B.Tech.

Data Structure

Unit - 5: Lec-1

CS / IT / CS Allied

By- Dr. Kapil Kumar
M.Tech.(CSE), Ph.D.(CSE)

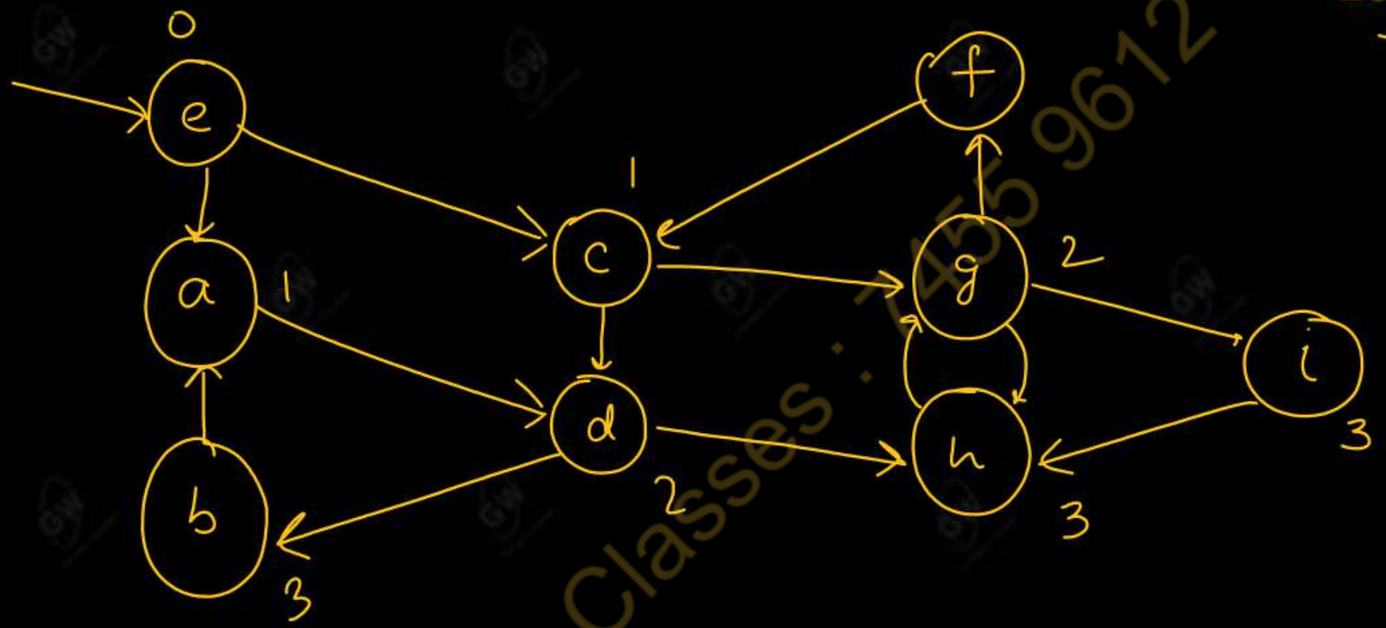
DATA STRUCTURE

Today's Target -

- ❑ Graph and Various Graph - Terminologies
- ❑ Representations of Graph (Sequential and Linked Representation)
- ❑ AKTU PYQs

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

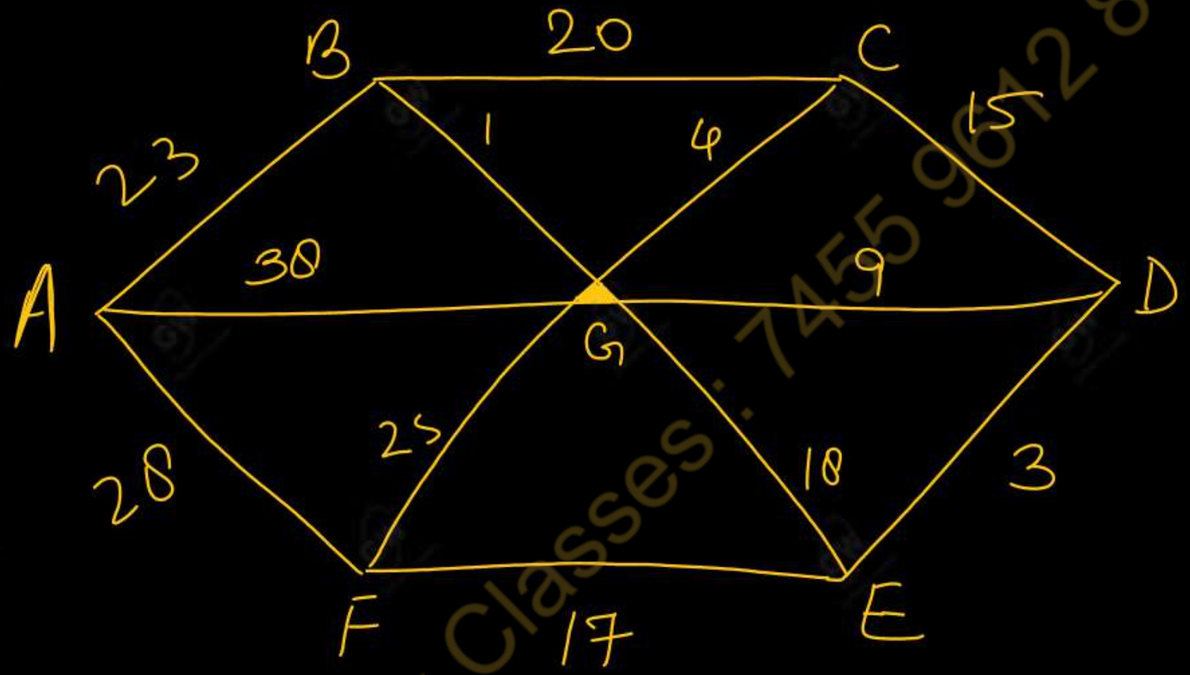
2014-15, 5 Marks



Q.2 Define connected and strongly connected graph.

2015-16, 2 Marks

Q.3 Consider the following undirected graph:



a) Find the adjacency list representation of the graph.

b) Find the minimum cost spanning tree by Kruskal's algorithm.

2015-16, 10 Marks

Q.4 When does a graph become tree. 2016-17, 2 Marks

Q.5 Define adjacency matrix with suitable example. 2016-17, 2 Marks

Q.6 List the different types of representation of graphs. 2017-18, 2 Marks

Q.7 How the graph can be represented in memory? Explain with example. 2018-19, 2 Marks

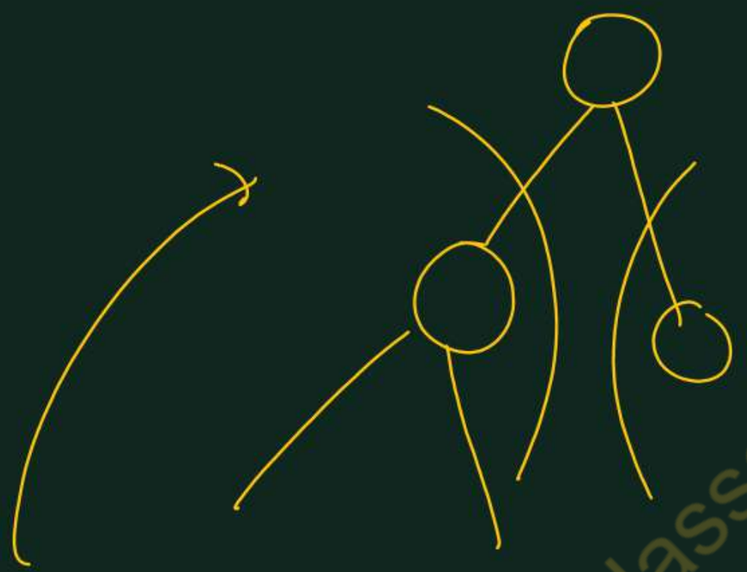
Q.8 Compare adjacency matrix and adjacency list representations of graph. 2019-20, 2 Marks

Q.9 Write a short note on adjacency multi list representation a Graph. 2020-21, 2 Marks

Q.10 Write different representation of graphs in memory. 2021-22, 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.

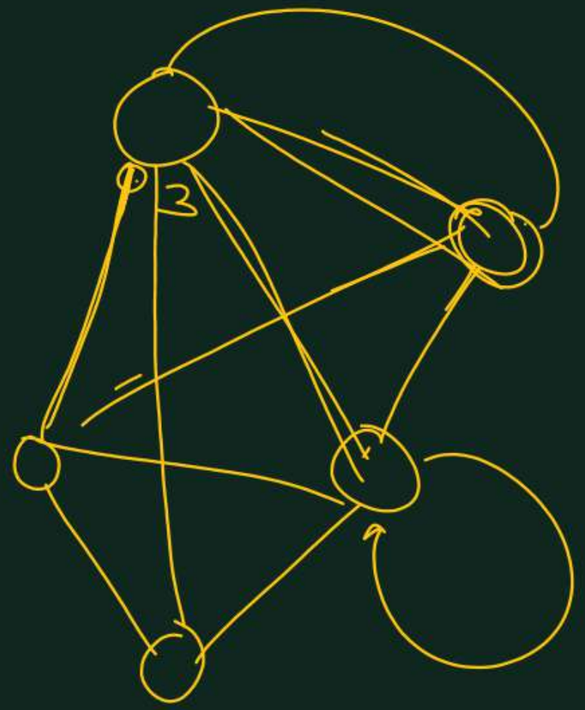


Root node

Binary = 2

I = 3

!



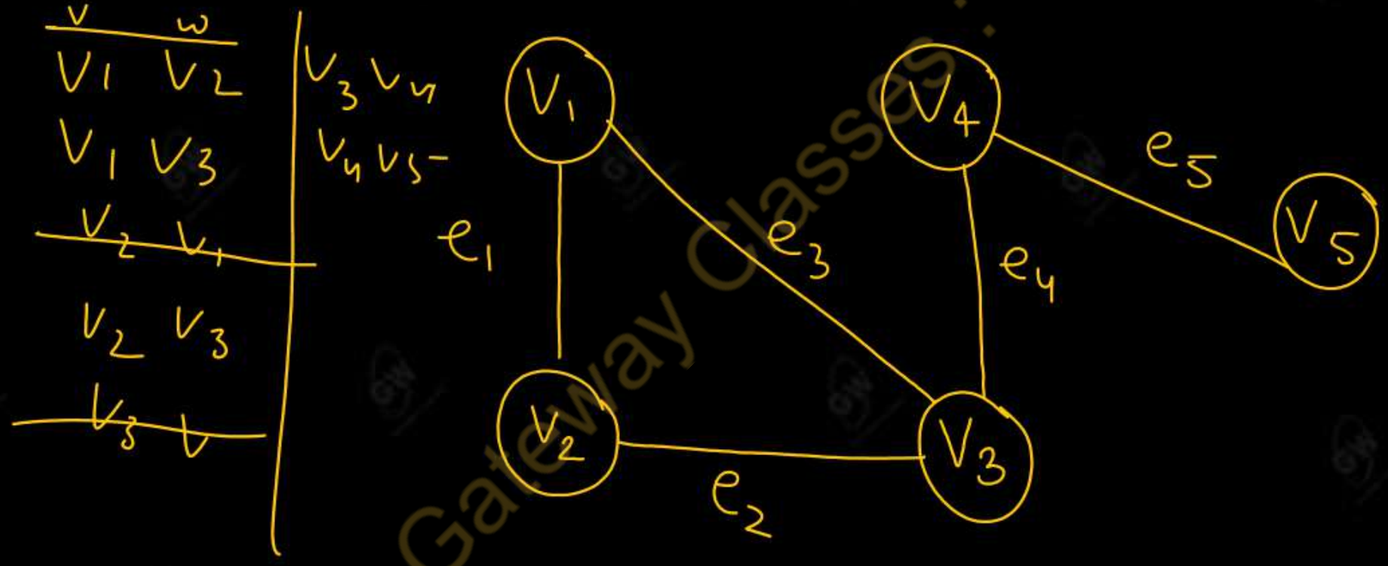
- In Computer science graphs are used to represent the flow of computation.
- Google maps uses graphs for building transportation systems, where intersection of two (or more) roads are considered to be a vertex and the road connecting two vertices is considered to be an edge, thus their navigation system is based on the algorithm to calculate the shortest path between two vertices.
- In Facebook, users are considered to be the vertices and if they are friends then there is an edge running between them.
- Facebook's Friend suggestion algorithm uses graph theory. Facebook is an example of undirected graph.
- In World Wide Web (WWW), web pages are considered to be the vertices. There is an edge from a page u to other page v if there is a link of page v on page u. This is an example of Directed graph. It was the basic idea behind Google Page Ranking Algorithm.

- In Operating System, we come across the Resource Allocation Graph where each process and resources are considered to be vertices. Edges are drawn from resources to the allocated process, or from requesting process to the requested resource. If this leads to any formation of a cycle then a deadlock will occur.
- In mapping system we use graph. It is useful to find out which is an excellent place from the location as well as your nearby location.
- In GPS we also use graphs. Global Positioning System is a satellite navigation system used to identify the ground position of an object.
- Facebook uses graphs. Using graphs suggests mutual friends. it shows a list of the f following pages, friends, and contact list.
- Microsoft Excel uses DAG means Directed Acyclic Graphs.

- In the Dijkstra algorithm, we use a graph. we find the smallest path between two or many nodes.
- On social media sites, we use graphs to track the data of the users. liked showing preferred post suggestions, recommendations, etc.
- Graphs are used in biochemical applications such as structuring of protein, DNA etc.

What is a Graph?

- A graph G consists of a set V of vertices (nodes) and a set E of edges (arcs).
- We write 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 = (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:



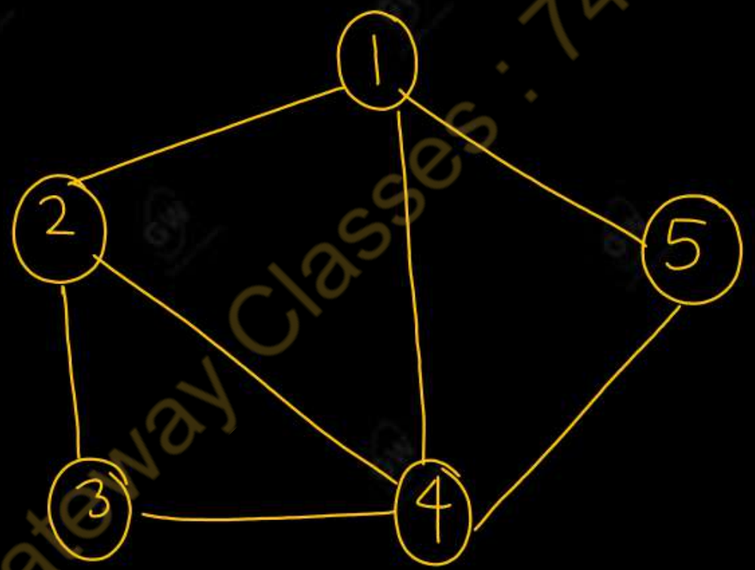
Set of $V = \{v_1, v_2, v_3, v_4, v_5\}$

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

- 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:



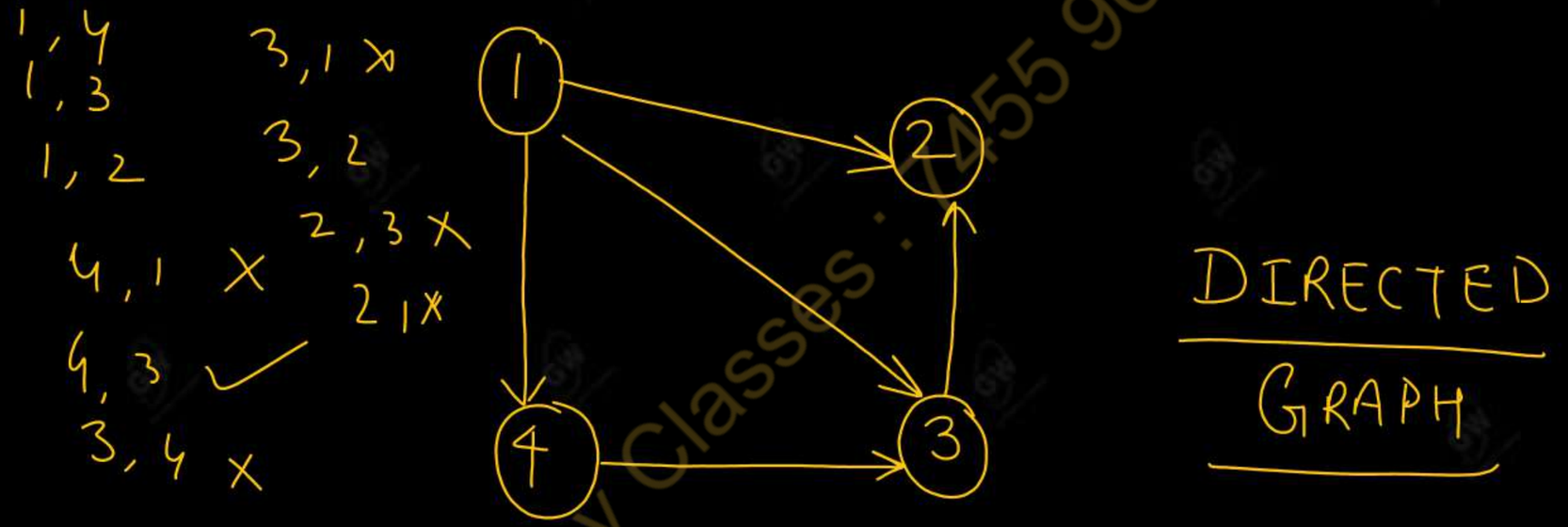
- We have numbered the nodes as 1, 2, 3, 4 and 5. Therefore

  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.

- If e = (v, w) then v is initial vertex and w is the final vertex.
- Subsequently (v, w) and (w, v) represent two different edges.
- A directed graph may be pictorially represented as shown in the following figure:



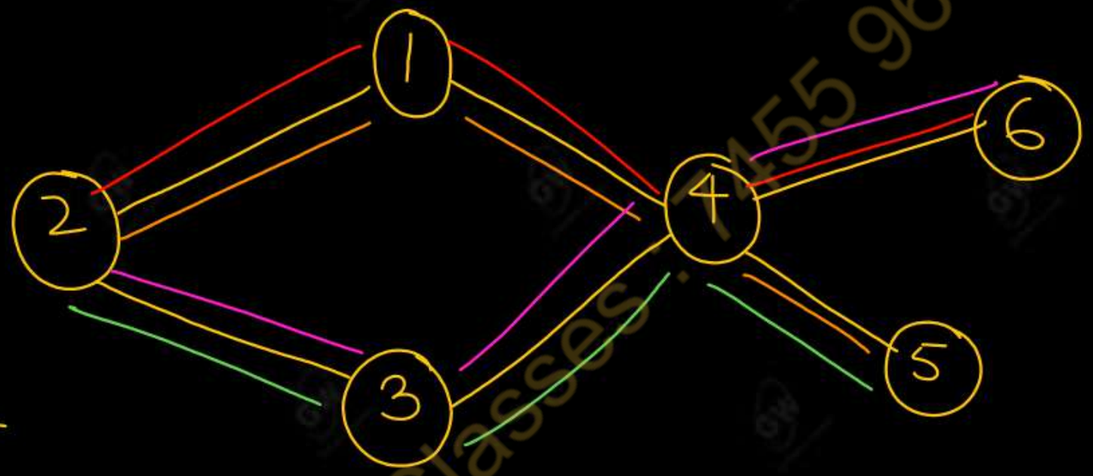
- The direction is indicated by an arrow. The set of vertices for this graph is

  V (G) = (1 , 2 , 3 , 4) and the set of edges would be

  E (G) = ((1 , 2) , (1 , 3) , (1 , 4) , (4 , 3), (3 , 2))

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)



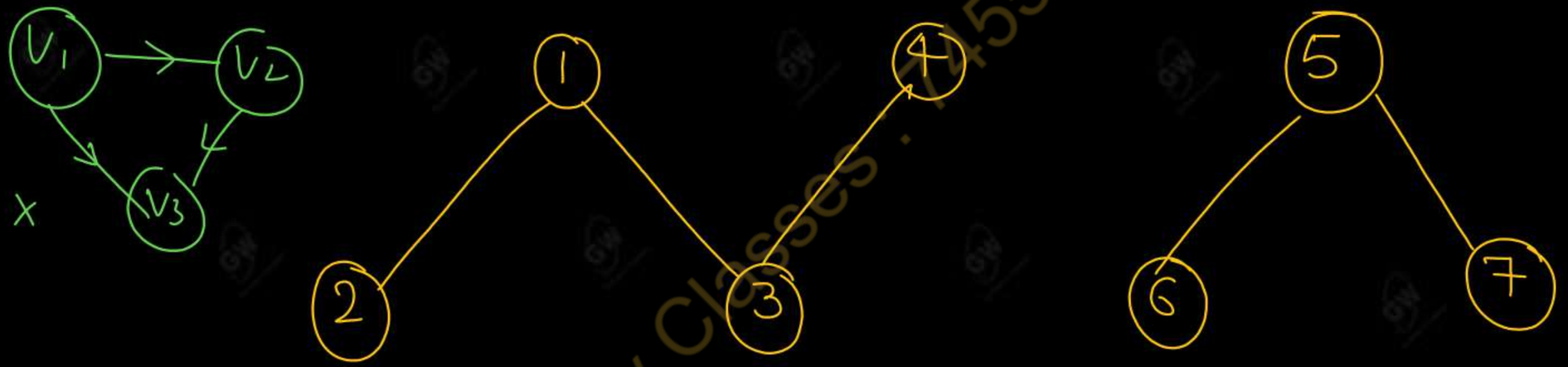
2-1-4-6
2-1-4-5
2-3-4-6
2-3-4-5

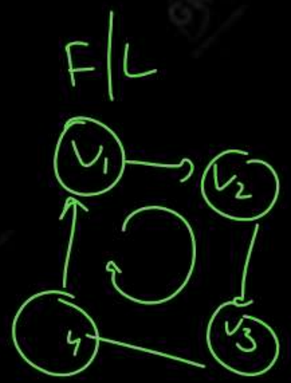
2-1-4-3-2
1-2-3-4-1

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

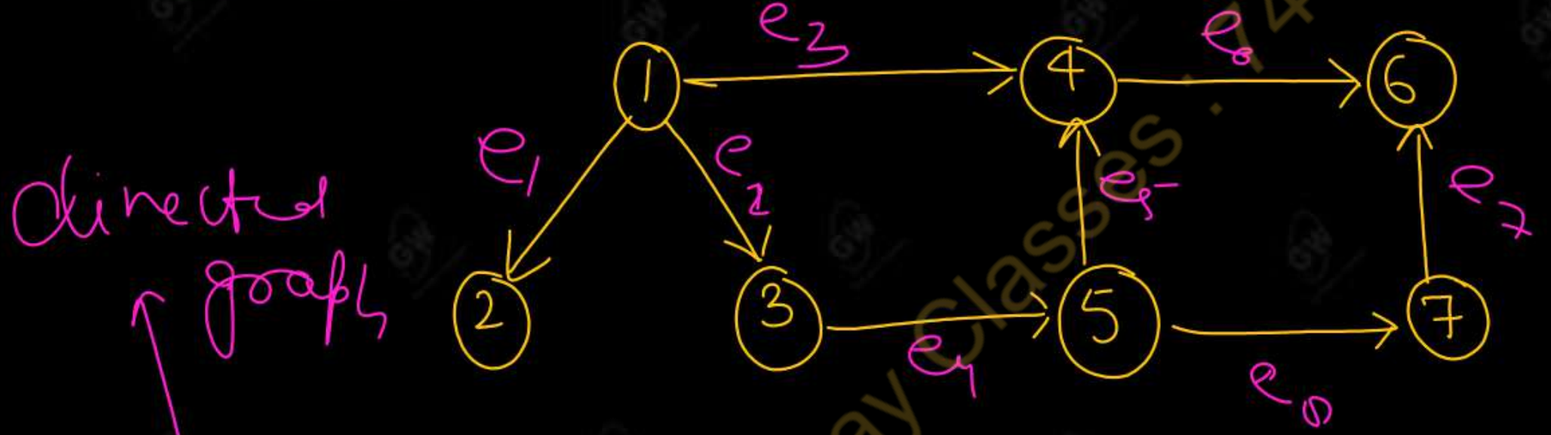
- **Path:-** A path from vertex w is a sequence of vertices, each adjacent to the next. Consider the above example again 1 , 2 , 3 is a path and 1, 4 and 3 is also a path.

