUS = 3).
 5. Push onto STACK all the neighbors of N that are still in the ready state (STATUS = 1), and
change their status to the waiting state (STATUS = 2).
- [End of Step 3 loop.]
6. Exit.



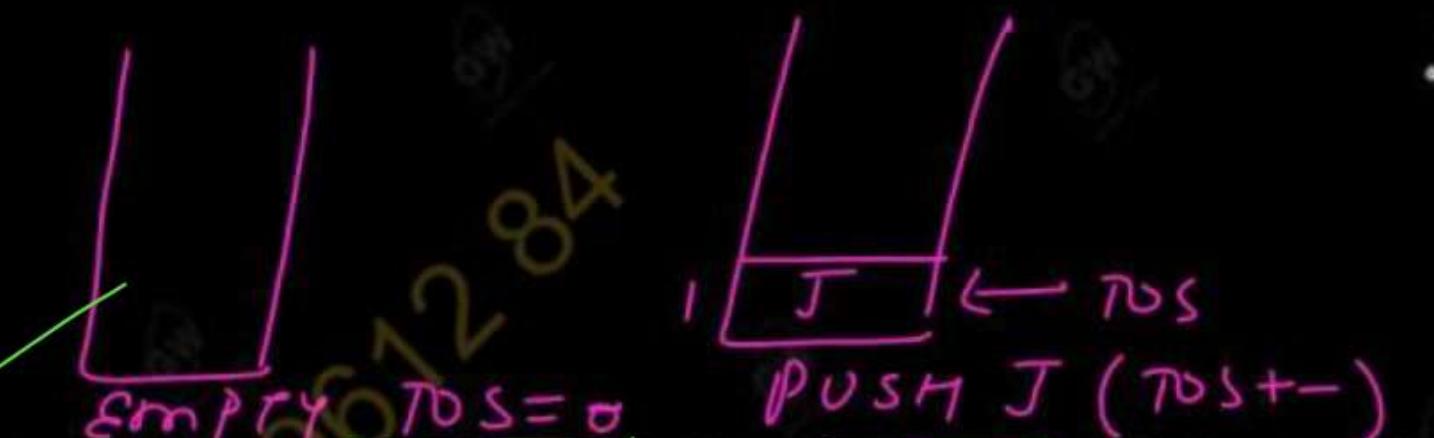
Adjacency list	
A	F, C, B
B	F, C
C	F
D	C
E	D, C, J
F	D
G	C
J	D, K
K	E, G

- Suppose we want to find and print all the nodes reachable from the node J (including J itself).
- One way to do this is to use a depth-first search of G starting at the node J.

The steps of our search are as follow:

(a) Initially, push J onto the stack as follows:

STACK : J

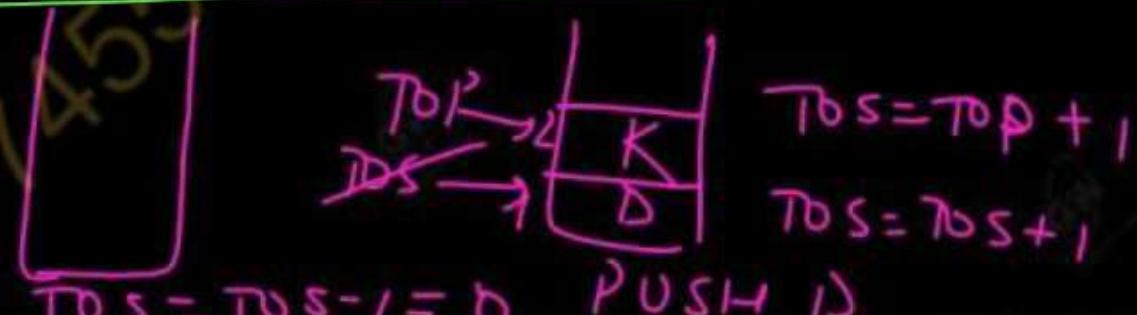


(b) Pop and print the top element J, and then push onto the stack all the neighbors of J

(those that are in the ready state) as follows:

Print J

STACK : D, K



(c) Pop and print the top element K, and then push onto the stack all the neighbors of K

(those that are in the ready state) as follows:

Print K

STACK : D, E, G

(d) Pop and print the top element G, and then push onto the stack all the neighbors of G

(those in the ready state) as follows:

Print G

STACK : D, E, C

Note that only C is pushed onto the stack, since the other neighbor, E, is not in the ready state (because E has already been pushed onto the stack).

(e) Pop and print the top element C, and then push onto the stack all the neighbors of C (those in the ready state) as follows:

Print C STACK : D, E, F

(f) Pop and print the top element F, and then push onto the stack all the neighbors of F (those in the ready state) as follows:

Print F STACK : D, E

Note that the only neighbor D of F is not pushed onto the stack, since D is not in the ready state (because D has already been pushed onto the stack).

(g) Pop and print the top element E, and push onto the stack all the neighbors of E (those in the ready state) as follows:

Print E

STACK : D

(Note that none of the three neighbors of E is in the ready state.)

(h) Pop and print the top element D, and push onto the stack all the neighbors of D (those in the ready state) as follows:

Print D

STACK: \emptyset NULL

The stack is now empty, so the depth-first search of G starting at J is now complete.

Accordingly, the nodes which were printed,

J, K, G, C, F, E, D

Dfs

are precisely the nodes which are reachable from J.

Q.6 Explain Kruskal's algorithm to find minimum spanning tree in a weighted directed graph. Can there be two minimum spanning trees of given weighted directed graph?

2014-15, 10 Marks

Q.7 Find MST of the following graph using Kruskal's algorithm.

2014-15, 5 Marks

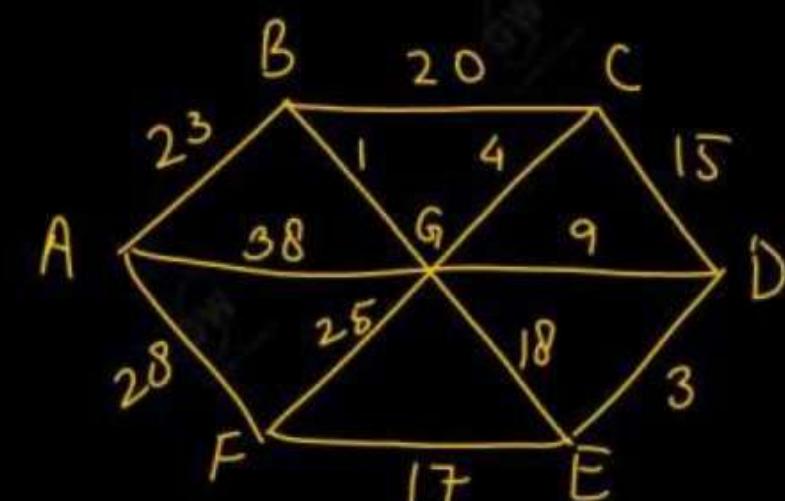


Min. Cost Spanning Tree:
1) Prim's Algorithm
2) Kruskal's Algorithm

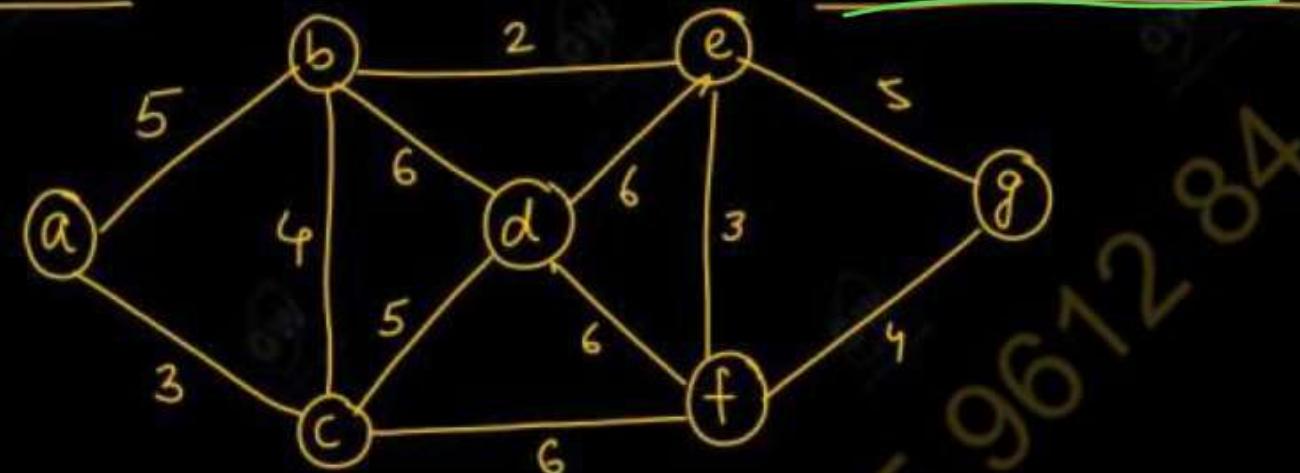
Q.8 Consider the following undirected graph:

- Find the adjacency list representation of the graph.
- Find the minimum cost spanning tree by Kruskal's algorithm.

2015-16, 10 Marks



Q.9 Find minimum spanning tree in the following graph using Prim's algorithm by considering vertex 'a' as root/source vertex.



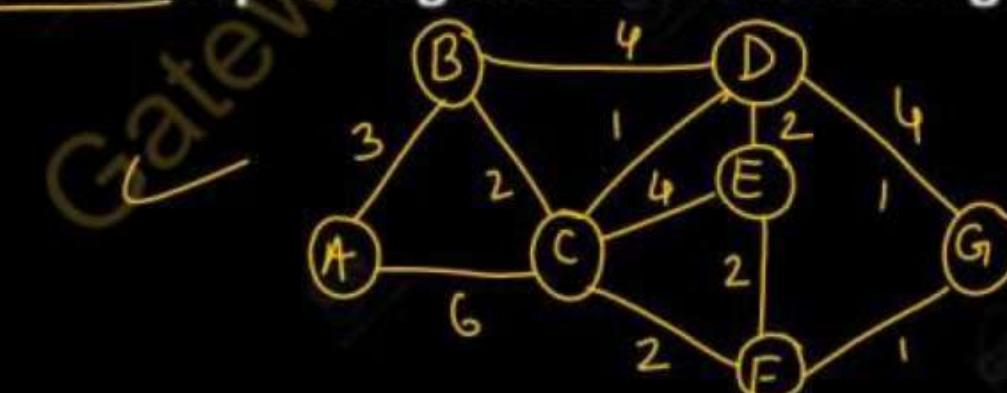
2020-21, 7 Marks

Q.10 Apply Prim's algorithm to find a minimum a spanning tree in the following weighted graph as shown below.



