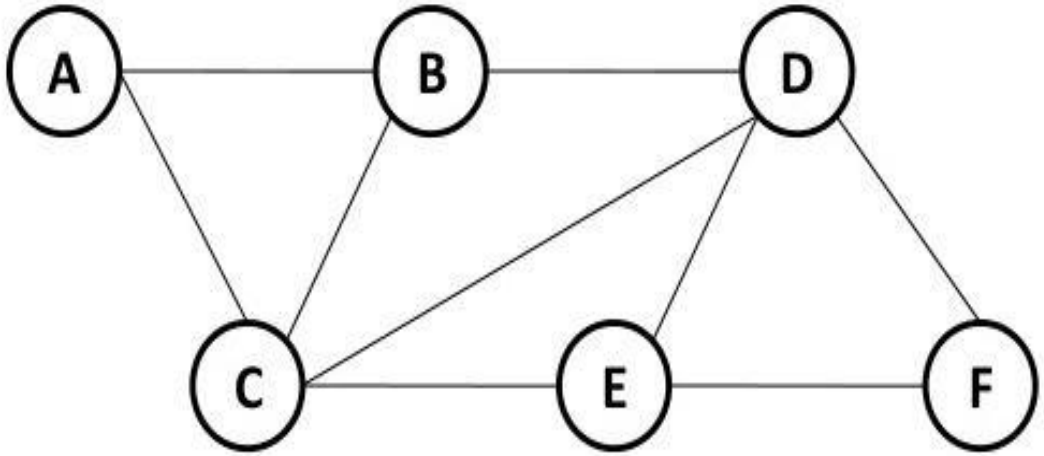
Unit-8
Graph

Graph is a non Linear Data structure which is a collection of two finite sets of V & E
Where V is vertices(node) and E is Edges (arcs)
i.e. G = (V, E)
Where V = {A, B, C, D, E, F} &
E = { (A, B}, (A,C), (B,C), (B,D), (C,D), (C,E), (D,E), (D,F), (E,F)}
The pair (A,B) defines an edge between A and B.

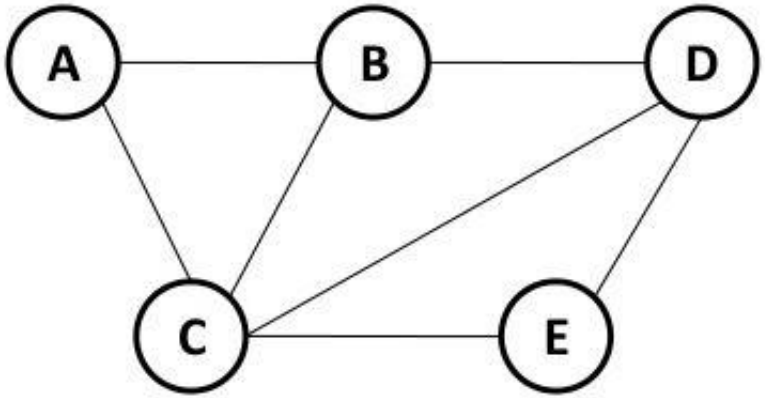
E.g.



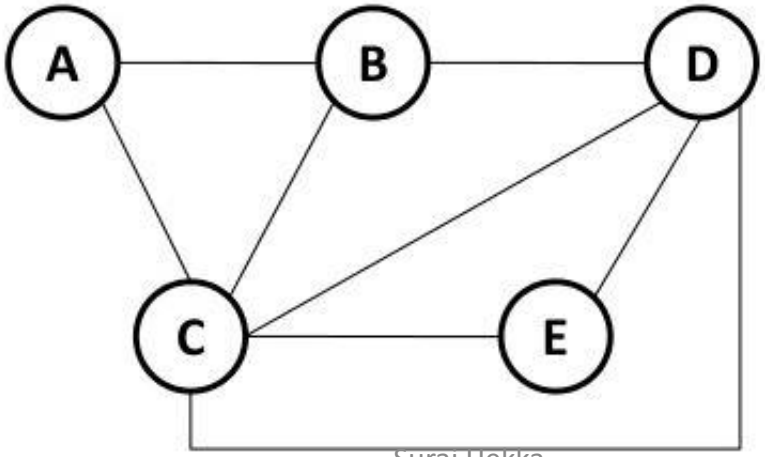
The number of vertices of graph constitute the order of graph. Hence In given Example the order of graph is **Six**

Types of Graph

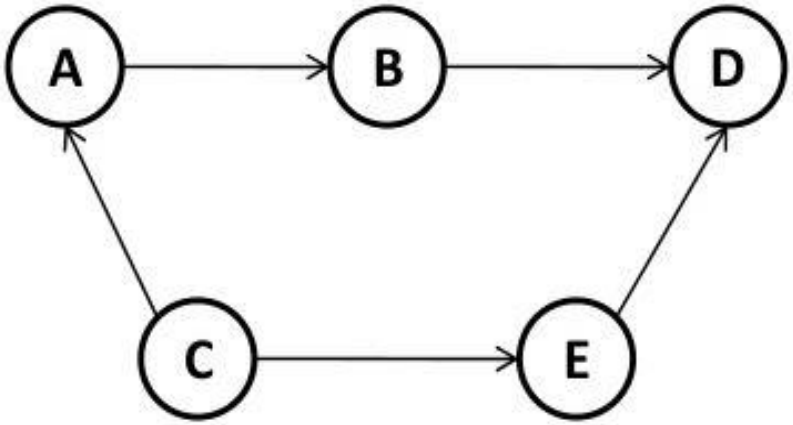
1. Simple Graph: A graph that does not contain any loop or multiple edges



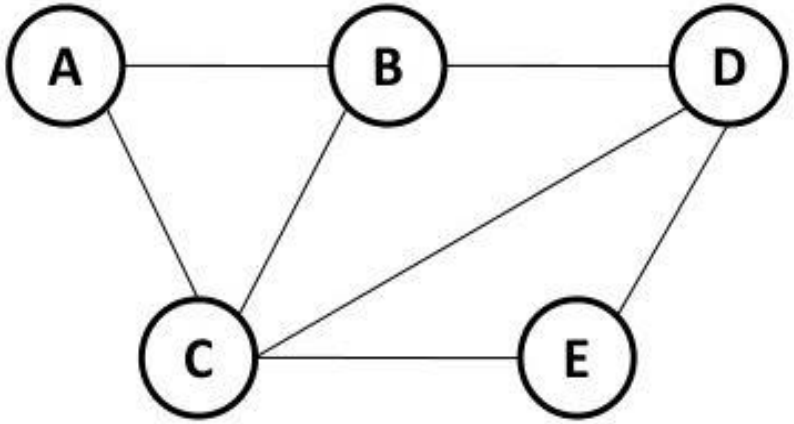
2. Multi graph: A graph that contain more then one edges between two vertices



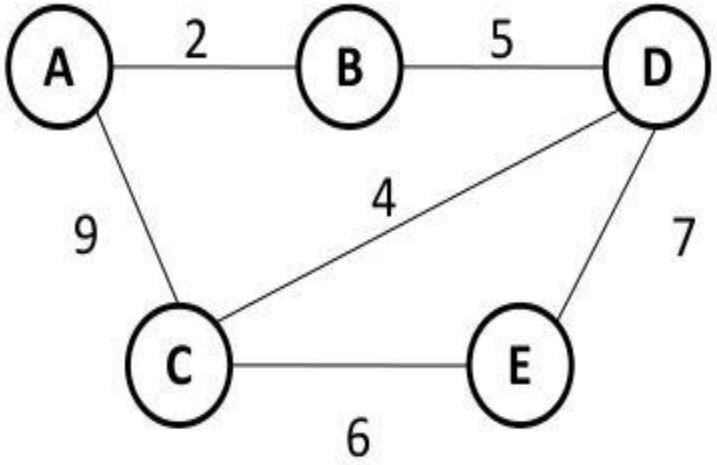
3. Directed Graph: A graph with directed edges is called directed graph



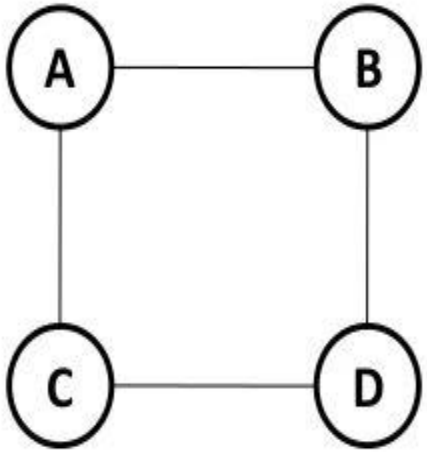
4. Undirected Graph: A graph with undirected edges is called undirected graph



5. Weighted Graph: A graph with Weight (value) assign to edges is called weighted graph

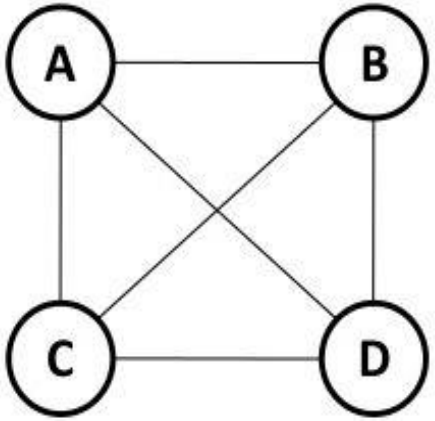


6. Regular Graph: A graph in which each vertex has same degree is called regular graph

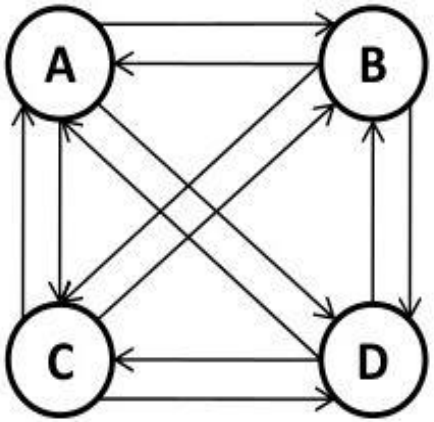


Note: k is degree of each vertex
hence it is called k regular graph

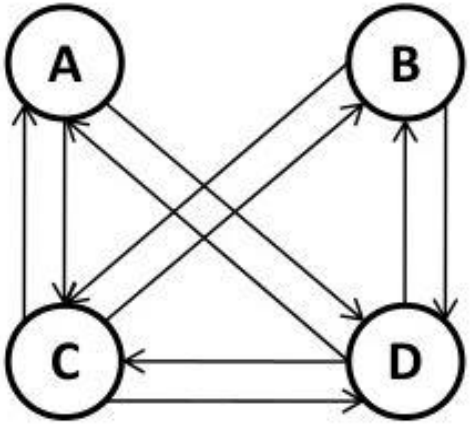
7. Complete Graph: A graph in which each vertex is connected to every other vertices called complete graph



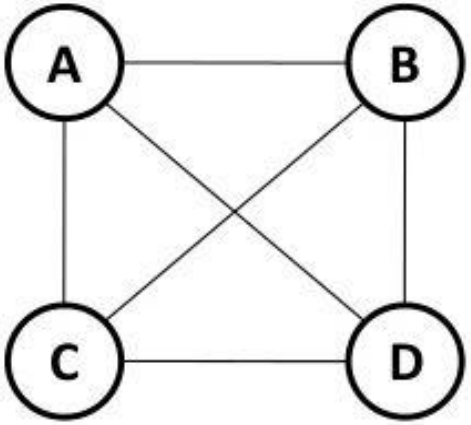
8. Strongly connected Graph: In Directed graph each vertex is connected to every other vertices called Strongly connected graph

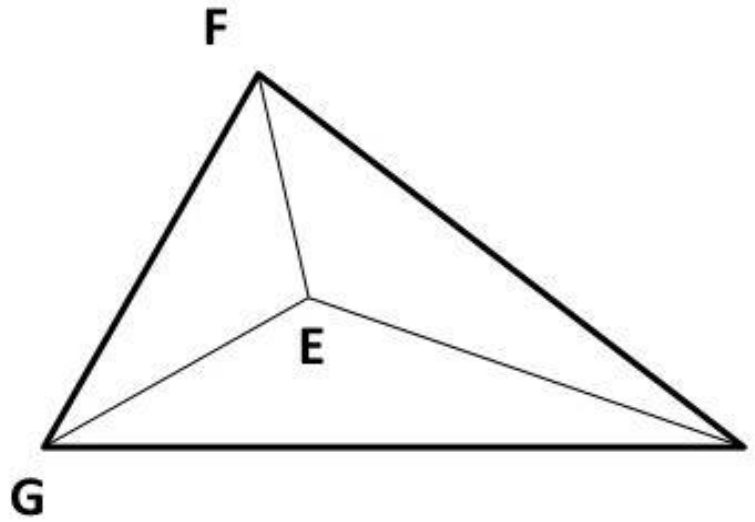


9. Weakly connected Graph: In Directed graph weakly connected if at least two vertices are not connected.



10. Isomorphic Graph: Two graph in which there is one to one correspondence between their vertices and their edges is said to be isomorphic graph .

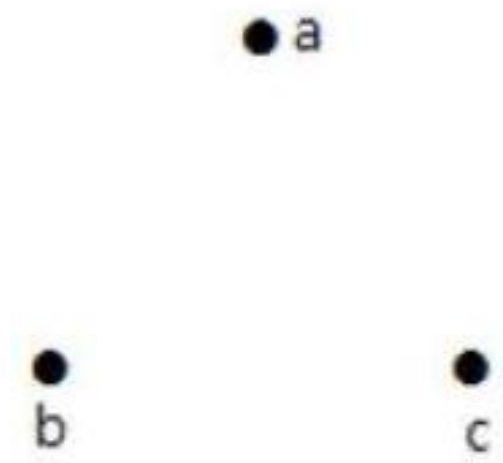




11. Null Graph

A **graph having no edges** is called a Null Graph.

Example



In the above graph, there are three vertices named 'a', 'b', and 'c', but there are no edges among them. Hence it is a Null Graph.

12. Trivial Graph

A **graph with only one vertex** is called a Trivial Graph.

Example

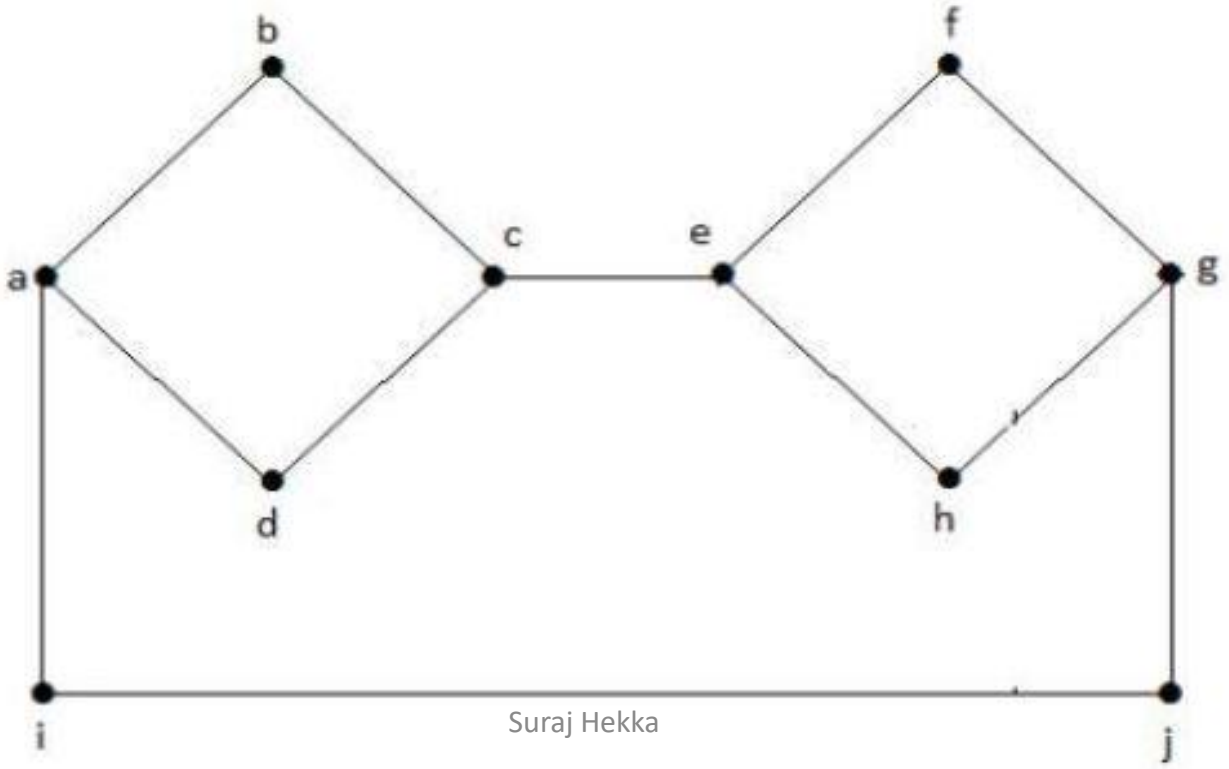
In the above shown graph, there is only one vertex 'a' with no other edges. Hence it is a Trivial graph.

13. Connected Graph

A graph G is said to be connected **if there exists a path between every pair of vertices**. There should be at least one edge for every vertex in the graph. So that we can say that it is connected to some other vertex at the other side of the edge.

Example

In the following graph, each vertex has its own edge connected to other edge. Hence it is a connected graph.

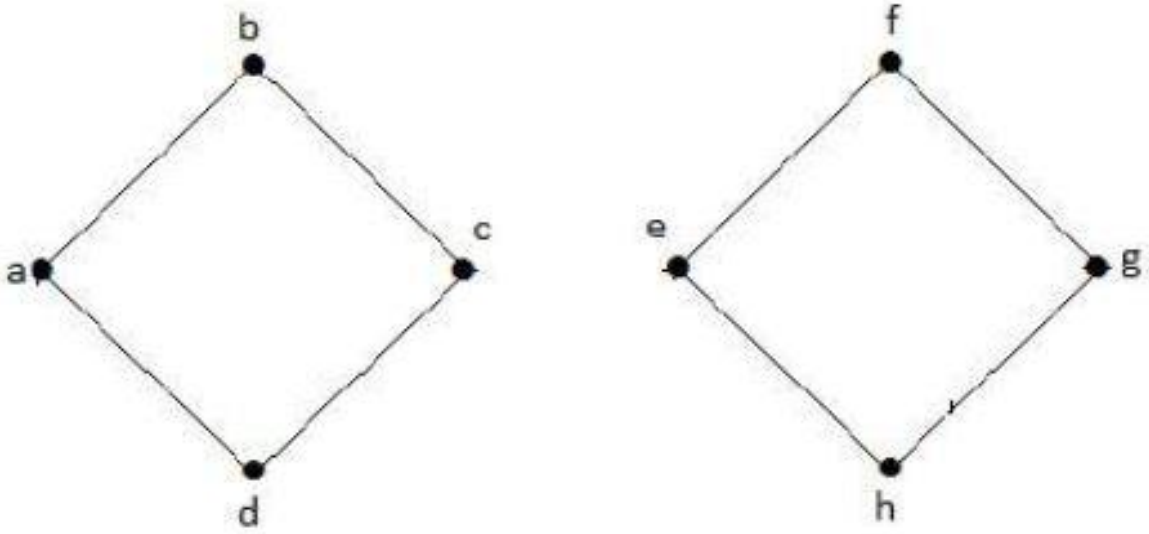


14. Disconnected Graph

A graph G is disconnected, if it does not contain at least two connected vertices.