- **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:





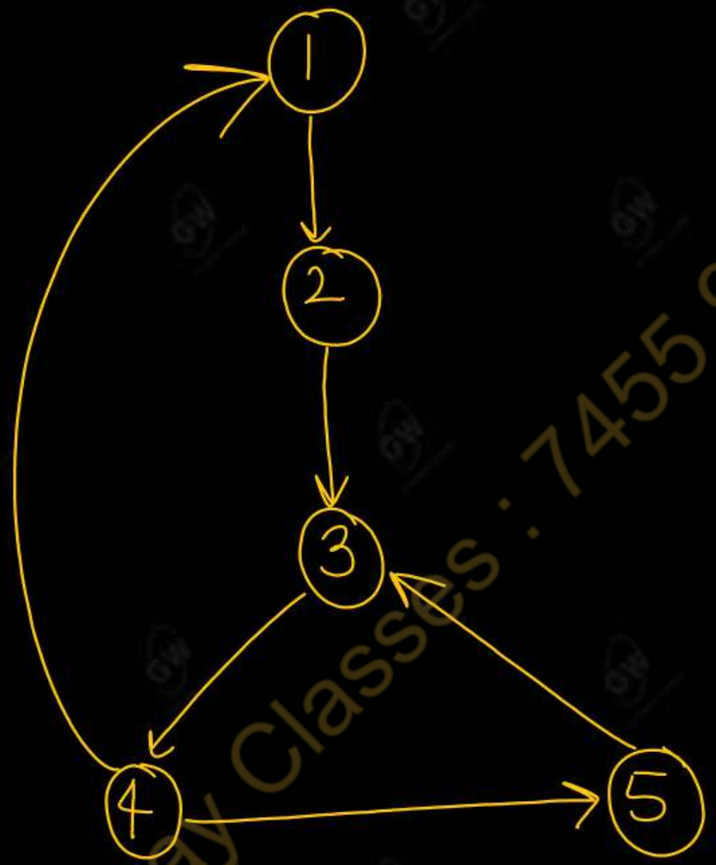
- It is an unconnected graph. You may say that these are two graphs and not one. But it is one graph having two unconnected components. Since there are unconnected components, it is an unconnected graph.
- So far we have talked about paths, cycles and connectivity of undirected graph. In a digraph the 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.

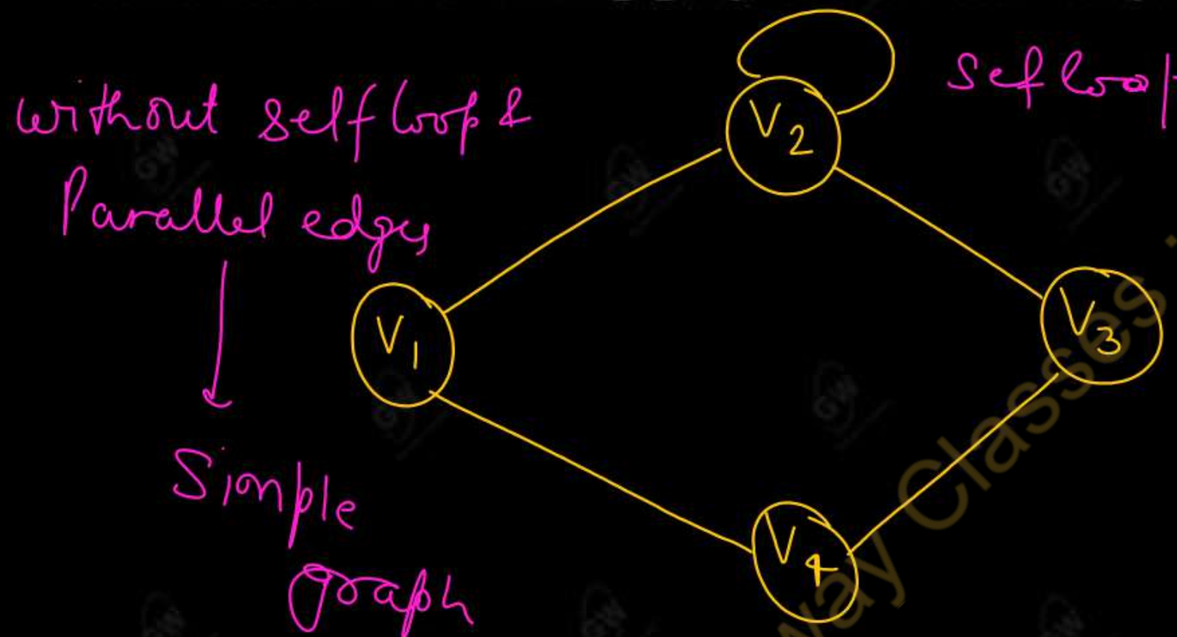
- Consider the following directed graph:

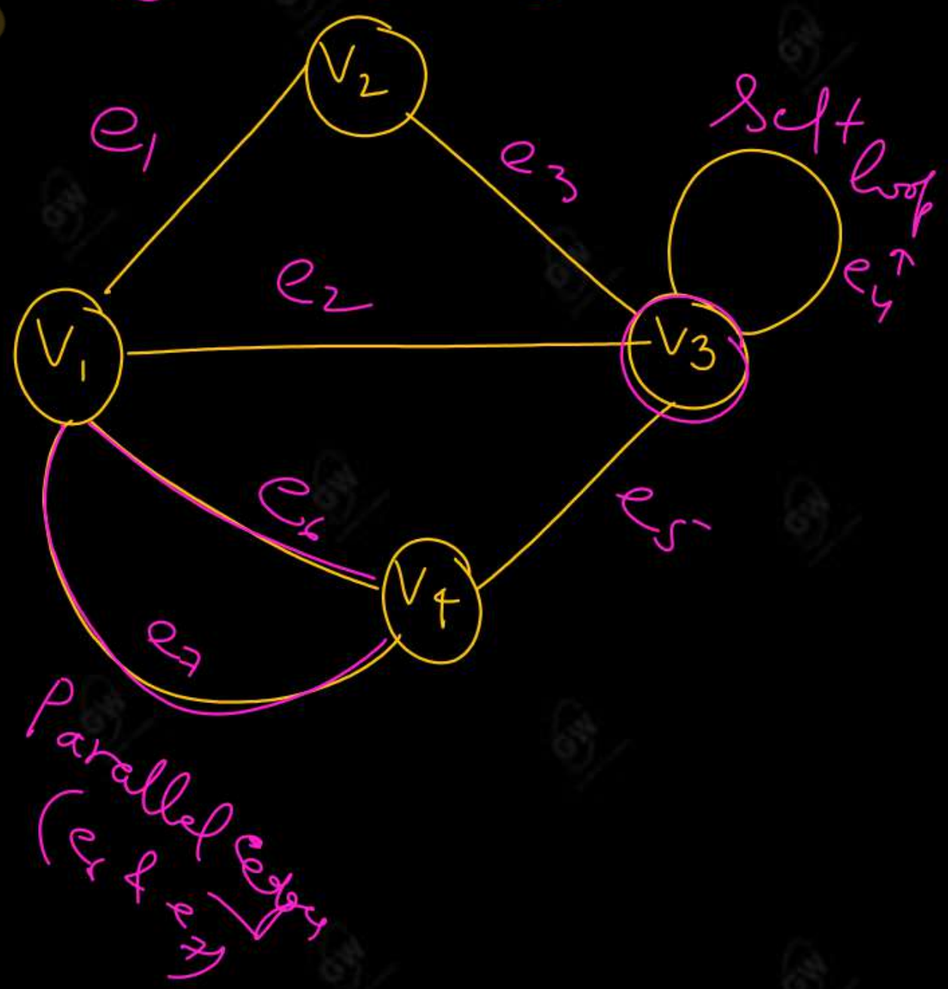
- There does not exist a directed path from vertex 1 to vertex 4, also from vertex 5 to any other vertices and so on.
- Therefore, it is a weekly connected graph. To make it strongly connected there are certain changes that are required given as in the next graph.



Self loop:-

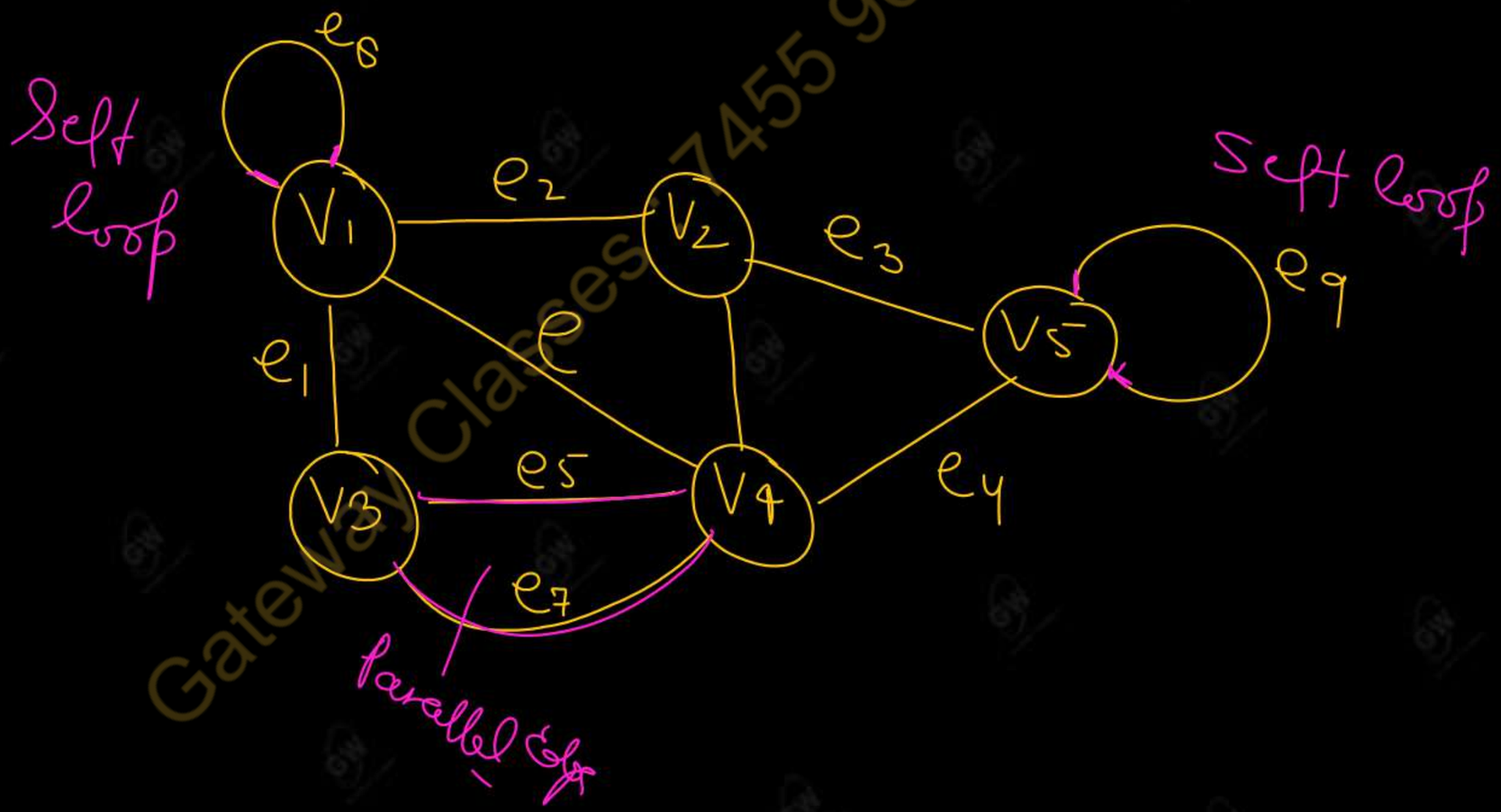
- If there is an edge whose starting and end vertices are same that is (v ,v) is an edge then it is called a self loop or simply a loop.
- Consider the following graphs with self loop:





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:



Multi–graph:-

- 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.

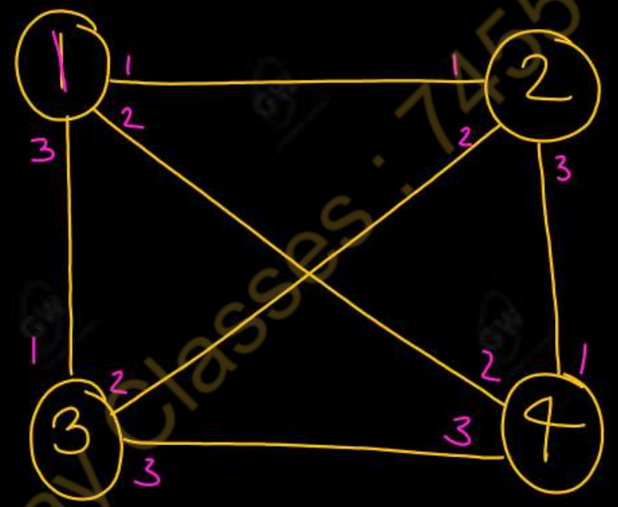
Simple–graph:-

- 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:

- This is a simple graph not a multigraph.



- In this graph every node has same degree.

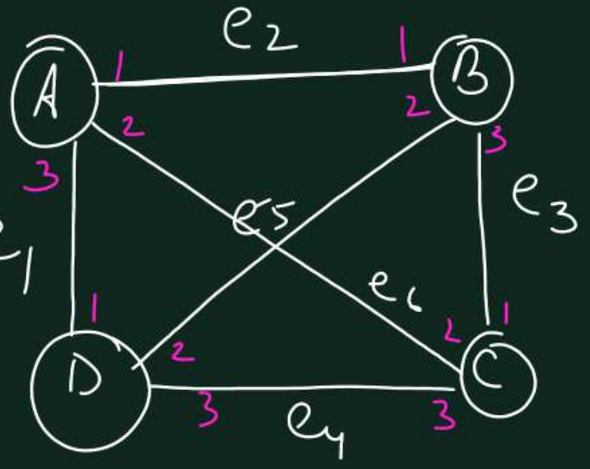
deg (1) = 3

deg (2) = 3

deg (3) = 3

deg (4) = 3

Consider the following graph -



Edges = 6

degree of every vertex =

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

n = number of nodes or vertices

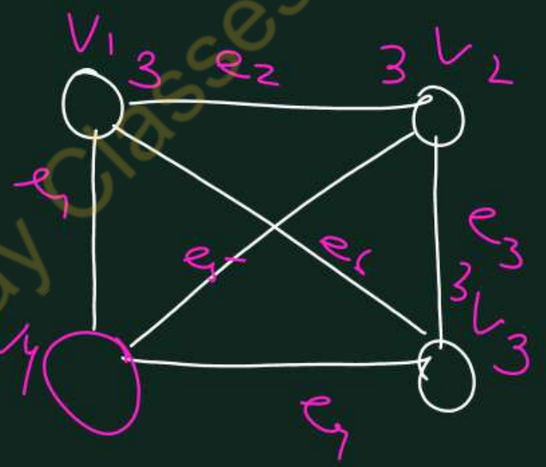
d = degree of a node

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

✓ Regular graph

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

⇒ In this graph every vertex is connected to other Vertex so it is a complete graph.

⇒ eg Degree of each node is same so it is a regular graph.

number of edges in complete graph =

$$= {}^{n}C_{2}$$

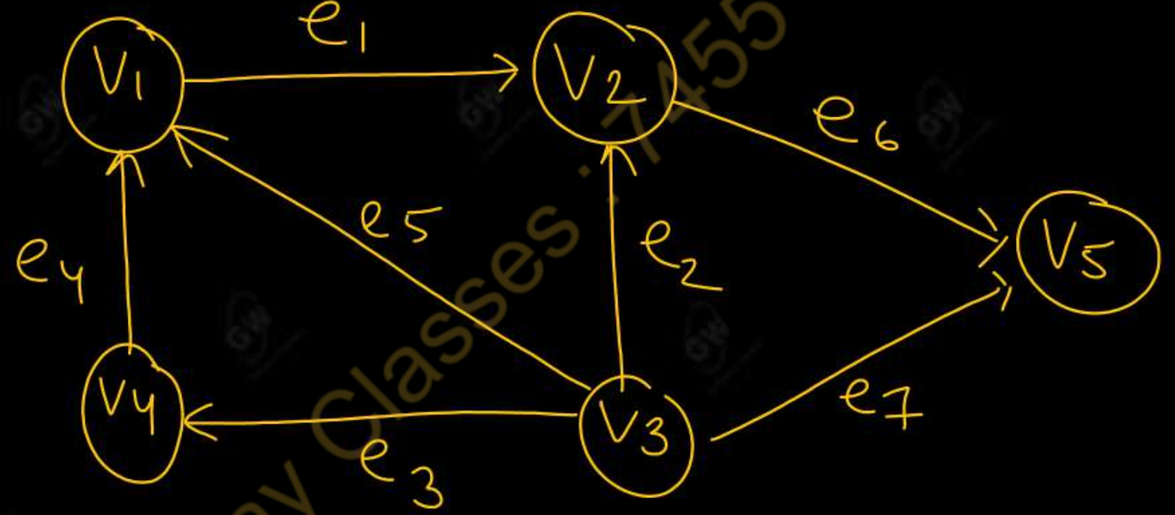
$$= \frac{n(n-1)}{2} = \frac{4 \times 3}{2} = \frac{12}{2} = 6$$

Degree of every vertex = $n-1$

$\Rightarrow$ If we will try to add a vertex in a complete graph, then it will become a multigraph, not a simple graph.

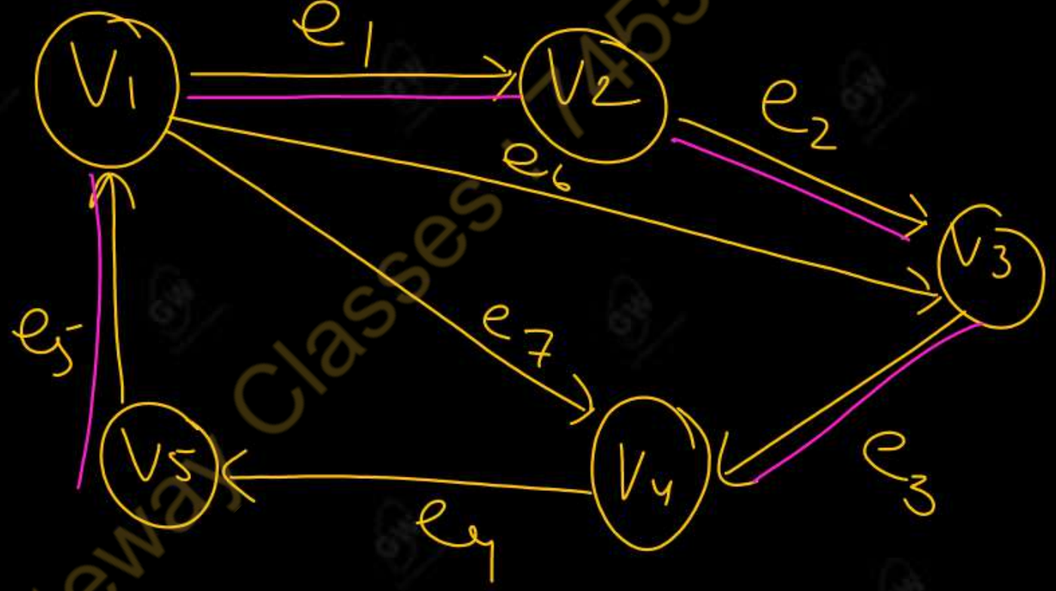
Acyclic graph:-

- If a graph (digraph) does not have any cycle then it is called as acyclic graph.
- Consider the following acyclic directed graph:



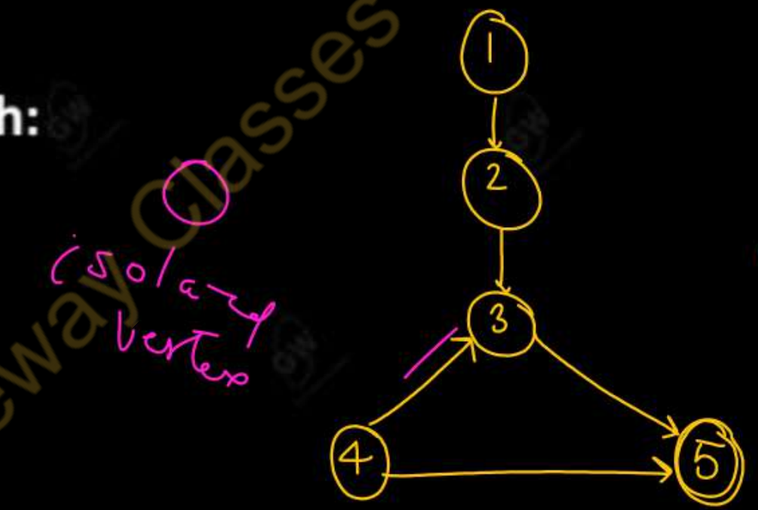
Cyclic graph:-

- If a graph have cycle then it is called as cyclic graph.
- Consider the following cyclic graph:



Degree of a vertex:-

- 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:



(Isolated vertex)

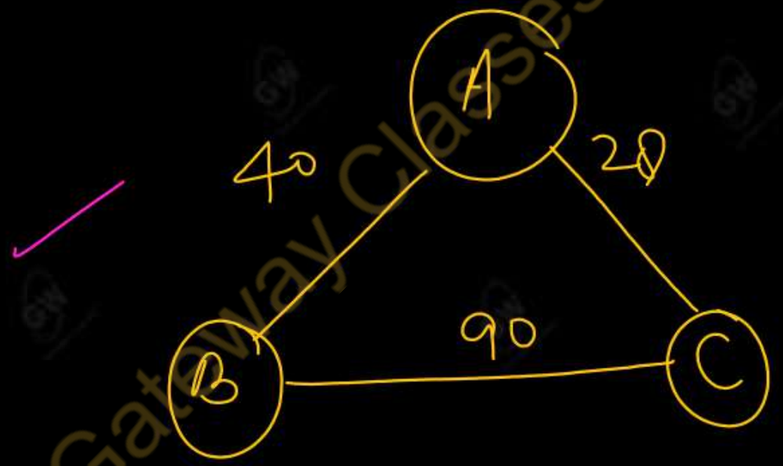
deg(1) = 1
deg(2) = 2
deg(3) = 3
deg(4) = 2
deg(5) = 2

	In	Out
1	x	1
2	1	1
3	2	1
4	x	1
5	2	2
6	2	x

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

Weighted Graph:-

- 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 -

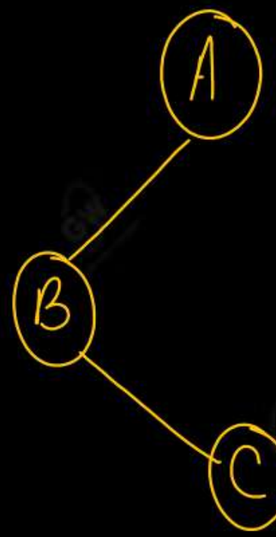


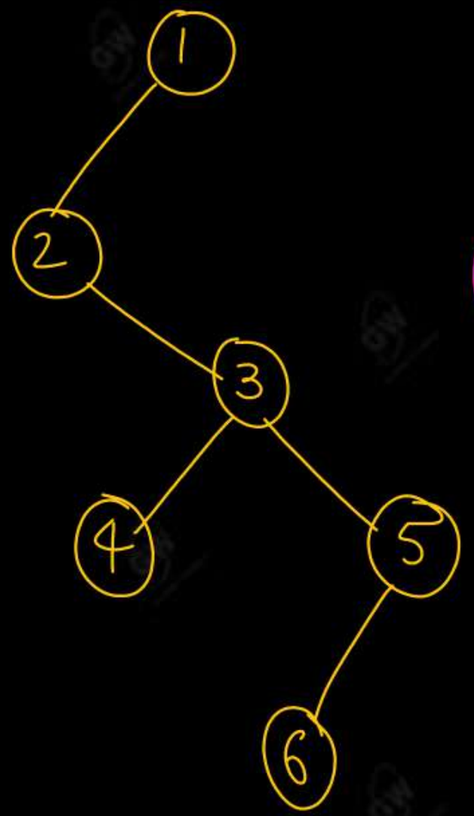
Tree:-

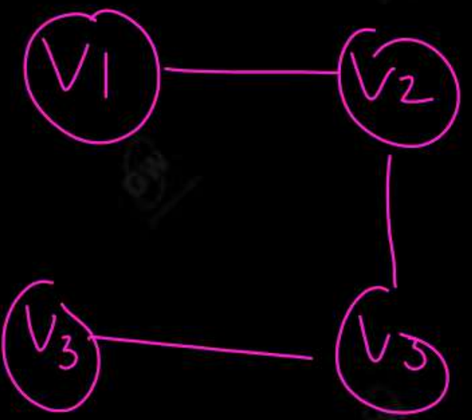
A graph is a tree if it has two properties:

1. It is connected, and
2. There are no cycles in the graph.

The following graphs are Tree:







without cycle a graph is a tree.