2021-22, 10 Marks

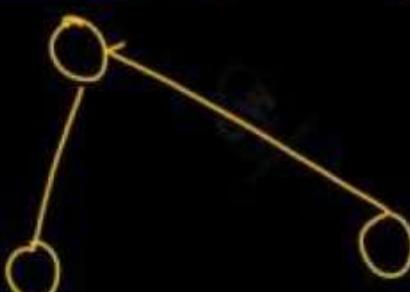
Q.11 What is spanning tree? Write down the Prim's algorithm to obtain minimum cost spanning tree. Use Prim's algorithm to find the minimum cost spanning tree in the following graph:



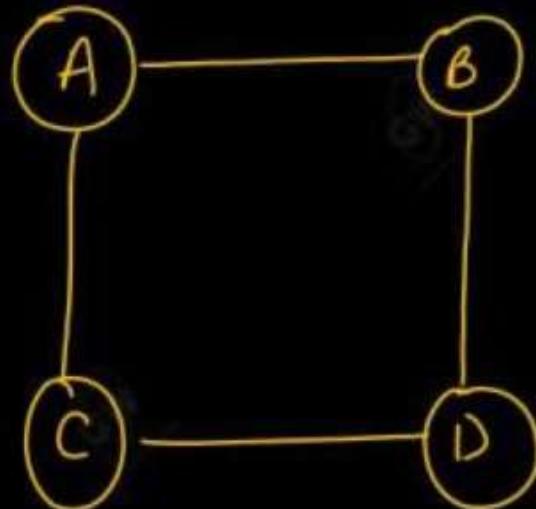
2022-23, 10 Marks

SPANNING TREE

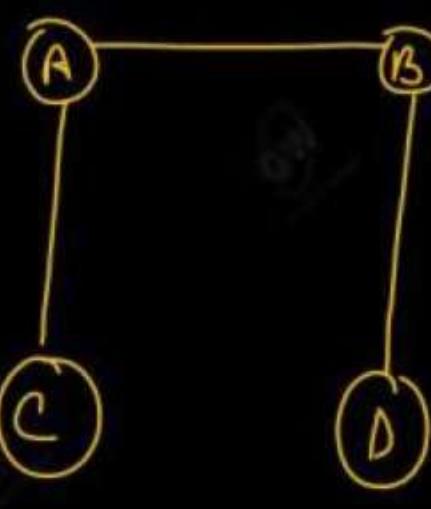
- A spanning tree of a graph is just a sub-graph that contains all the vertices and is a tree.
- A spanning tree is a subset of graph G, such that all the vertices are connected using minimum possible number of edges.
- A spanning tree does not have cycles and a graph may have more than one spanning tree.
- That is, a spanning tree of a connected graph G contains all the vertices and has the edges which connects all the vertices. So, the number of edges will be 1 less than the number of nodes.



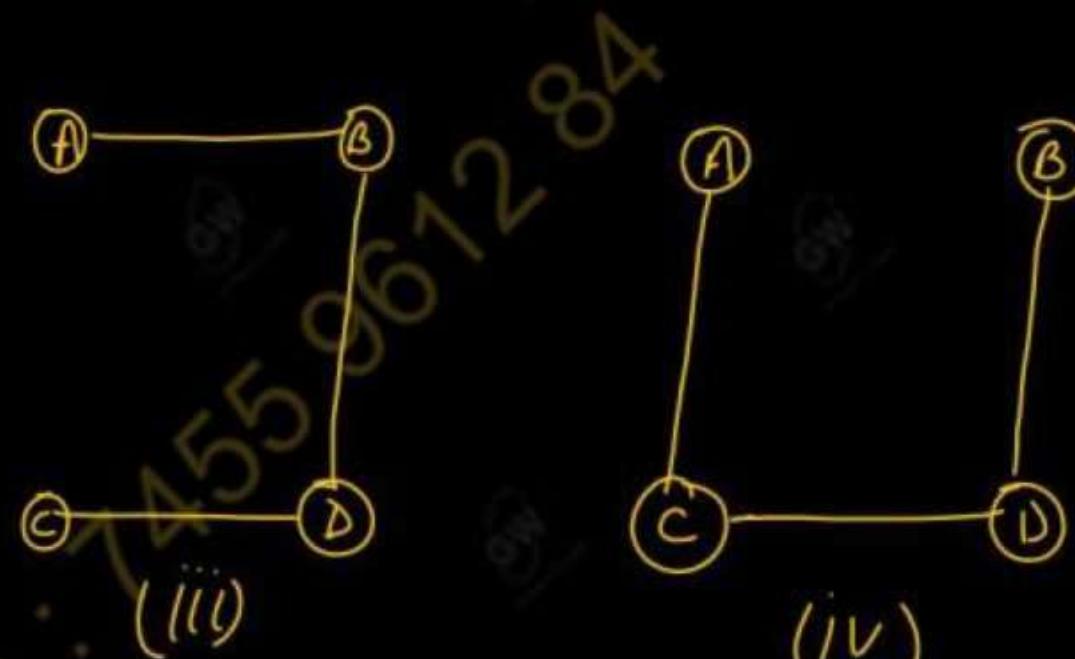
□ Let us take a connected graph:



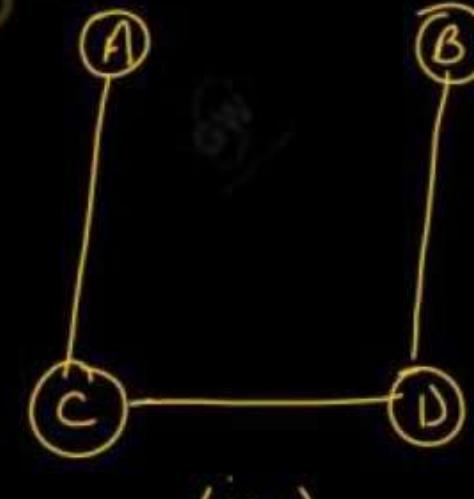
(i)



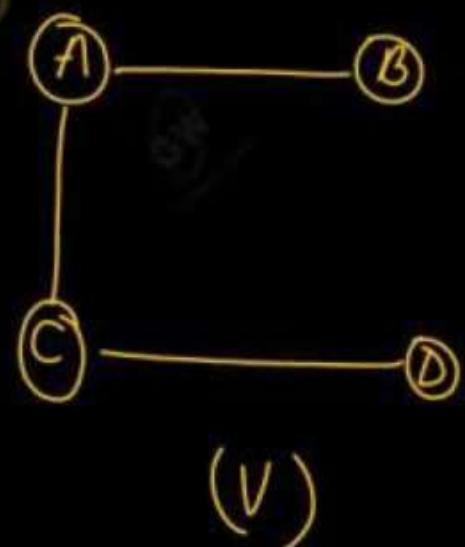
(ii)



(iii)



(iv)



(v)

□ Here all these trees are spanning tree and the number of edges is one less than the number of nodes.

MINIMUM SPANNING TREE OR MINIMUM COST SPANNING TREE

- Given a connected weighted graph G , it is often desired to create a spanning tree T for G such that the sum of the weights of the tree edges in T is as minimum as possible.
- Such a tree is called a minimum spanning tree or minimum cost spanning tree and represents the "cheapest" way of connecting all the nodes in G .
- There are number of techniques for creating a minimum spanning tree for a weighted graph but the most famous methods are as follows:-
 1. Kruskal's Algorithm and
 2. Prim's Algorithm.

KRUSKAL'S ALGORITHM

- This algorithm creates a forest of trees.
- Initially, the forest consists of n single node trees and no edges. That is, in the method initially we take n distinct trees for all n nodes of the graph.
- At each step, we add one (the cheapest one or an edge with minimum weight) edge, so that it joins two trees together.
- If it forms a cycle, it does simply links two nodes that were already part of a single connected tree, so that this edge does not included.
- The steps are as follows:
 1. Initially construct a separate tree for each node in a graph.
 2. Edges are placed in a priority queue, i.e., we take edges in ascending order.

3. Until we have added $n - 1$ edges.

(a) Extract the cheapest edge from the queue.

(b) If it forms a cycle, reject it.

else

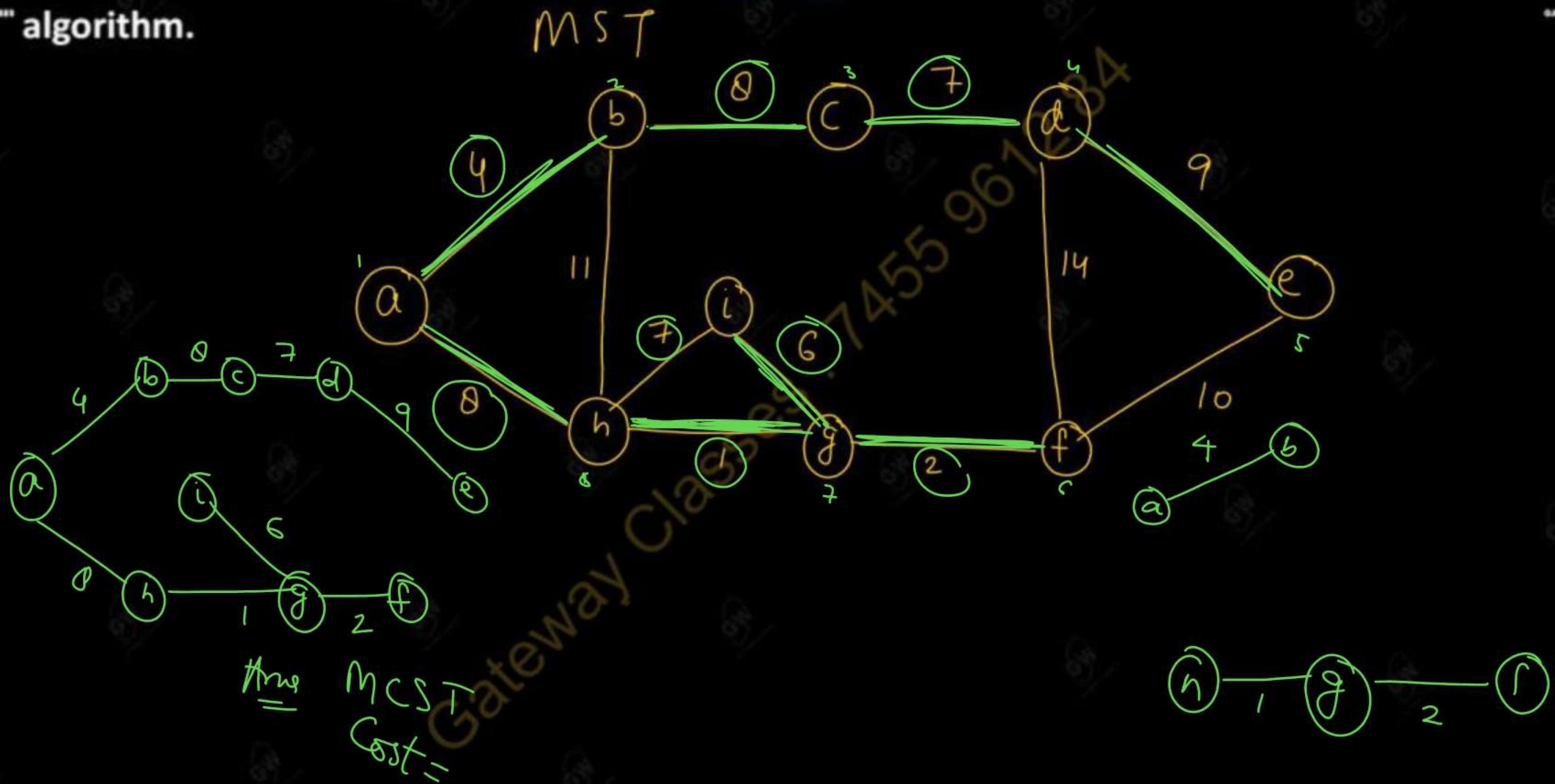
Add it to the forest.

4. Whenever we insert an edge, two trees will be joined, i.e., every step will have joined two trees in the forest together so that at the end, there will be only one tree.

- In this method, first we examine all the edges one-by-one starting from the smallest edge.
- To decide whether the selected edge should be included in the spanning tree or not, we will examine the two nodes connecting the edge.
- If the two nodes belong to the same tree then we will not include the edge in the spanning tree, since the two nodes are in the same tree, they are already connected and adding this edges would result in a cycle.
- So we will insert an edge in the spanning tree only if it's nodes are in different trees.

1. $A \leftarrow \phi.$
2. For each vertex $v \in V [G]$
do MAKE-SET (v)
3. Sort the edges of E into increasing order by weight w.
4. For each edge $(u, v) \in E$ taken in increasing order.
do if FIND-SET (u) \neq FIND-SET (v)
then $A \leftarrow A \cup (u, v)$ UNION (u, v)
5. Return A.

EXAMPLE-1. Find the minimum spanning tree of the following graph using Kruskal's algorithm.