Example 1

The following graph is an example of a Disconnected Graph, where there are two components, one with 'a', 'b', 'c', 'd' vertices and another with 'e', 'f', 'g', 'h' vertices.



The two components are independent and not connected to each other. Hence it is called disconnected graph.

15. Cycle Graph

A simple graph with 'n' vertices (n >= 3) and 'n' edges is called a cycle graph if all its edges form a cycle of length 'n'.

If the **degree of each vertex in the graph is two,** then it is called a Cycle Graph.

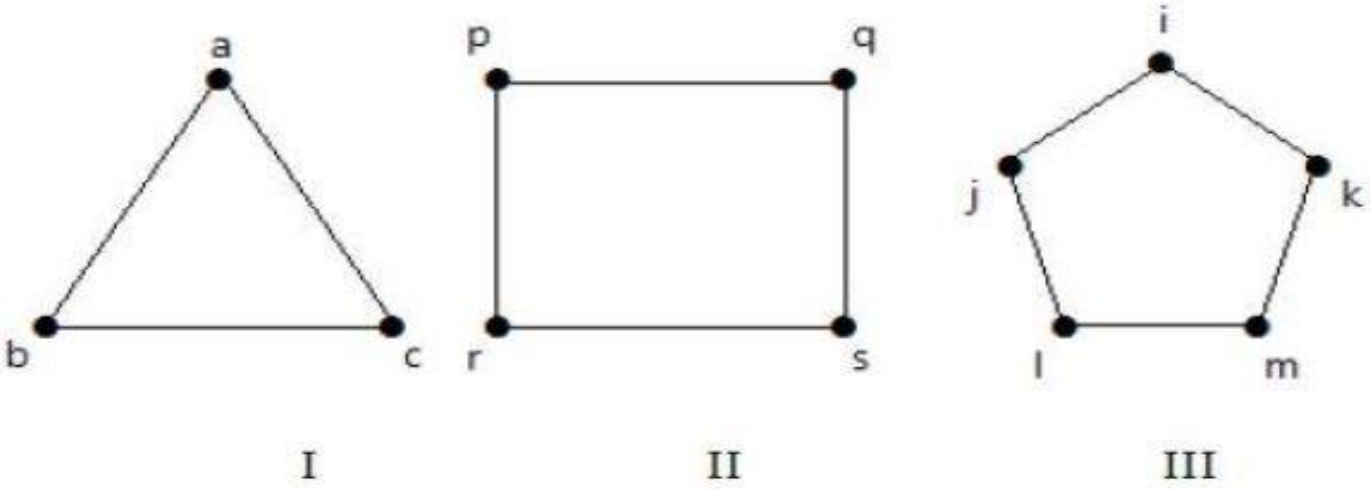
Example

Take a look at the following graphs –

Graph I has 3 vertices with 3 edges which is forming a cycle 'ab-bc-ca'.

Graph II has 4 vertices with 4 edges which is forming a cycle 'pq-qs-sr-rp'.

Graph III has 5 vertices with 5 edges which is forming a cycle 'ik-km-ml-lj-ji'.

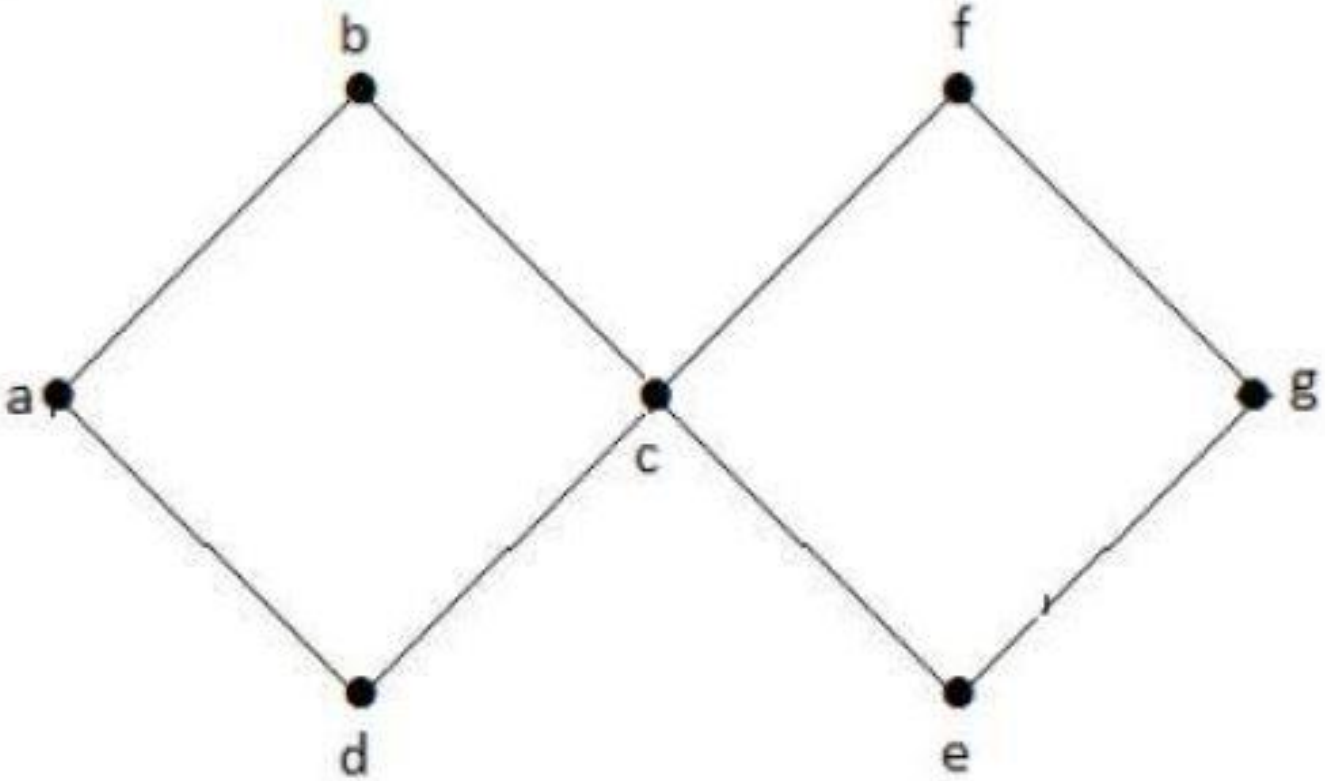


Hence all the given graphs are cycle graphs.

16. Cyclic Graph

A graph **with at least one** cycle is called a cyclic graph.

Example

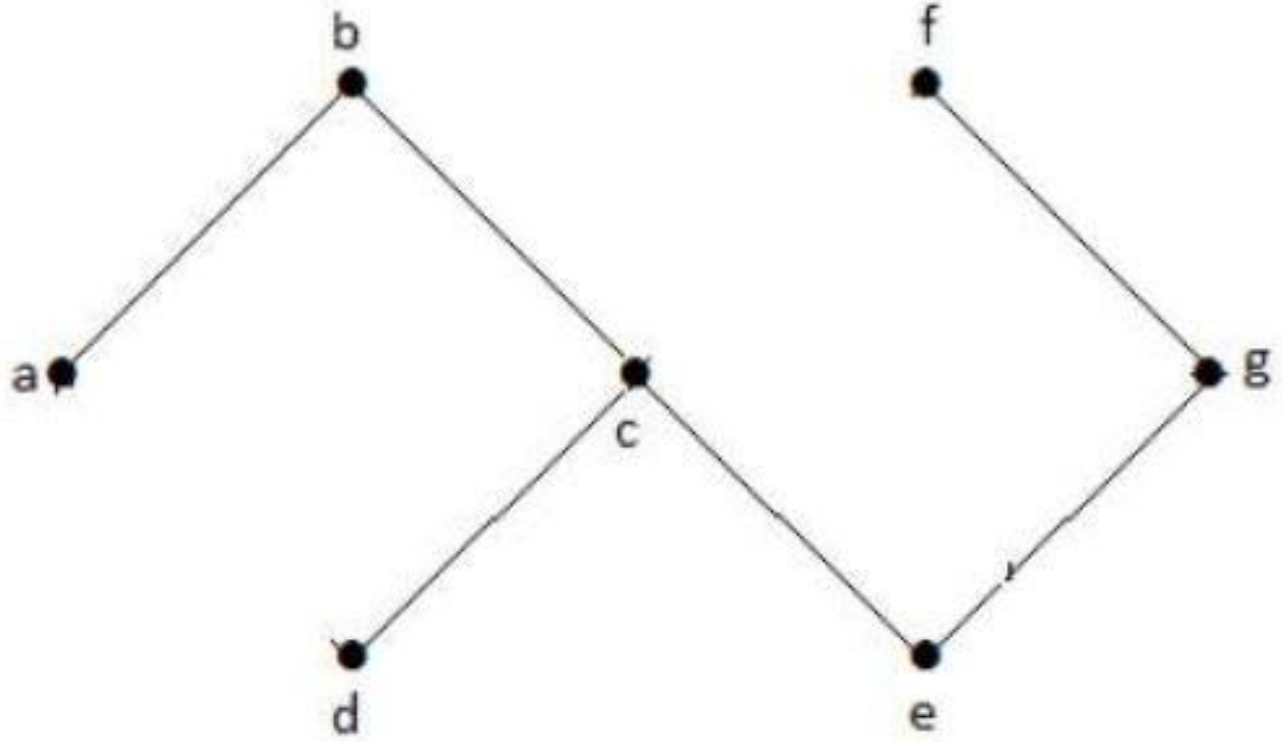


In the above example graph, we have two cycles a-b-c-d-a and c-f-g-e-c. Hence it is called a cyclic graph.

17. Acyclic Graph

A graph **with no cycles** is called an acyclic graph.

Example



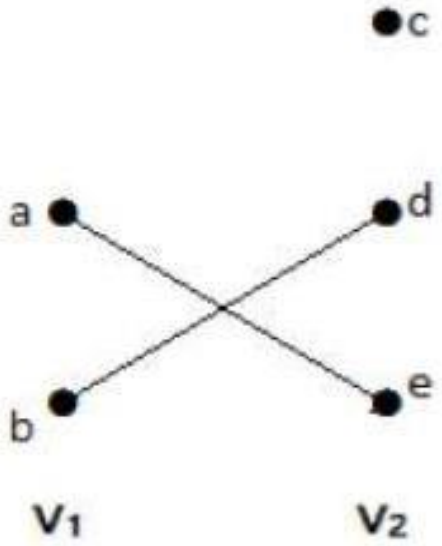
In the above example graph, we do not have any cycles. Hence it is a non-cyclic graph.

18. Bipartite Graph

A simple graph G = (V, E) with vertex partition V = {V_1, V_2} is called a bipartite graph **if every edge of E joins a vertex in V_1 to a vertex in V_2.**

In general, a Bipertite graph has two sets of vertices, let us say, V_1 and V_2, and if an edge is drawn, it should connect any vertex in set V_1 to any vertex in set V_2.

Example



In this graph, you can observe two sets of vertices – V_1 and V_2. Here, two edges named 'ae' and 'bd' are connecting the vertices of two sets V_1 and V_2.

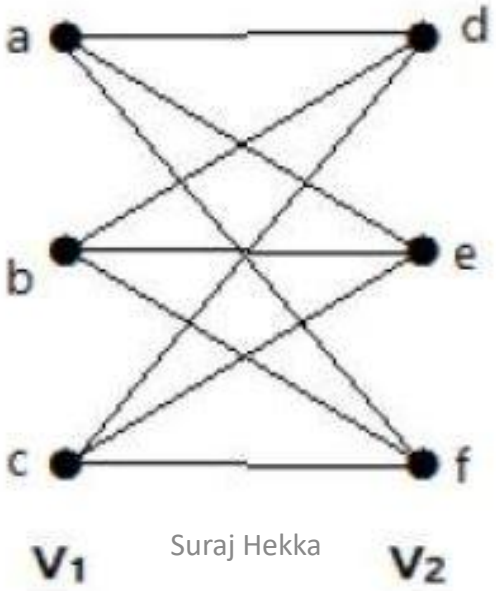
19. Complete Bipartite Graph

A bipartite graph 'G', G = (V, E) with partition V = {V_1, V_2} is said to be a complete bipartite graph if every vertex in V_1 is connected to every vertex of V_2.

In general, a complete bipartite graph connects each vertex from set V_1 to each vertex from set V_2.

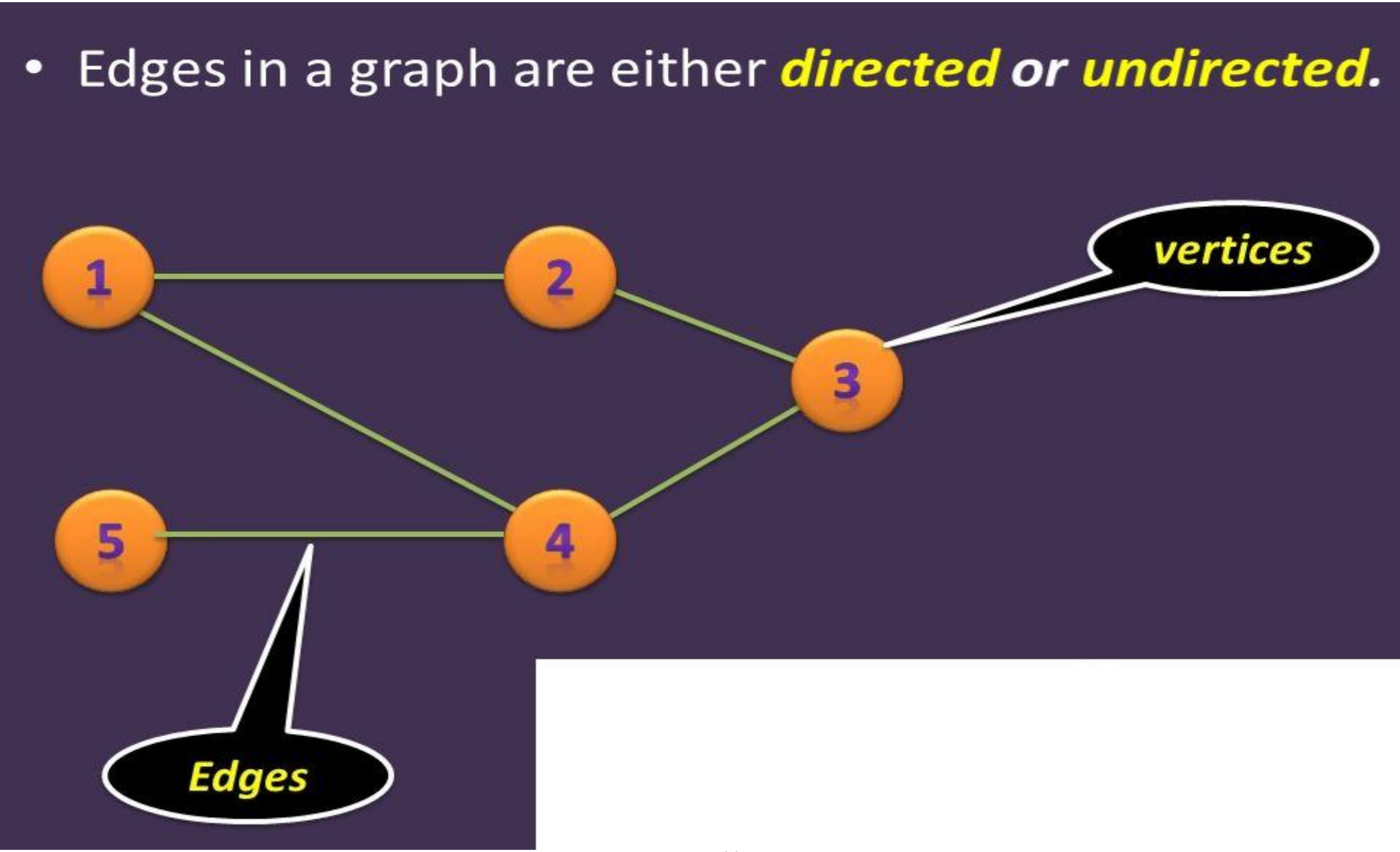
Example

The following graph is a complete bipartite graph because it has edges connecting each vertex from set V_1 to each vertex from set V_2.