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
   (b) Incidence Matrix representation of a graph
3. Linked Representation of a graph

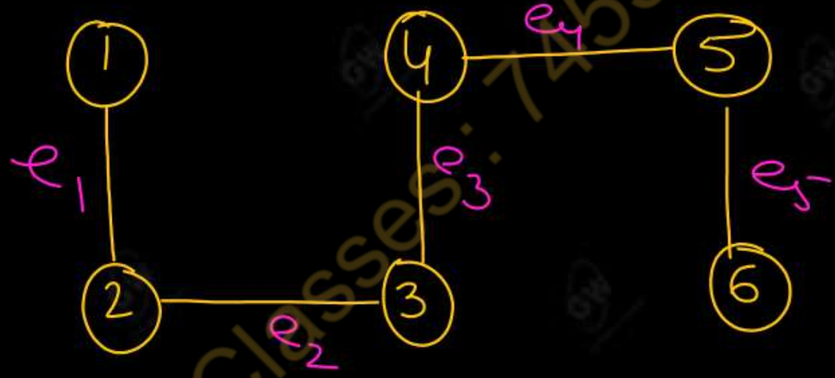
1. Set Representation of a Graph: -

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:-

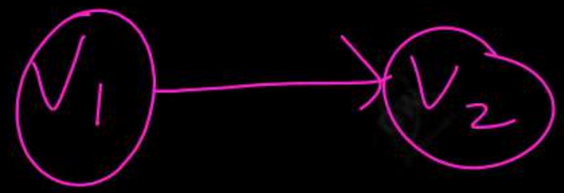
- Adjacency matrix is the matrix, which keeps the information of adjacency nodes or vertices.
- In other words, we can say that the matrix contains the information that whether this vertex is adjacent to any other vertex or not.

- 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 X 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.

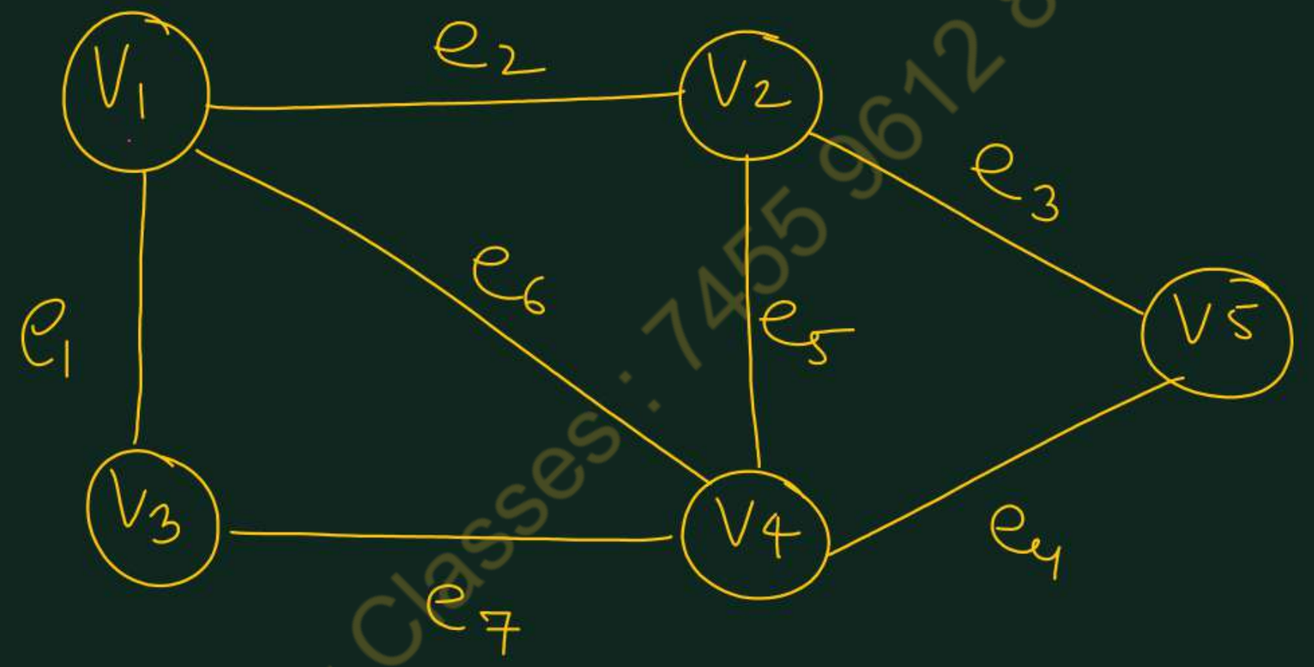
from V_i to V_j



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:



UNDIRECTED GRAPH

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

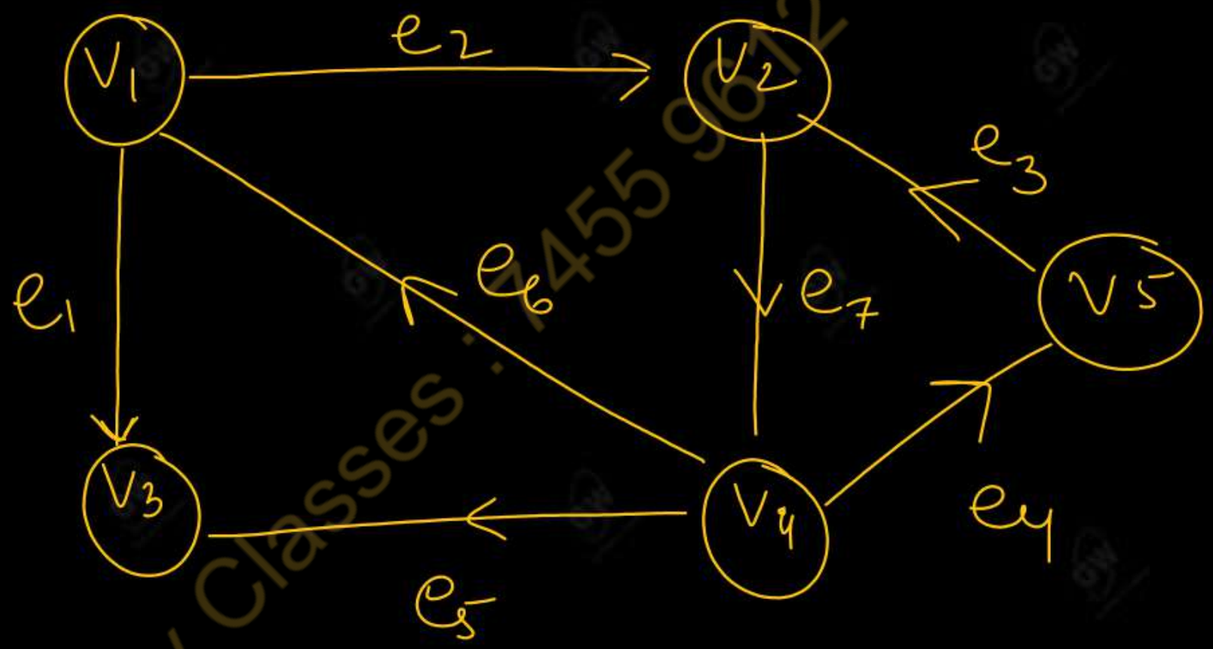
	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	1	0
V_2	1	0	0	1	1
V_3	1	0	0	1	0
V_4	1	1	1	0	1
V_5	0	1	0	1	0

ADJACENCY MATRIX

❑ Consider the following directed graph:-

$V_1 - V_2 = 1$

$V_1 - V_3 = 1$



DIRECTED GRAPH

- 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	+1 (out)	+1	−1 (indegree)	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

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

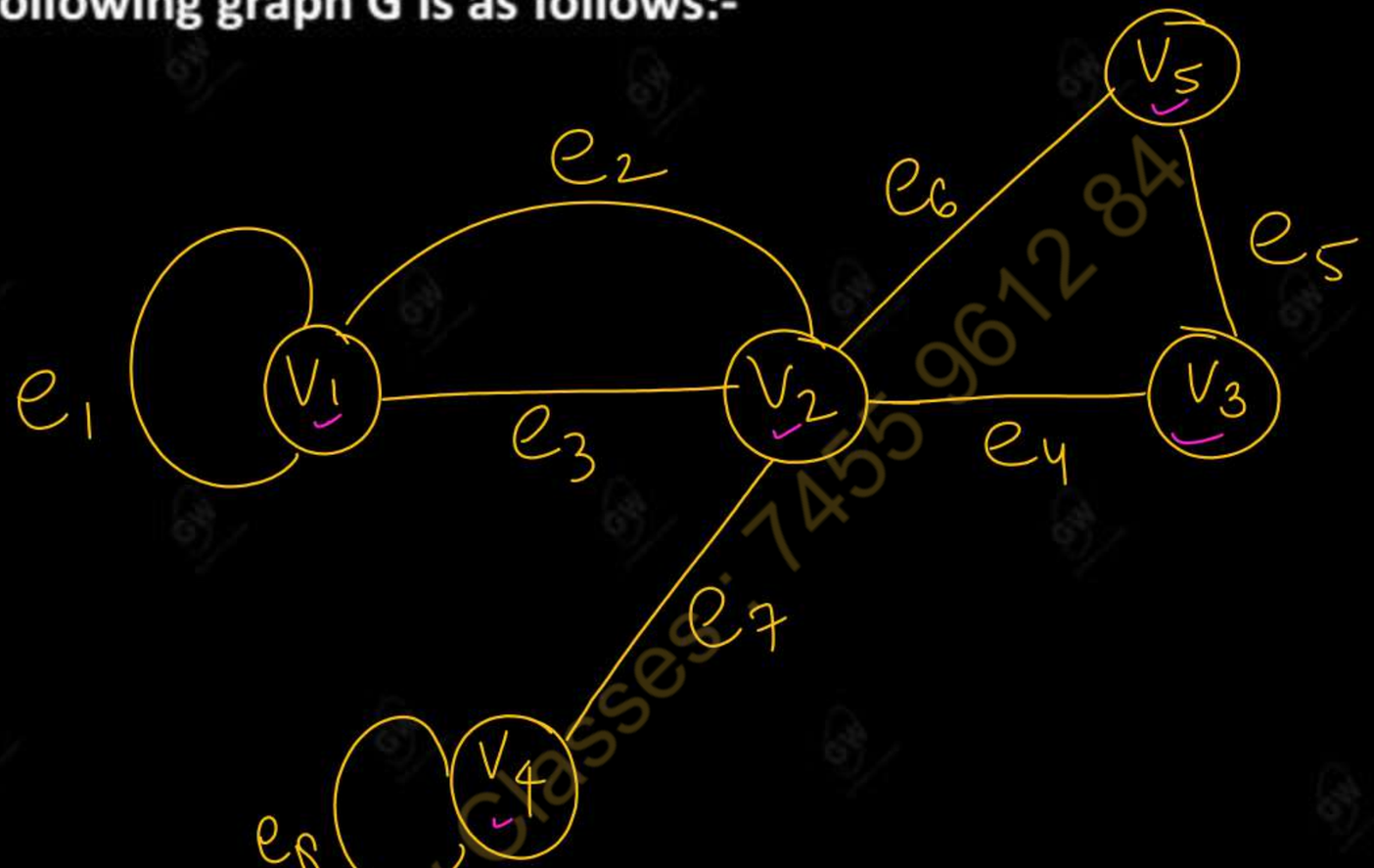
Let G be a graph with n vertices and e edges then the incidence matrix is a matrix of order n X e, whose n rows corresponds to the n vertices and e columns corresponds to the e edges as follows:-

$$a_{ij} = \begin{cases} 1 & \text{if the } j^{th} \text{ edge } e_j \text{ is incident on } i^{th} \text{ vertex } V_i \\ 0 & \text{otherwise} \end{cases}$$

n x n
5 x 5

V = 5
e = 7
5 x 7

❑ Consider the following graph G is as follows:-



$V = 5$

$E = 8$

$5X$

The incidence matrix for the previous graph is:

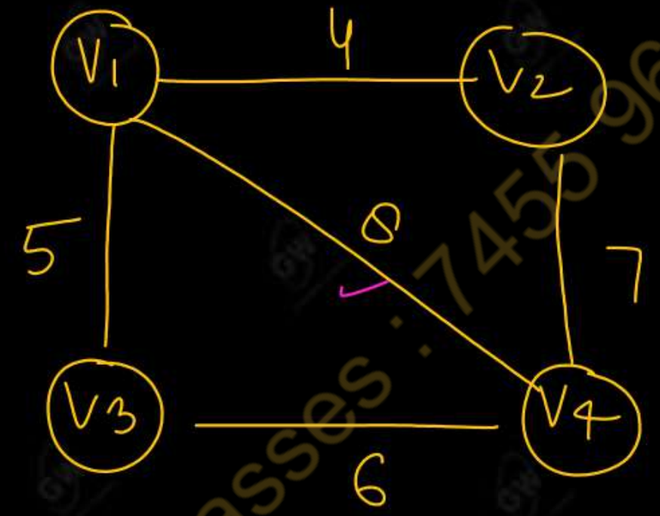
Incidence matrix Rep^{n} of a Graph G.

	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:

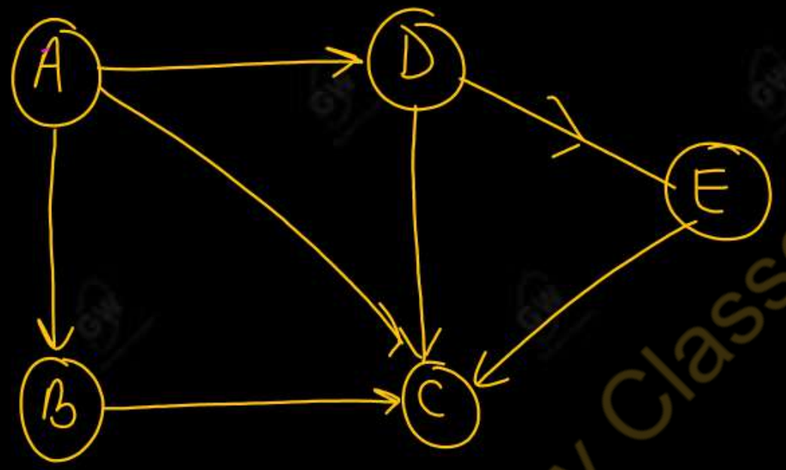


- The matrix representation of above weighted graph is as follows:-

	V_1	V_2	V_3	V_4
V_1	0	4	5	8
V_2	4	0	0	7
V_3	5	0	0	6
V_4	8	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:



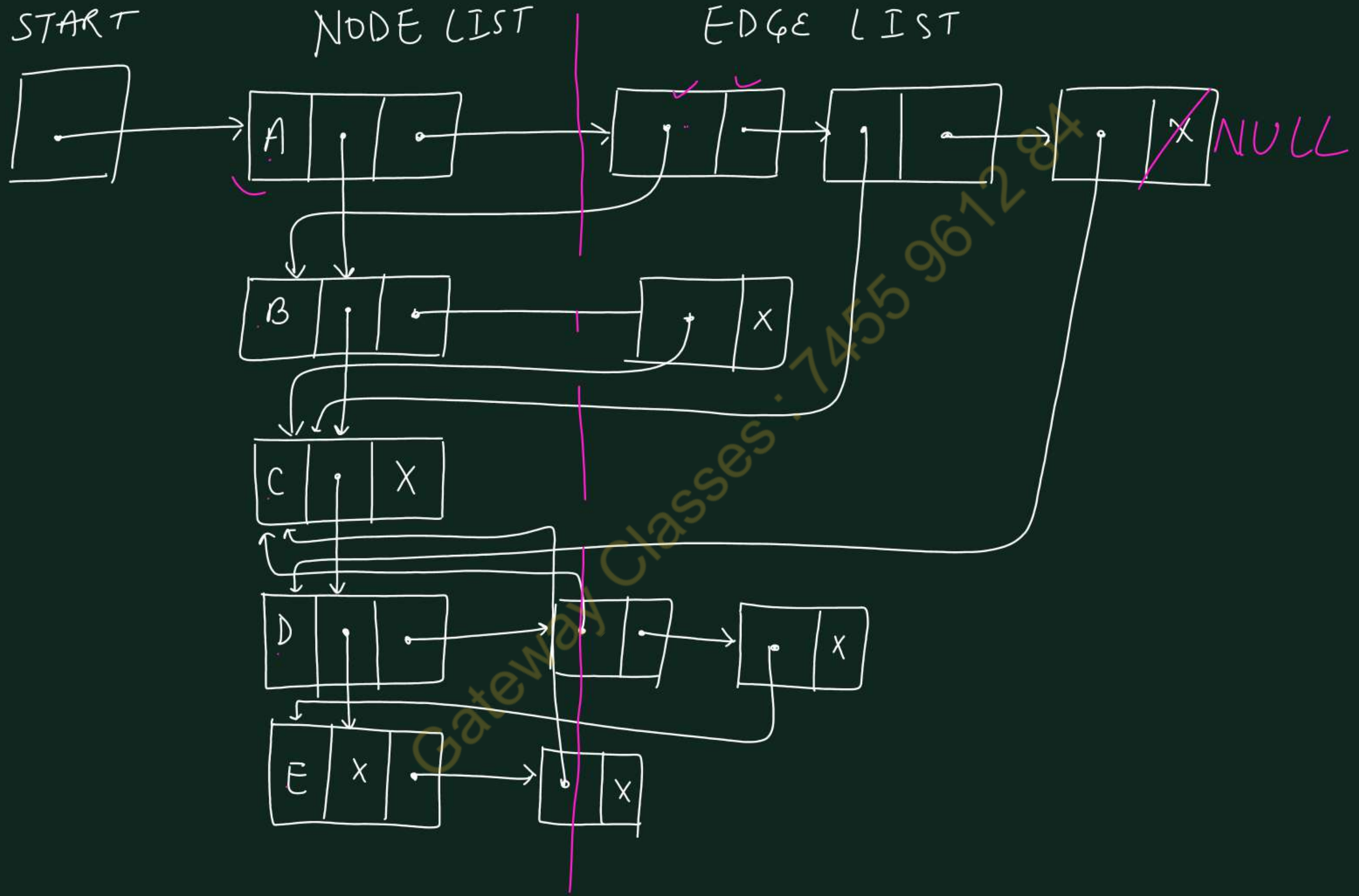
Directed graph

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

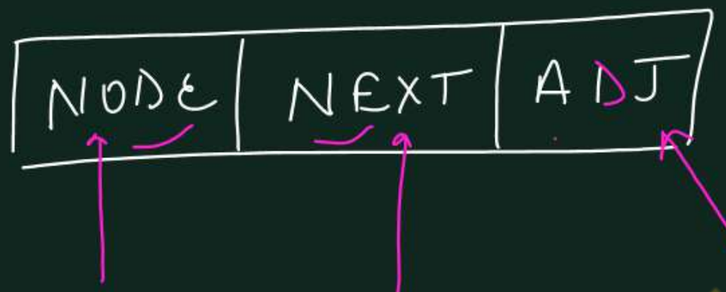
Adjacency list

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.