Solution: First we initialize the set A to the empty set and create v trees, one containing each vertex with MAKE-SET procedure.

Then sort the edges in E into order by non-decreasing weight, i.e.,

Edge	Weight
(h, g)	1
(g, f)	2
(a, b)	4
(i, g)	6
(h, i)	7
(c, d)	7
(b, c)	8
(a, h)	8
(d, e)	9
(e, f)	10
(b, h)	11
(d, f)	14

←
↑
↑
↑
↑
↑
↑
↑
↑
↑
↑
↑

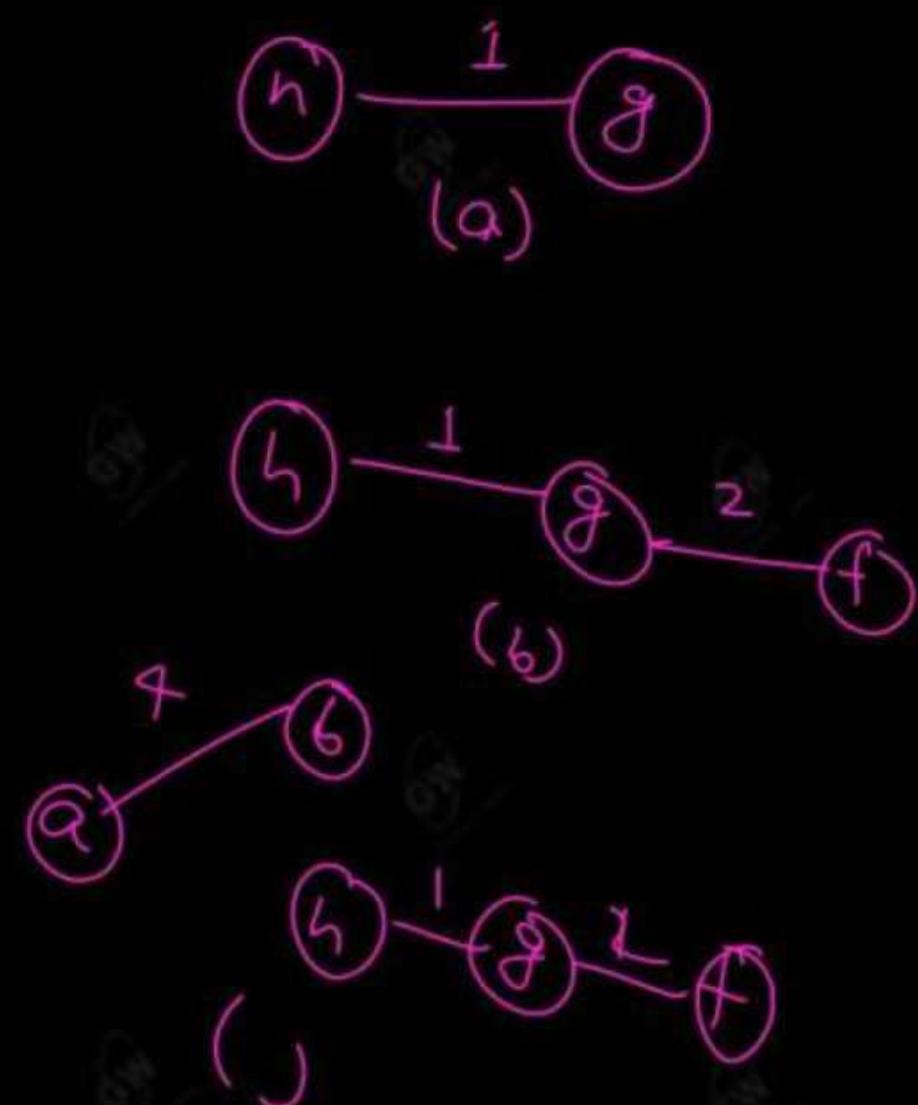
Not considered due to cycle
not considered due to cycle
cycle
cycle

□ Now, check for each edge (u, v) , whether the end points u and v belong to the same tree.

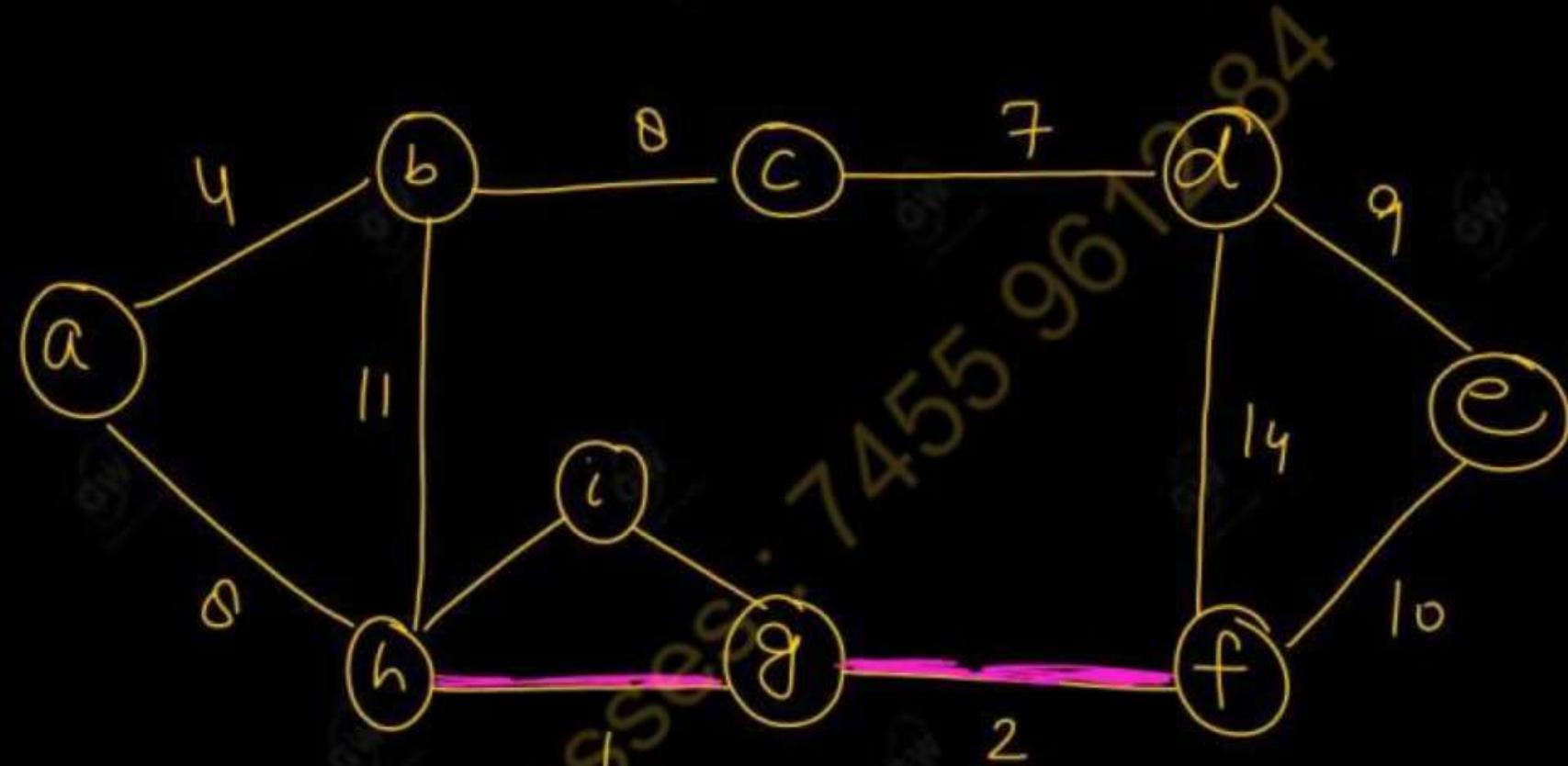
□ If they do, then the edge (u, v) cannot be added.

□ Otherwise, the two vertices belong to different trees and the edge (u, v) is added to A and the vertices in the two trees are merged.

□ So, first take (h, g) edge -



□ Then (g, f) edge.



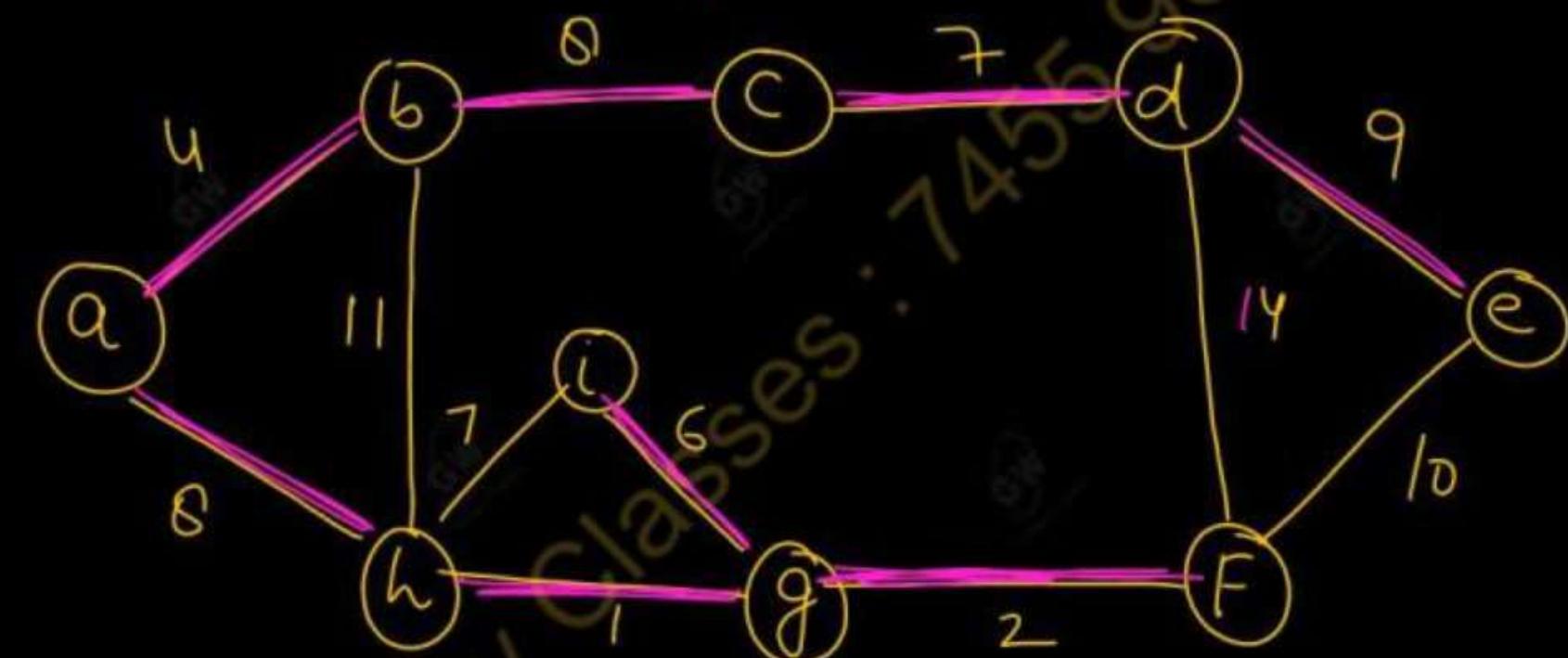
□ Then (a, b) and (i, g) edges are considered and forest becomes.