Graph Terminologies:

1. Edge: It is also called a arc.



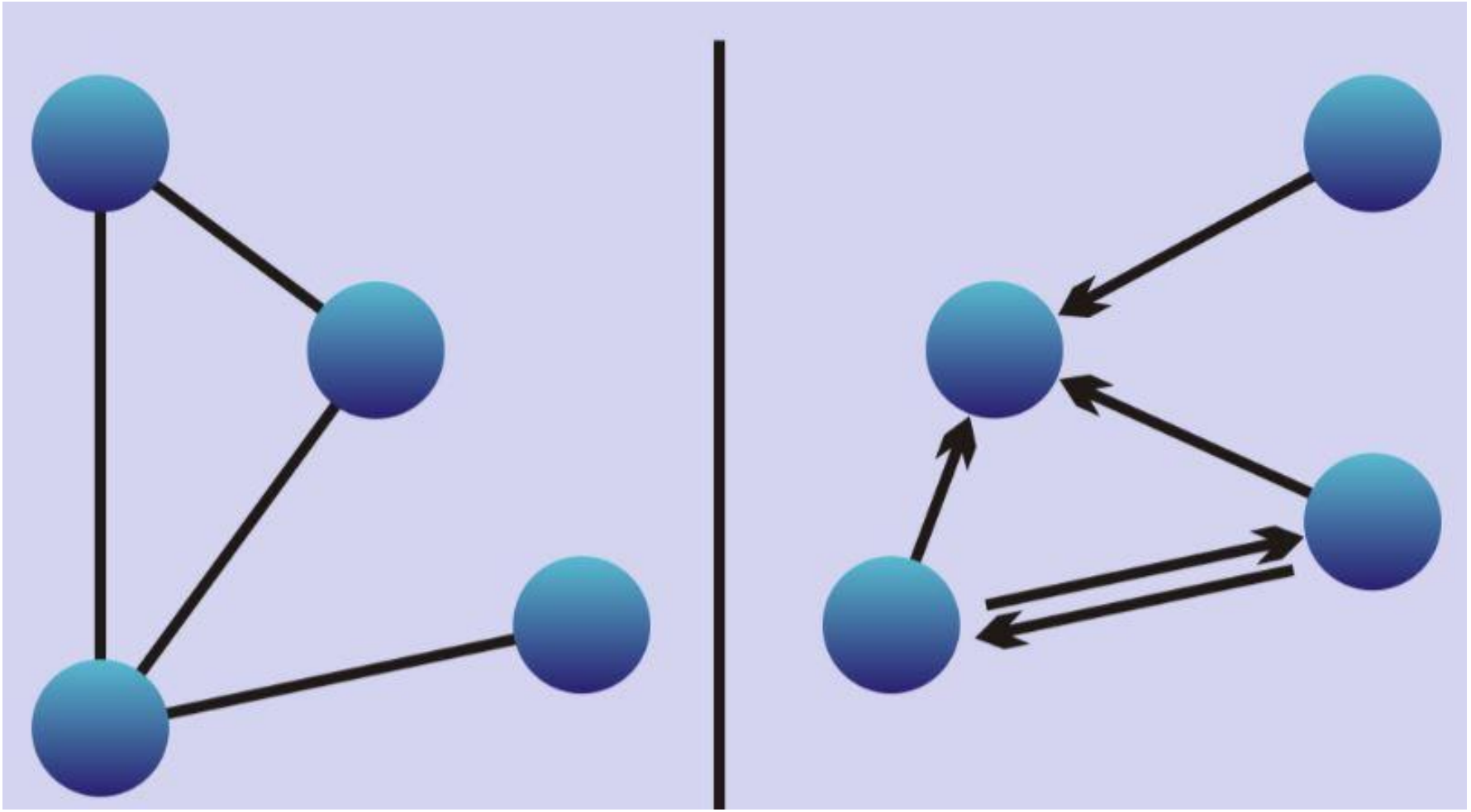


Fig: Undirected Edge

Fig: Directed Edge

2. Vertex:

- A vertex is a point where multiple lines meet.
- A vertex is also called a node in graph.
- A, B, C, D, E, F are vertices in a Graph ‘G’.

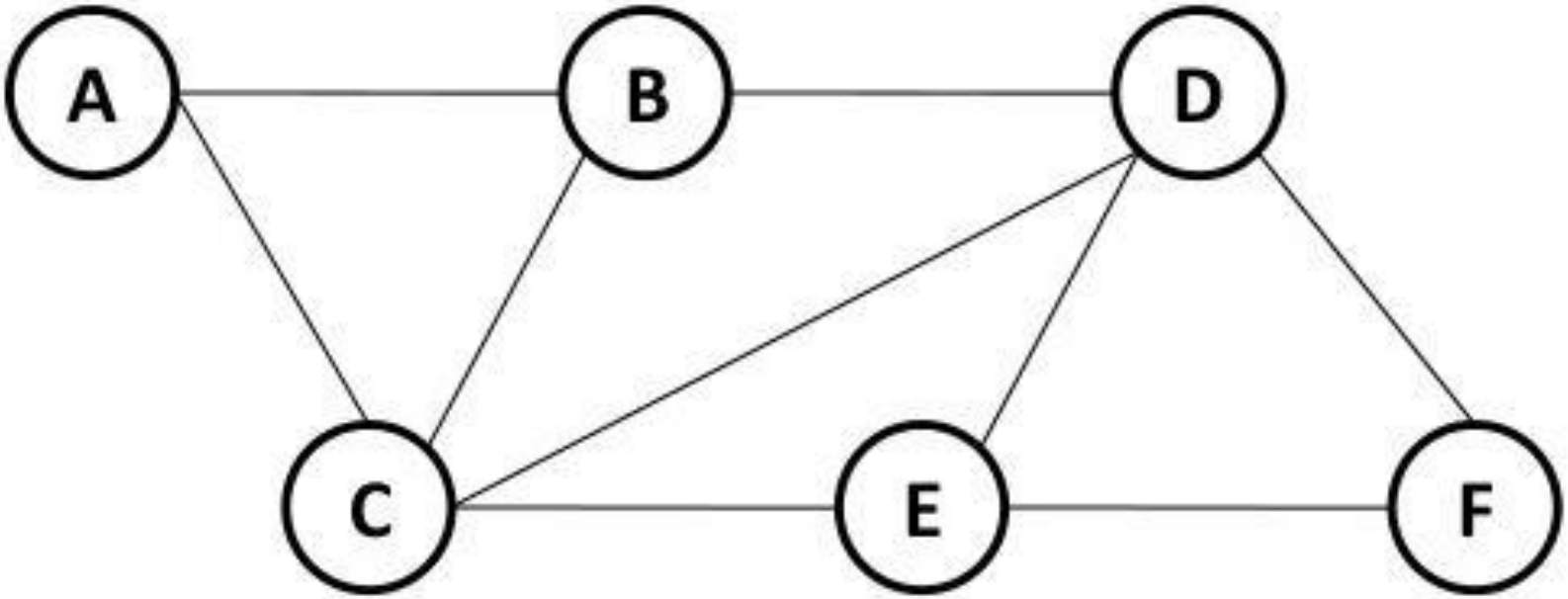


Fig 1: Graph ‘G’

3. **Adjacent vertices:** Two vertices are said to be adjacent if they are connected by an edge.
   e.g. B & C **In Fig 1: Graph 'G'**
4. **Adjacent Edges:** Two edges are said to be adjacent if they are incident on the common vertex.
   e.g. CD & CE. **In Fig 1: Graph 'G'**
5. **Path:** A Path is a sequence of distinct vertices in which each vertex is adjacent to next one.
   e.g. ABCDEF **In Fig 1: Graph 'G'**
6. **Cycle:** A cycle is a path consisting of at least three vertices where last vertex is adjacent to the first vertex.
   e.g. C D F E C **In Fig 1: Graph 'G'**
7. **Loop:** It is special cycle that start and end with same vertex without visiting any other vertex.
8. **Degree of Vertex:** The degree of a vertex is the number of lines incident on it
   e.g. degree of D is 4 **In Fig 1: Graph 'G'**

9. **Out-Degree:** The out Degree of directed graph is the number of arcs leaving the vertex.
10. **In-Degree:** In degree of directed graph is number of arcs entering the vertex.
11. **Multiple Edges:** The distinct parallel edges that connect the same end point.

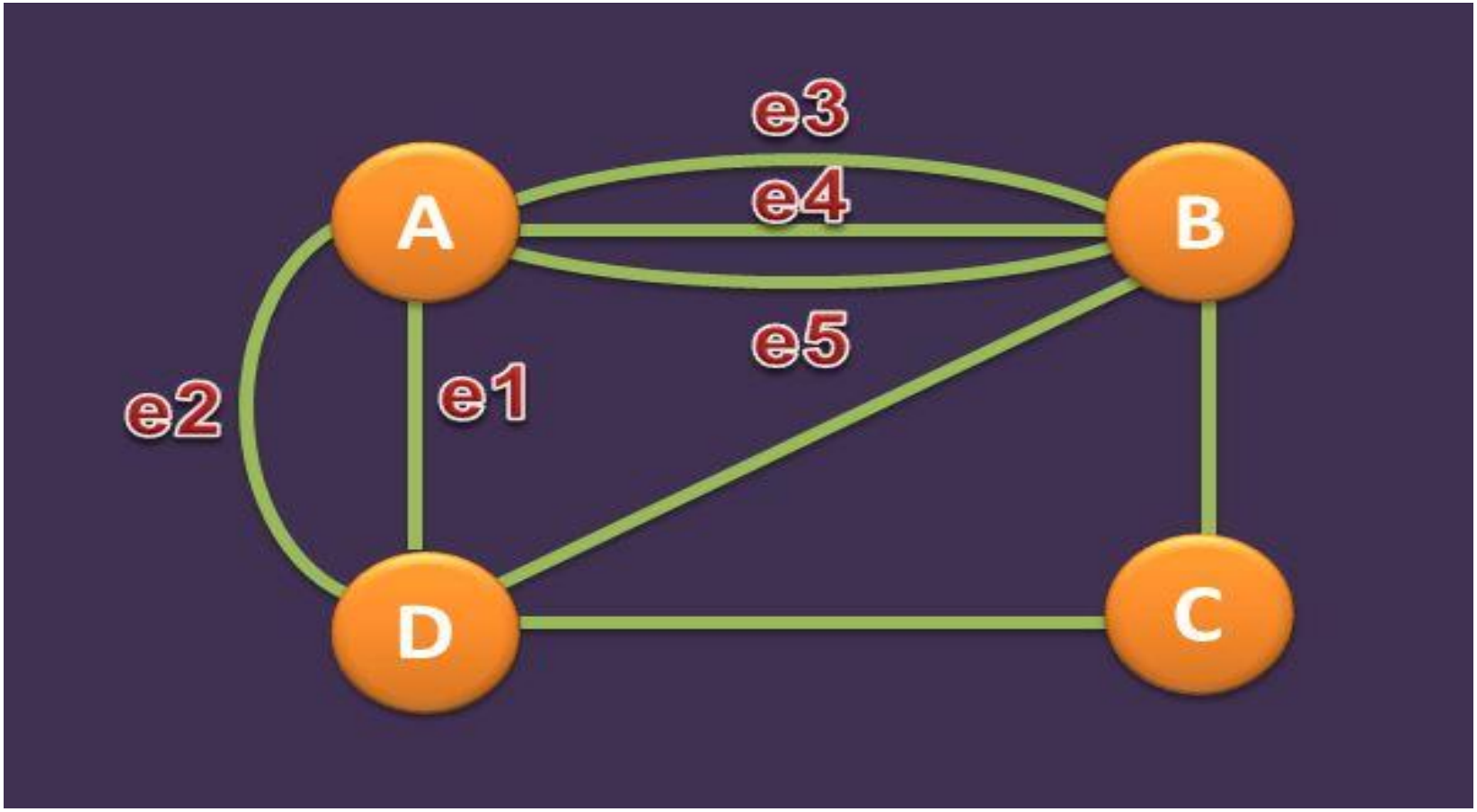
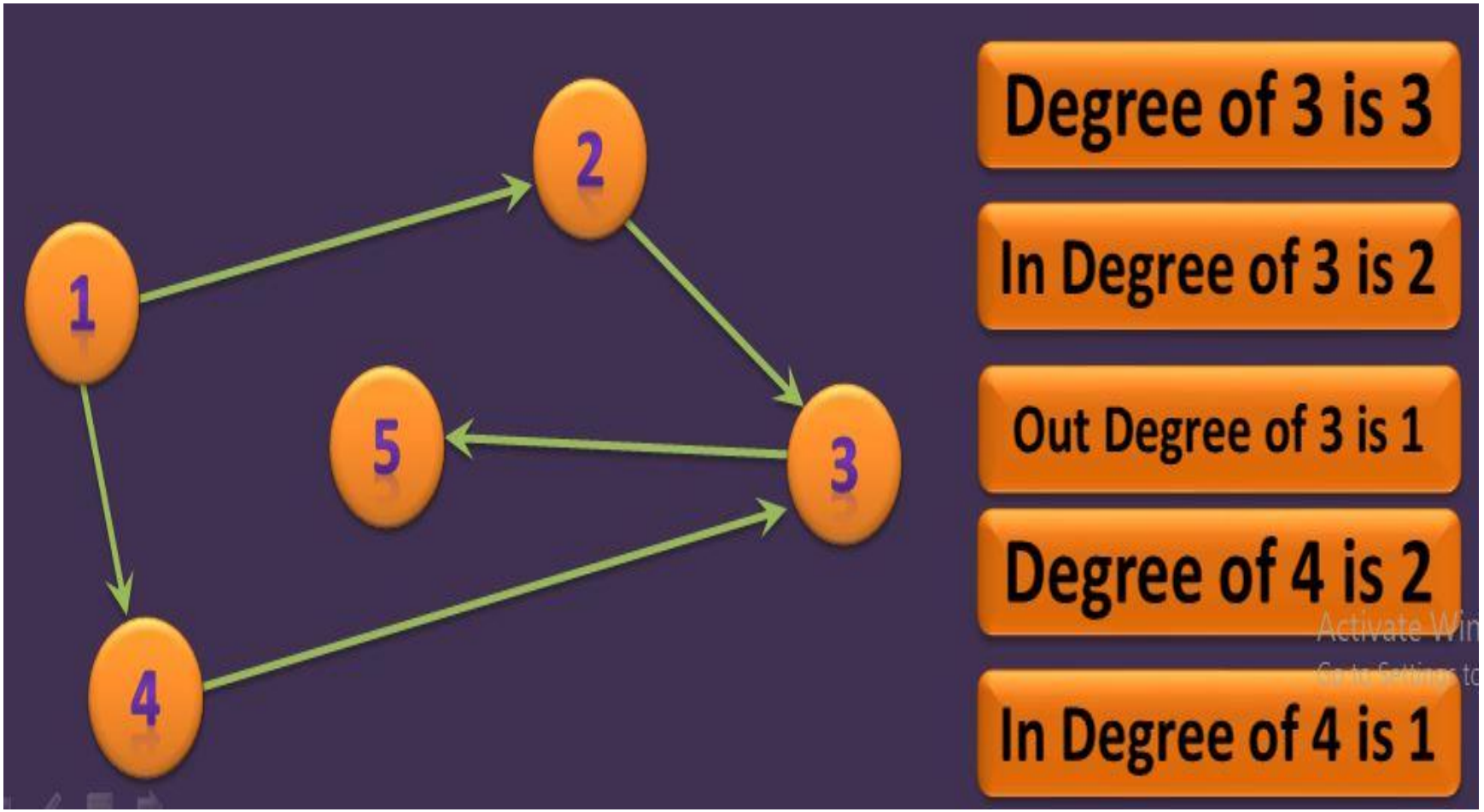
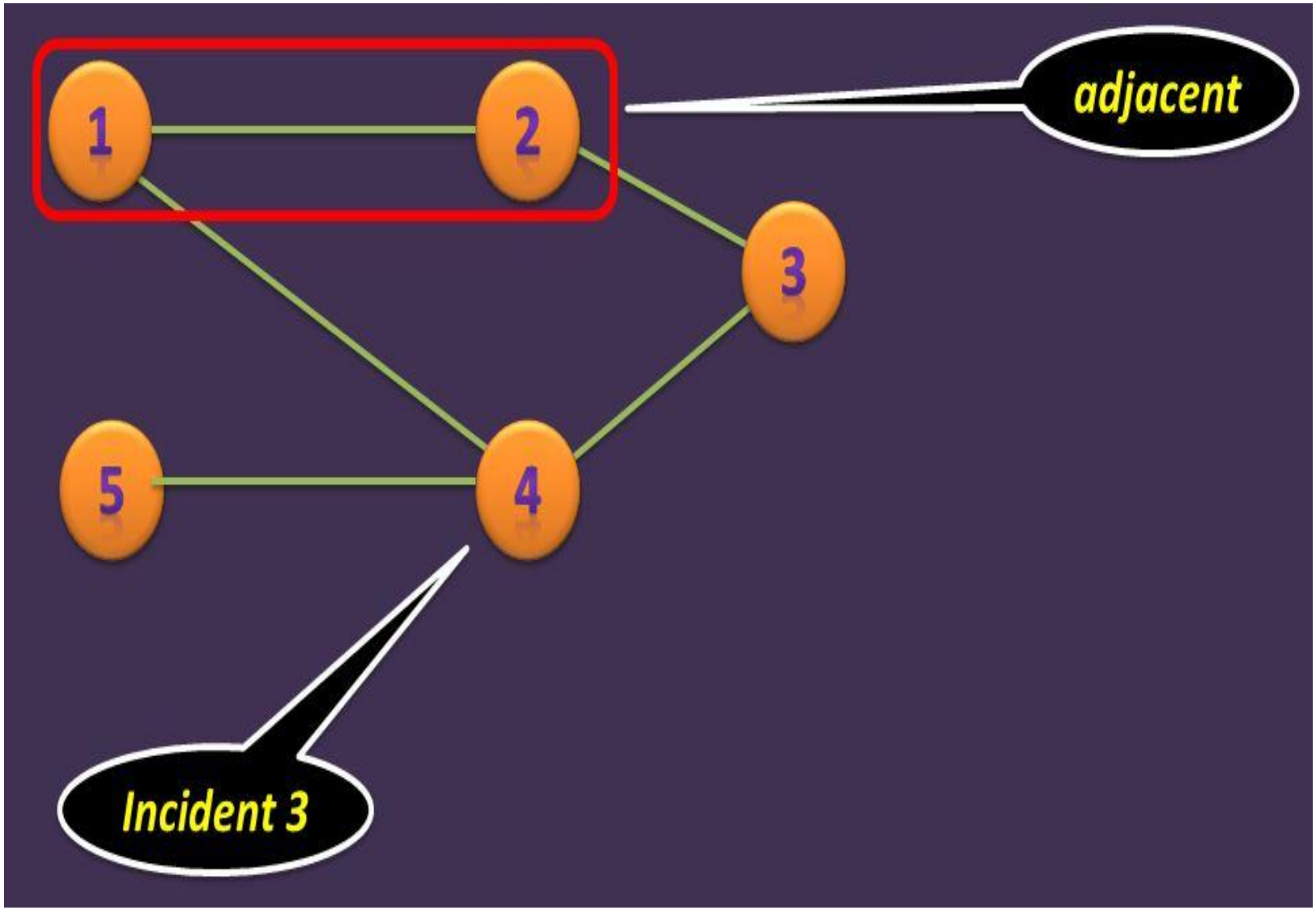
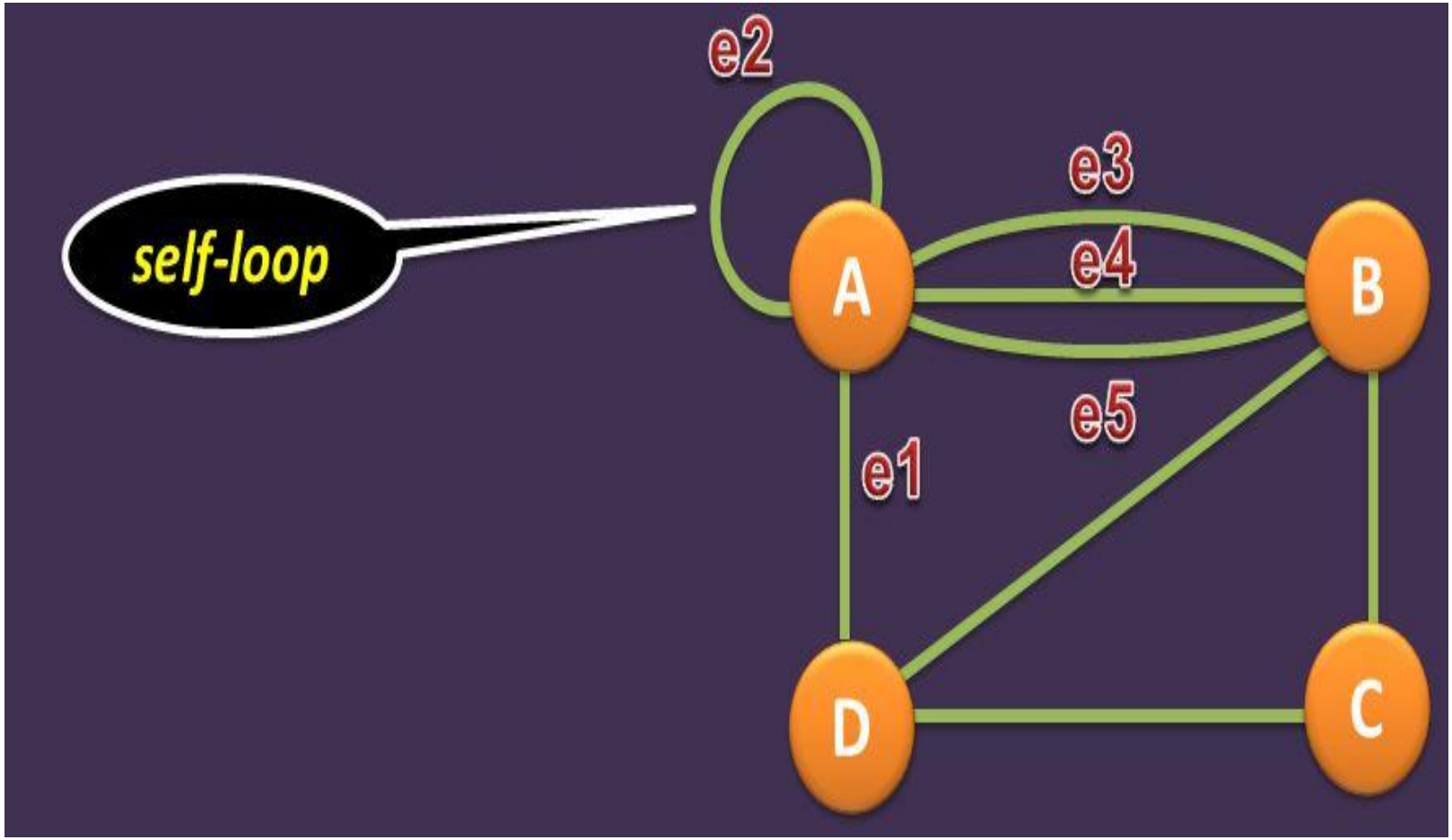