NODE LIST-



Key value of node

pointer to next node in list NODE

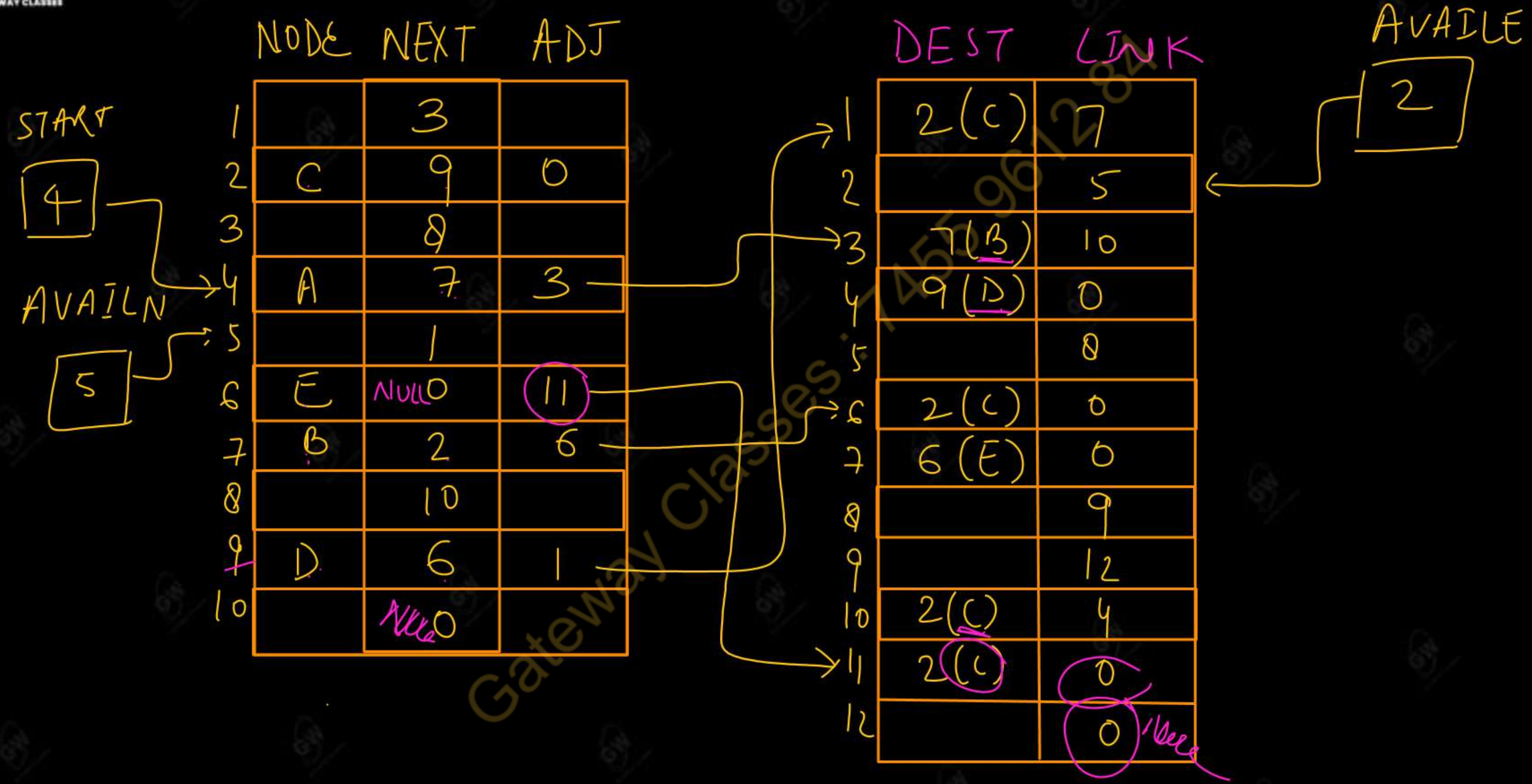
pointer to a I element in adjacency list which is maintained in the list EDGE

EDGE LIST :-

point to the location in the LIST NODE

link the edges with the same initial node

❑ Specifically graph can be represented in memory as per the following:



Unit - 5 : Lec - 2

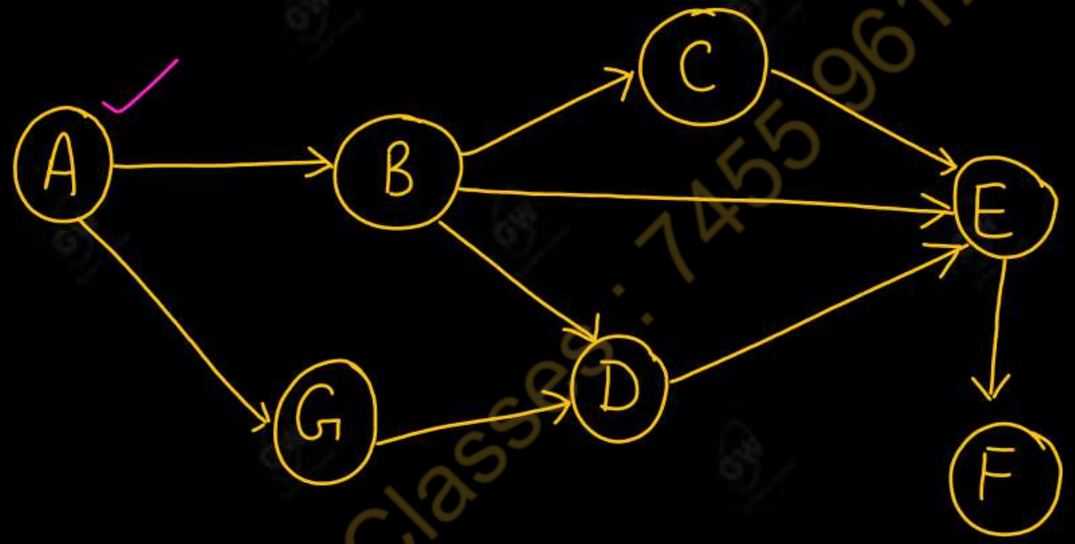
Today's Target -

- ❑ Graph Traversal (BFS & DFS)
- ❑ AKTU PYQs

AKTU PYQs

Q.1 Explain Depth First Search. Give example to support your explanation.	2014-15, 10 Marks
Q.2 Describe the Breadth First Search traversal of a Graph.	2014-15, 5 Marks
Q.3 Explain Breadth First Search with suitable example.	2016-17, 10 Marks
Q.4 Write short note on Depth first Search.	2016-17, 3 Marks

Q.5 Discuss the data structure used in DFS. Write an algorithm for DFS, Traverse the given graph starting from node A using DFS.



stack

2017-18, 7 Marks

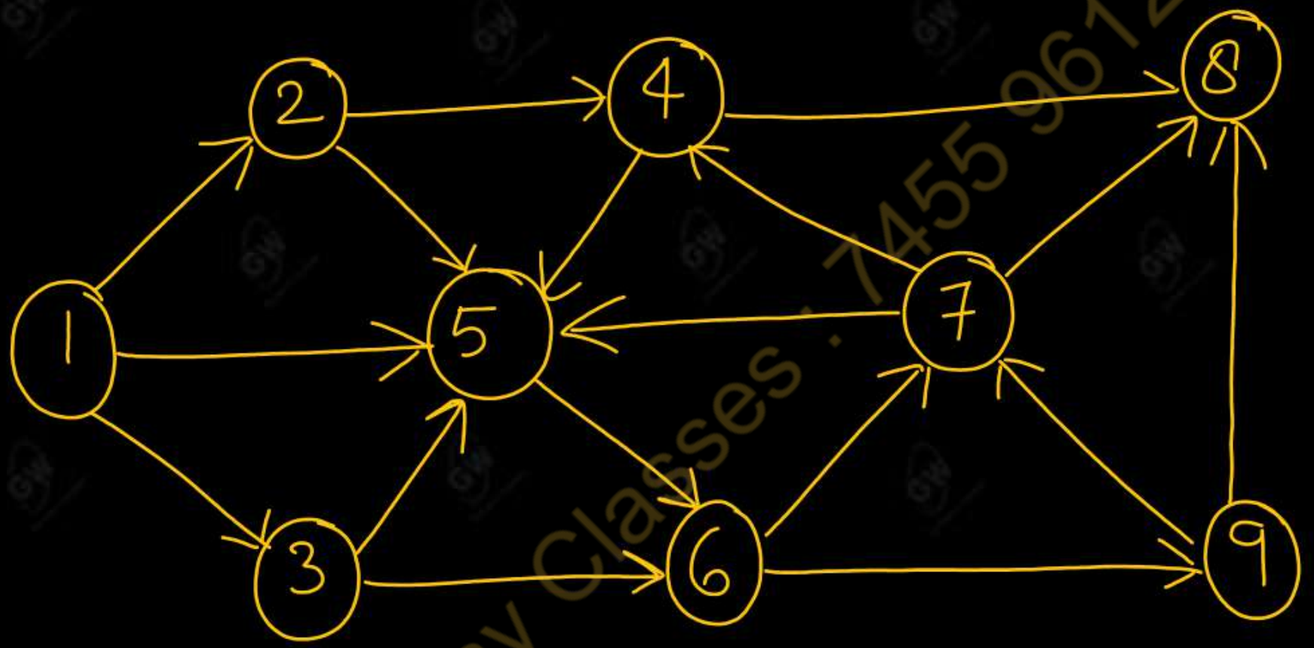
Q.6 Write an algorithm for Breadth First Search (BFS) and explain with the help of suitable example.

2020-21, 10 Marks

Q.7 Explain Depth First Search Traversal in Graph with the help of an example. 2020-21, 7 Marks

BFS

Q.8 Differentiate between DFS and BFS. Draw the breadth first tree for the ~~above~~ given graph.



2021-22, 10 Marks

Q.9 Write an algorithm for Breadth First Search (BFS) traversal of a graph.

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.

Breadth-First Search

- The general idea behind a breadth-first search beginning at a starting node A is as follows.
- First we examine the starting node A.
- Then we examine all the neighbors of A.
- Then we examine all the neighbors of the neighbors of A.
- And so on.
- We need to keep track of the neighbors of a node, and we need to guarantee that no node is processed more than once.
- This is accomplished by using a queue data structure to hold nodes that are waiting to be processed, and by using a field STATUS which tells us the current status of any node.

Adjacency list of graph.

During the execution of algorithm, each node N of graph G will be one of three states, called the status of N, as per the following:

- STATUS = 1, Ready state, The initial state of the node N.
- STATUS = 2, Waiting state, The node N is on the queue or stack, waiting to be processed.
- STATUS = 3, Processed state, The node has been processed.

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

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

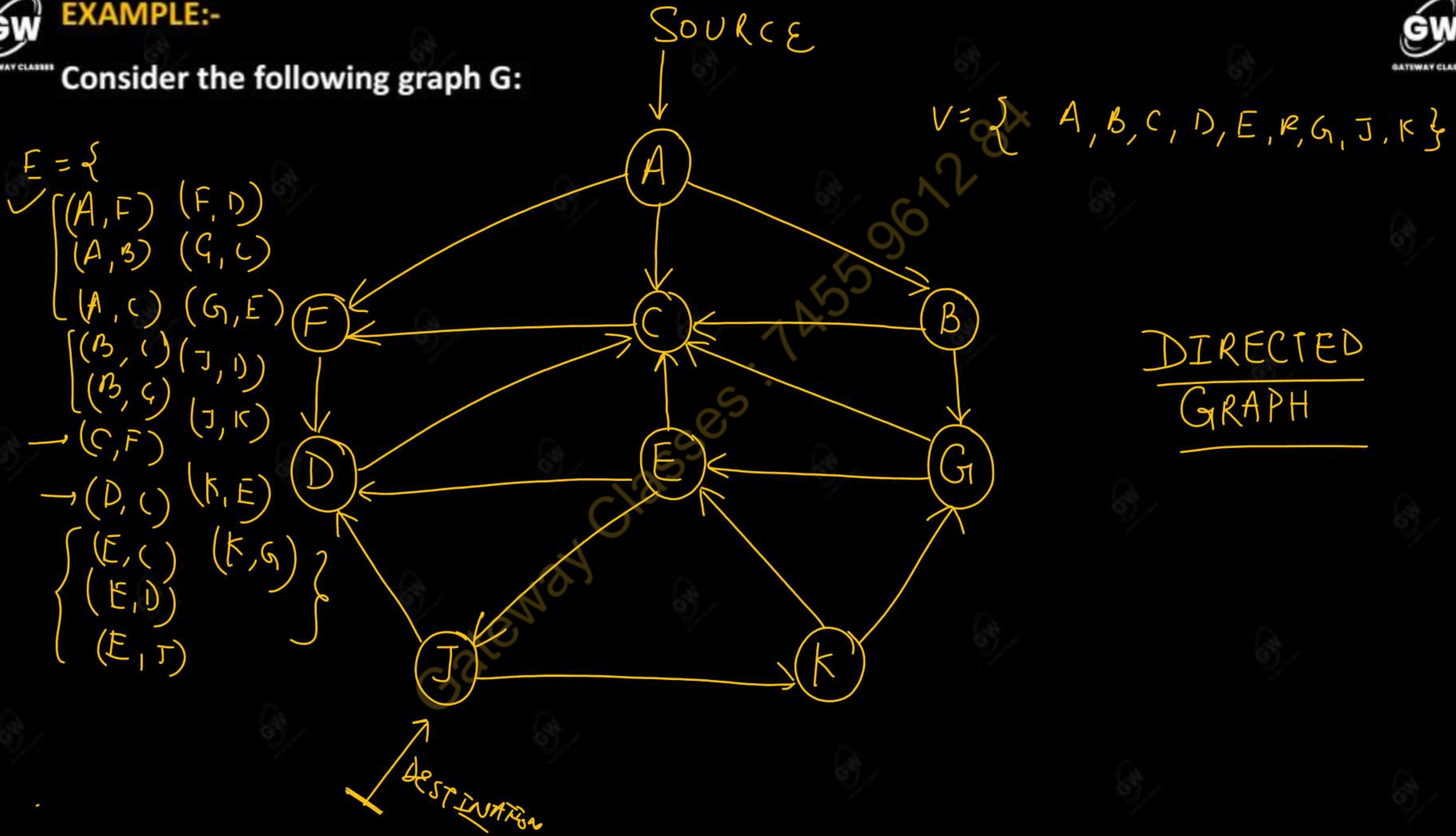
1. Initialize all nodes to the ready state (STATUS = 1).
2. Put the starting node A in QUEUE and change its status to the waiting state (STATUS=2).
3. Repeat Steps 4 and 5 until QUEUE is empty.
4. Remove the front node N of QUEUE. Process N and change the status of N to the processed state (STATUS = 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

EXAMPLE:-

Consider the following graph G:



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
J	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 by using an array ORIG together with the array QUEUE.

ORIGIN

- The steps of our search follow.

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

FRONT = 1	QUEUE : A
REAR = 1	ORIG : Ø

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

FRONT

REAR

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

	1	2	3	5
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

	1	2	3	4	5
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

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

FRONT ↓ 5, REAR ↓ 6

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

FRONT ↓ 6, REAR ↓ 6

(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	QUEUE : A, F, C, B, D, G, E
REAR 7	ORIG : Ø, A, A, A, F, B, G

(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	QUEUE : A, F, C, B, D, G, E, J
REAR = 8	ORIG : Ø, A, A, A, F, B, G, E

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.

Depth-First Search (DFS) (STACK)

- The general idea behind a depth-first search beginning at a starting node A is as follows.
- First we examine the starting node A.
- Then we examine each node N along a path P which begins at A, that is, we process a neighbor of A, then a neighbor of a neighbor of A, and so on.
- After coming to a "dead end," that is, to the end of the path P, we backtrack on P until we can continue along another path P and so on.
- The algorithm is very similar to the breadth-first search except now we use a stack instead of the queue.
- Again, a field STATUS is used to tell us the current status of a node.

Algorithm : Depth First Search (DFS)

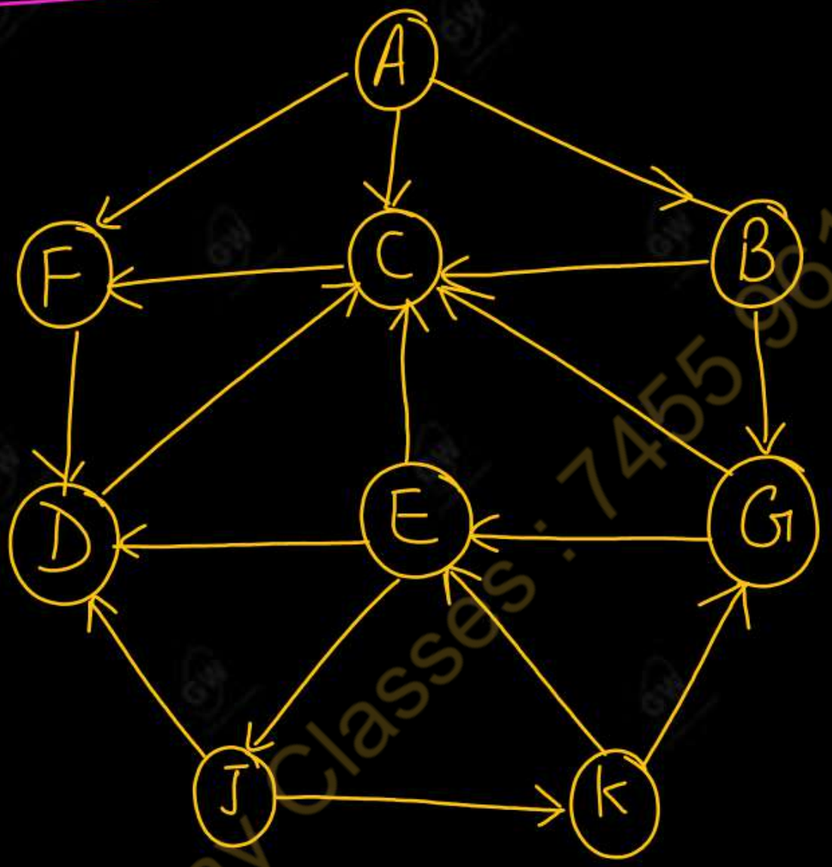
This algorithm executes a depth-first search on a graph G beginning at a starting node A.

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.

EXAMPLE:- Consider again the same graph G:



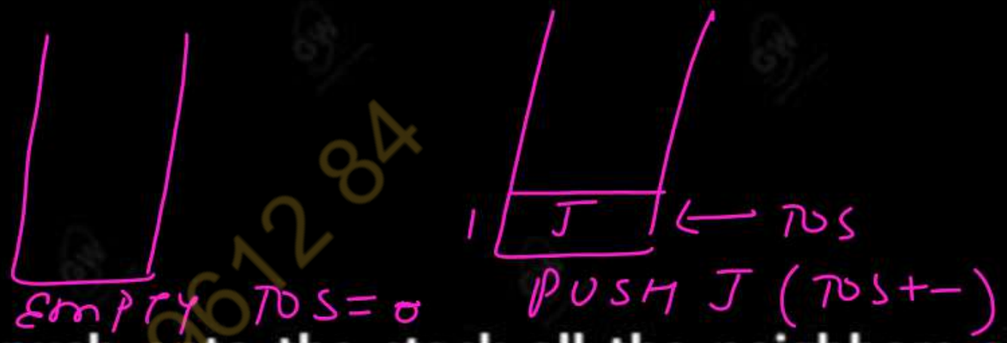
Adjacency list	
A	F, C, B
B	F, C
C	F
D	C
E	D, C, J
F	D
G	C
J	D, K
K	E, G

- 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:

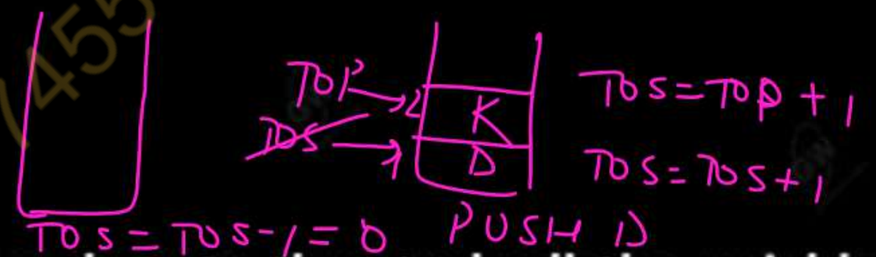
(a) Initially, push J onto the stack as follows:

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: ϕ

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.

Unit-5 : Lec- 3

Today's Target

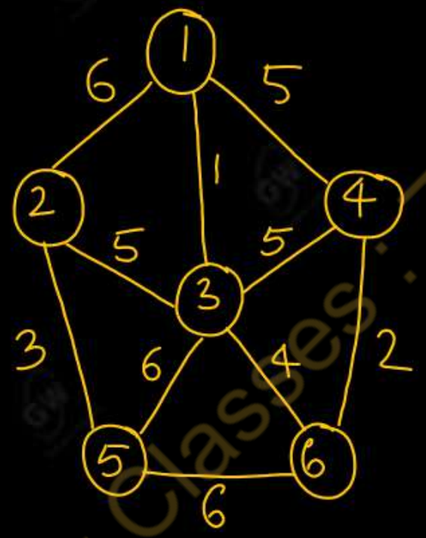
- Spanning Tree
- Minimum Spanning Tree (Prim's and Kruskal's Algorithms)
- AKTU PYQs

MST

Minimum Cost Spanning Tree

Q.1 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**

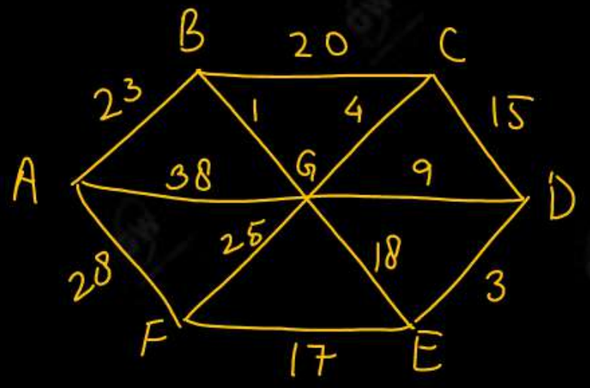
Q.2 Find MST of the following graph using Kruskal's algorithm. **2014-15, 5 Marks**



Q.3 Consider the following undirected graph:

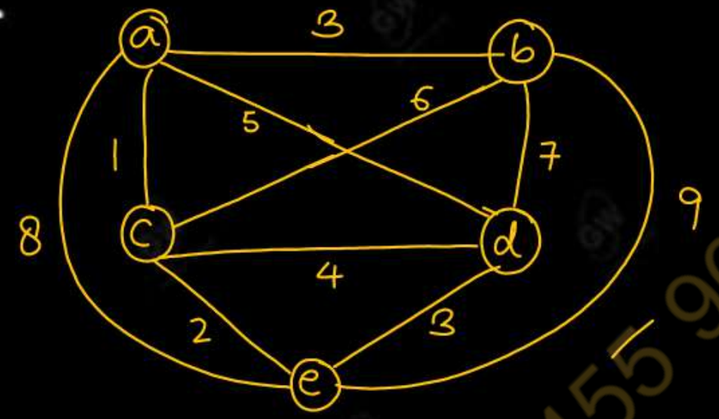
a) Find the adjacency list representation of the graph.

b) Find the minimum cost spanning tree by Kruskal's algorithm. **2015-16, 10 Marks**



Q.4 Discuss Prim's and Kruskal's algorithm. Construct minimum spanning tree for the below given graph using prim's algorithm(source node = a).

2016-17, 10 Marks



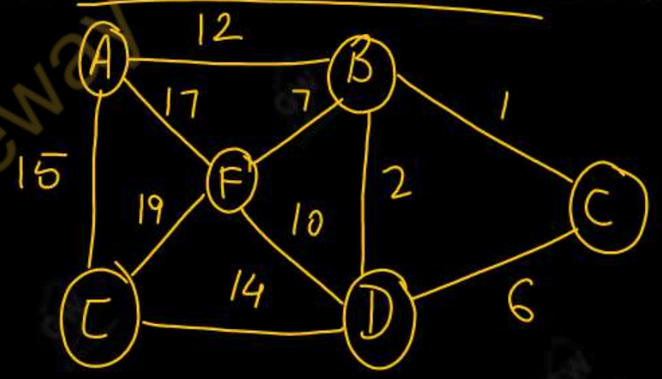
Same Question.

Q.5 Explain Kruskal's algorithm to find minimum spanning tree in a weighted directed graph. Can there be two minimum spanning tree of given weighted directed graph?