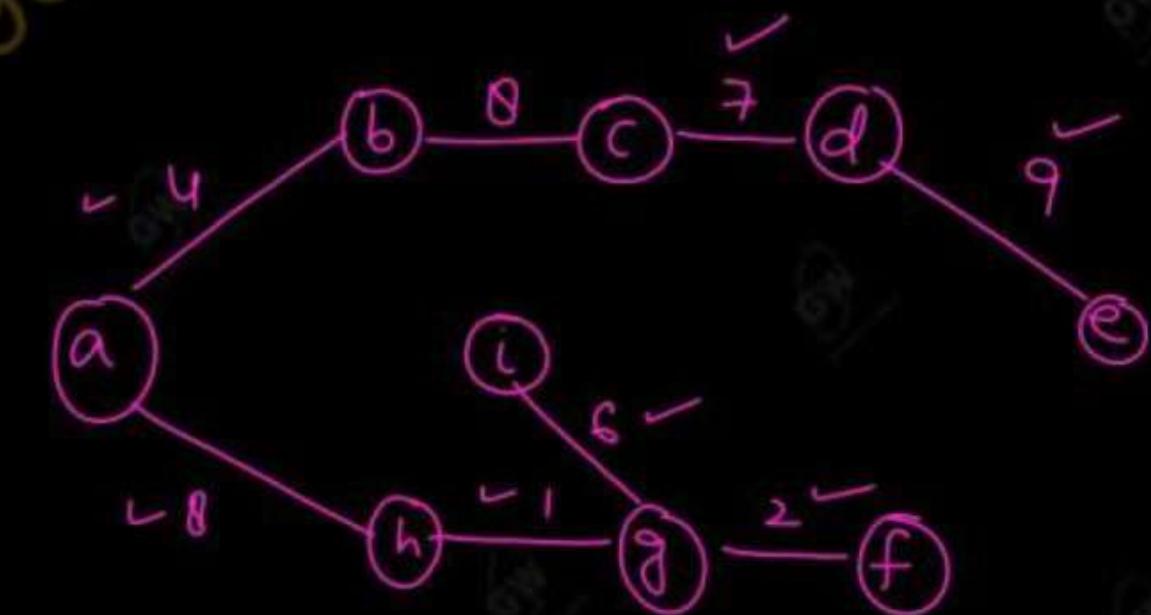
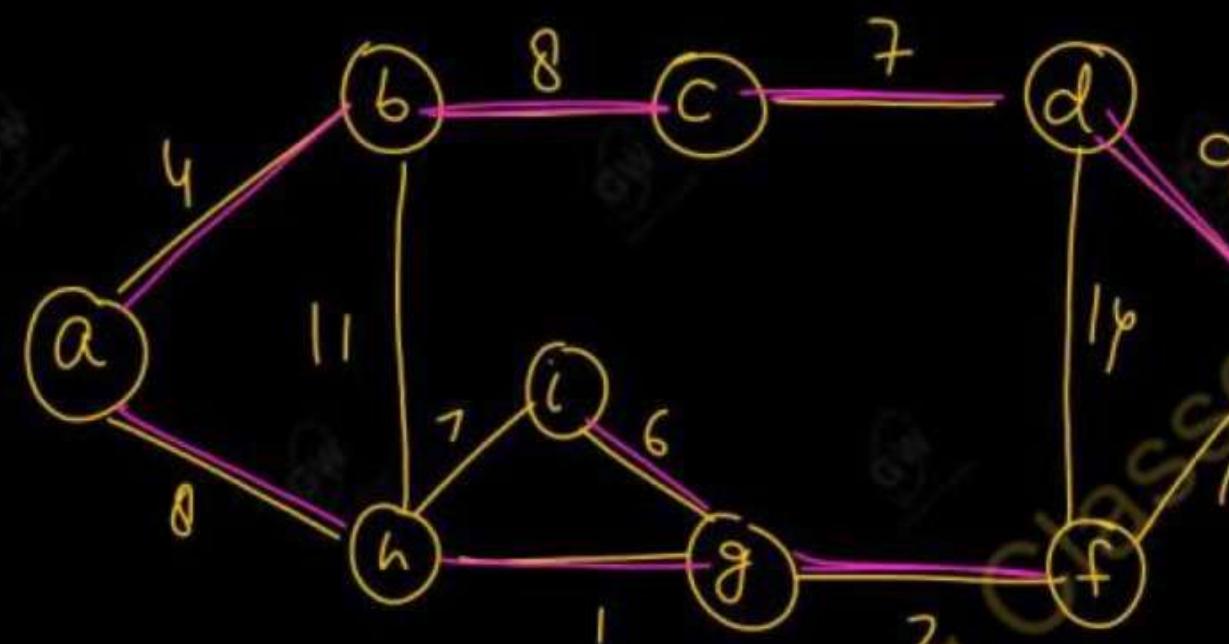
□ Now, edge (h, i) .

□ Both h and i vertices are in same set, thus it creates a cycle. So this edge is discarded.

□ Then edge (c, d) , (b, c) , (a, h) , (d, e) , (e, f) are connected as per the following:



- In (e, f) edge both end point e and f exist in same tree so discarded this edge.
- Then (b, h) edge, it also creates a cycle.
- After that edge (d, f) and the final spanning tree is shown as in dark lines.

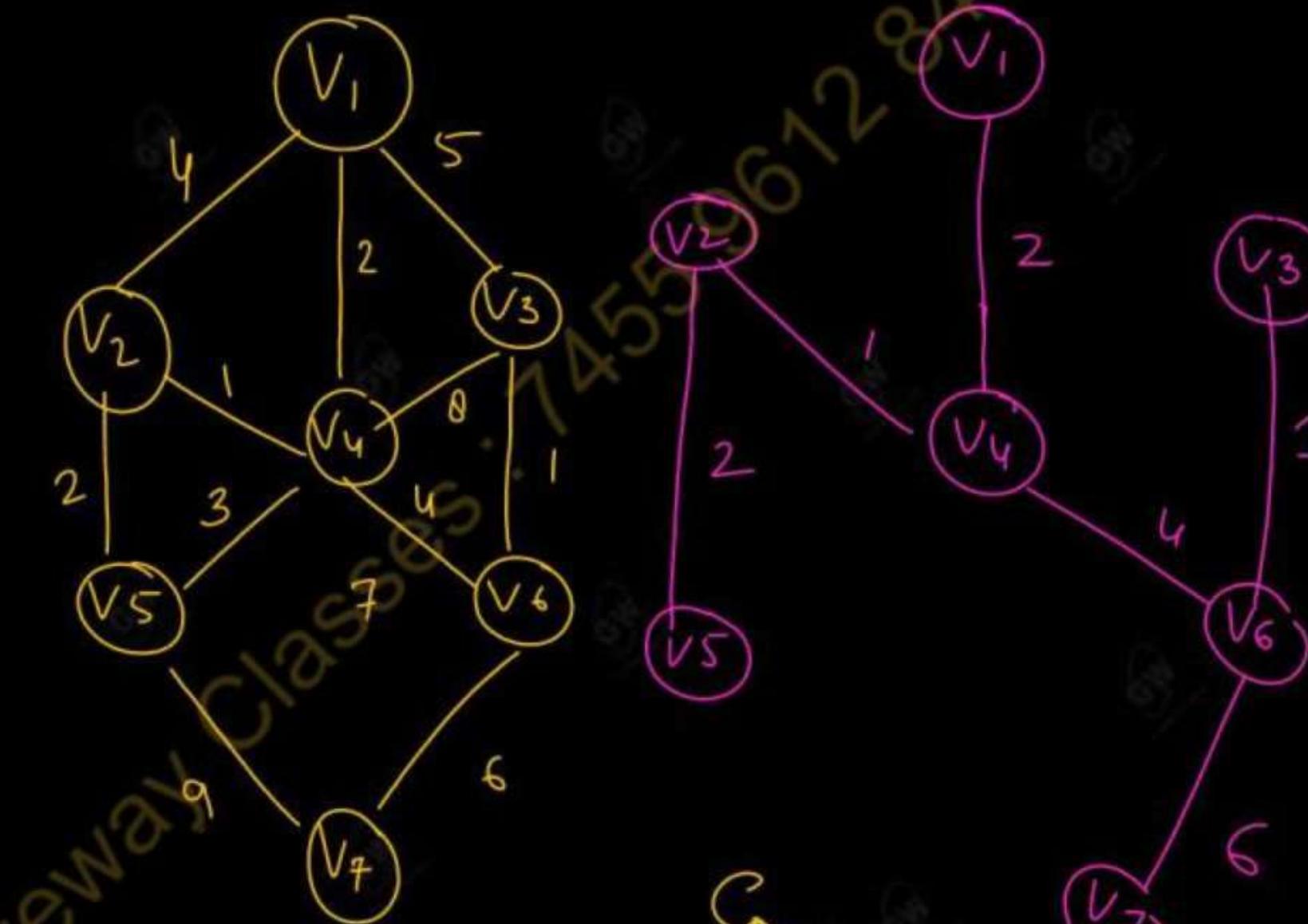


Cost = 45
Minimum Cost Spanning Tree

1. for each $u \in V[G][u]$
2. do $u \text{ key}[u] \leftarrow \infty$
3. $N[u] \leftarrow NIL$
4. $\text{key}[r] \leftarrow Q$
5. $Q \leftarrow V[G]$
6. While $Q \neq \phi$
7. do $u \leftarrow EXTRACT - MIN(Q)$
8. For each $u \in Adj[u]$
9. Do if $v \in Q$ and $w(u, v) < \text{key}[v]$
10. then $u(v) \leftarrow u$
11. $\text{Key}[v] \leftarrow w(u, v)$

EXAMPLE-2. Find the minimum spanning tree using Prim's algorithm for the given graph.**SOLUTION :-**

Edges Cost

 $V_1 - V_2 = 4 \times \text{cycle}$ $V_1 - V_4 = 2 \times \text{choosed}$ $V_1 - V_3 = 5 \times \text{cycle}$ $V_4 - V_2 = 1 \times \text{choosed}$ $V_4 - V_5 = 3 \times \text{cycle}$ $V_4 - V_6 = 4 \times \text{choose}$ $V_4 - V_3 = 8$ $V_2 - V_1 = 4 \times \text{cycle}$ $V_1 - V_5 = 2 \times \text{choosed}$ $V_5 - V_4 = 3 \times \text{cycle}$ $V_5 - V_7 = 9$ 

$$\text{Cost} = 16$$

Ans

$V_6 - V_3 = 1 \times \text{choosed}$
 $V_6 - V_2 = 5 \times \text{choosed}$
 $V_3 - V_1 = 5 \times \text{cycle}$
 $V_3 - V_4 = 8$