Fig: Multiple Edges



- In degree: Number of edges entering a node
- Out degree: Number of edges leaving a node
- Degree = Indegree + Outdegree


1
2
3
4
5
adjacent
Incident 3

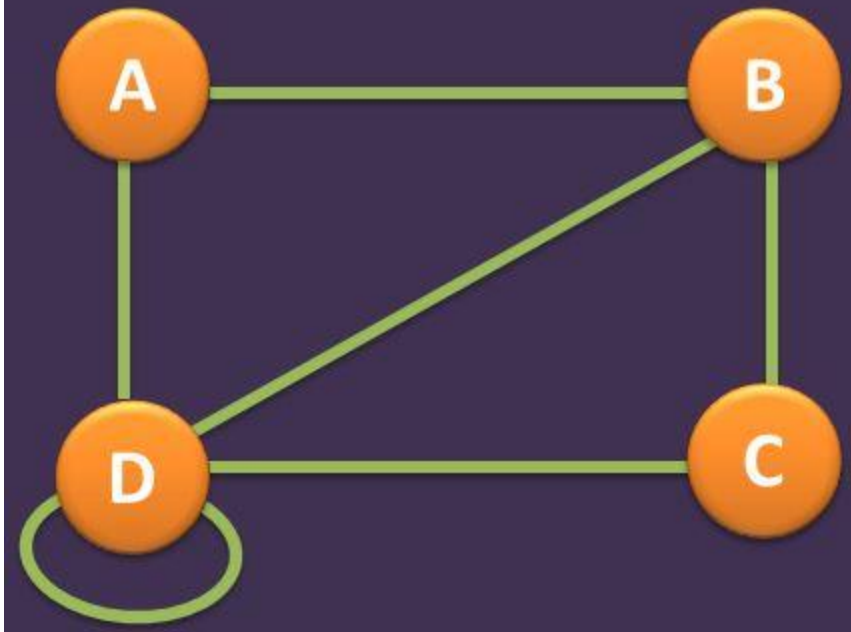

e2
self-loop
e3
e4
A
B
e5
e1
D
C

Graph Representation

1. Adjacency Matrix
2. Incidence Matrix
3. Adjacency List

1. Adjacency Matrix

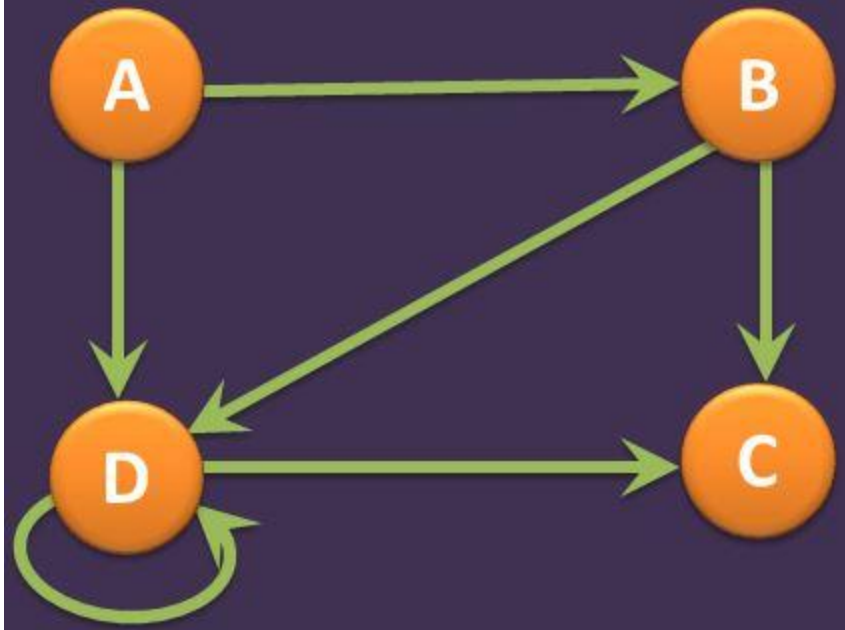
In Undirected Graph



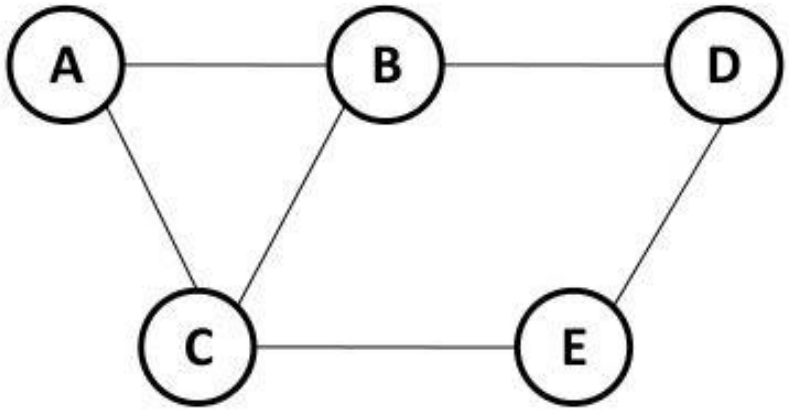
	A	B	C	D
A	0	1	0	1
B	1	0	1	1
C	0	1	0	1
D	1	1	1	1

1. Adjacency Matrix

In Directed Graph

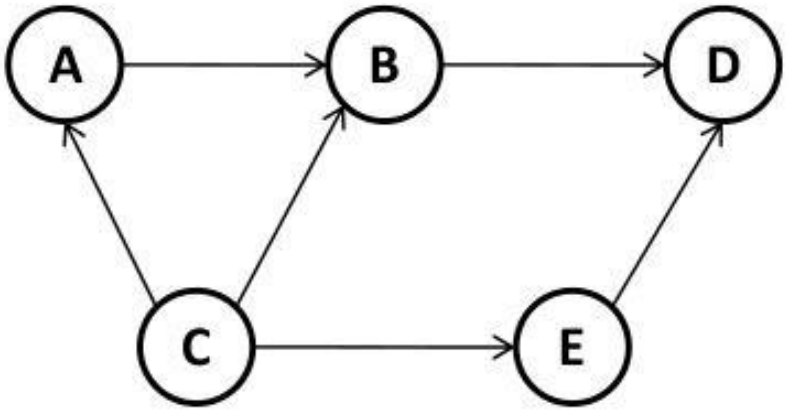


	A	B	C	D
A	0	1	0	1
B	0	0	1	1
C	0	0	0	0
D	0	0	1	1



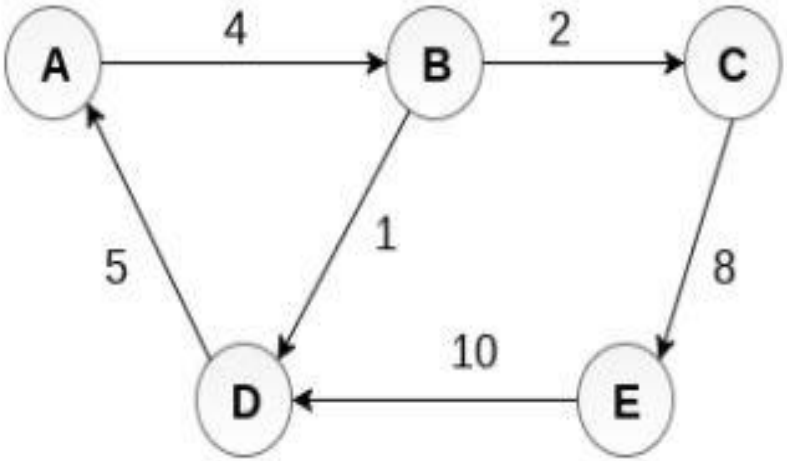
	A	B	C	D	E
A	0	1	1	0	0
B	1	0	1	1	0
C	1	1	0	0	1
D	0	1	0	0	1
E	0	0	1	1	0