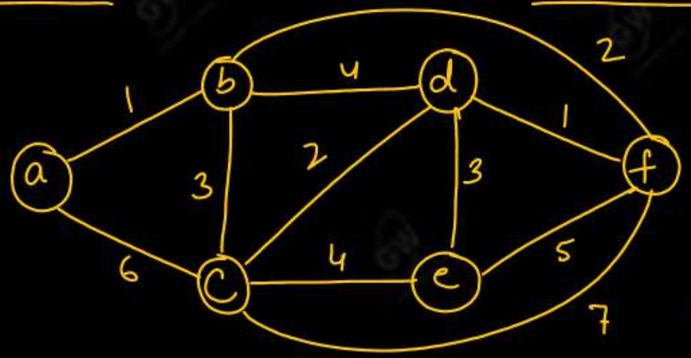
2016-17, 10 Marks

Q.6 Define spanning tree. Also construct minimum spanning tree using Prim's algorithm for the given graph.

2017-18, 7 Marks

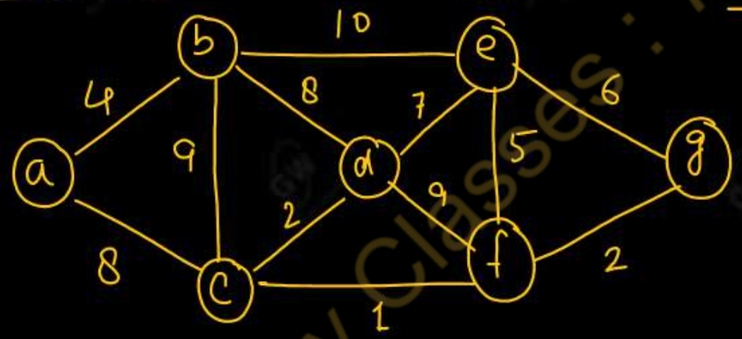


Q.7 Write Prim's algorithms and Find the Minimum Spanning tree for the following graph. **2017-18, 7 Marks**

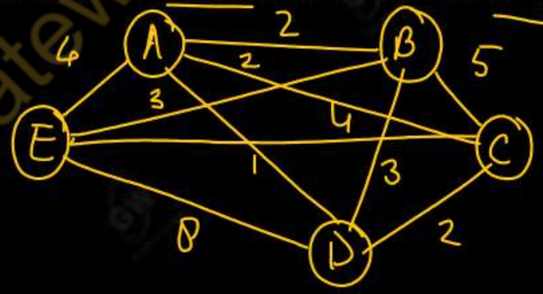


Q.8 What is minimum spanning tree? Give its applications. **2019-20, 2 Marks**

Q.9 Find the minimum spanning tree in the following graph using kruskal's algorithm: **2019-20, 10 Marks**

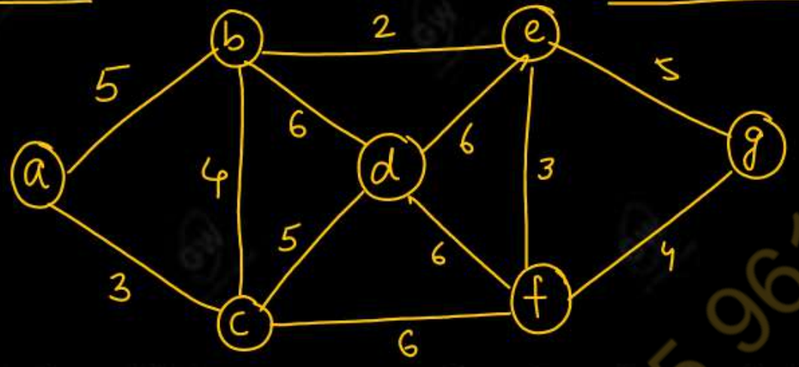


Q.10 Describe Prim's algorithm and find the cost of minimum spanning tree using Prim's algorithm. **2020-21, 10 Marks**



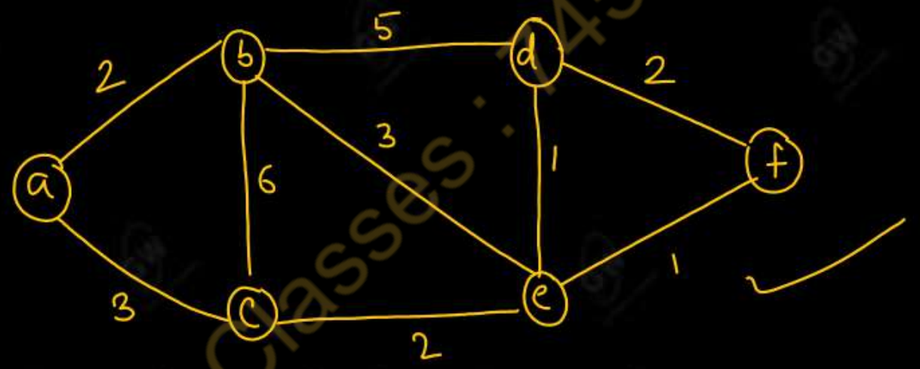
Q.11 Find minimum spanning tree in the following graph using Prim's algorithm by considering vertex 'a' as root/source vertex.

2020-21, 7 Marks



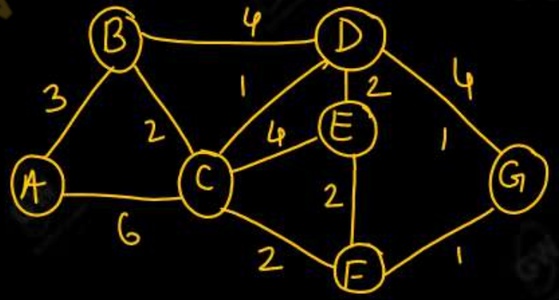
Q.12 Apply Prim's algorithm to find a minimum a spanning tree in the following weighted graph as shown below.

2021-22, 10 Marks



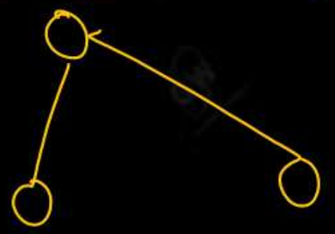
Q.13 What is spanning tree? Write down the Prim's algorithm to obtain minimum cost spanning tree. Use Prim's algorithm to find the minimum cost spanning tree in the following graph:

2022-23,10 Marks

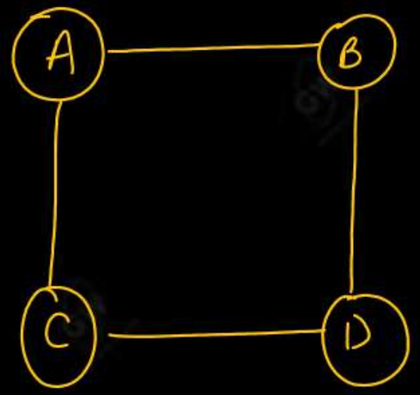


SPANNING TREE

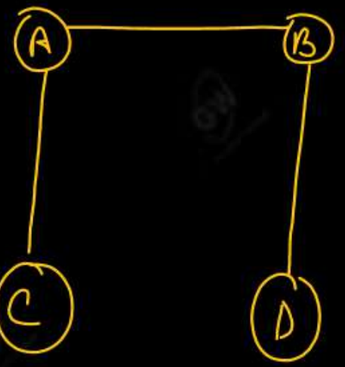
- A spanning tree of a graph is just a sub-graph that contains all the vertices and is a tree.
- A spanning tree is a subset of graph G, such that all the vertices are connected using minimum possible number of edges.
- A spanning tree does not have cycles and a graph may have more than one spanning 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.



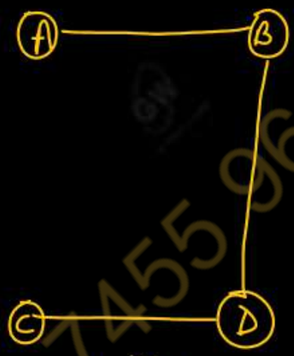
- Let us take a connected graph:



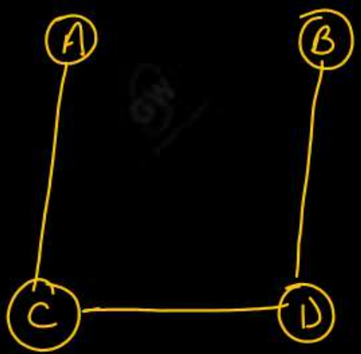
(i)



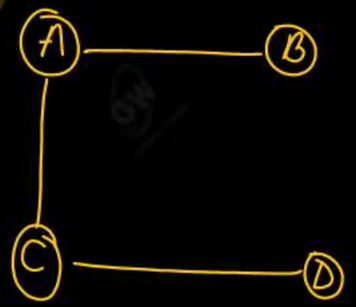
(ii)



(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.

ALGORITHM : KRUSKAL MST (G, W)

1. $A \leftarrow \phi$.
2. For each vertex $v \in V\,[G]$

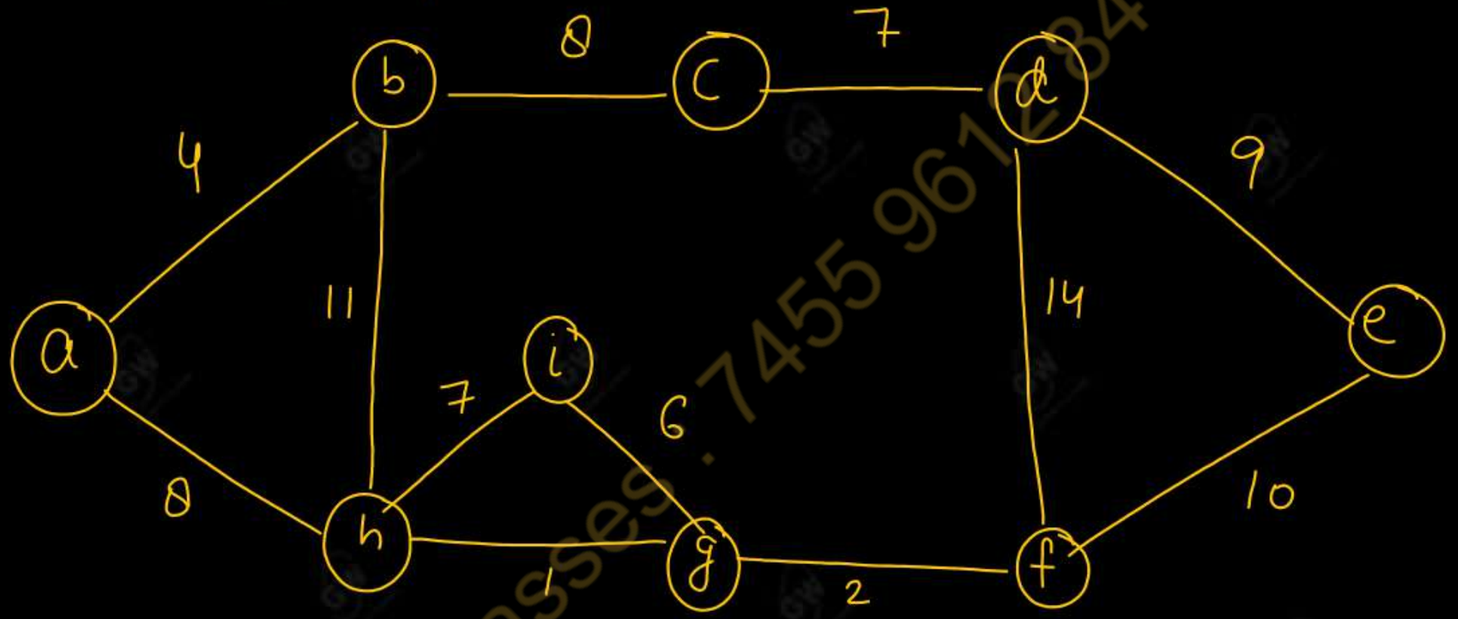
   do MAKE-SET (v)
3. Sort the edges of E into increasing order by weight w.
4. For each edge $(u, v) \in E$ taken in increasing order.

   do if FIND-SET $(u) \neq$ FIND-SET (v)

   then $A \leftarrow U(u, v)$ UNION (u, v)
5. Return A.

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

MST



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

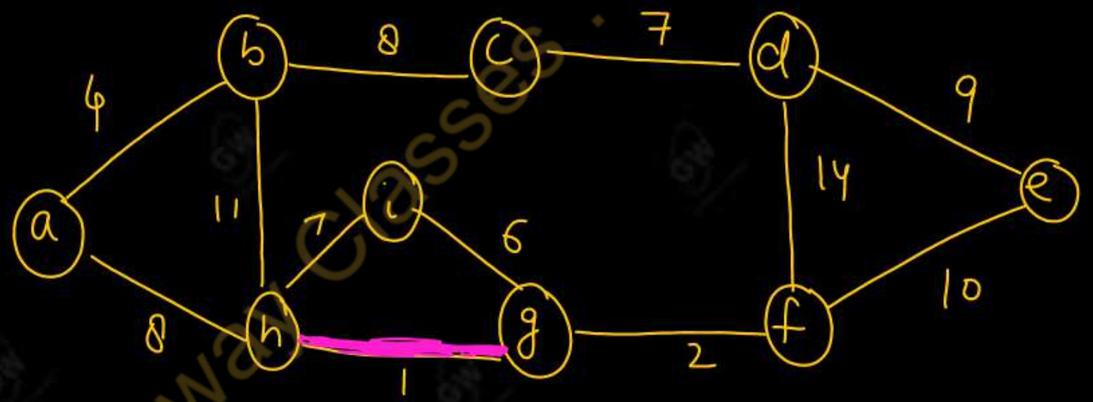
Then sort the edges in E into order by non-decreasing weight, i.e.,

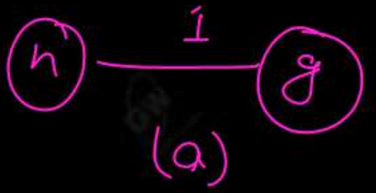
Edge	Weight
(h, g)	1
(g, f)	2
(a, b)	4
(i, g)	6
(h, i)	7
(c, d)	7
(b, c)	8
(a, h)	8
(d, e)	9
(e, f)	10
(b, h)	11
(d, f)	14

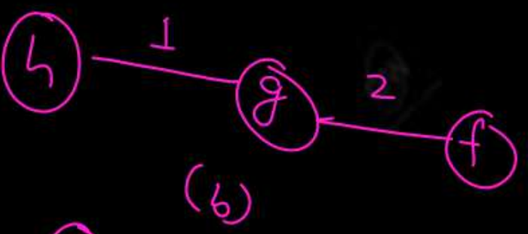
Not considered due to cycle

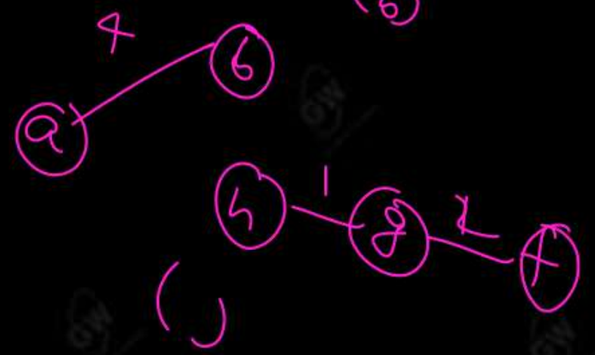
not considered due to cycle

- 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 -

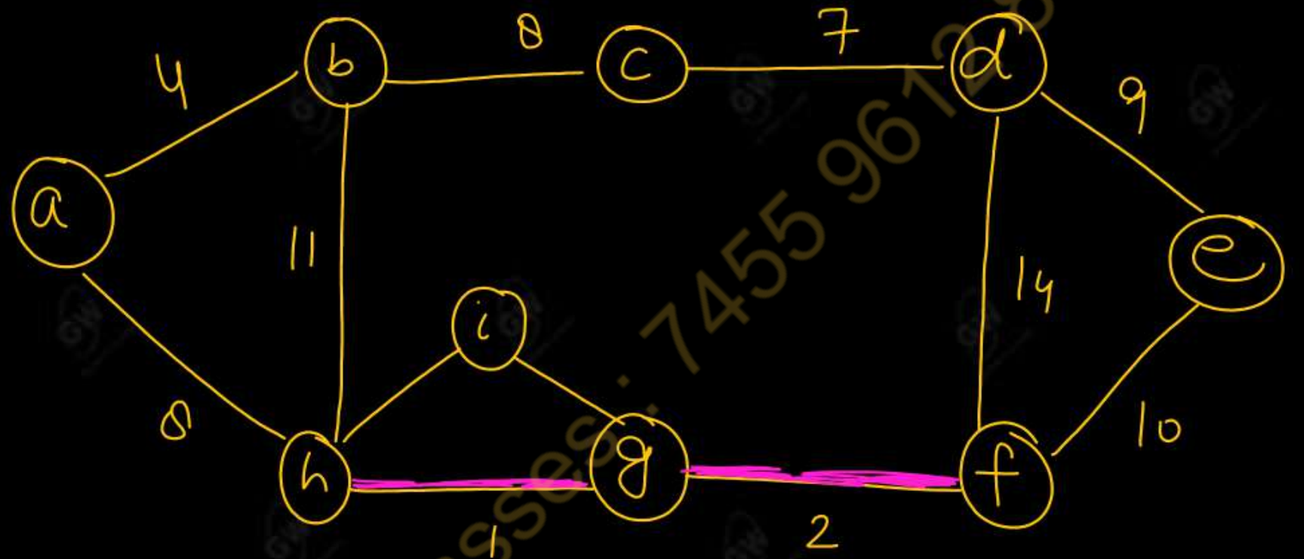




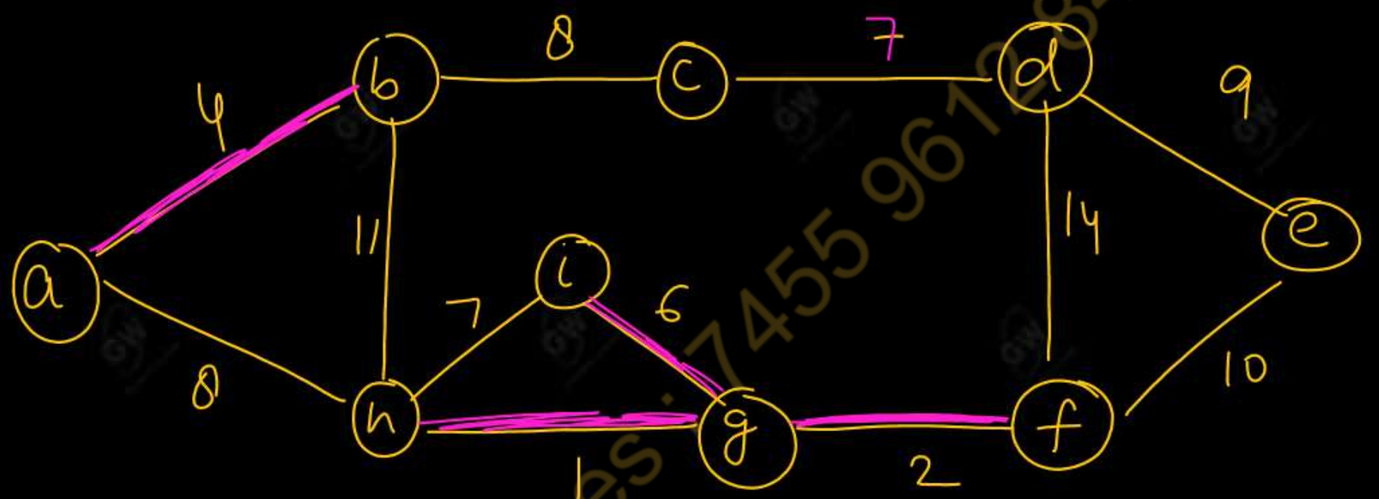




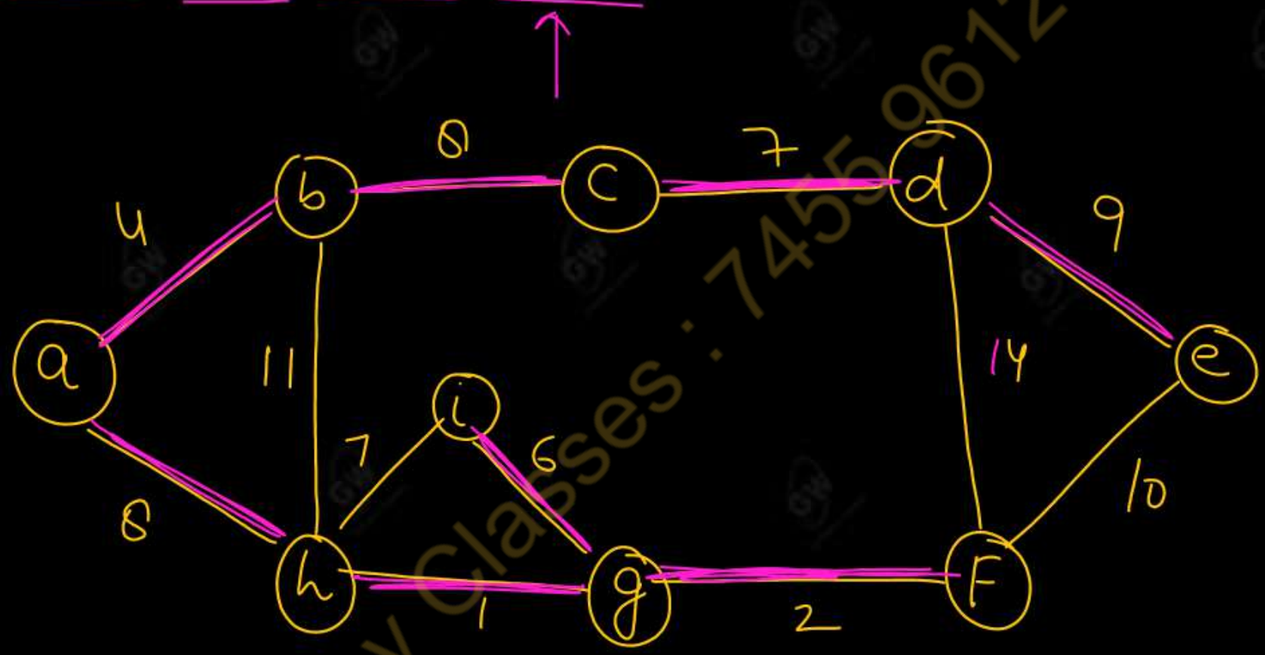
- ❑ Then (g, f) edge.