Prim's MST

$v_1 - v_2 = 4 \times \text{cycle}$

$v_1 - v_4 = 2 \text{ chosen}$

$v_1 - v_3 = 5 +$

$v_4 - v_2 = 1 \text{ chosen}$

$v_4 - v_5 = 3 \checkmark$

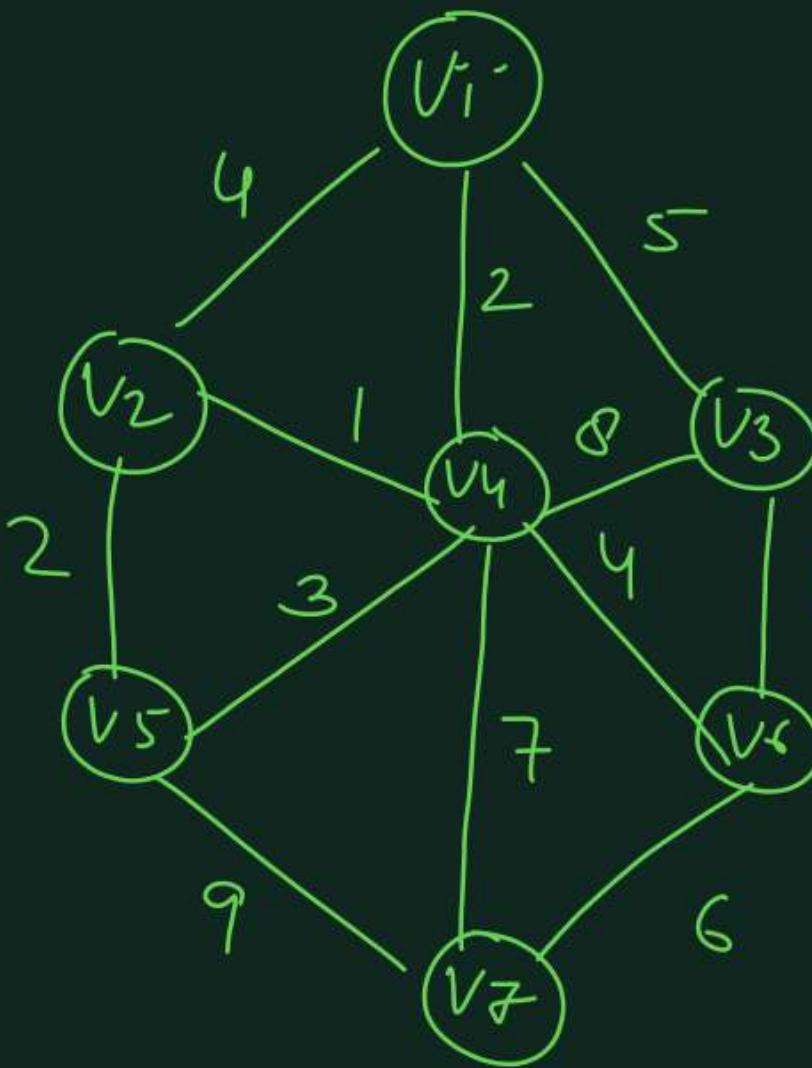
$v_4 - v_7 = 7$

$v_4 - v_6 = 4 \text{ chosen}$

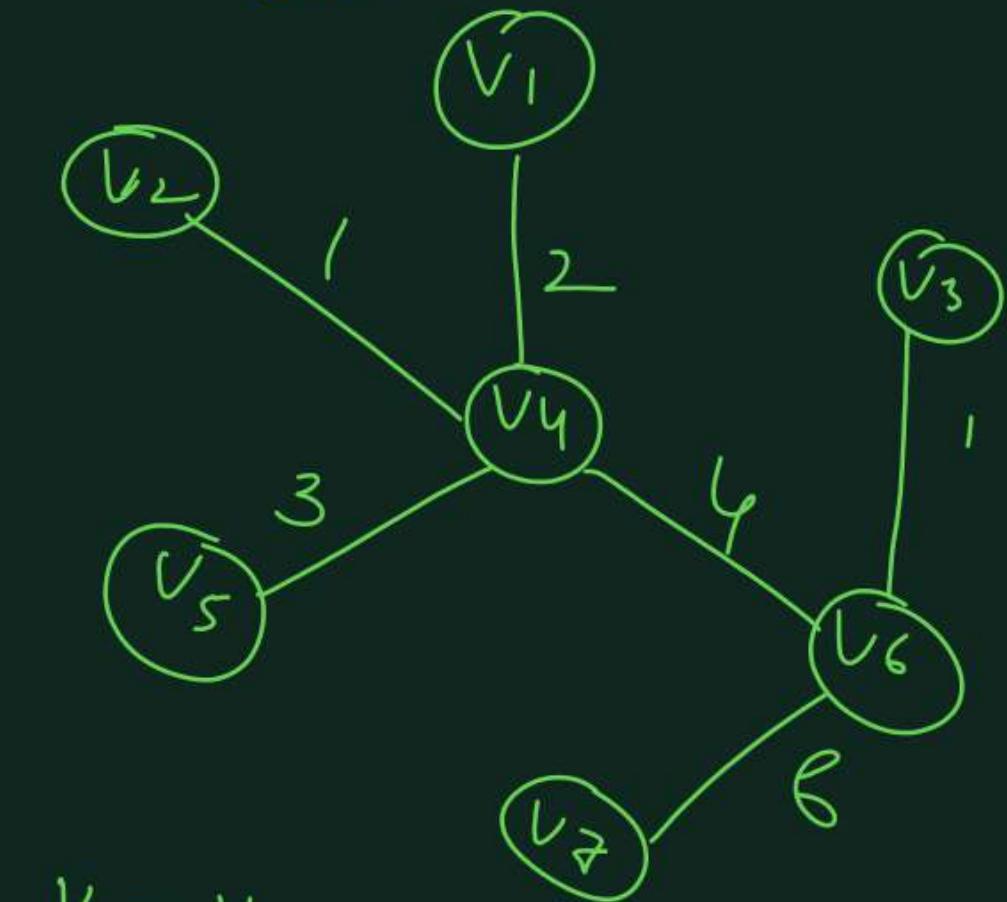
$v_4 - v_3 = 8$

$v_2 - v_1 = 4 \times \text{cycle}$

$v_4 - v_5 = 2 \text{ chosen}$



An MST
Cost =



$v_5 - v_2 = 2 \times \text{cannot be chosen due to cycle}$

$v_5 - v_7 = 9$

$v_6 - v_3 = 1 \text{ chosen}$

$v_3 - v_7 = 6 \text{ chosen}$

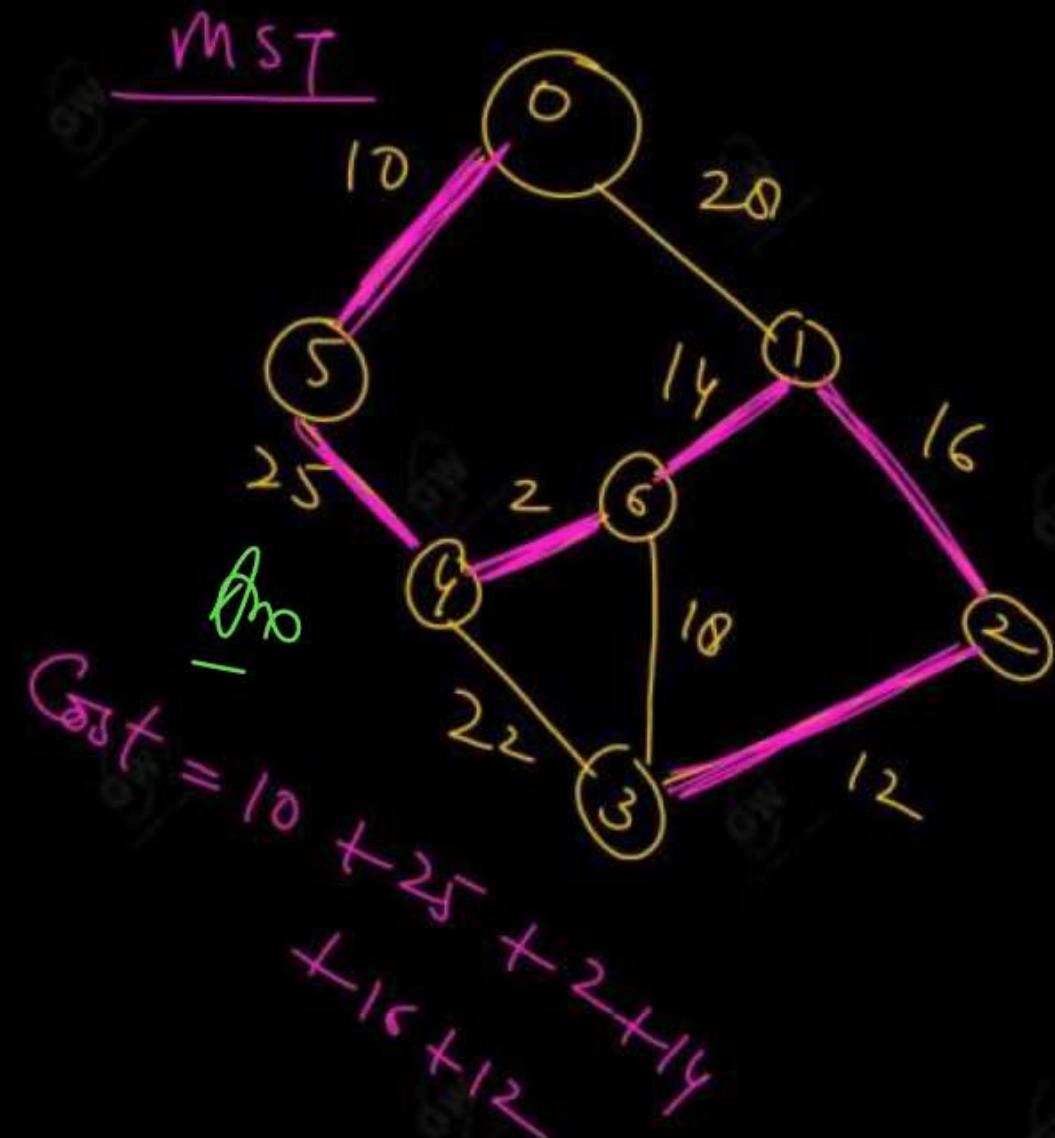
EXAMPLE-3. Generate minimum cost spanning tree for the following graph using