Adjacency Matrix of Undirected Graph



	A	B	C	D	E
A	0	1	0	0	0
B	0	0	0	1	0
C	1	1	0	0	1
D	0	0	0	0	0
E	0	0	0	1	0

Adjacency Matrix of Directed Graph

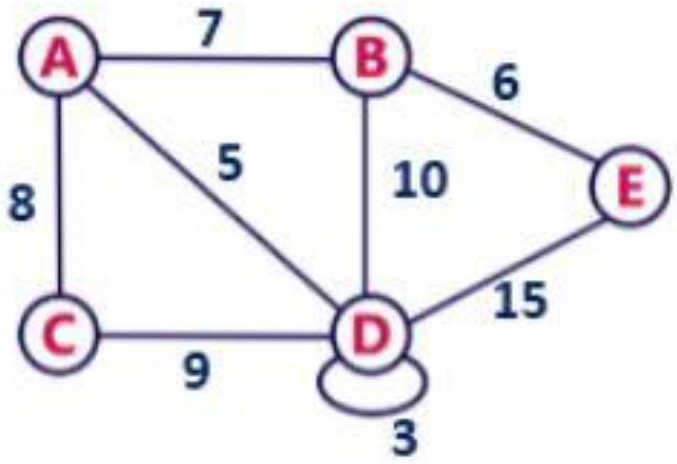


Weighted Directed Graph

	A	B	C	D	E
A	0	4	0	0	0
B	0	0	2	1	0
C	0	0	0	0	8
D	5	0	0	0	0
E	0	0	0	10	0

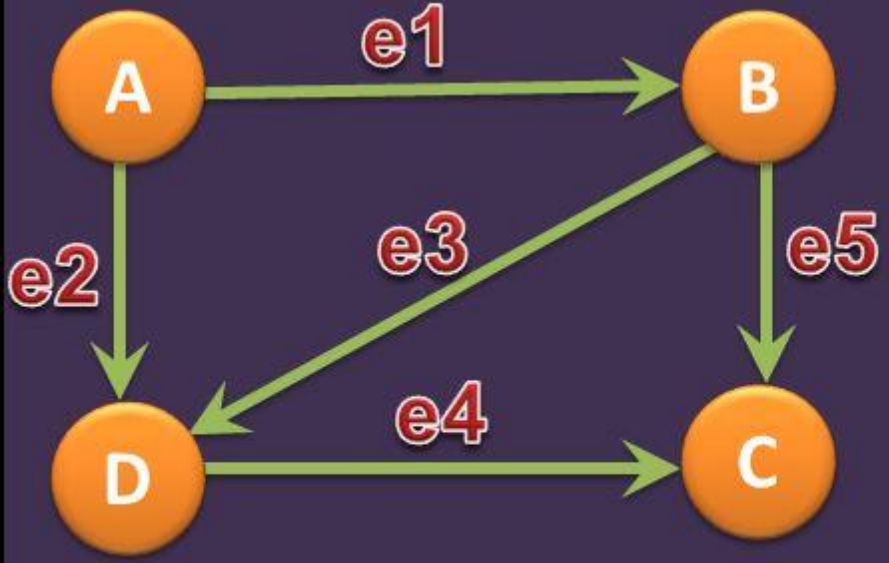
Adjacency Matrix

Undirected weighted graph represenation



	A	B	C	D	E
A	0	7	8	5	0
B	7	0	0	10	6
C	8	0	0	9	0
D	5	10	9	3	15
E	0	6	0	15	0

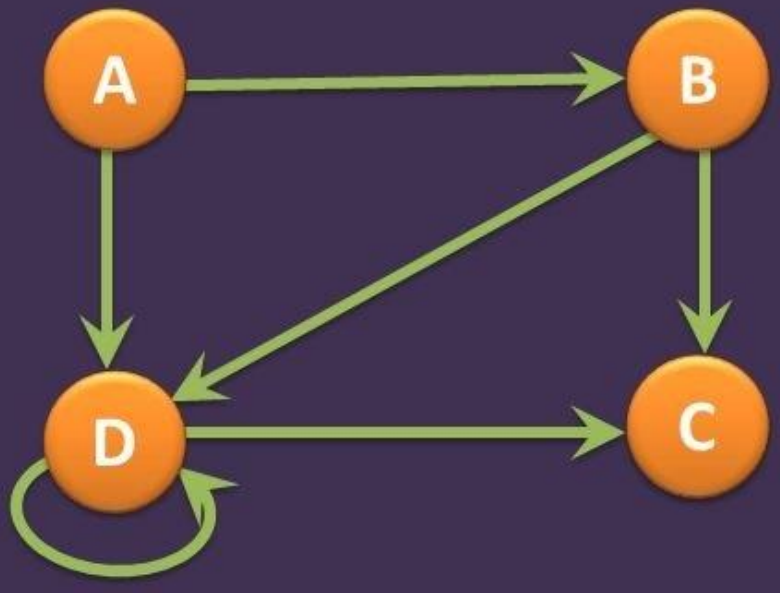
2. Incidence Matrix



	e1	e2	e3	e4	e5
A	1	1	0	0	0
B	-1	0	1	0	1
C	0	0	0	-1	-1
D	0	-1	-1	1	0

3. Adjacency List

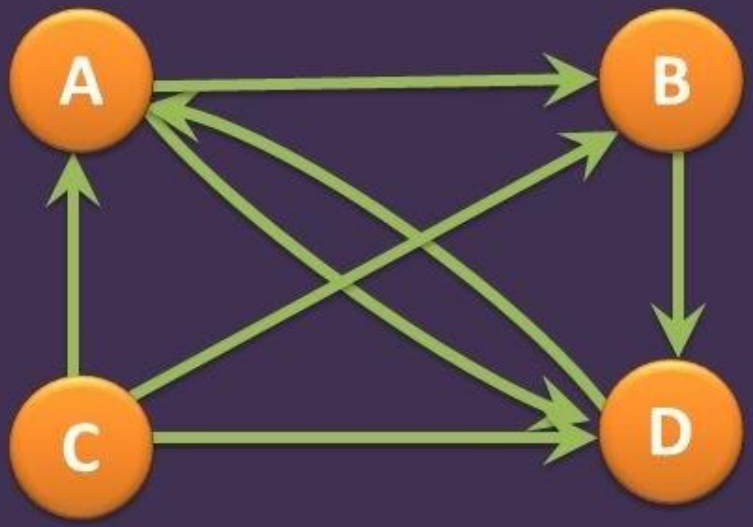
Array Representation



A	B	D
B	C	D
C		
D	C	D

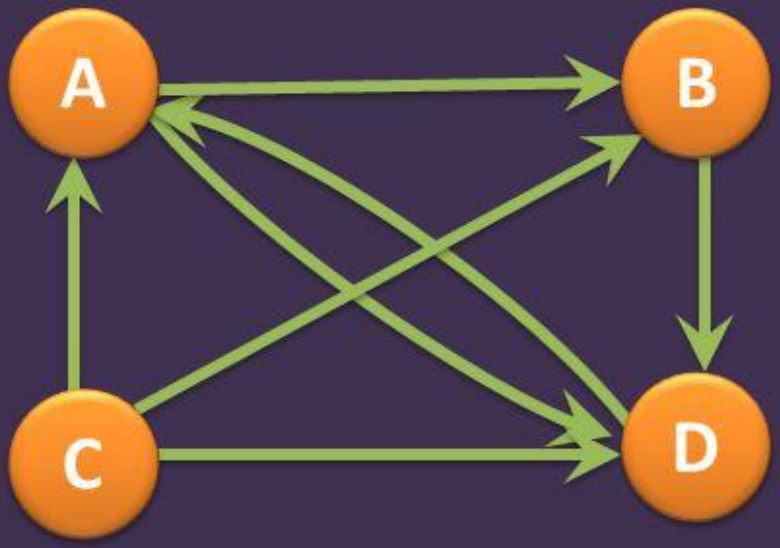
3. Adjacency List

Array Representation

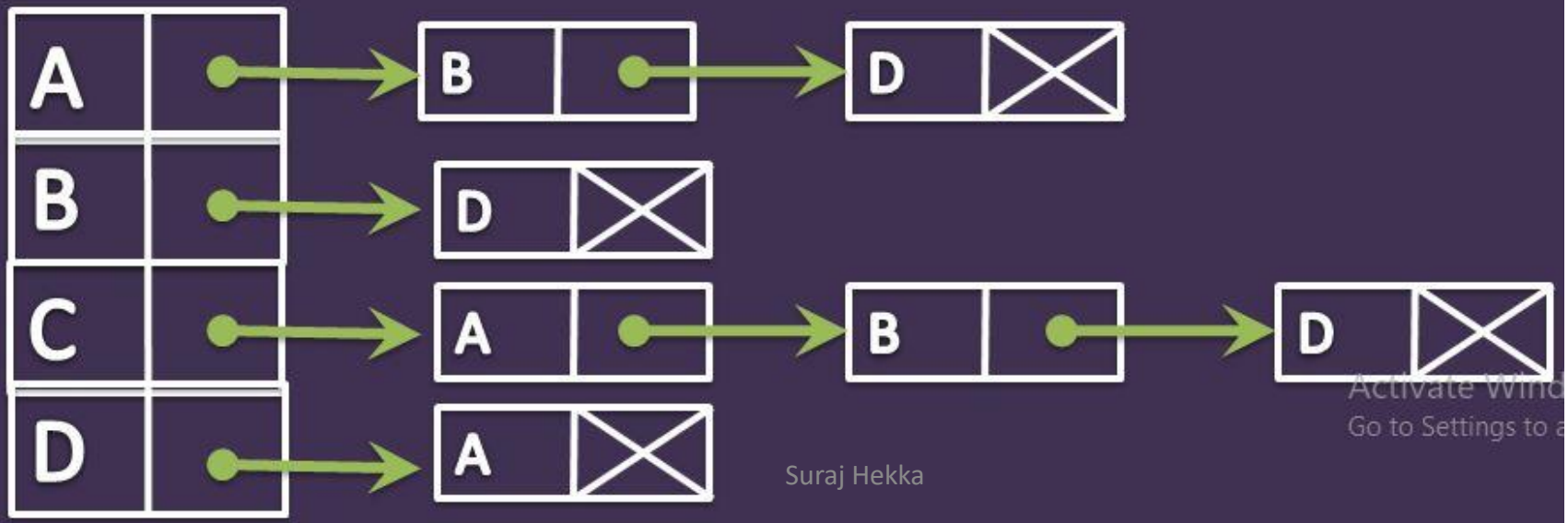


A	B D
B	D
C	A B D
D	A

3. Adjacency List

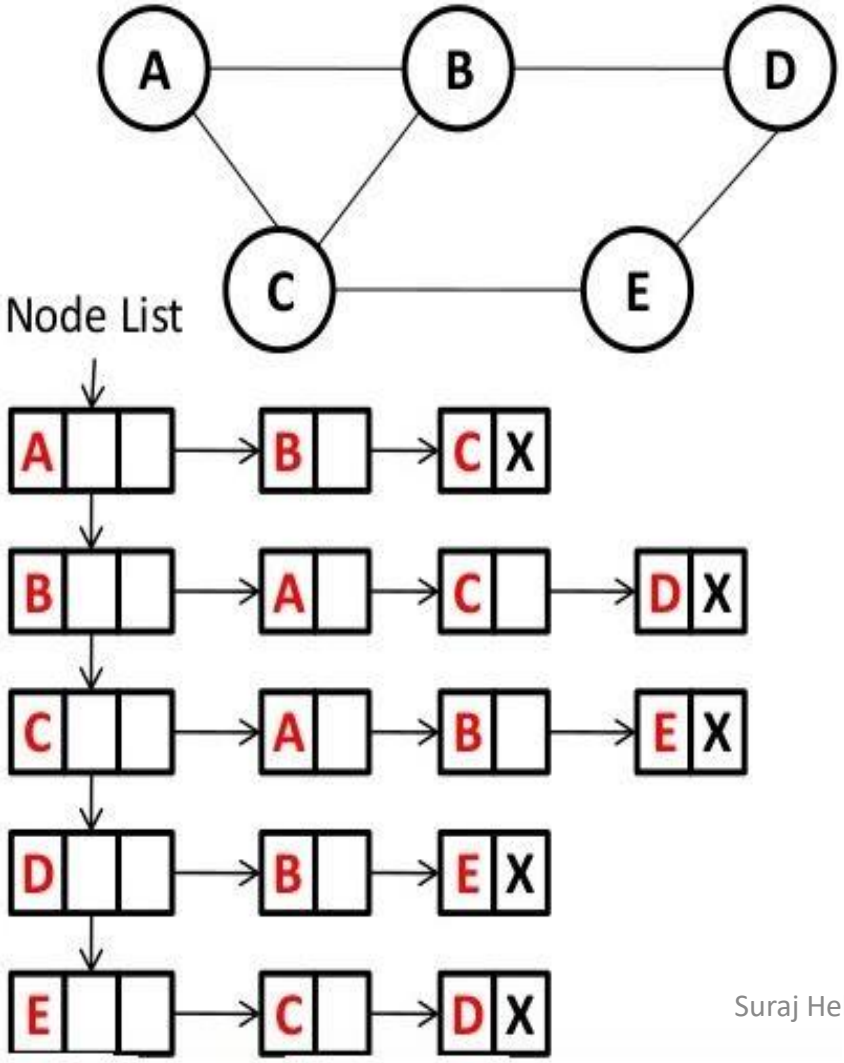


Linked list Representation

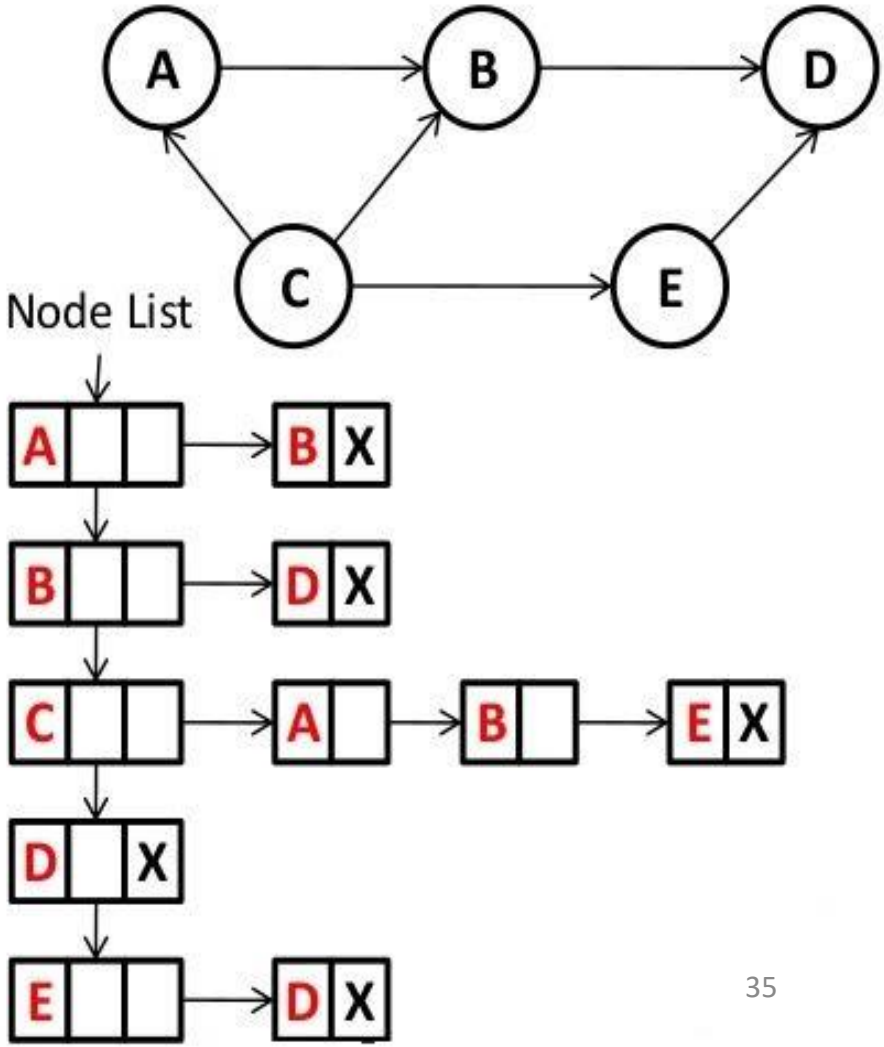


Linked Representation using adjacency List

3. Adjacency List of undirected graph

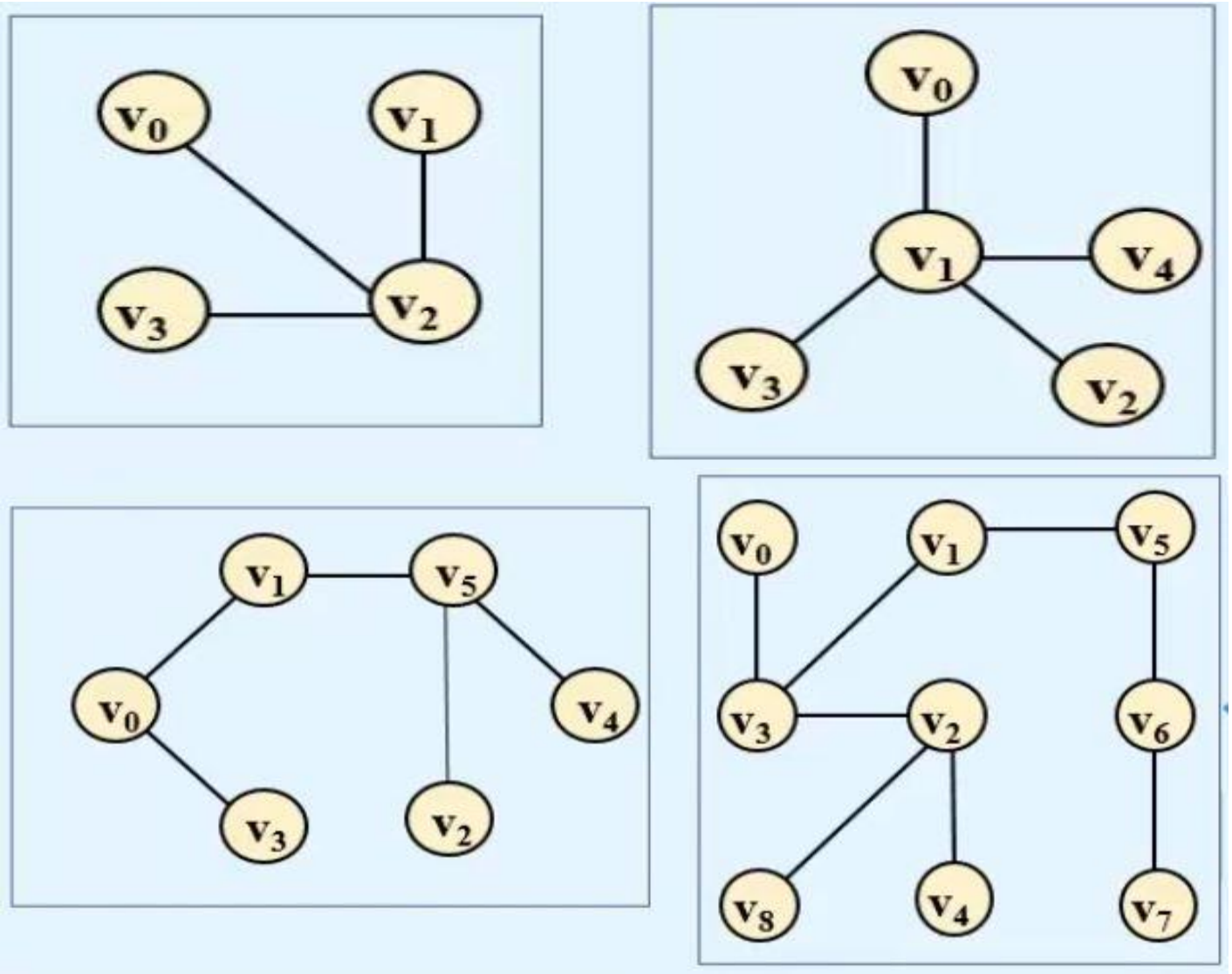


3. Adjacency List of directed graph

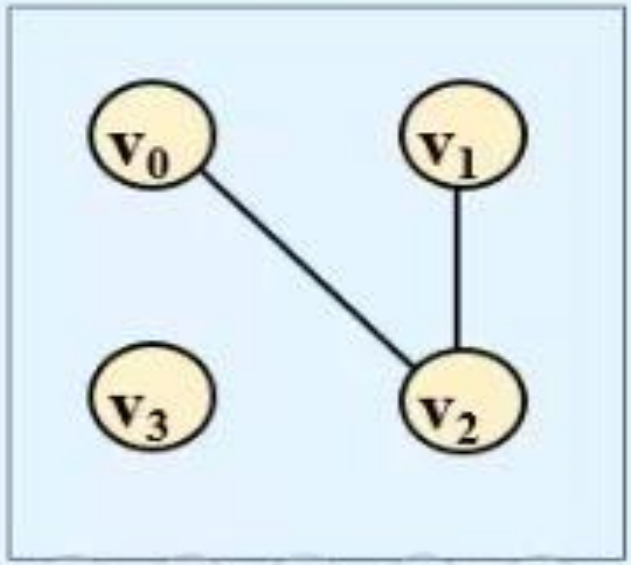


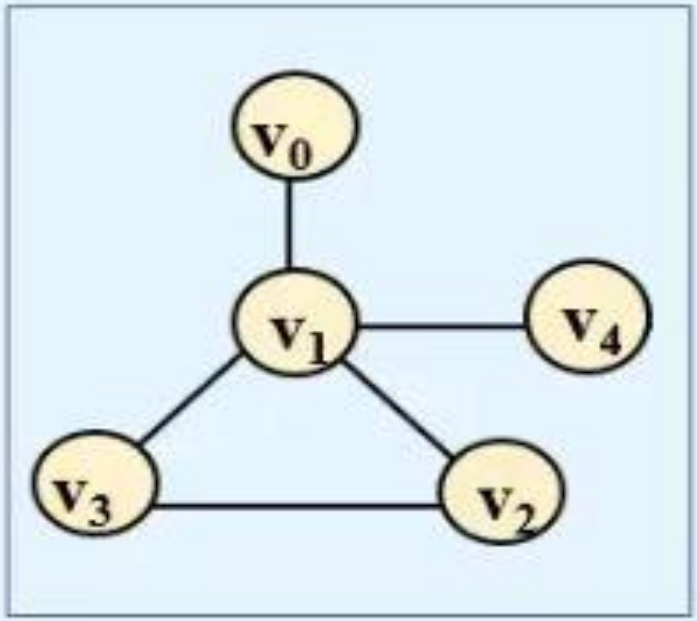
Tree:

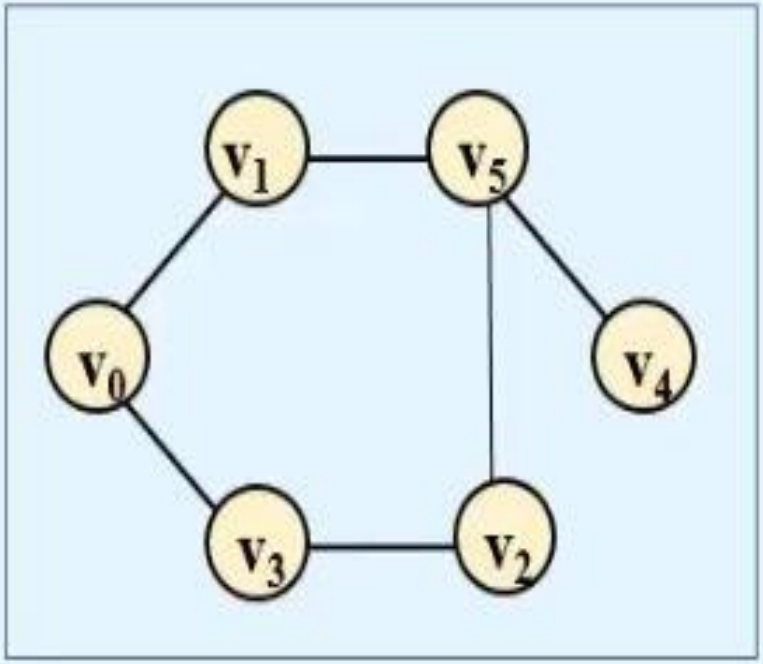
- A connected graph is a tree if there are no cycle in it.
- A tree with n vertices has exactly n-1 edges.