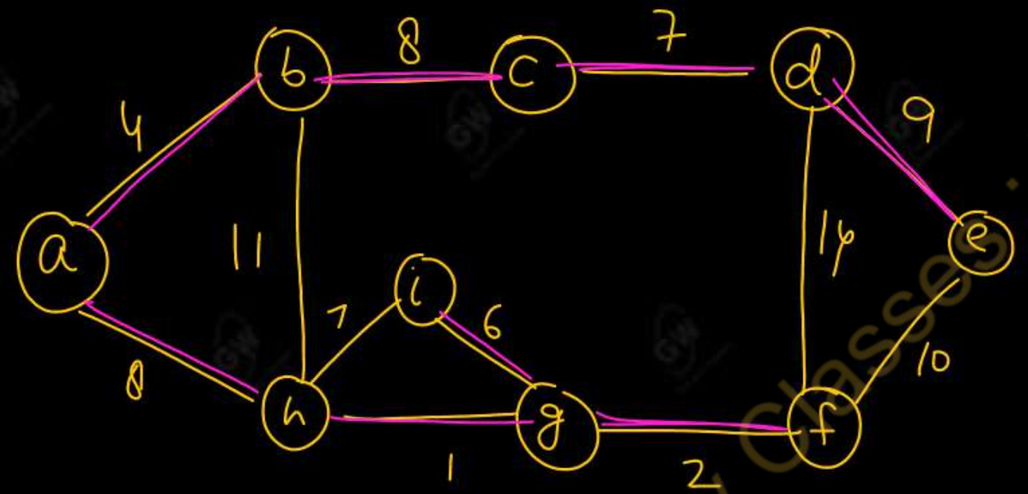
❑ Then (a, b) and (i, g) edges are considered and forest becomes.

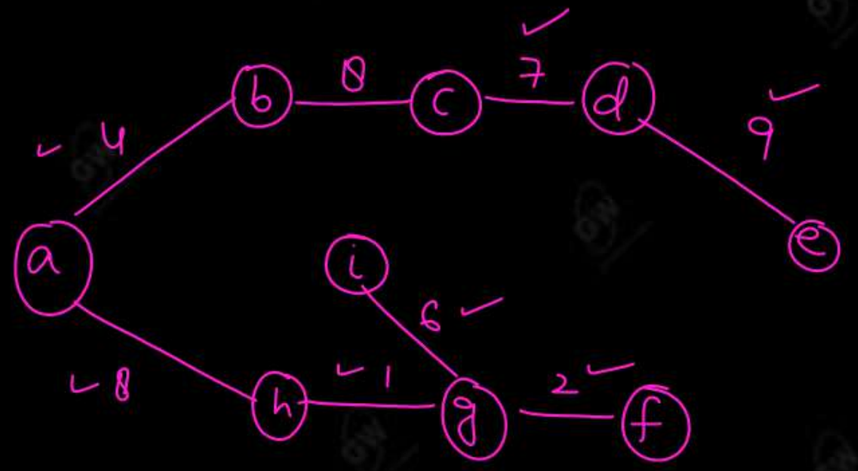


- Now, edge (h, i).
- Both h and i vertices are in same set, thus it creates a cycle. So this edge is discarded.
- Then edge (c, d), (b, c), (a, h), (d, e), (e, f) are connected as per the following:



- 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.





Minimum Cost spanning Tree

Cost = 45

PRIM'S ALGORITHM

- In this, we start with any node and add the other node in spanning tree on the basis of weight of edge connecting to that node.
- Suppose we start with the node "u" then we have a need to see all the connecting edges and which edge has minimum weight.
- Then we will add that edge and node to the spanning tree.
- Suppose if two nodes u_1 and u_2 are in spanning tree and both have edge connecting to v_3 then edge. Which has minimum weight will be added in spanning tree.
- Prim's algorithm has the property that the edges in the set A always from a single tree.
- We begin with some vertex u in a given graph G = (V, E), defining the initial set of vertices A.
- Then, in each iteration, we choose a minimum- weight edge (u, v) connecting a vertex v in the set A to the vertex u outside to set A.

- Then vertex u is brought in to A.
- This process is repeated until a spanning tree is formed.
- Like Kruskal's algorithm, here too, the important fact about MST's is we always choose the smallest weight edge joining a vertex inside set A to the one outside the set A.
- The implication of this fact is that it adds only edges that are safe for A; therefore when the algorithm terminates, the edge in set A form a MST.

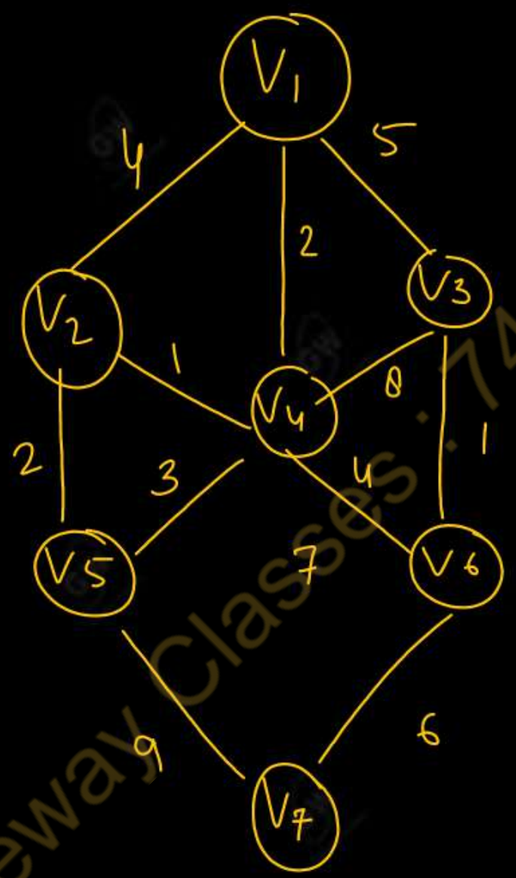
ALGORITHM : MST- Prim (G, u, r)

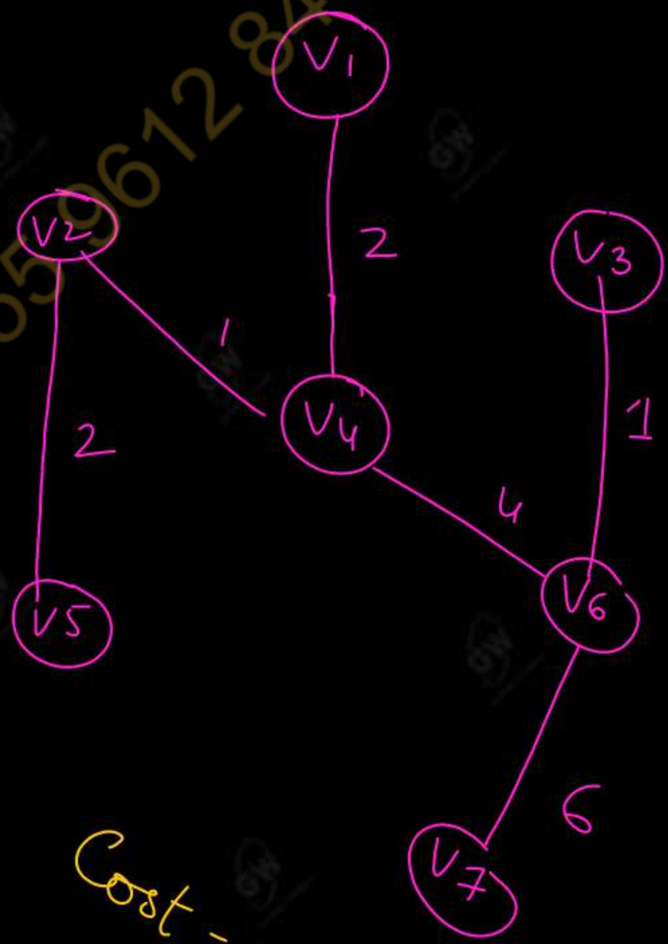
1. for each $u \in V[G][u]$
2. do u key [u] $\leftarrow \infty$
3. N[u] $\leftarrow NIL$
4. key [r] $\leftarrow Q$
5. Q $\leftarrow V[G]$
6. While Q $\neq \phi$
7. do u $\leftarrow EXTRACT - MIN\ (Q)$
8. For each $u \in Adj\ [u]$
9. Do if $v \in Q\ and\ w\ (u, v) < key\ [v]$
10. then $u\ (u) \leftarrow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

EXAMPLE-2. Find the minimum spanning tree vising Prim's algorithm for the given graph.

SOLUTION :-

Edges	Cost
$V_1 - V_2$	4 × cycle
$V_1 - V_4$	2 choosed
$V_1 - V_3$	5 × cycle
$V_4 - V_2$	1 choosed
$V_4 - V_5$	3 × cycle
$V_4 - V_6$	4 Choose
$V_4 - V_3$	8
$V_2 - V_1$	4 × cycle
$V_2 - V_5$	2 choosed
$V_5 - V_4$	3 × cycle
$V_5 - V_7$	9





Cost = 16

Ans

↓ Choosed

$V_6 - V_3 = 1$

$V_6 - V_7 = 6$ Choosed

$V_3 - V_1 = 5$ ×

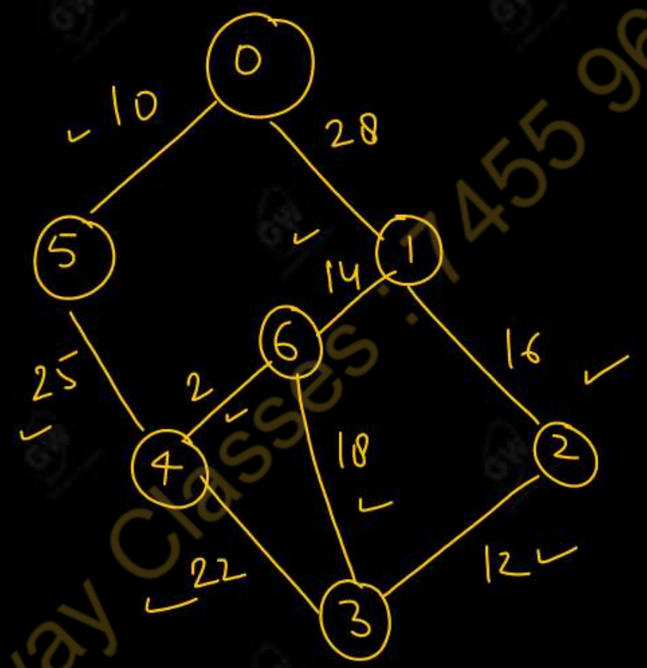
$V_3 - V_4 = 8$

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

(i) Kruskal's algorithm and

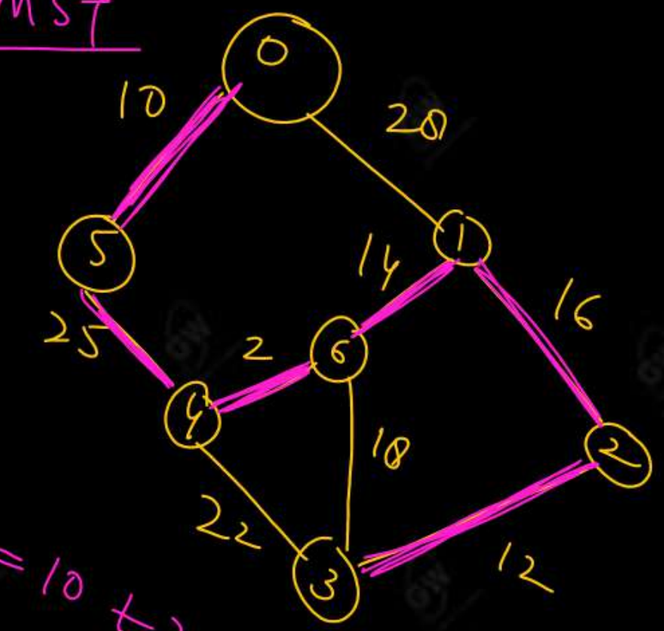
(ii) Prim's algorithm.

4 – 6	2 ✓
0 – 5	10 ✓
2 – 3	12 ✓
1 – 6	14 ✓
1 – 2	16 ✓
6 – 3	18 ✓ × cycle
3 – 4	22 × cycle
4 – 5	25 ✓
0 – 1	28



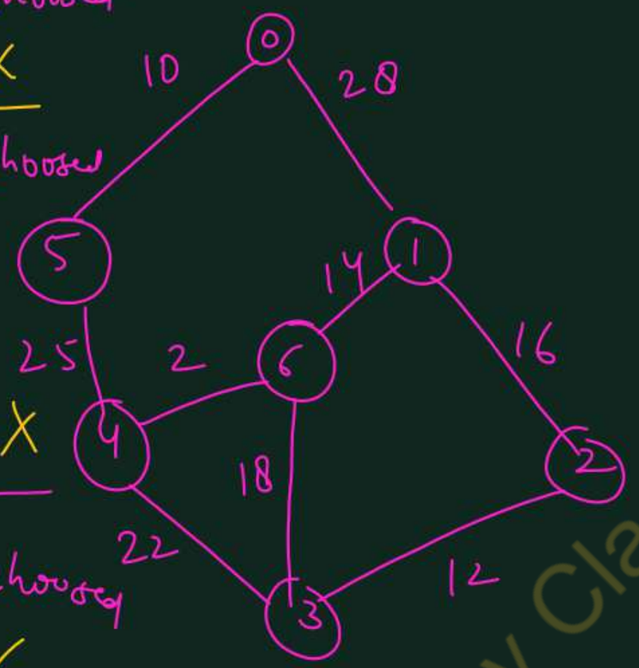
(i) Kruskal's Algorithm (MST)

MST



Cost = 10 + 25 + 2 + 14 + 16 + 12

Prim's Algorithm



0-5 = 10 choosed

0-1 = 28 X

5-4 = 25 choosed

4-6 = 2 ch

4-3 = 22 X

6-1 = 14 choosed

6-3 = 18 X

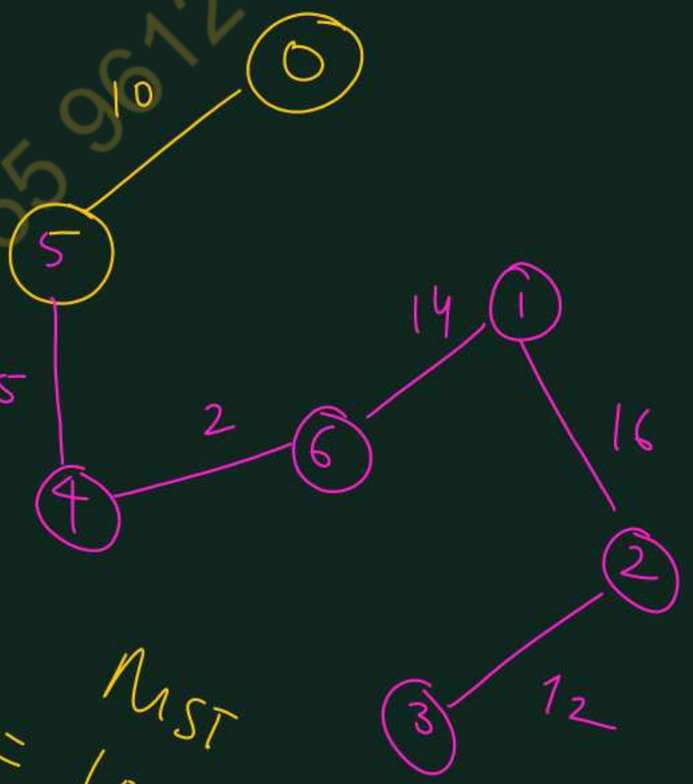
1-0 = 28 X

1-2 = 16 choosed

2-3 = 12 Choosed

3-4 = 22 X

3-6 = 18 X



MST

Cost = 10 + 25 + 2 + 14 + 16 + 12

= Ans

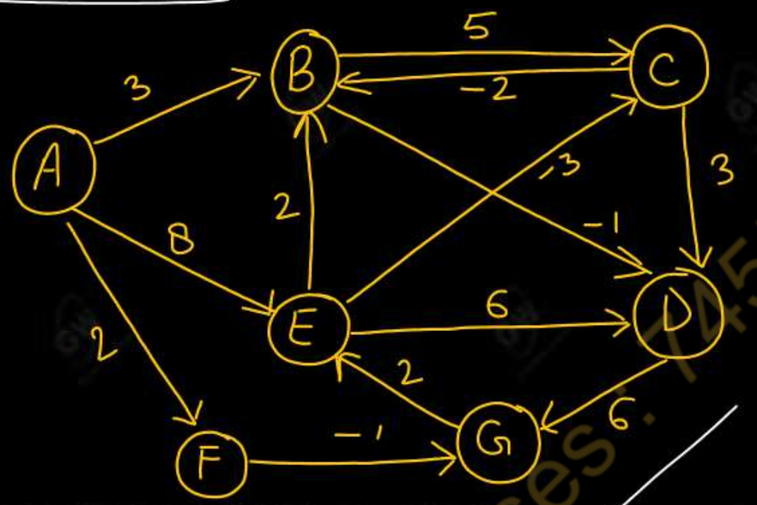
Unit - 5 : Lec- 4

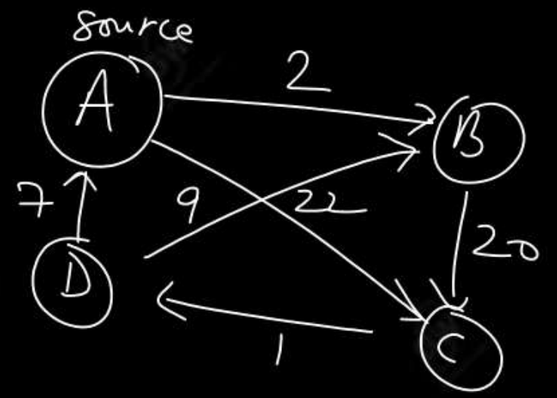
Today's Target

- Dijkstra's Algorithm
- AKTU PYQs

AKTU PYQs

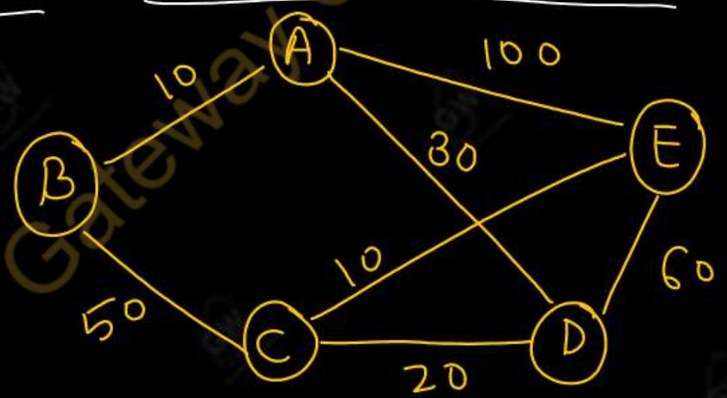
Q.1 Find the single source shortest path from the following graph using Dijkstra's algorithm. **2014–15,10 Marks**





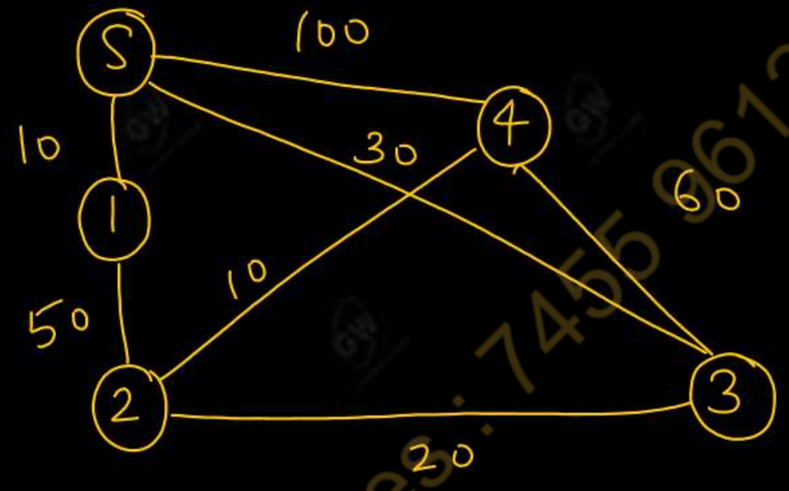
Q.2 Explain Dijkstra's algorithm with suitable example. **2015-16, 10 Marks**

Q.3 Describe Dijkstra's algorithm for finding shortest path. Describe its working for the graph given below. **2017-18, 7 Marks**



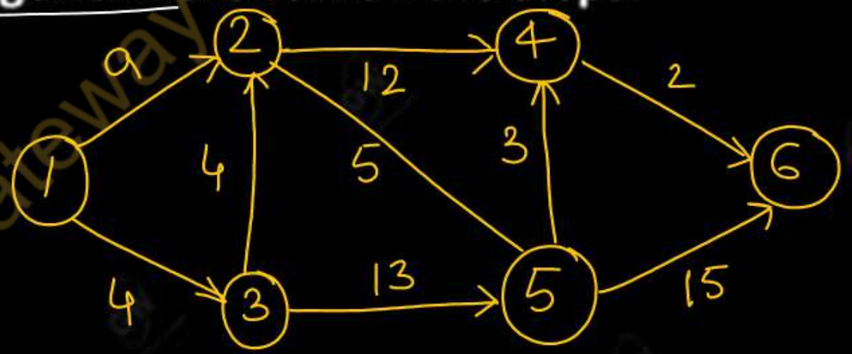
Q.4 Find the shortest path from S to all remaining vertices of graph using Dijkstra's algorithm

2018-19, 7 Marks



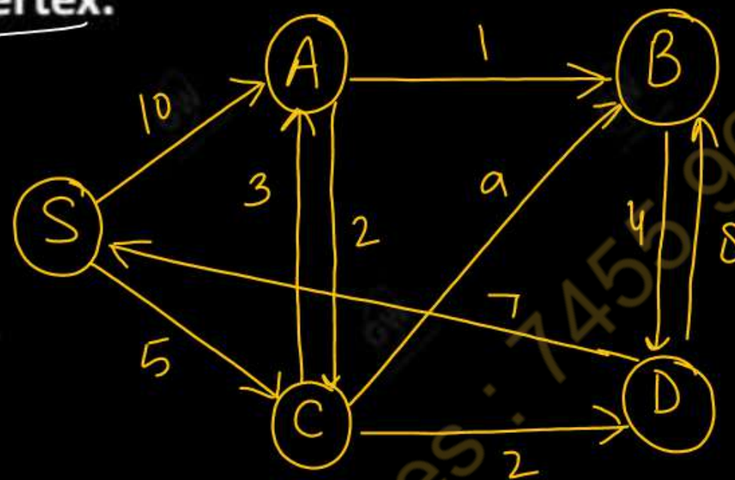
Q.5 By considering vertex '1' as a source vertex, Find the shortest paths to all other vertices in the following graph using Dijkstra's algorithms. Show all the steps.

2018-19, 7 marks



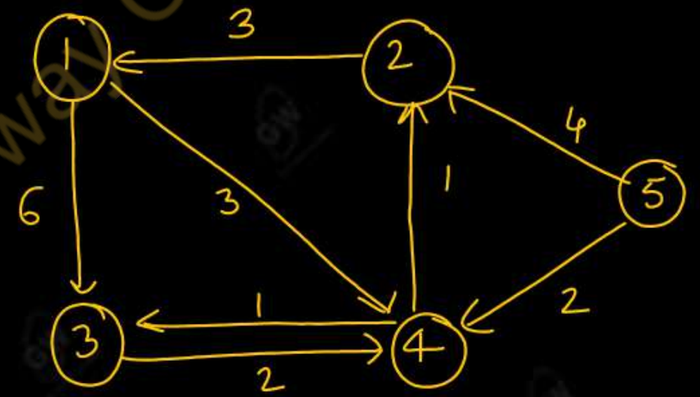
Q.6 Describe the Dijkstra's algorithm to find the shortest path. Find the shortest path in the following graph with vertex 'S' as source vertex.