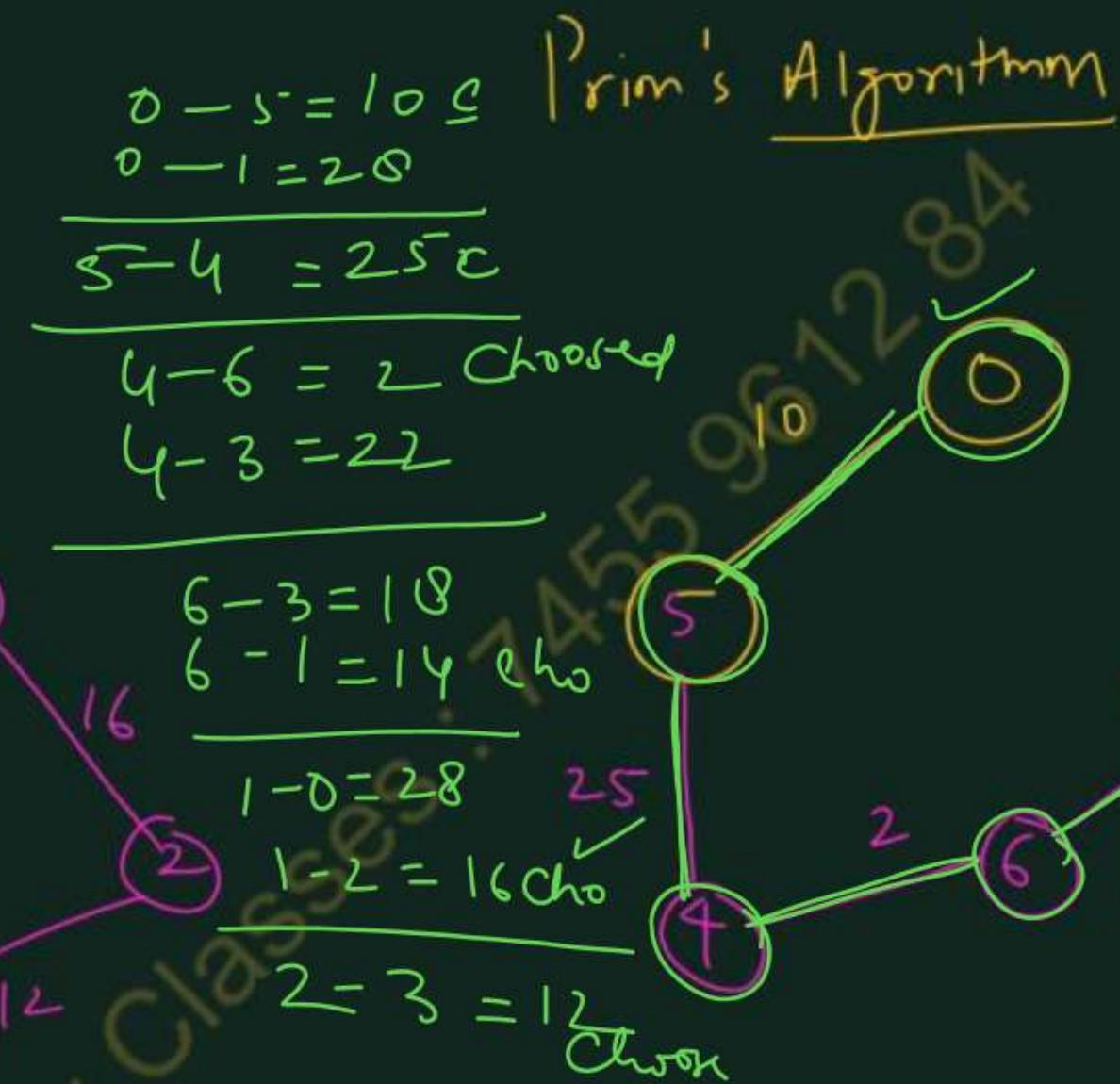
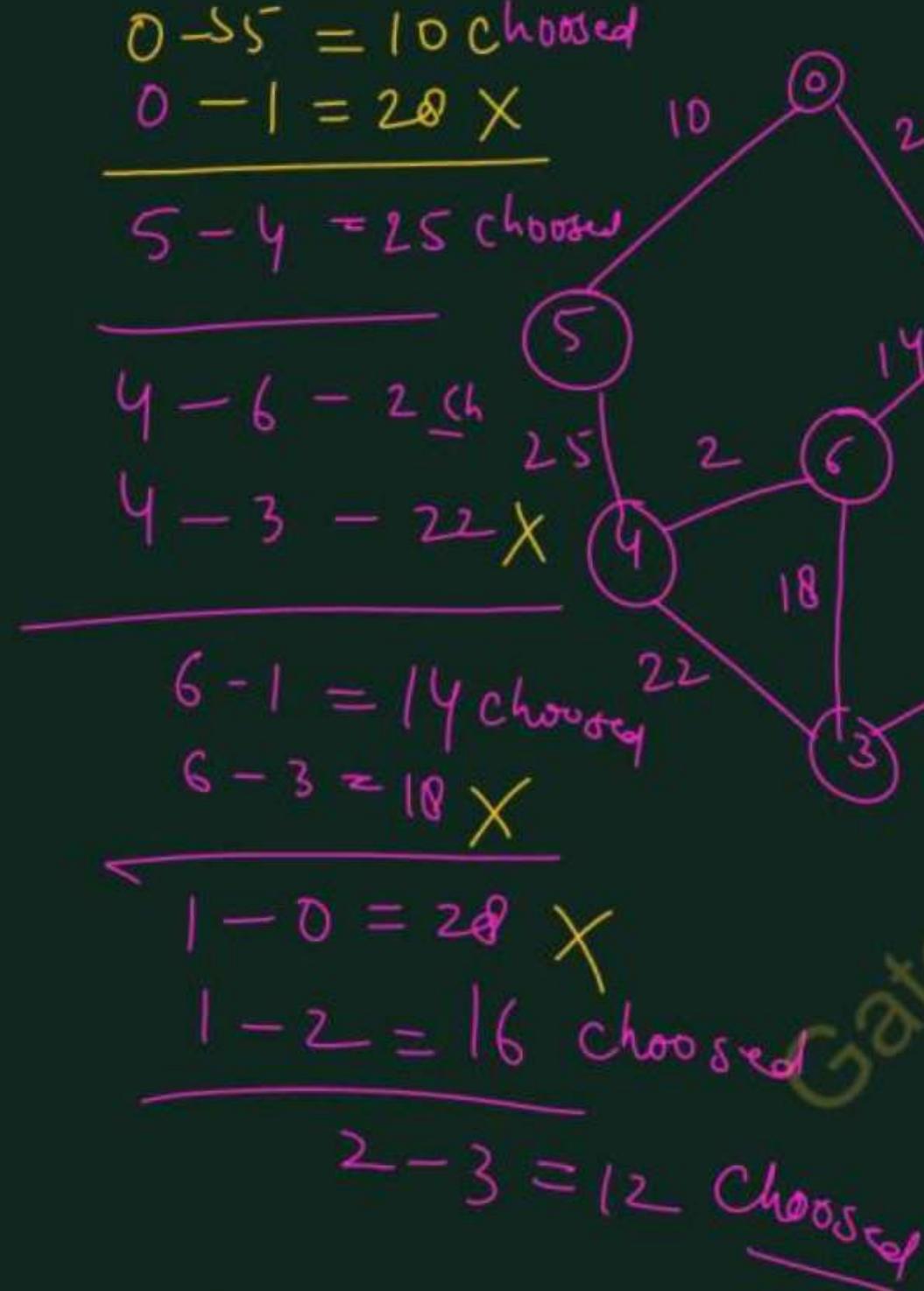
- (i) Kruskal's algorithm and
(ii) Prim's algorithm.

4 - 6	2 ✓✓
0 - 5	10 ✓✓
2 - 3	12 ✓✓
1 - 6	14 ✓✓
1 - 2	16 ✓✓
6 - 3	18 ✓✓
3 - 4	22 X cycle
4 - 5	25 X cycle
0 - 1	20 X cycle



(i) Kruskal's Algo Thm
(mst)



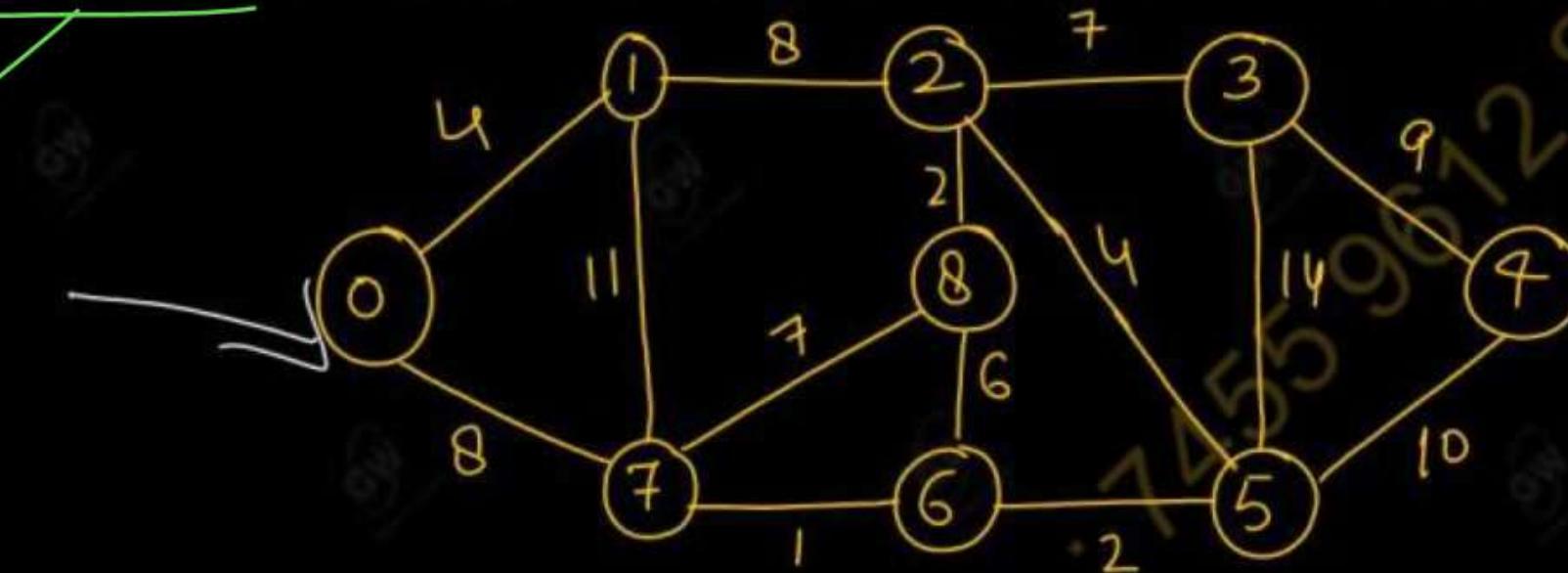


$$\begin{aligned}
 \text{Cost} &= \text{MST} \\
 &= 10 + 25 + 2 + 14 + 18 \\
 &= 70
 \end{aligned}$$

$$\begin{aligned}
 3-4 &= 22 \times \\
 3-6 &= 10 \times
 \end{aligned}$$

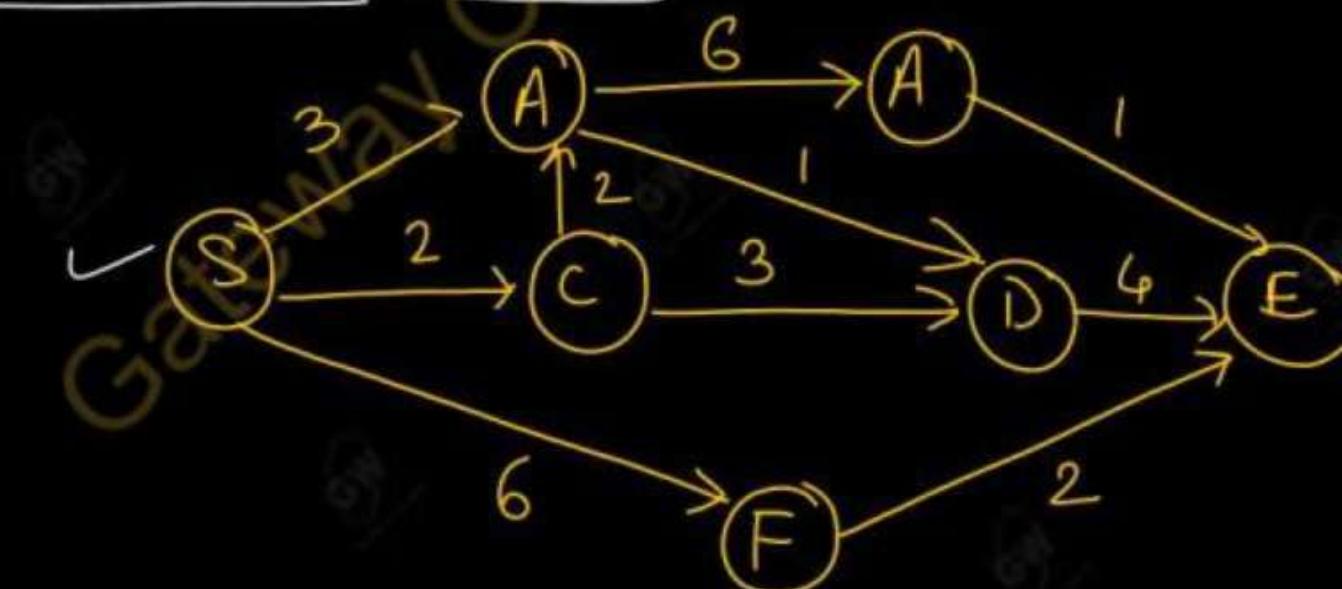
Q.12 Use Dijkstra's algorithm to find the shortest paths from source to all other vertices in the following graph.

2021-22, 10 Marks



Q.13 Write the Dijkstra's algorithm for shortest path in a graph and also find the shortest path from 'S' to all remaining vertices of graph in the following graph:

2022-23, 10 Marks



DIJKSTRA'S ALGORITHM

- Dijkstra's algorithm, named after its discoverer, Dutch computer scientist Edsger Dijkstra, is a greedy algorithm that solves the single-source shortest path problem for a directed graph / undirected graph.
- $G = (V, E)$ with non-negative edge weights, i.e., we assume that $w(u, v) \geq 0$ each edge $(u, v) \in E$.
- Dijkstra's algorithm maintains a set S of vertices whose final shortest-path weights from the source s have already been determined.
- That is, for all vertices $u \in S$. we have $d(v) = \delta(s, v)$. The algorithm repeatedly selects the vertex $u \in V - S$ with the minimum shortest-path estimate, inserts u into S , and relaxes all edges leaving u .
- We maintain a priority queue Q that contains all the vertices in $u - s$, keyed by their d values. Graph G is represented by adjacency lists.