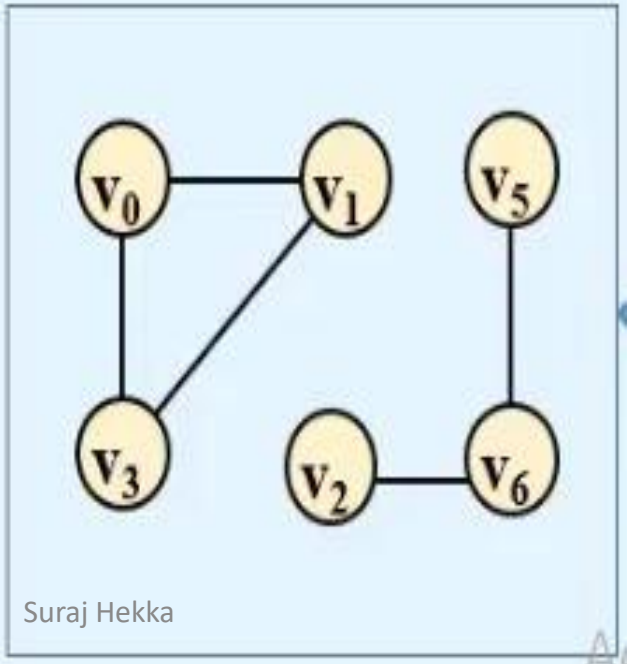


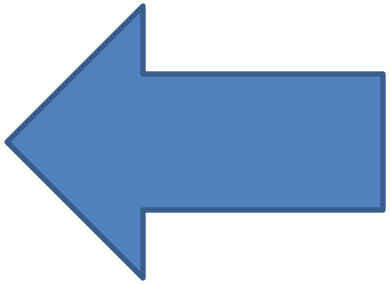
Trees







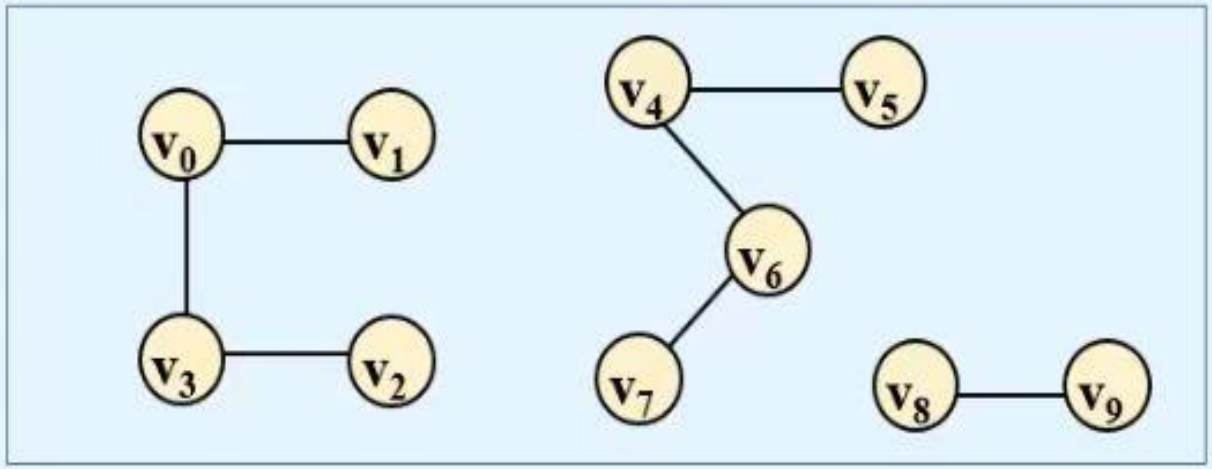




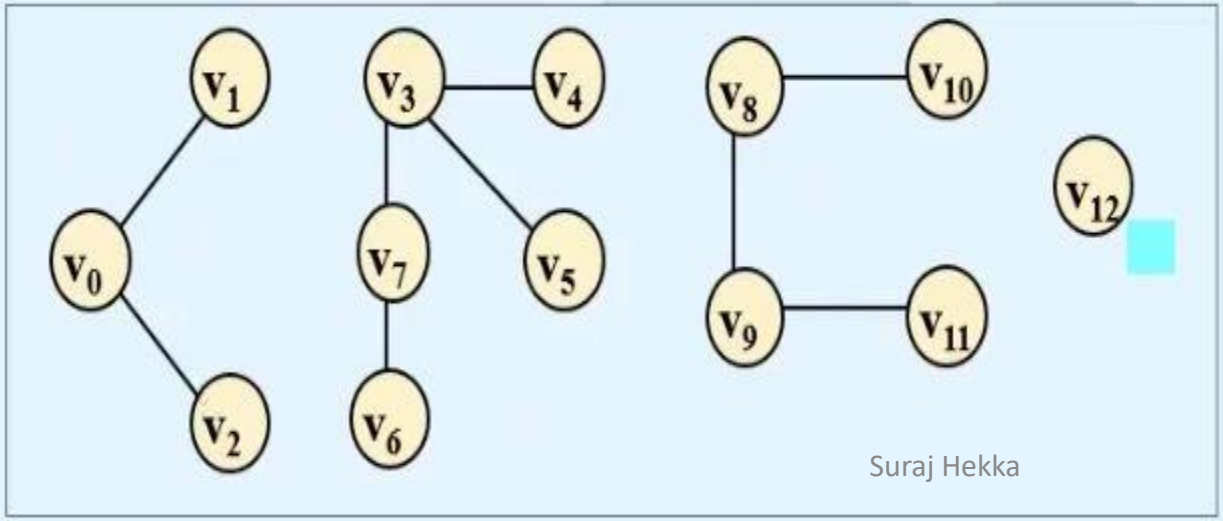
Not Trees

Forest:

- A forest is a **disjoint** union of trees.
- In a forest, there is at most one path between any two vertices.



In this forest there are 3 Trees.



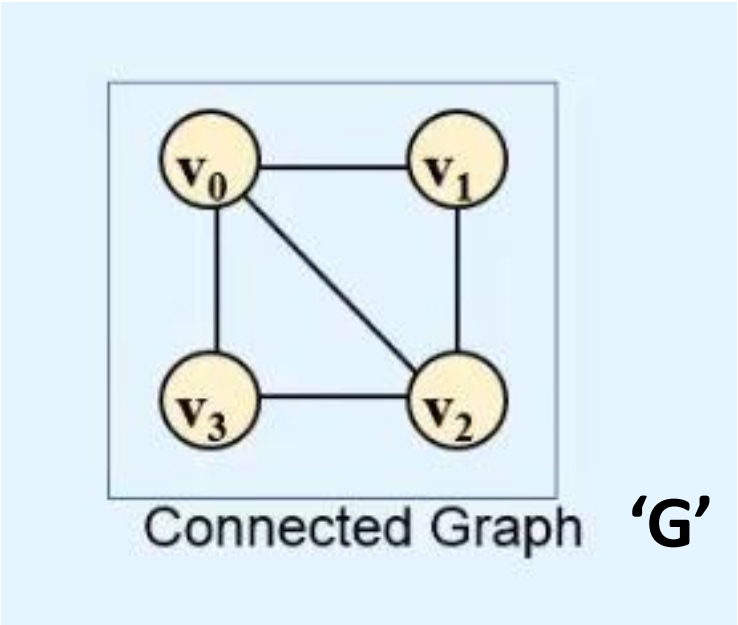
In this forest there are 4 Trees.

Spanning Tree:

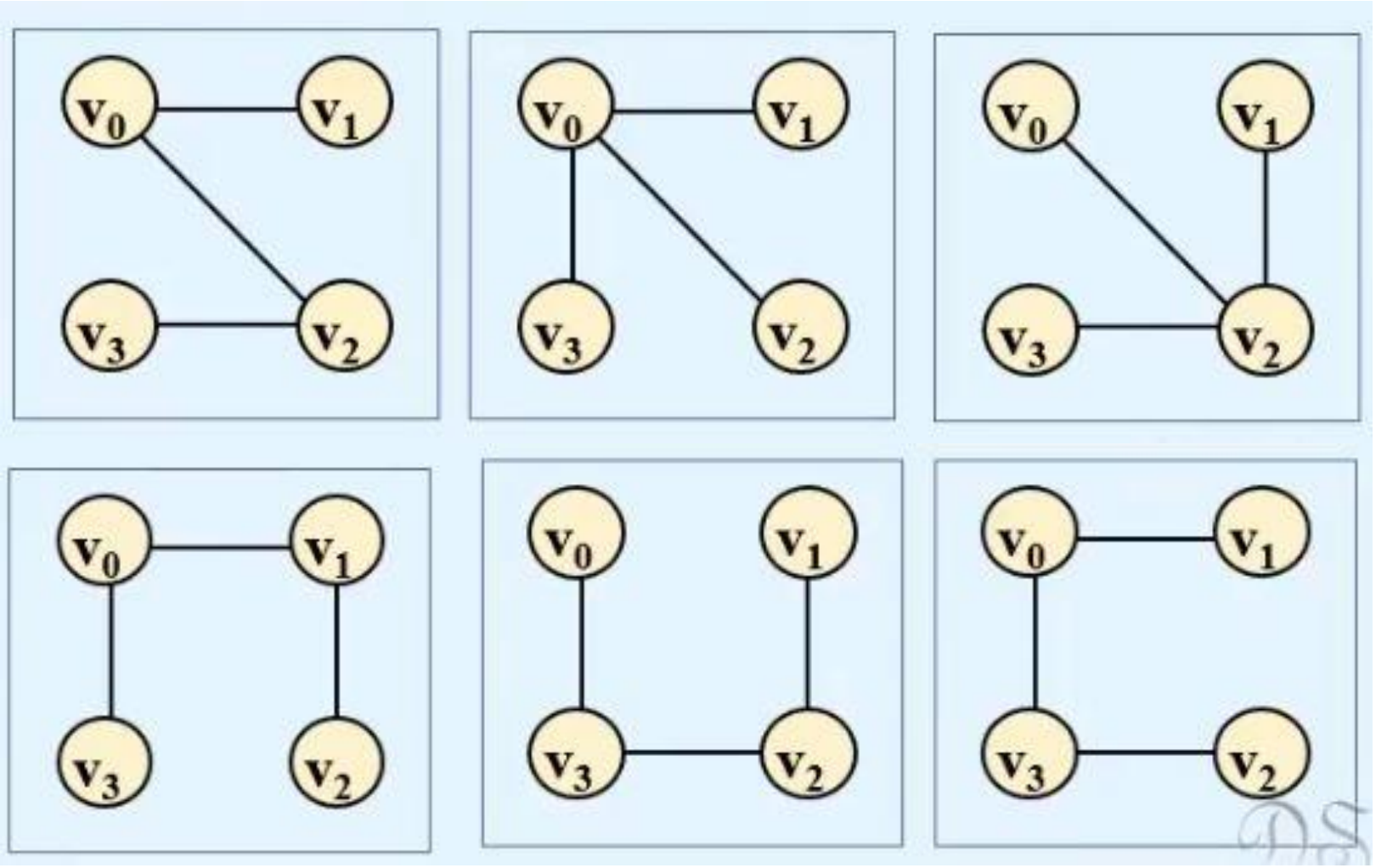
- Sub graph T of connected graph G is a spanning tree of G.

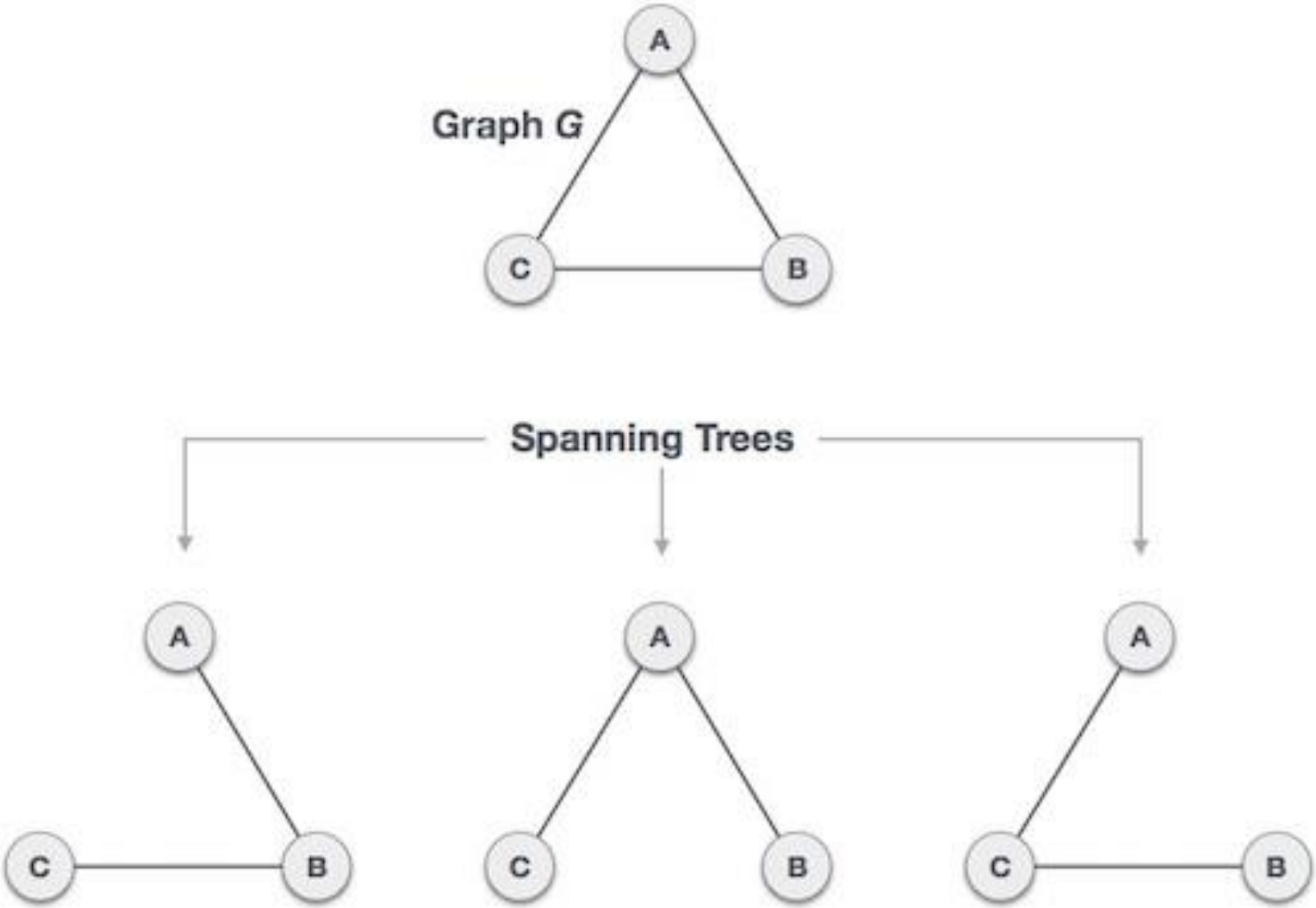
  Where,

  - T should contains all the vertices of G.
  - T should be a Tree.



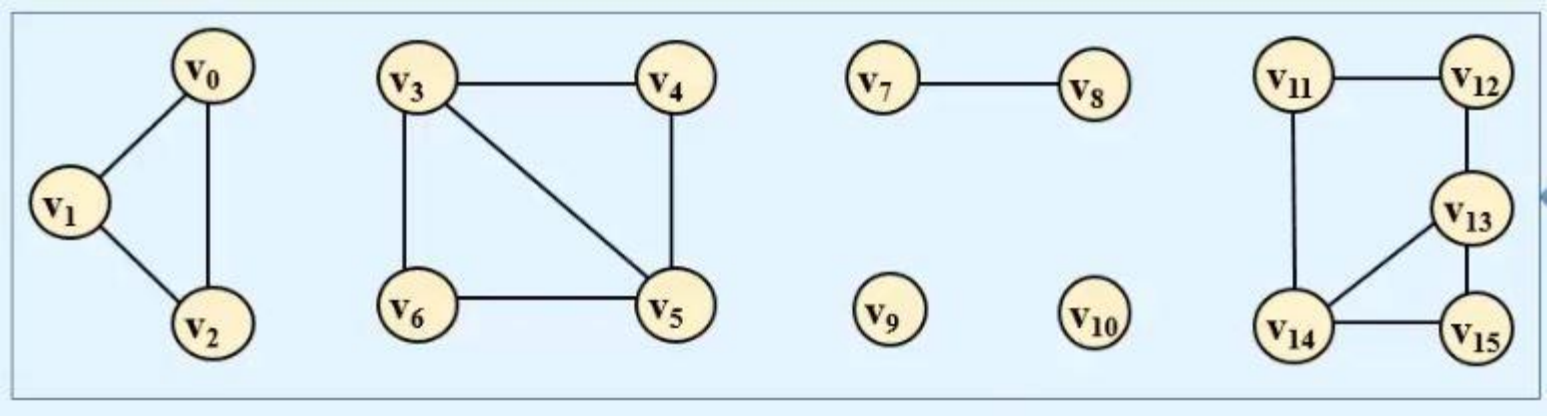
Many possible Spanning Trees of connected graph G are as follows:



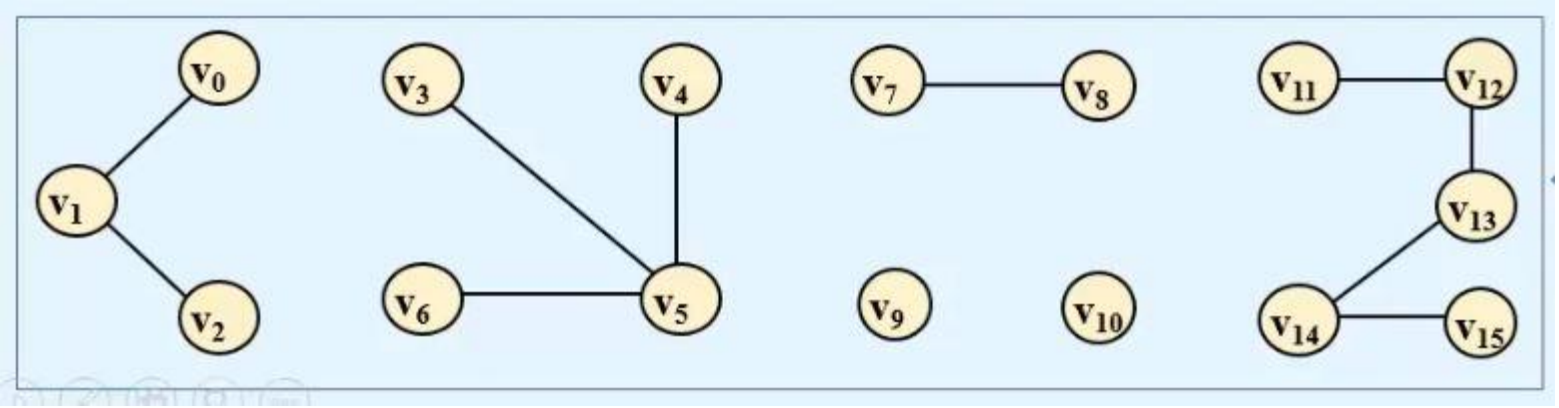

Graph G
A
C
B
Spanning Trees
A
C
B
A
C
B
A
C
B

Spanning Forest:

A spanning forest is a sub graph that consists of a spanning tree corresponding to each connected component.



Disconnected Graph G



A Spanning Forest of G

What is a Minimum Spanning Tree (MST)?

- A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.

or

- Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees.
- The cost of the spanning tree is the sum of the weights of all the edges in the tree.
- Minimum spanning trees are used for network designs (i.e. telephone or cable networks, water supply networks, and electrical grids). An example is a cable company wanting to lay line to multiple neighborhoods by minimizing the amount of cable laid, so that the cable company will save money.

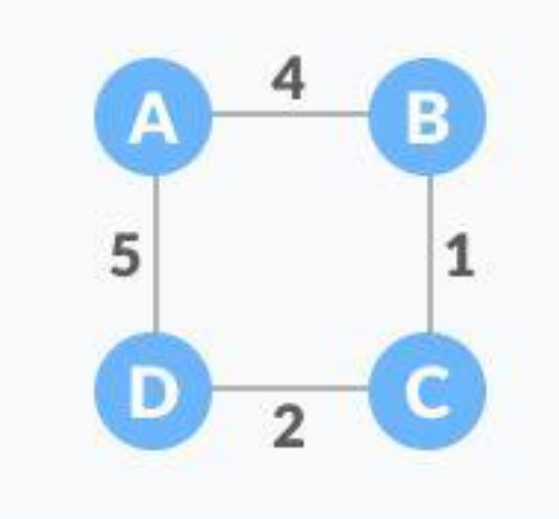
- Another useful application of MST would be finding airline routes. The vertices of the graph would represent cities, and the edges would represent routes between the cities. MST can be applied to optimize airline routes by finding the least costly paths with no cycles.

Properties of MST:

1. Should have (n-1) edges.
2. Weight of MST=sum of weight of edges of MST.
3. Maximum path length between 2 vertices is (n-1).
4. Only one path from one vertex to another vertex.
5. Removal of any edge disconnects the graph.