2019-20, 10 Marks



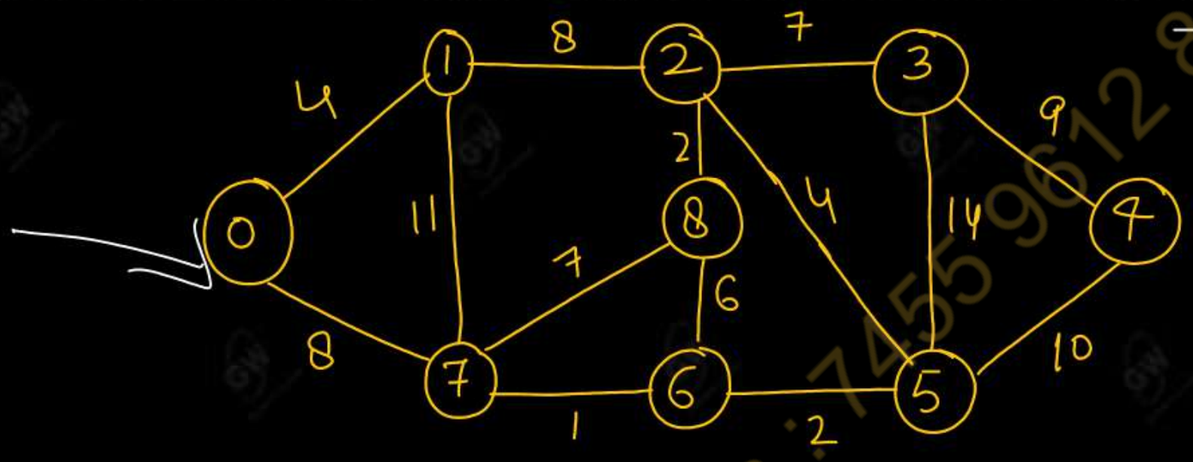
Q.7 Find the shortest path distance from source vertex '5' to all vertices in the following graph. Also draw the shortest path tree.

2020-21, 7 Marks



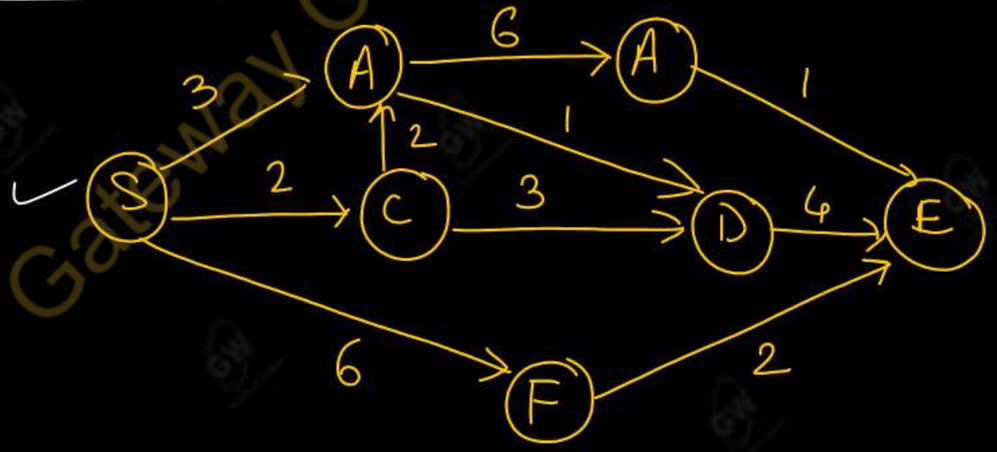
Q.8 Use Dijkstra's algorithm to find the shortest paths from source to all other vertices in the following graph.

2021-22, 10 Marks



Q.9 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) ≥ 0 each edge (u, v) ∈ 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 ∈ S. we have $d(v) = \delta(s, v)$. The algorithm repeatedly selects the vertex u ∈ $V - S$ 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).

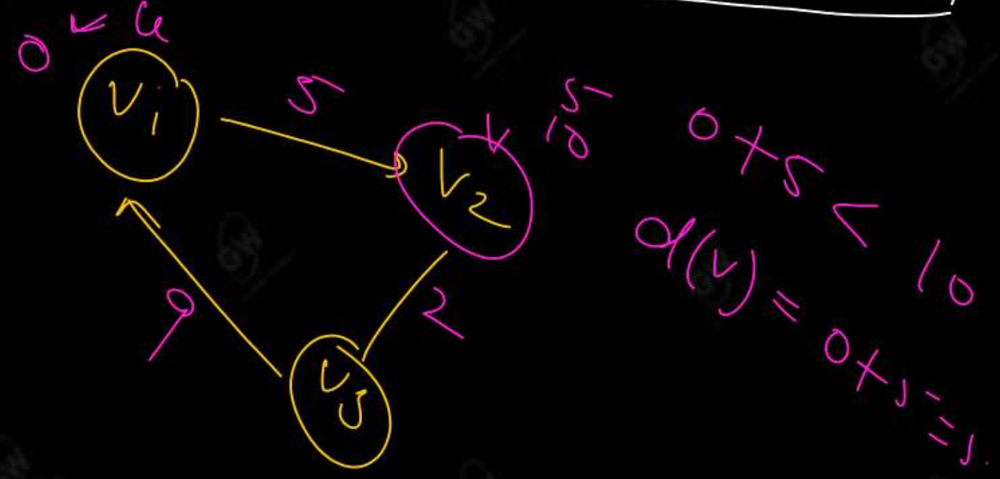
ALGORITHM : DIJKSTRA (G, W, S)

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

main formula :-

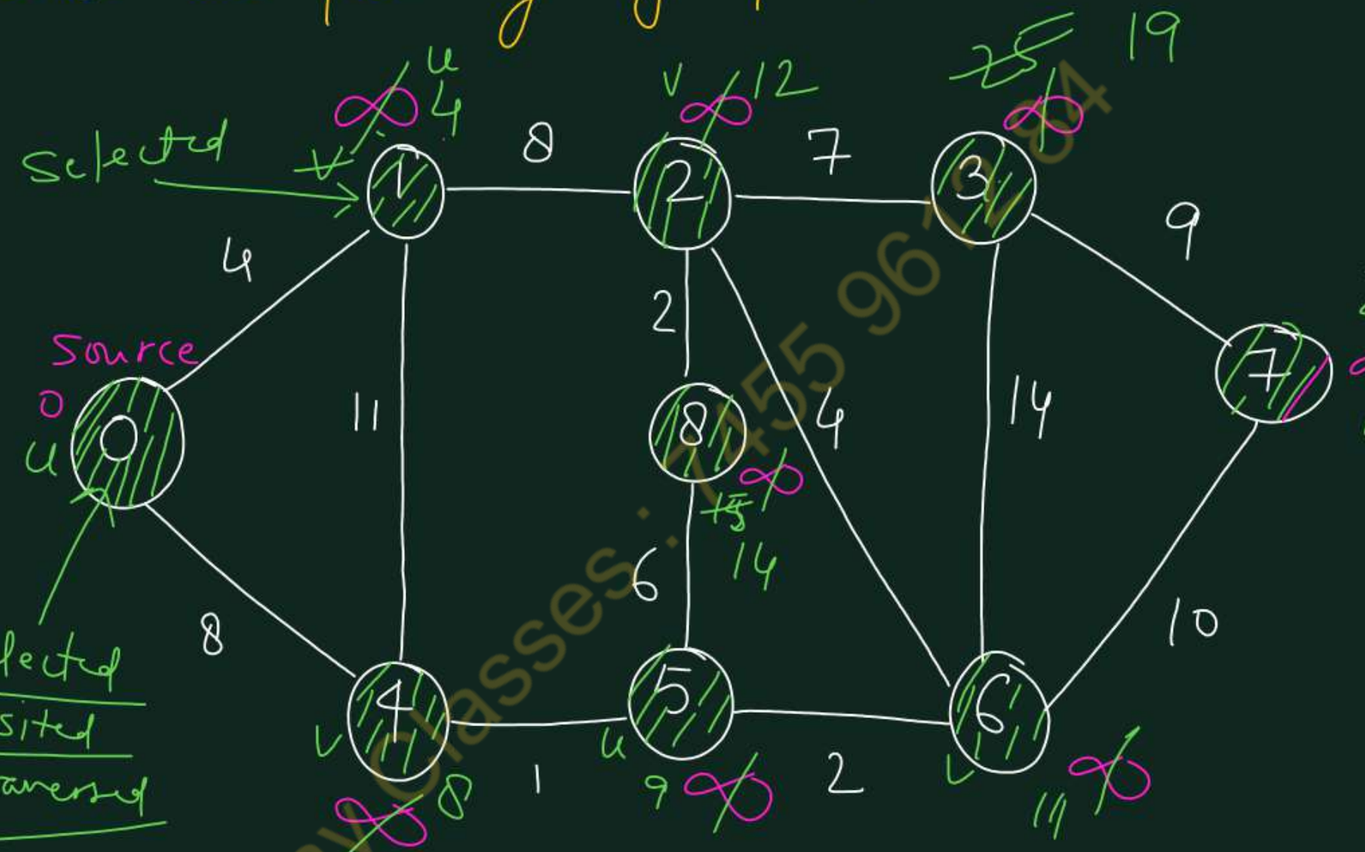
if $[d(u) + \text{cost}(u,v) < d(v)]$

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



- Because Dijkstra's always chooses the "lightest" or "closest" vertex in V - S to insert into set S, we say that it uses a greedy strategy.
- Dijkstra's algorithm bears some similarity to both breadth-first search and Prim's algorithm for computing minimum spanning trees.
- It is like breadth-first search in that set S correspond to the set of black vertices in a breadth-first search; just as vertices in Shave their final shortest path weights, so black vertices in a breadth-first search have their correct breadth-first distances.
- Dijkstra's algorithm is like Prim's algorithm in that both algorithms use a priority queue to find the "lightest" vertex outside a given set (the set S in Dijkstra's algorithm and the tree being grown in Prim's algorithm), insert this vertex into the set, and adjust the weights of the remaining vertices outside the set accordingly.

Example :- Consider the following graph :-



Undirected graph

0 to 1 = 4
0 to 4 = 8
0 to 2 = 12
0 to 5 = 9
0 to 8 = 14
0 to 3 = 19
0 to 6 = 11
0 to 7 = 21

Say Source Vertex = 0 ie find the shortest path 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 $[d(u) + \text{Cost}(u,v) < d(v)]$

ie $0 + 4 < \infty$

$(4 < \infty)$ True

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

ie. $d(v) = 4$

* Now update the distance at vertex 1, ~~∞~~ 4 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 $[\ d(u) + \text{Cost}((u,v) < d(v)\ ]$

ie $0 + 8 < \infty$

$(8 < \infty)$ True

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

$= 8$

* 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.

Choose

inimum

* 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 $d(u) + Cost(u,v) < d(v)$

$4 + 8 < \infty$

$12 < \infty$ True

then

$d(v) = d(u) + Cost(u,v)$

$d(v) = 12$

* Now Update the distance from 1 to 4

if $d(u) + c(u,v) < d(v)$

$4 + 11 < 8$

$15 < 8$ false

ie no need to modify the distance of Vertex 4.

Ans

Now, shortest distance from

source 0 to 1 = 4

4 = 8

2 = 12

8 = 14

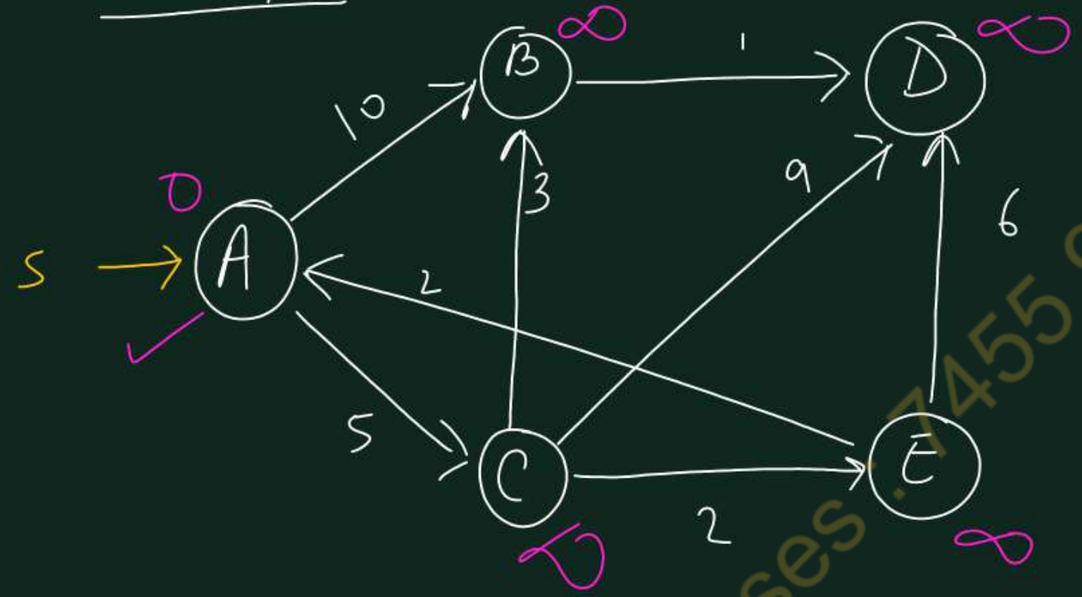
5 = 9

3 = 19

6 = 11

7 - 21

Another Example. Consider the following directed graph-



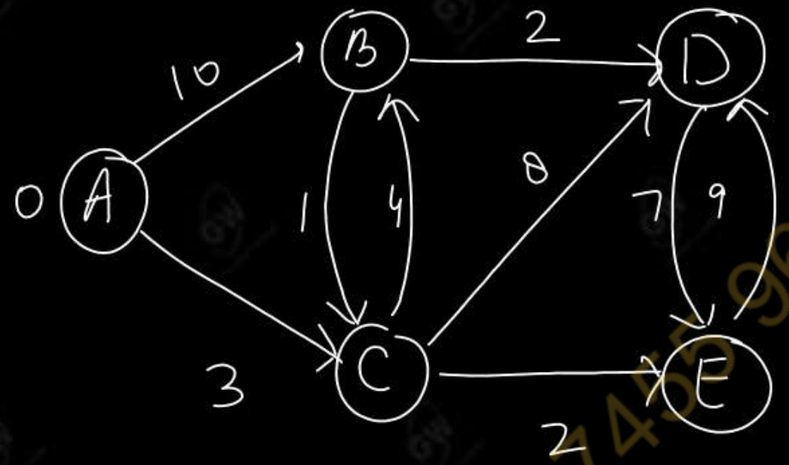
Traversal Select

Visited	A	B	C	D	E
A	[0]	∞	∞	∞	∞
C		10	[5]	∞	∞
E		8		14	[7]
B		[8]		13	
D				[9]	

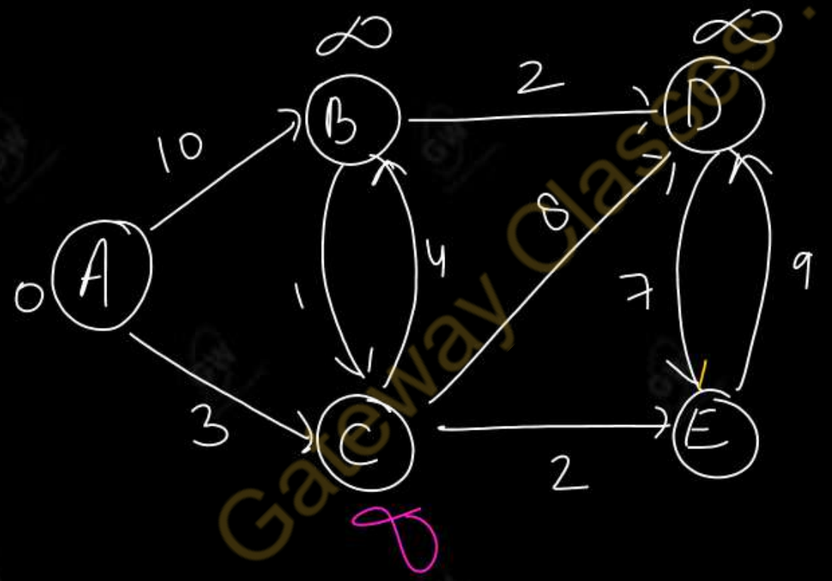
i.e. shortest distance
from A to B = 8
to C = 5
to D = 9
& A to E = 7

EXAMPLE: Consider the following graph with A as a source vertex.

3

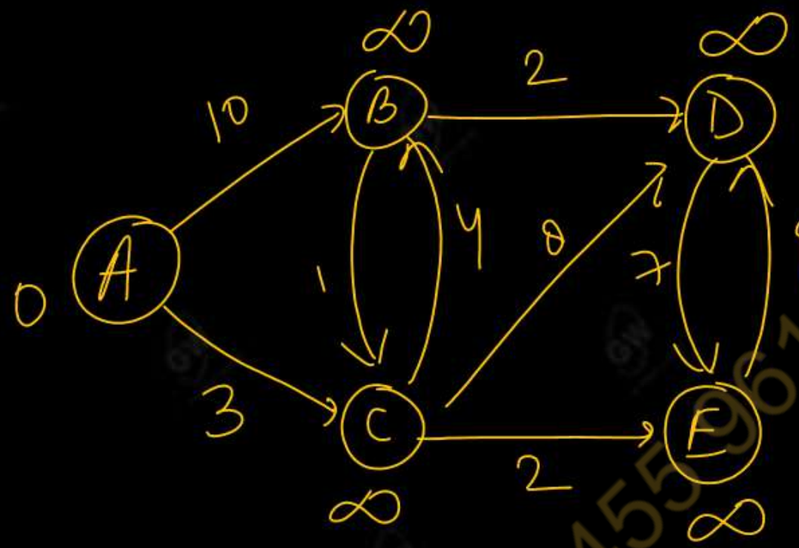


Initialise



Q	A	B	C	D	E
	0	∞	∞	∞	∞

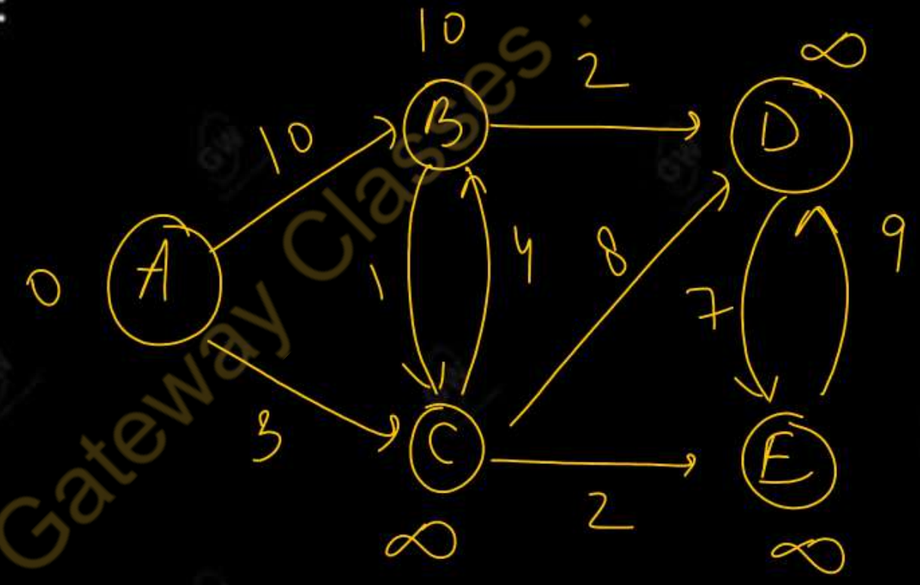
"A" ← EXTRACT-MIN (Q) :



Q:	A	B	C	D	E
	0	∞	∞	∞	∞

$S = \{A\}$

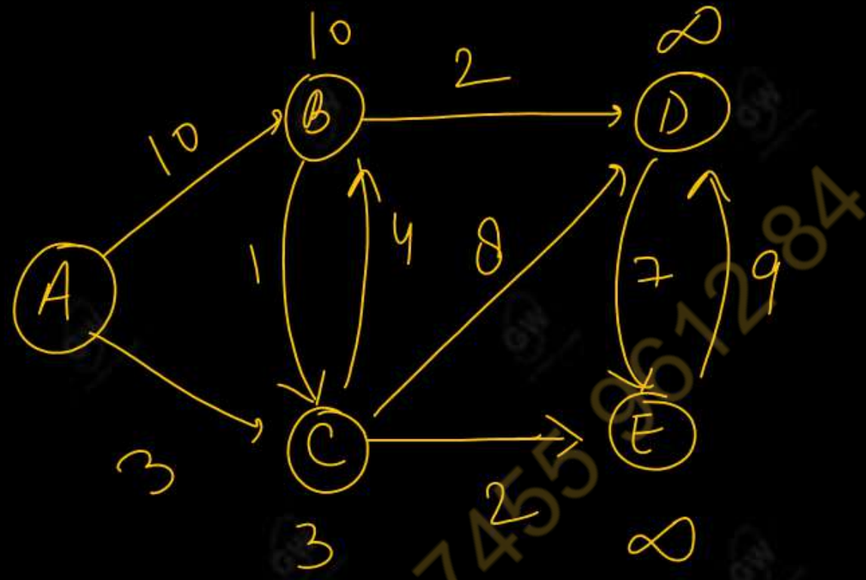
Relax all edges leaving A:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–

$S = \{A\}$

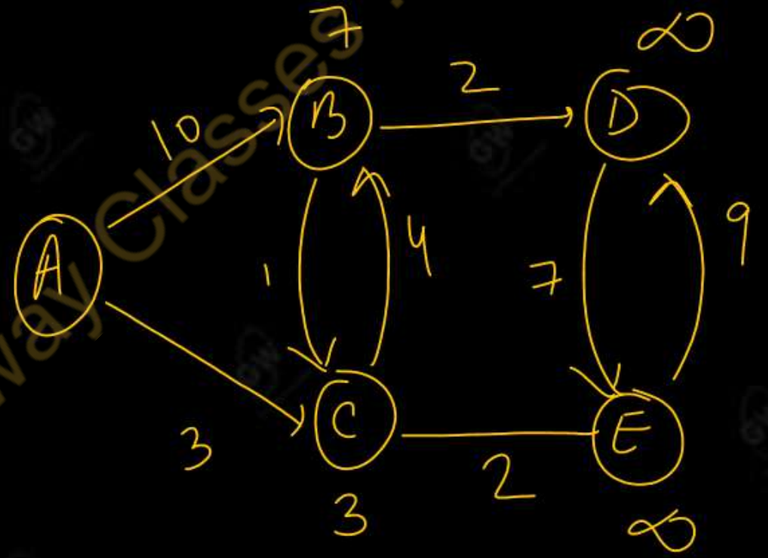
"C" ← EXTRACT-MIN (Q) :