NOTE:-

- * There must be a source vertex.
- * This algorithm finds the shortest path from a single source vertex to all other vertices.
- * If source node is not mentioned in the question then choose any vertex as a source vertex.
- * All vertices must have nonnegative weights. (if weights are negative then Bellman Ford algorithm is more suitable).

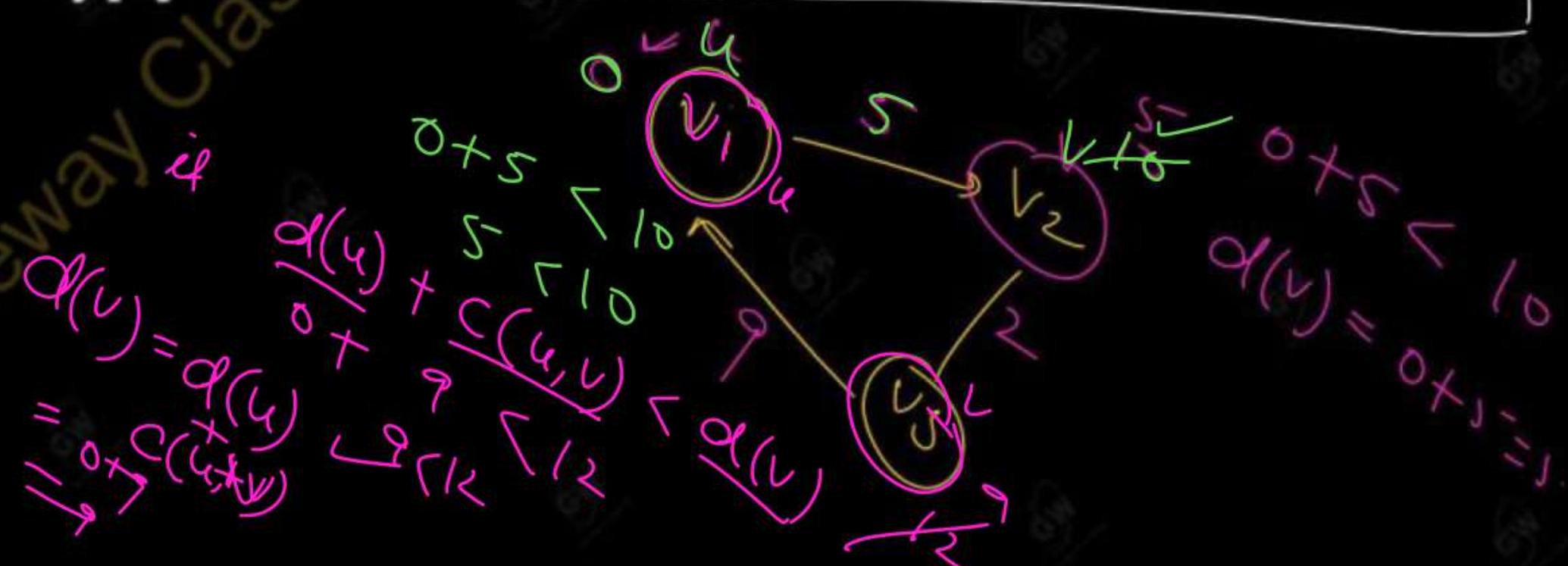
ALGORITHM : DIJKSTRA (G, W, S)

1. INITIALIZE - SINGLE - SOURCE (G, s)
2. $s \leftarrow \phi$
3. $Q \leftarrow V(G)$
4. While $Q \neq \phi$
5. Do $u \leftarrow \text{EXTRACT-MIN } (Q)$
6. $S \leftarrow S \cup \{u\}$
7. For each vertex $v \in \text{Adj}[u]$
8. Do RELAX (u, v, w)

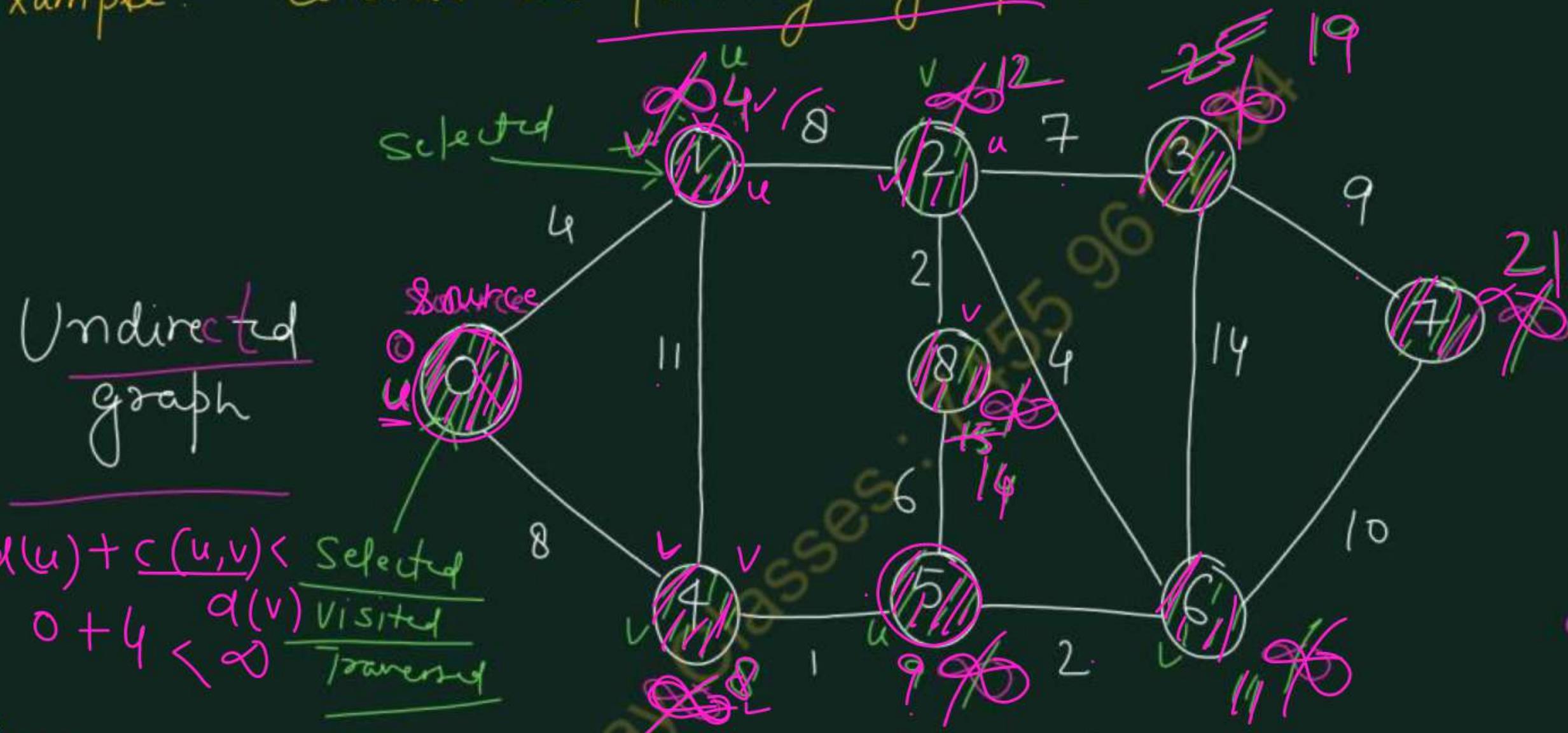


if $[d(u) + \text{cost}(u, v) < d(v)]$

then $d(v) = d(u) + \text{cost}(u, v)$



Example :- Consider the following graph :-



if $d(u) + c(u,v) < \frac{\text{Selected}}{\text{Visited}}$
 $0 + 4 < \infty \frac{d(v)}{\text{Traversed}}$

$d(v) = d(u)$ say Source Vertex = 0 ie find the shortest path
 $= 0 + c(u, v)$ from 0 to all other vertices

Solution :-

- * The distance from Vertex 0 to 0 is 0 & say distance for all other vertex is ∞ .
- * The vertex 1 and 4 are connected directly to vertex 0 so we can find out the distance from vertex 0 to vertex 1 and vertex 0 to vertex 4.
- * Suppose calculating distance from 0 to 1, say 0 is u and 1 is v. Now consider the formula -

if $[d(u) + \text{cost}(u, v) < d(v)]$

i.e. $0 + 4 < \infty$

$(4 < \infty)$ Tree

then $d(v) = d(u) + \text{cost}(u, v)$

i.e. $d(v) = 4$

* Now update the distance at vertex 1, ~~2~~ to 4.

* Now calculate the distance from 0 to 4. Now vertex 0 is u and vertex 4 is v.

Now apply the formula -

if $\left[\underline{d(u)} + \underline{\text{cost}(u,v)} < \underline{d(v)} \right]$

i.e. $0 + 0 < \infty$
 $(0 < \infty)$ True

then $d(v) = d(u) + \text{cost}(u,v)$
 $= 0$

* Now update the distance at vertex 4, ∞ to 8.

* In this way, vertex 0 is visited / selected vertex.
mark vertex 0 as selected vertex in the graph

- * Now check the distance of all remaining vertices
(i.e. except vertex 0, as it is already visited / selected)
 - & select the shortest distance. The shortest distance
is 4 at vertex 1.
- * Mark vertex 1 as selected vertex in the graph.
- * Now from vertex 1 we can update the distance from
vertex 1 to 2 and 1 to 4. we will not update 1 to 0
as it has already been visited.