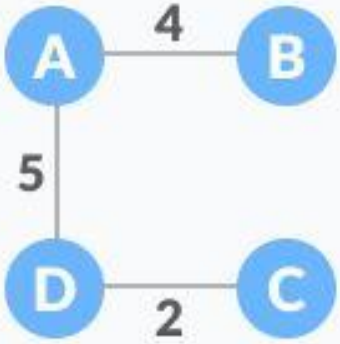
Example1 of Minimum Spanning Tree:

- Lets understand the minimum spanning tree with example.
- The initial graph is:



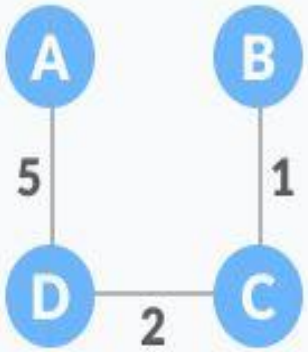
Weighted graph

- The possible spanning trees from the above graph are:



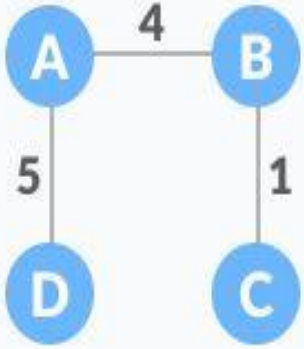
sum = 11

Minimum spanning tree - 1



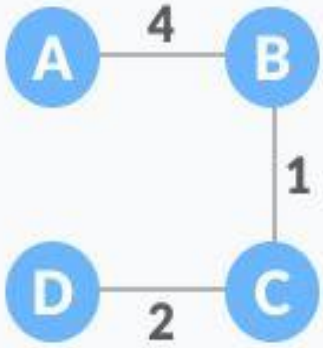
sum = 8

Minimum spanning tree - 2



sum = 10

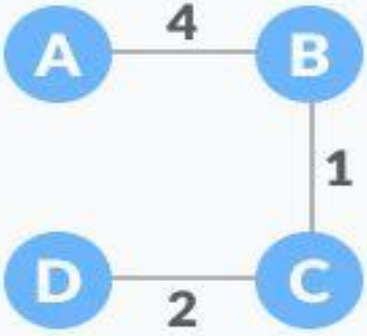
Minimum spanning tree - 3



sum = 7

Minimum spanning tree - 4

The minimum spanning tree from the above spanning trees is:

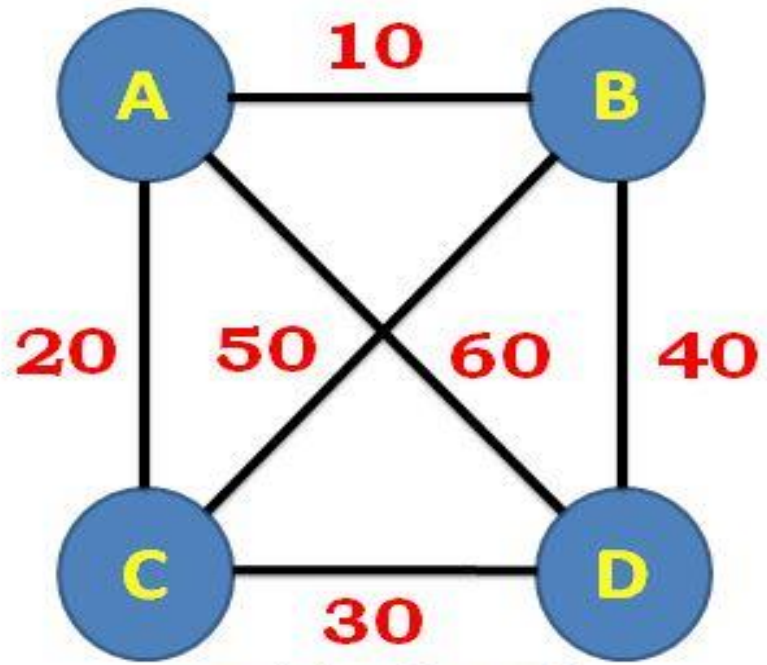


sum = 7

Minimum spanning tree

Example2 of Minimum Spanning Tree:

- Lets understand the minimum spanning tree with example.
- The initial graph is:



Weighted Graph

- The total number of spanning trees that can make from above graph is:

n^{n-2} where 'n' is number of vertices in the graph.

n^{n-2}

$=4^{4-2}$

$=4^2$

$=16$

- The MST for above respective graph is:

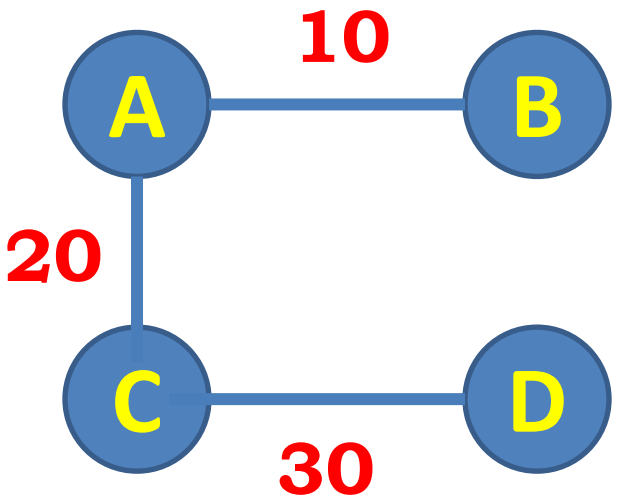


Fig: MST

- Minimum spanning tree, can be constructed using any of the following two algorithms:
  1. **Kruskal's algorithm**
  2. **Prim algorithm**

- Both algorithms differ in their methodology, but both eventually end up with the MST. Kruskal's algorithm uses edges, and Prim's algorithm uses vertex connections in determining the MST.

Note:

- https://www.tutorialspoint.com/data_structures_algorithms/prims_spanning_tree_algorithm.htm
- https://www.javatpoint.com/spanning-tree

1. Kruskal's Algorithm:

- Kruskal's Algorithm is a famous greedy algorithm.
- It is used for finding the Minimum Spanning Tree (MST) of a given graph.
- To apply Kruskal's algorithm, the given graph must be weighted, connected and undirected.
- To understand Kruskal's algorithm let us consider the following examples.

https://www.gatevidyalay.com/kruskals-algorithm-kruskals-algorithm-example/

Kruskal's Algorithm Implementation:

The implementation of Kruskal's Algorithm is explained in the following steps:

Step-01:

- Sort all the edges from low weight to high weight.

Step-02:

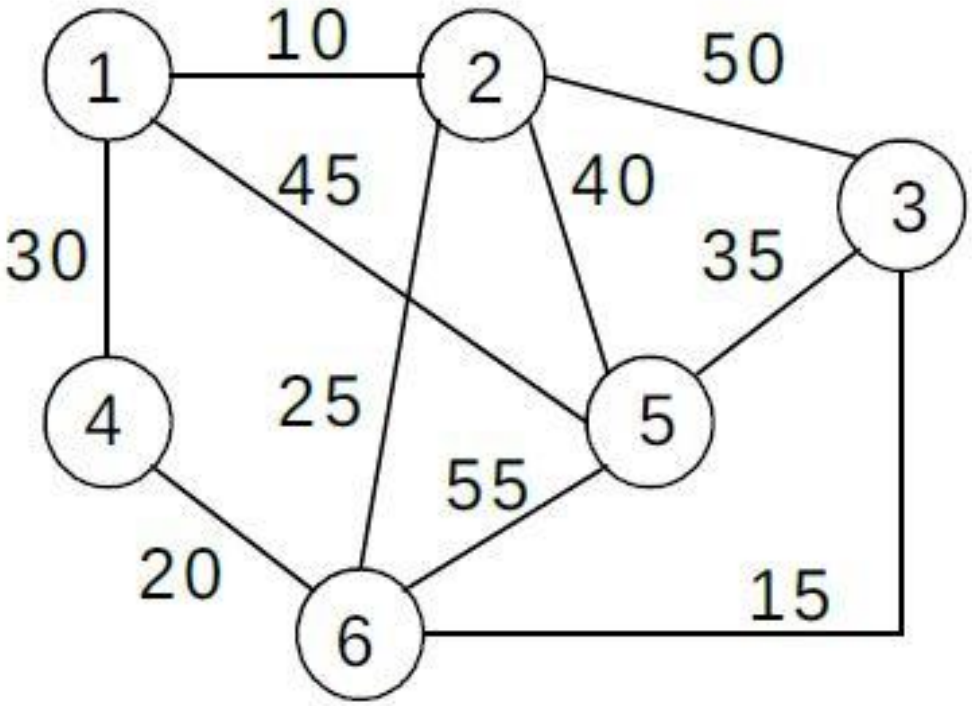
- Take the edge with the lowest weight and use it to connect the vertices of graph.
- If adding an edge creates a cycle, then reject that edge and go for the next least weight edge.

Step-03:

- Keep adding edges until all the vertices are connected and a Minimum Spanning Tree (MST) is obtained

Example 1:

Construct the minimal spanning tree for the graph shown below using Kruskal's Algorithm.

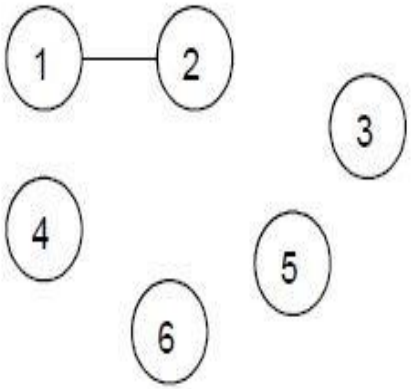


Solution:

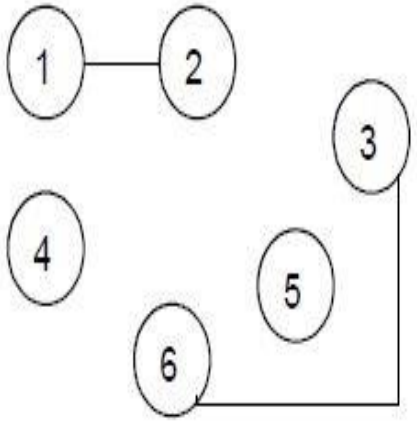
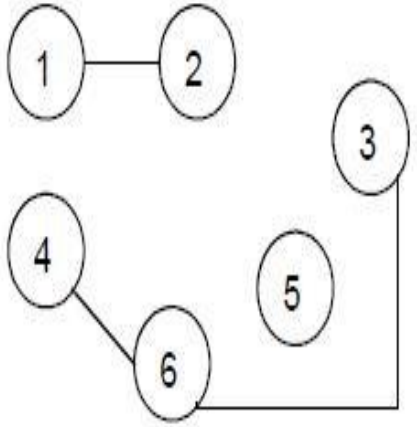
Arrange all the edges in increasing order of their cost:

Cost	10	15	20	25	30	35	40	45	50	55
Edge	(1, 2)	(3, 6)	(4, 6)	(2, 6)	(1, 4)	(3, 5)	(2, 5)	(1, 5)	(2, 3)	(5, 6)

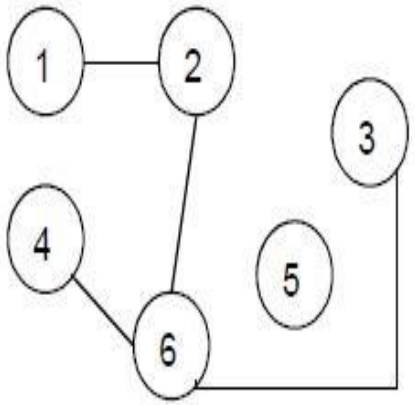
The stages in Kruskal's algorithm for minimal spanning tree is as follows:

Edge	Cost	Stages in Kruskal's algorithm	Remarks
(1, 2)	10		The edge between vertices 1 and 2 is the first edge selected. It is included in the spanning tree.

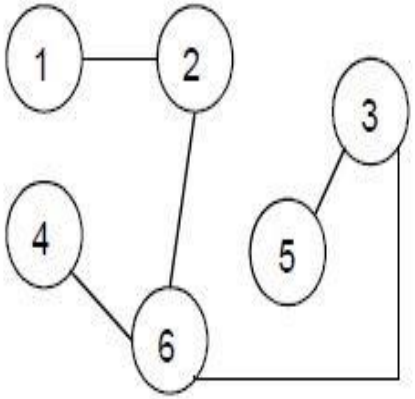
Cost	10	15	20	25	30	35	40	45	50	55
Edge	(1, 2)	(3, 6)	(4, 6)	(2, 6)	(1, 4)	(3, 5)	(2, 5)	(1, 5)	(2, 3)	(5, 6)

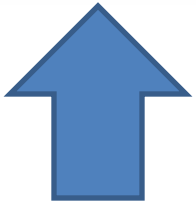
(3, 6)	15		Next, the edge between vertices 3 and 6 is selected and included in the tree.
(4, 6)	20		The edge between vertices 4 and 6 is next included in the tree.

Cost	10	15	20	25	30	35	40	45	50	55
Edge	(1, 2)	(3, 6)	(4, 6)	(2, 6)	(1, 4)	(3, 5)	(2, 5)	(1, 5)	(2, 3)	(5, 6)

(2, 6)	25		The edge between vertices 2 and 6 is considered next and included in the tree.
(1, 4)	30	Reject	The edge between the vertices 1 and 4 is discarded as its inclusion creates a cycle.

Cost	10	15	20	25	30	35	40	45	50	55
Edge	(1, 2)	(3, 6)	(4, 6)	(2, 6)	(1, 4)	(3, 5)	(2, 5)	(1, 5)	(2, 3)	(5, 6)

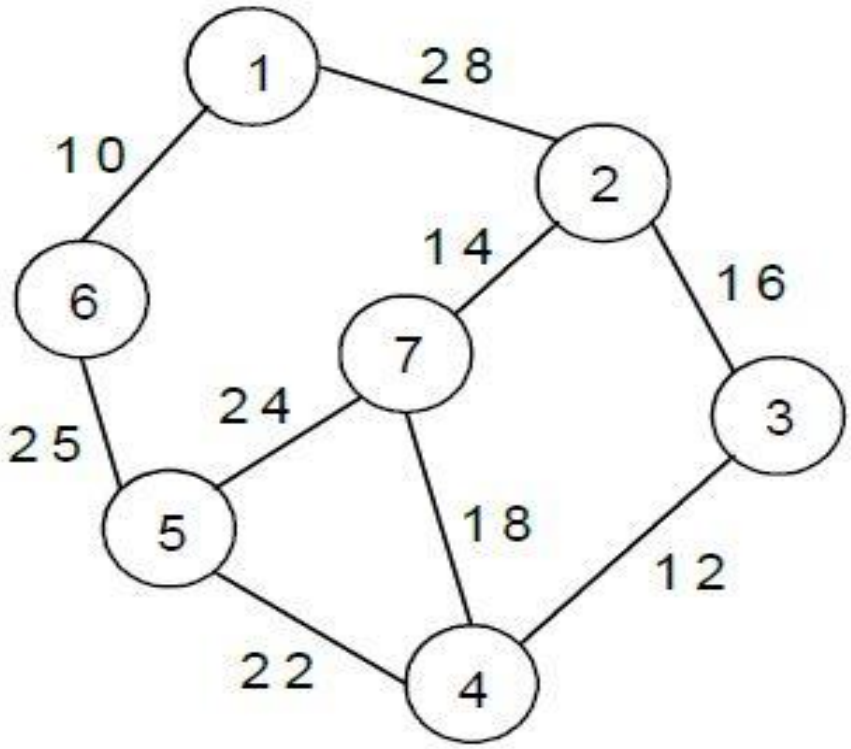
(3, 5)	35		Finally, the edge between vertices 3 and 5 is considered and included in the tree built. This completes the tree. The cost of the minimal spanning tree is **105** .



Minimum Spanning Tree

Example 2:

Construct the minimal spanning tree for the graph shown below using Kruskal's Algorithm.

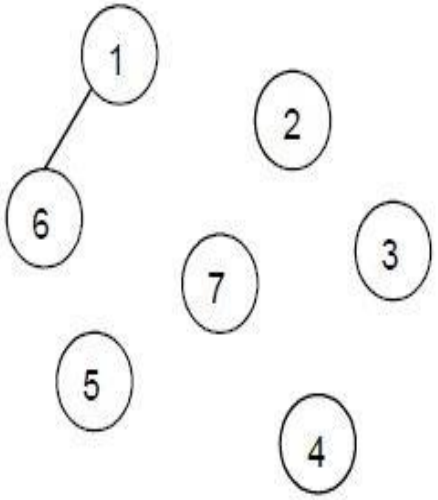


Solution:

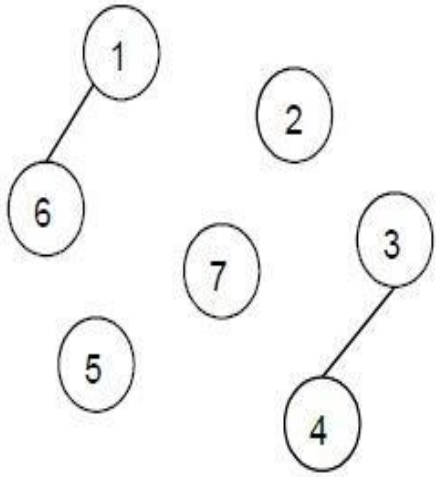
Arrange all the edges in increasing order of their cost:

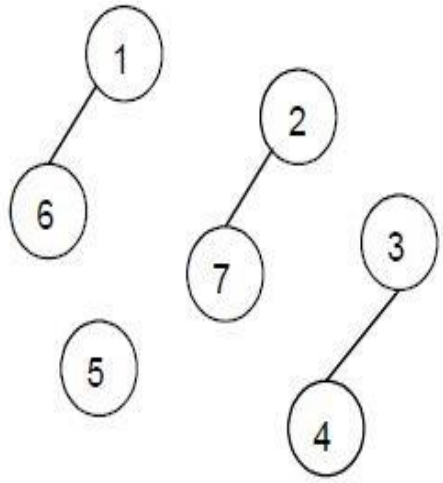
Cost	10	12	14	16	18	22	24	25	28
Edge	(1, 6)	(3, 4)	(2, 7)	(2, 3)	(4, 7)	(4, 5)	(5, 7)	(5, 6)	(1, 2)

The stages in Kruskal's algorithm for minimal spanning tree is as follows:

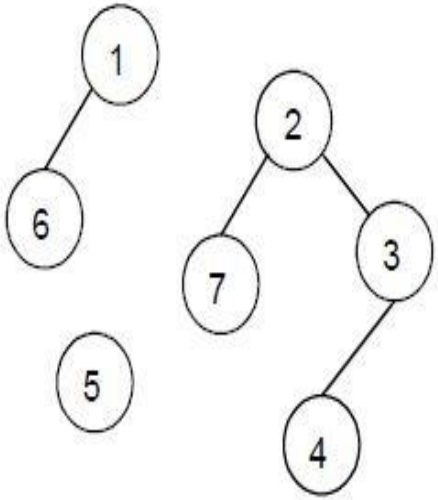
Edge	Cost	Stages in Kruskal's algorithm	Remarks
(1, 6)	10		The edge between vertices 1 and 6 is the first edge selected. It is included in the spanning tree.