Q:	A	B	C	D	E
	[0]	∞	∞	∞	∞
		10	[3]	-	-

S = { A, C }

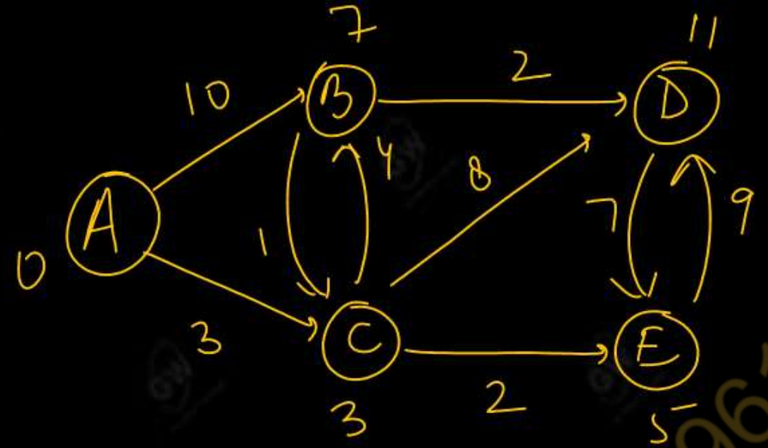
Relax all edges leaving C:



Q:	A	B	C	D	E
	[0]	∞	∞	∞	∞
		10	[3]	-	-
		7		11	5

S = { A, C }

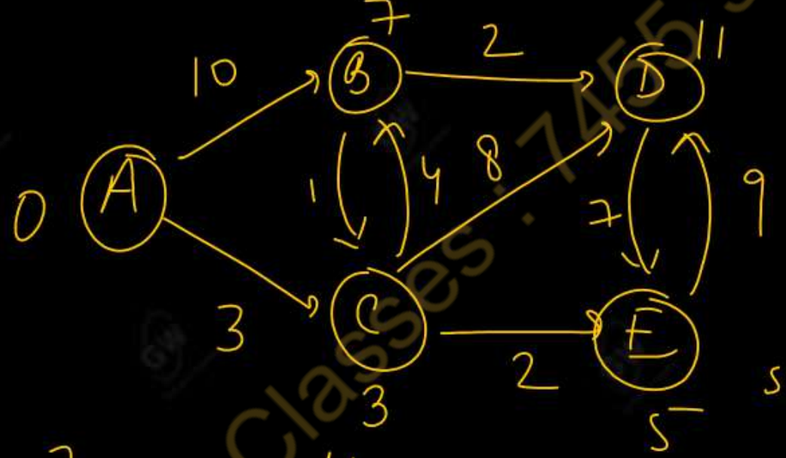
"E" ← EXTRACT-MIN (Q) :



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	–	–
		7		11	5

$S = \{A, C, E\}$

Relax all edges leaving E:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

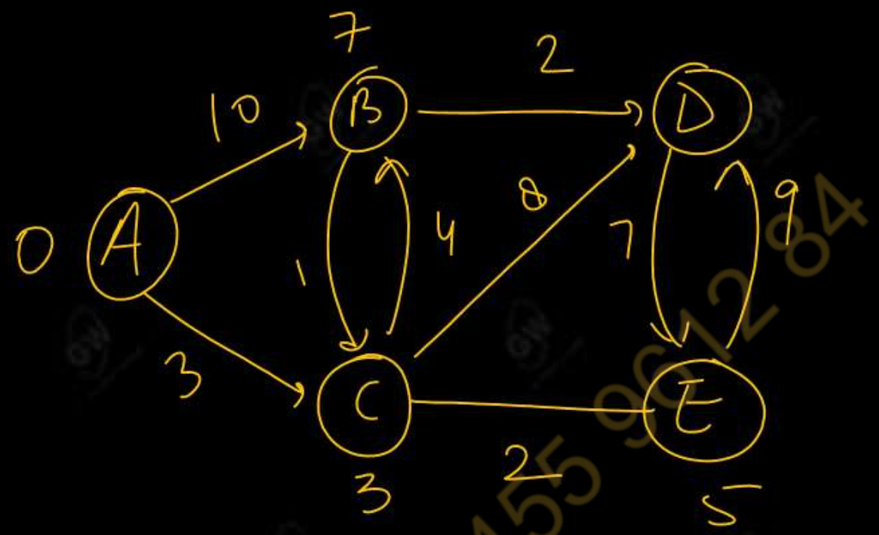
$S = \{A, C, E\}$

"B" ← EXTRACT-MIN (Q) :

Q	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
				11	

$S = \{A, C, E, B\}$

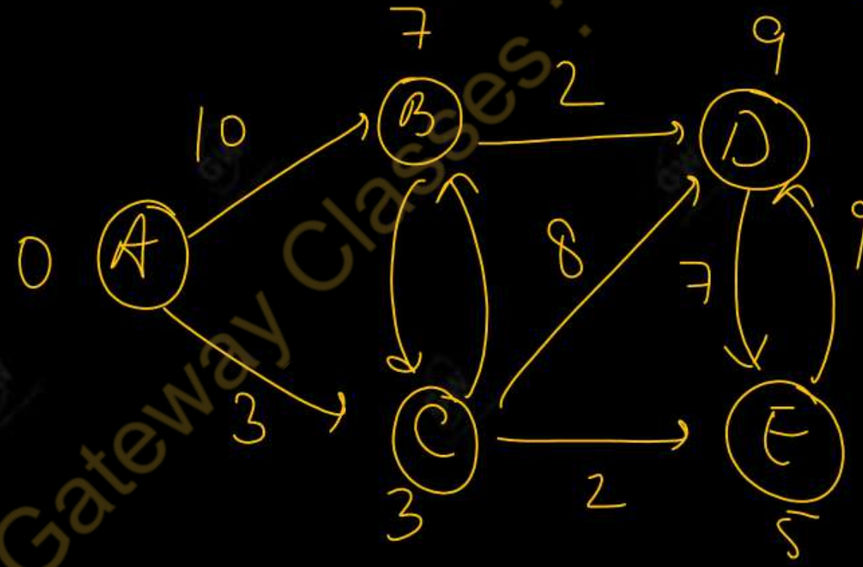
Relax all edges leaving B:



Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

"D" ← EXTRACT-MIN (Q) :

S = { A C E B }



S = { A S E B D }

Q	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

UNIT-5

1. Graph Terminology (Repn) — 2 marks
2. DFS / BFS
3. MST (Prim's / Kruskal's Algo)
4. Dijkstra Algo
5. Floyd Warshall's
6. Topological Sorting — 2/5

Unit - 5: Lec- 5

Today's Target

- ❑ Floyd Warshall's Algorithm
- ❑ Transitive Closure
- ❑ AKTU PYQs

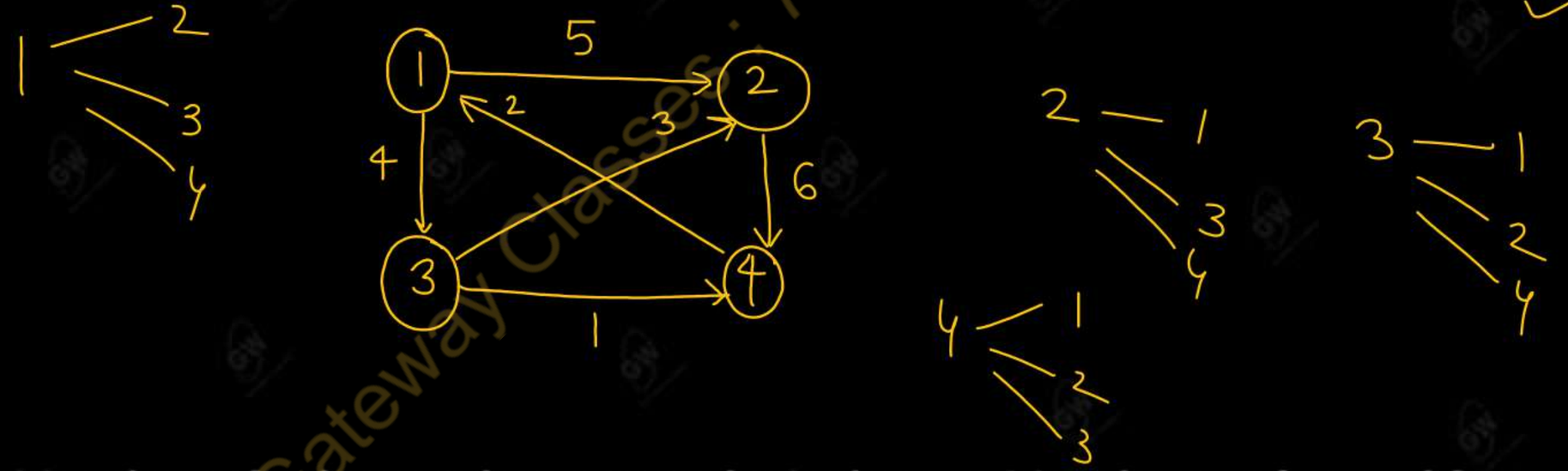
AKTU PYQs

Q.1. Explain Transitive Closure. **2017-18, 2 Marks**

Q.2. Write algorithm for Floyd Warshall algorithm also explains with a suitable example.

2017-18, 7 Marks

Q.3. Implement Floyd Warshall algorithm on the following graph. **2018-19, 7 Marks**

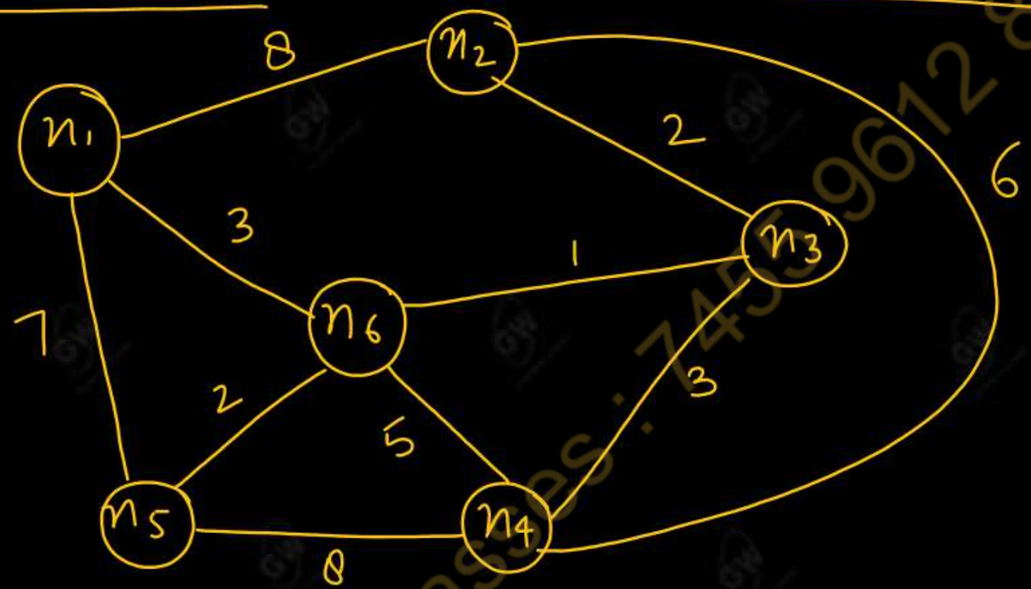


Q.4. What is transitive closure? What are the steps to obtain the transitive closure of a Graph?

2018-19, 8 Marks

Q.5. Write the Floyd Warshall algorithm to compute the all pair shortest path. Apply the algorithm on the following graph:

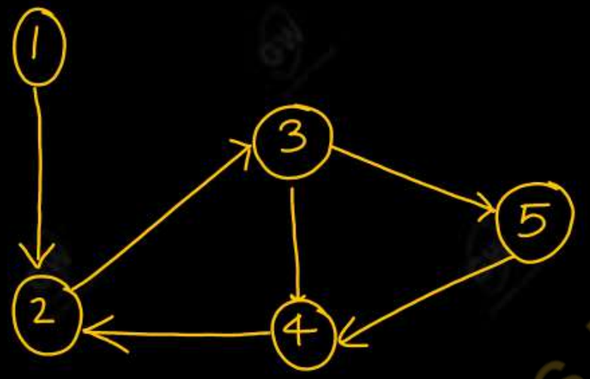
2018-19, 7 Marks



Q.6. Explain Warshall algorithm with the help of an example.

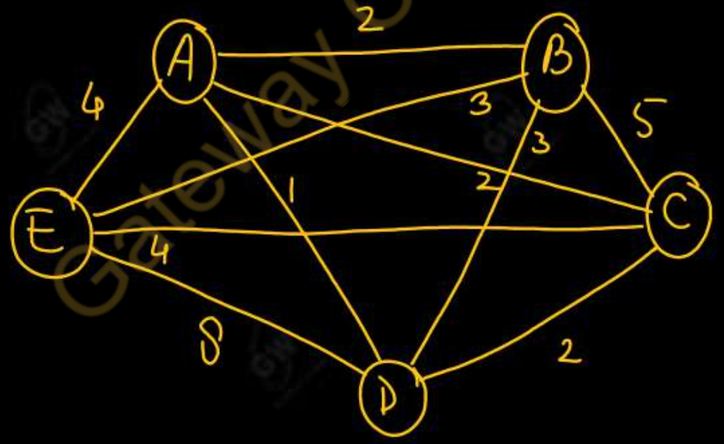
2019-20, 10 Marks

Q.7. Compute the transitive closure of the following graph: **2020-21, 2 Marks**

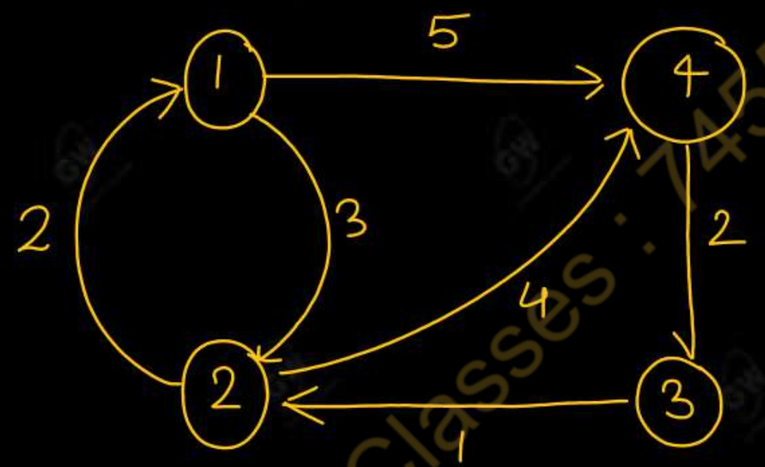


(AKTU Q 6a) (this is 66)

Q.8. Apply the Floyd Warshall's algorithm in above-mentioned graph. **2020-21, 10 marks**



Q.9. 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



WARSHALL'S ALGORITHM

- The algorithm considers the "intermediate" vertices of the shortest path, where an intermediate vertex of a simple path p = < v_1,v_2, ... v_m> is any vertex of p other than v_1 or v_m, that is, any vertex in the set { v_1, v_2 ... ... v_{m_1}}.
- The Floyd – Warshall's algorithm is based on the following observation -
- Let the vertices of G be V = {1, 2, ..., n}, and consider a sub-set { 1, 2, ..., k) of vertices for some k.
- For any pair of vertices i, j ∈ V, consider all paths from i to j whose intermediate vertices are all drawn from {1, 2, ..., k), and let p be a minimum - weight path from among them. (Path p is simple, since we assume that G contains no negative - weight cycles).
- The Floyd – Warshall's algorithm exploits a relationship between path p and shortest paths from i to j with all intermediate vertices in the set { 1, 2, ..., k – 1 }.
- The relationship depends on whether or not k is an intermediate vertex of path p.

- If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set { 1, 2, ..., k – 1 }.
- Thus, a shortest path from vertex i to vertex j with all intermediate vertices in the set { 1, 2, ..., k – 1 } is also a shortest path from i to j with all intermediate vertices in the set (1, 2, ..., k).
- If k is intermediate vertex of path p, then we break p down into $i\ P_1\ k\ P_2\ j$.
- Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in the set { 1, 2, ..., k }.

A recursive definition is given by -

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

$$w_{ij} = \begin{cases} 0 & if\ i = k \\ w(i,j) & if\ i \neq j\ and\ (i,j) \in E \\ \infty & if\ i \neq j\ and\ (i,j) \notin E \end{cases}$$

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

Let us define the V x *V* matrix -

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

$$d_{ij}^{(m)} = \text{the length of the shortest path from } i \text{ to } j \text{ with } \leq m \text{ edges.}$$

$$When\ m = 0, there\ is\ a\ shortest\ path\ from\ i\ to\ j\ with\ no\ edges\ if\ and\ only\ if\ i = j.$$

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

ALGORITHM: FLOYD – WARSHELL'S (W)

Step 1. $n \leftarrow rows\,[W]$

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

Step 3. $for\ k \leftarrow 1\ to\ n$

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

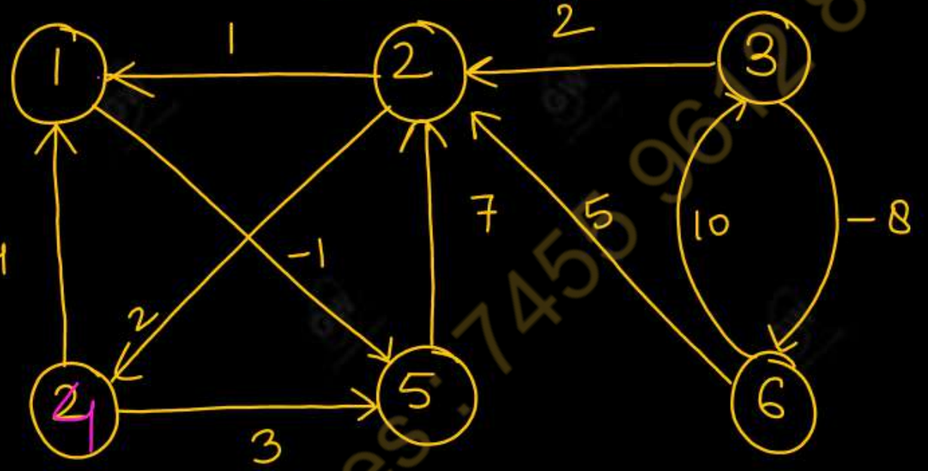
Step 5. $do\ for\ j \leftarrow 1\ to\ n$

Step 6. $do\ d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)\ 1\ to\ n$

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

EXAMPLE 1. Apply Floyd – Warshall's algorithm for constructing shortest path. Show the matrix $D^{(k)}$ that results each iteration.

All Pair shortes Paths



Directed graph
weighted graph

Distance matrix

$D^0 =$

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

$$d_{ij}^{(K)} = \min\left[d_{ij}^{(K-1)} , d_{iK}^{(K-1)} + d_{Kj}^{(K-1)} \right]$$

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

$D^1(2,3) = \min\{ D^0(2,3), D^0(2,1) + D^0(1,3) \}$

$= \min\{ \infty, 1 + \infty \}$

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

$= \infty$

$D^1(2,5) = \min\{ D^0(2,5), D^0(2,1) + D^0(1,5) \}$

$= \min\{ \infty, 1 + (-1) \}$

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

$= 0$

$D^1(2,4) = \min\{ D^0(2,4), D^0(2,1) + D^0(1,4) \}$

$= \min\{ 2, 1 + \infty \}$

$= \min\{ 2, \infty \}$

$= 2$

$D^{(2)}$ =

	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	6	5	10	7	5	0

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

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

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

$= \infty$

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

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

$= -1$

$D^{(3)} =$

	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

$D^3(5,1) = \min\left(D^2(5,1),\ D^2(5,3) + D^2(3,1)\right)$

$= \min\{8,\ \infty + \}$

$= \min\{8, \infty\}$

$= 8$

$D^{(4)} =$

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

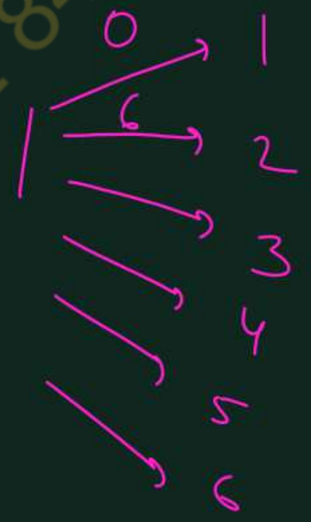
$D^{(5)} =$

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

$D^{(6)} =$

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

Ans



The shortest path from 3 to 1 is -

$3 \rightarrow 6 \rightarrow 2 \rightarrow 4 \rightarrow 1$ and is -5 units.

The shortest path from from 6 to 2 is - 5 units

etc.

Transitive Closure of a Graph

- The transitive closure of a graph G is defined to be the graph G′has the same nodes as G and there is an edege(v_i, v_j)in G′whenever there is a path from vi to vj in G.
- Accordingly the path matrix P of the graph G is precisely the adjacency matrix of its transitive closure G′.
- The transitive closure of a graph G is defined as G ∗ or G′ = (V, E ∗).

where,

$E* = \{(i,j)$there is a path from vertex i to vertex j in $G\}$

- For $i, j, k = 1,2, \ldots\ldots\ldots, n$ we define $t_{ij}^{(k)}$ to be 1 if there exists a path in graph G from vertex i to vertex j with all intermediate vertices in the set $\{1,2, \ldots\ldots, k\}$and 0 otherwise.
- We construct the transitive closure −

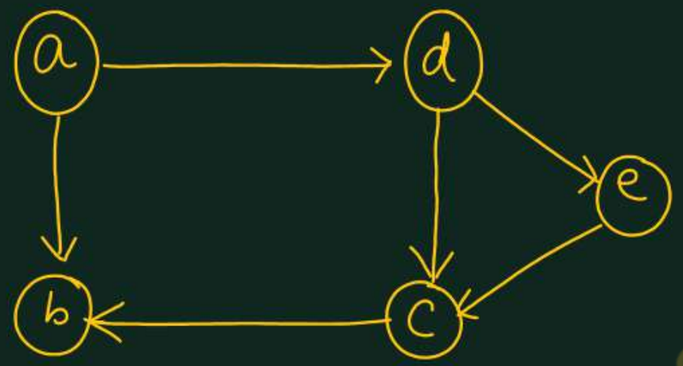
$G* = (V, E*)$by putting edge (i,j)into $E*$ if and only if $t_{ij}^{(k)}$ is

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i,j) \in E \\ 1 & \text{if } i = j \text{ or } (i,j) \in E \end{cases} \quad \text{and} \quad \text{for } k \geq 1 \quad t_{ij}^{(k)} = t_{ij}^{(k-1)} \quad t_{ik}^{(k-1)} \quad t_{kj}^{(k-1)}$$

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 $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \quad t_{ik}^{(k-1)} \quad t_{kj}^{(k-1)}$

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

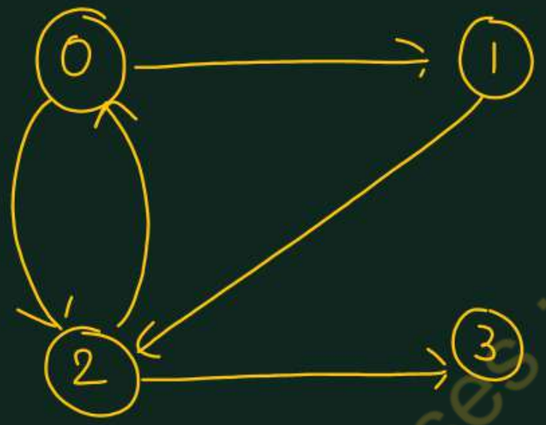


Reachability matrix :

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

Transitive Closure of this graph.

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



reachability matrix -

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

Transitive closure

Download Gateway Classes Application
From Google Play store
Link in Description

Thank You