* Now update distance from 1 to 2 i.e

if $d(u) + \text{cost}(u, v) < d(v)$

4 + 8 \rightarrow ∞

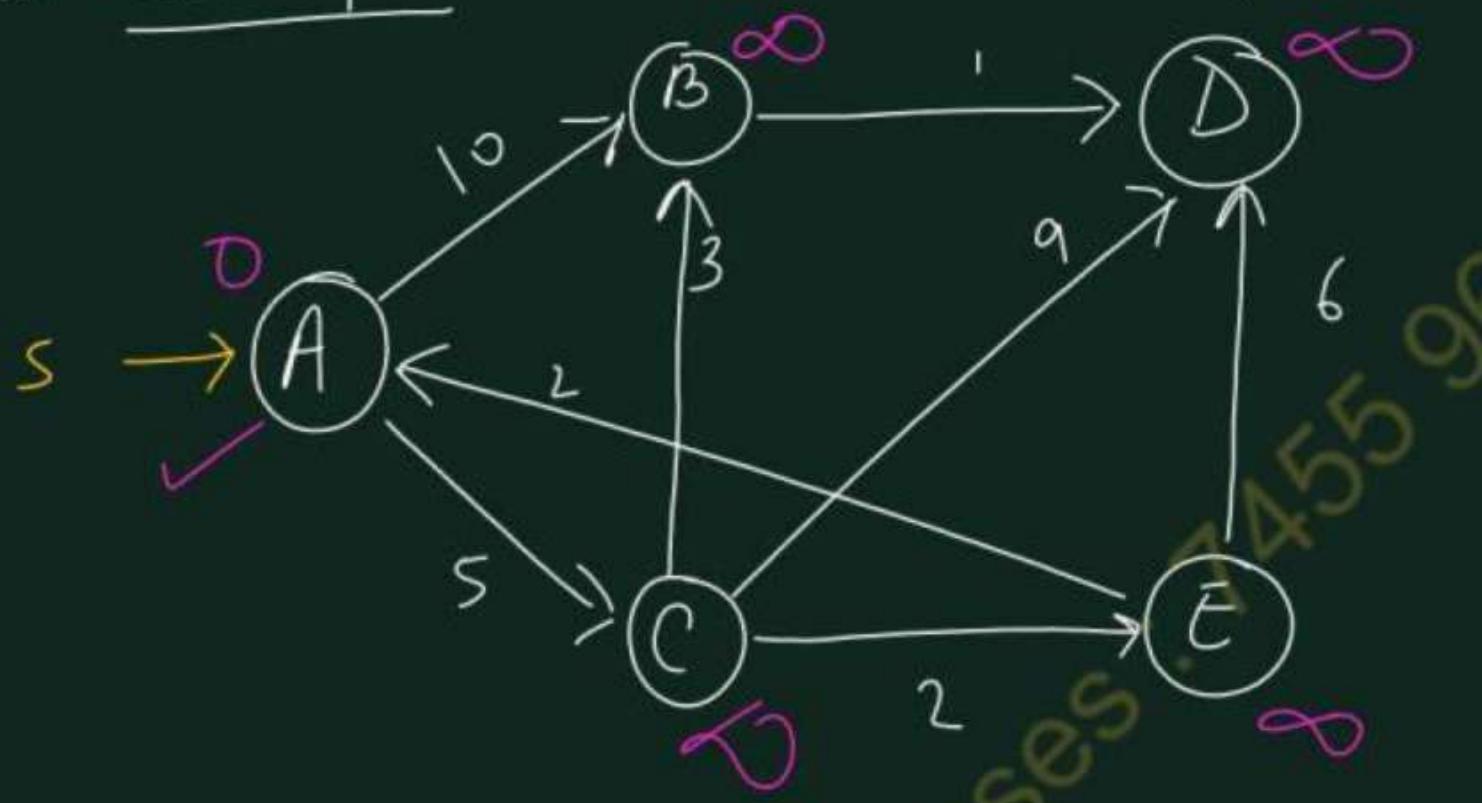
12 \leftarrow ∞ True

then

$d(v) = d(u) + \text{cost}(u, v)$

$d(v) = 12$

Another example. Consider the following directed graph -



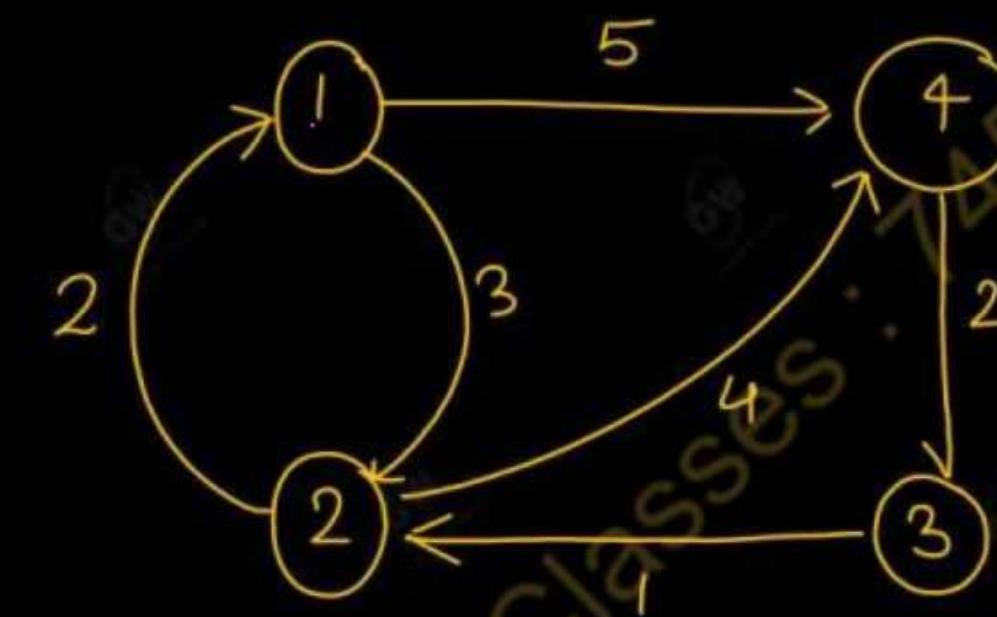
i.e. shortest distance
from A to B = 8 ✓
to C = 5 ✓
to D = 9 ✓
& A to E = 7 ✓

Trove
Selected
Visited

	A	B	C	D	E
A	0	∞	∞	∞	∞
C	10	5	∞	∞	
E	8		14	13	
B	8	✓			
D				9	

Q.14 Write and explain Floyd Warshall's algorithm to find the all-pair shortest path. Use the Floyd Warshall algorithm to find the path among all the vertices in the given graph:

2022-23, 10 Marks



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

ALGORITHM: FLOYD – WARSHELL'S (W)

Step 1. $n \leftarrow \text{rows}[W]$

Step 2. $D^{(0)} \leftarrow W$ {weights, weighted graph \Leftarrow distance matrix}

Step 3. $\text{for } k \leftarrow 1 \text{ to } n$

Step 4. $\text{do for } i \leftarrow 1 \text{ to } n$

Step 5. $\text{do for } j \leftarrow 1 \text{ to } n$

Step 6. $\boxed{\text{do } d_{ij}^{(k)} \leftarrow \min \{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \} \text{ for } i, j \in \{1, 2, \dots, n\}}$

Step 7. $\text{return } D^{(n)}$

that results each iteration.

Distance
matrix

$$\checkmark \boxed{D^0} = \begin{bmatrix} 0 & \infty & \infty & \infty & -1 & \infty \\ \infty & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & \infty & \infty & -8 \\ -4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 1 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{bmatrix}$$

all Pair shortest Path



Directed graph
weighted graph

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

So, the $D^{(1)}$ matrix is -

$$D^{(1)} =$$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	∞	2	0	∞	∞	-8
4	-4	∞	∞	0	-5	∞
5	∞	7	∞	∞	0	∞
6	∞	5	10	∞	∞	0

$$\begin{aligned} D^{(2,5)} &= \min \left\{ \frac{D^0(2,5)}, \frac{D^0(2,1) + D^0(1,5)}{1+(-1)} \right\} \\ &= \min \left\{ \infty, 0 \right\} \\ &= 0 \end{aligned}$$

$$\begin{aligned} D^{(2,3)} &= \min \left\{ \frac{D^0(2,3)}, \frac{D^0(2,1) + D^0(1,3)}{1+(-1)} \right\} \\ &= \min \left\{ \infty, \frac{1}{1+\infty} \right\} \\ &= \min \left\{ \infty, \infty \right\} \\ &= \infty \\ D^{(2,4)} &= \min \left\{ \frac{D^0(2,4)}, \frac{D^0(2,1) + D^0(1,4)}{1+(-1)} \right\} \\ &= \min \left\{ 2, \frac{1}{1+0} \right\} \\ &= 2 \end{aligned}$$

$$D^{(2)} =$$

	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	2	0	4	2	-8	
4	∞	∞	0	-5	∞	
5	7	∞	9	0	∞	
6	5	10	7	5	0	

$$D^2(1,3) = \min \left\{ \underbrace{D^1(1,3)}, \underbrace{D^1(1,2) + D^1(2,3)} \right\}$$

$$= \min(\infty, 0 +$$

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

$$= 0$$

$$D^2(1,5) = \min \left\{ D^1(1,5), D^1(1,2) + D^1(2,5) \right\}$$

$$= \min \left\{ -1, \infty \right\}$$

$$= -1$$

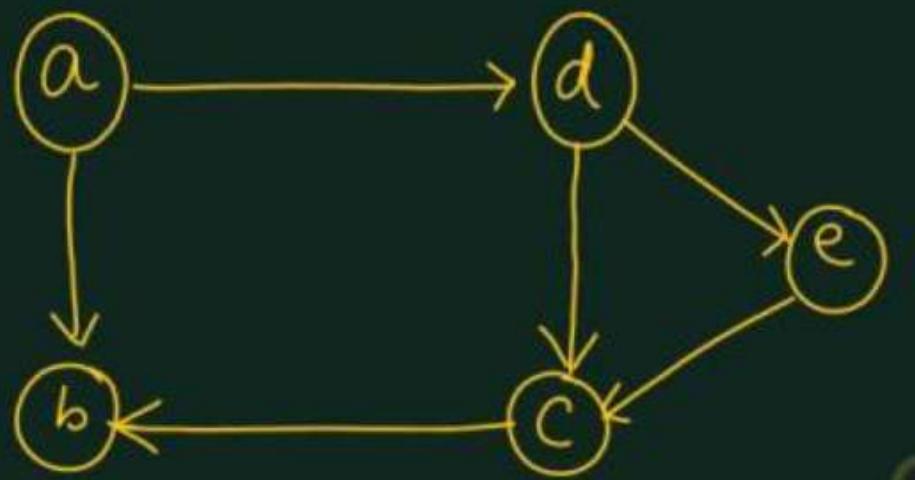
	1	2	3	4	5	6
1	0	∞	∞	∞	-1	∞
2	1	0	∞	2	0	∞
3	3	2	0	4	2	-8
4	-4	∞	∞	0	-5	∞
5	8	7	∞	9	0	∞
6	5	10	7	5	0	∞

$$\begin{aligned}
 D^3(5, 1) &= \min \left(D^2(5, 1), \right. \\
 &\quad \left. D^2(5, 3) + D^2(3, 1) \right) \\
 &= \min \left\{ 0, \infty + \right. \\
 &\quad \left. \min \left\{ 8, \infty \right\} \right\} \\
 &= \min \left\{ 8, \infty \right\} \\
 &= 8
 \end{aligned}$$

Algorithm: Transitive – closure (G)

1. $n \leftarrow v(G)$
2. for $i \leftarrow 1$ to n
3. Do for $j \leftarrow 1$ to n
4. Do if $i = j$ or $(i, j) \in E(G)$
5. then $t_{ij}^{(0)} \leftarrow 1$
6. else
7. $t_{ij}^{(0)} \leftarrow 0$
8. For $k \leftarrow 1$ to n
9. Do for $i \leftarrow 1$ to n
10. Do for $j \leftarrow 1$ to n
11. Do $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} + t_{ik}^{(k-1)} \cdot t_{kj}^{(k-1)}$

Example - find the Transitive closure of the following graph :-

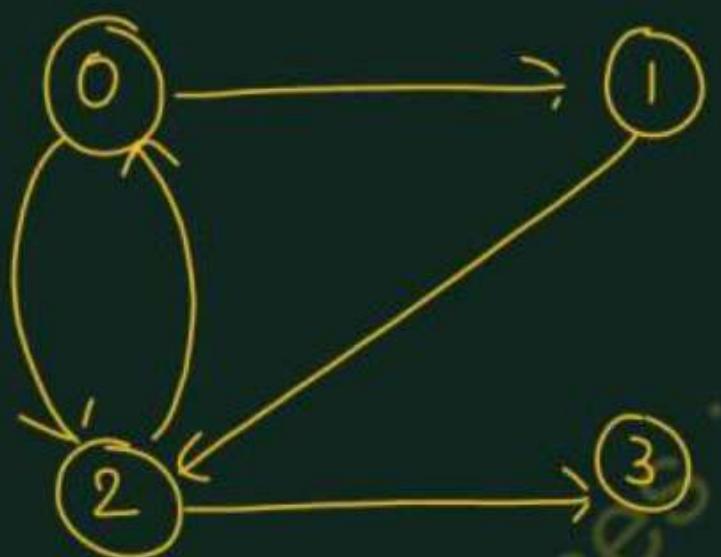


Gateway
Transitive
Classes :
This closure of
the graph

Reachability matrix

	a	b	c	d	e
a	1	0	0	1	1
b	0	1	0	0	0
c	0	0	1	1	0
d	0	0	1	1	1
e	0	0	0	0	1

Example - find the transition closure of the following graph -



reachability matrix -

		0	1	2	3
0	0	1	1	1	1
	1	1	1	1	1
2	0	1	1	1	1
3	0	0	0	1	

Transitive Closure

Download **Gateway Classes Application**
From Google Play store
Link in Description

Thank You