Cost	10	12	14	16	18	22	24	25	28
Edge	(1, 6)	(3, 4)	(2, 7)	(2, 3)	(4, 7)	(4, 5)	(5, 7)	(5, 6)	(1, 2)

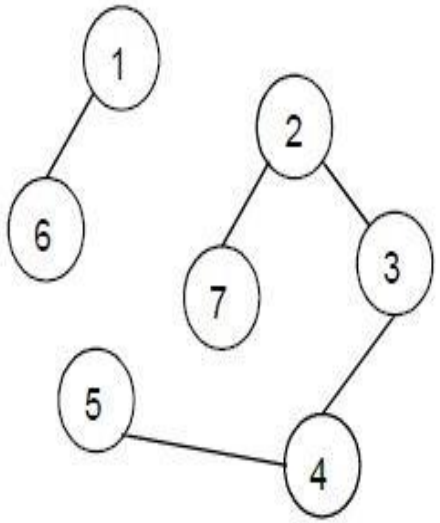
(3, 4)	12		Next, the edge between vertices 3 and 4 is selected and included in the tree.

(2, 7)	14		The edge between vertices 2 and 7 is next included in the tree.

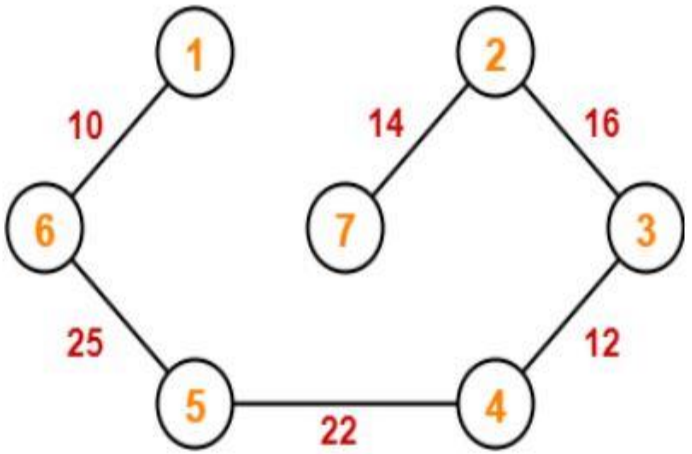
Cost	10	12	14	16	18	22	24	25	28
Edge	(1, 6)	(3, 4)	(2, 7)	(2, 3)	(4, 7)	(4, 5)	(5, 7)	(5, 6)	(1, 2)

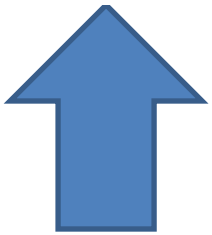
(2, 3)	16		The edge between vertices 2 and 3 is next included in the tree.
(4, 7)	18	Reject	The edge between the vertices 4 and 7 is discarded as its inclusion creates a cycle.

Cost	10	12	14	16	18	22	24	25	28
Edge	(1, 6)	(3, 4)	(2, 7)	(2, 3)	(4, 7)	(4, 5)	(5, 7)	(5, 6)	(1, 2)

(4, 5)	22		The edge between vertices 4 and 7 is considered next and included in the tree.
(5, 7)	24	Reject	The edge between the vertices 5 and 7 is discarded as its inclusion creates a cycle.

Cost	10	12	14	16	18	22	24	25	28
Edge	(1, 6)	(3, 4)	(2, 7)	(2, 3)	(4, 7)	(4, 5)	(5, 7)	(5, 6)	(1, 2)

(5, 6)	25		Finally, the edge between vertices 5 and 6 is considered and included in the tree built. This completes the tree. The cost of the minimal spanning tree is 99.



Minimum Spanning Tree

Weight of the MST

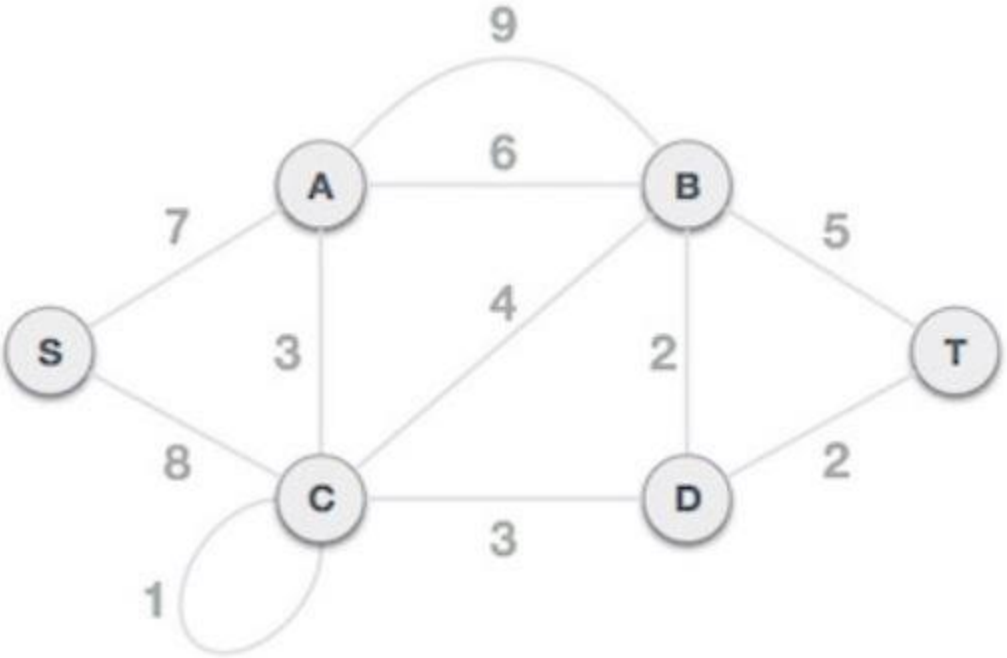
= Sum of all edge weights

= 10 + 25 + 22 + 12 + 16 + 14

= 99 units

Example 3: DO THIS QUESTION BY YOURSELF

Construct the minimal spanning tree for the graph shown below using Kruskal's Algorithm. (show all necessary steps).



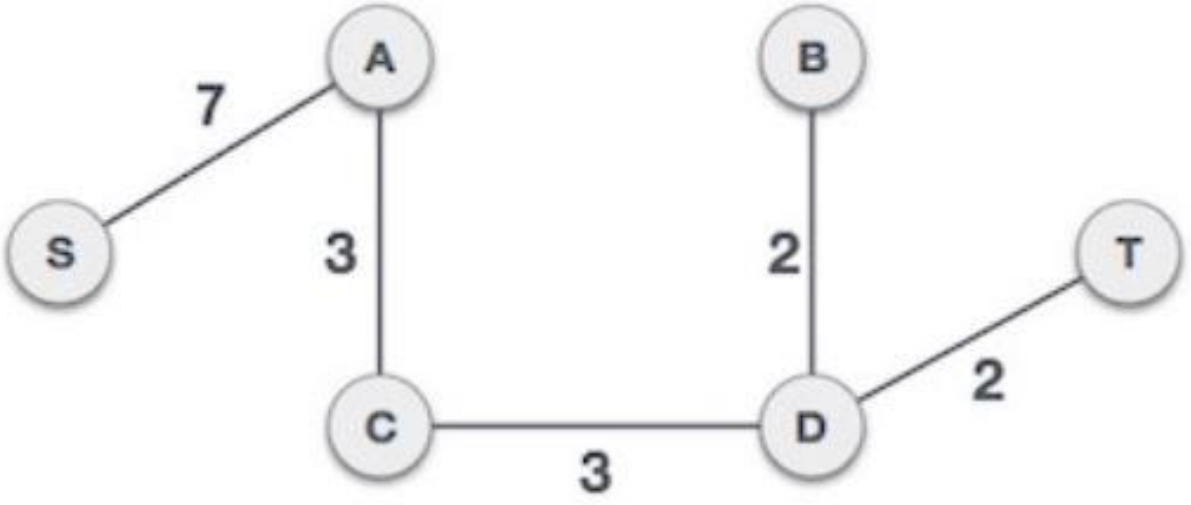
Hint:

- Remove all loops and parallel edges from the given graph.
- In case of parallel edges, keep the one which has the least cost associated and remove all others.

- Create a set of edges and weight, and arrange them in an ascending order of weightage (cost) like this:

B, D	D, T	A, C	C, D	C, B	B, T	A, B	S, A	S, C
2	2	3	3	4	5	6	7	8

- Your final minimum spanning tree should look like this:



2. Prim algorithm:

- Prim's Algorithm is a famous greedy algorithm.

- It is used for finding the Minimum Spanning Tree (MST) of a given graph.

- To apply Prim's algorithm, the given graph must be weighted, connected and undirected.

Prim's Algorithm Implementation:

The implementation of Prim's Algorithm is explained in the following steps:

Step-01:

- Randomly choose any vertex.
- The vertex connecting to the edge having least weight is usually selected.

Step-02:

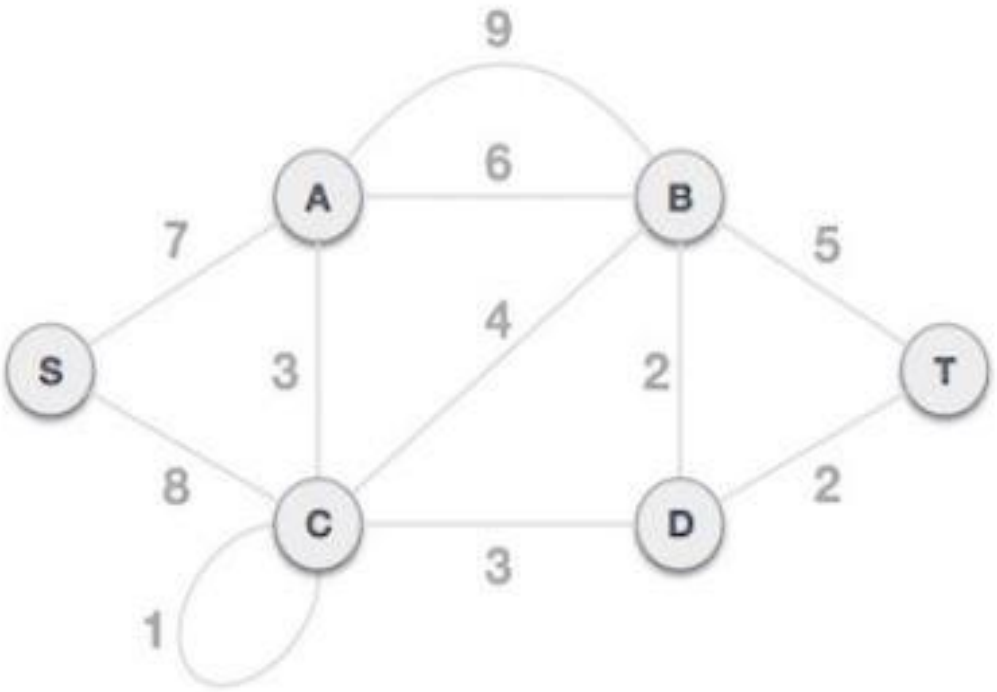
- Find all the edges that connect the tree to new vertices.
- Find the least weight edge among those edges and include it in the existing tree.
- If including that edge creates a cycle, then reject that edge and look for the next least weight edge

Step-03:

- Keep repeating step-02 until all the vertices are included and Minimum Spanning Tree (MST) is obtained.

Example 1:

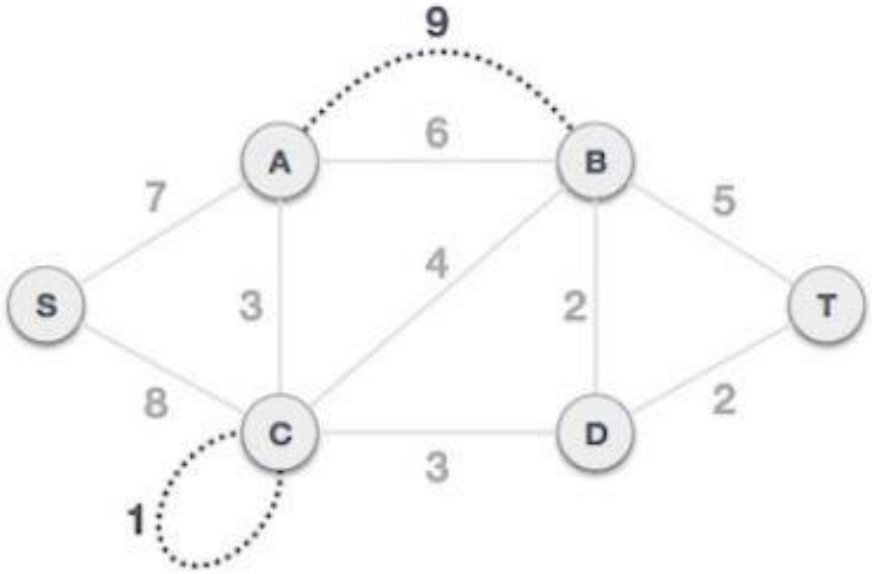
Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm.



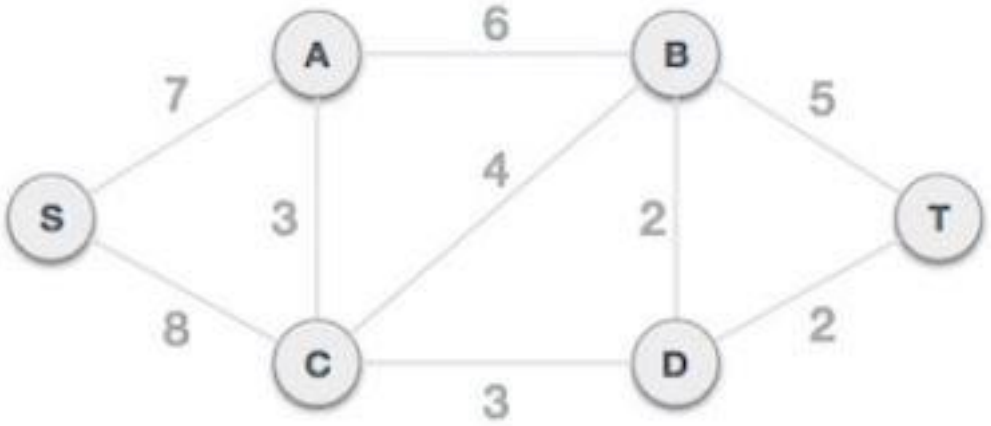
Solution:

Following are the steps to find the minimum spanning tree of above graph using prim algorithm.

Step 1: Remove all loops and parallel edges



- In case of parallel edges, keep the one which has the least cost associated and remove all others.

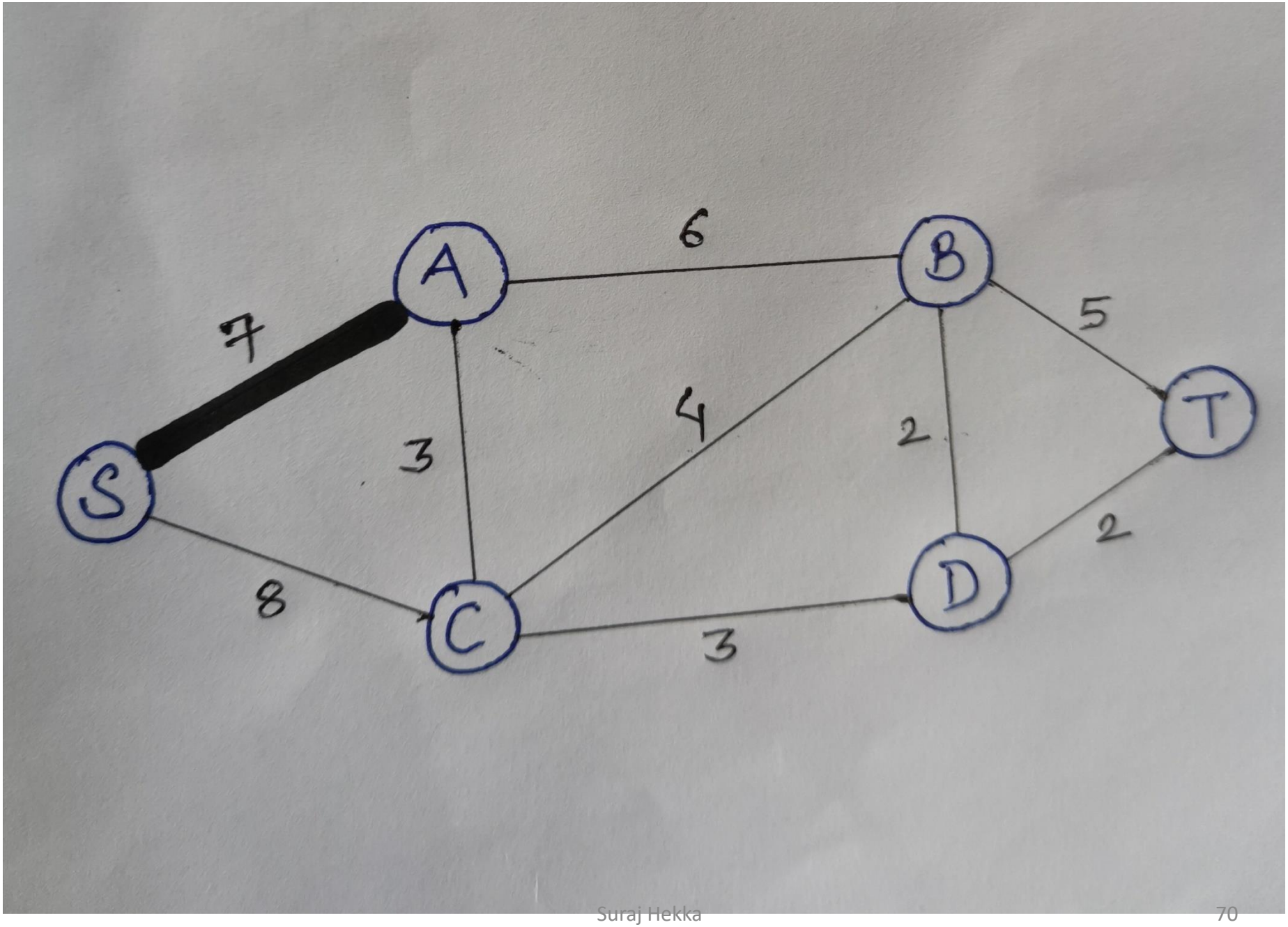


Step 2: Choose any arbitrary node as root node.

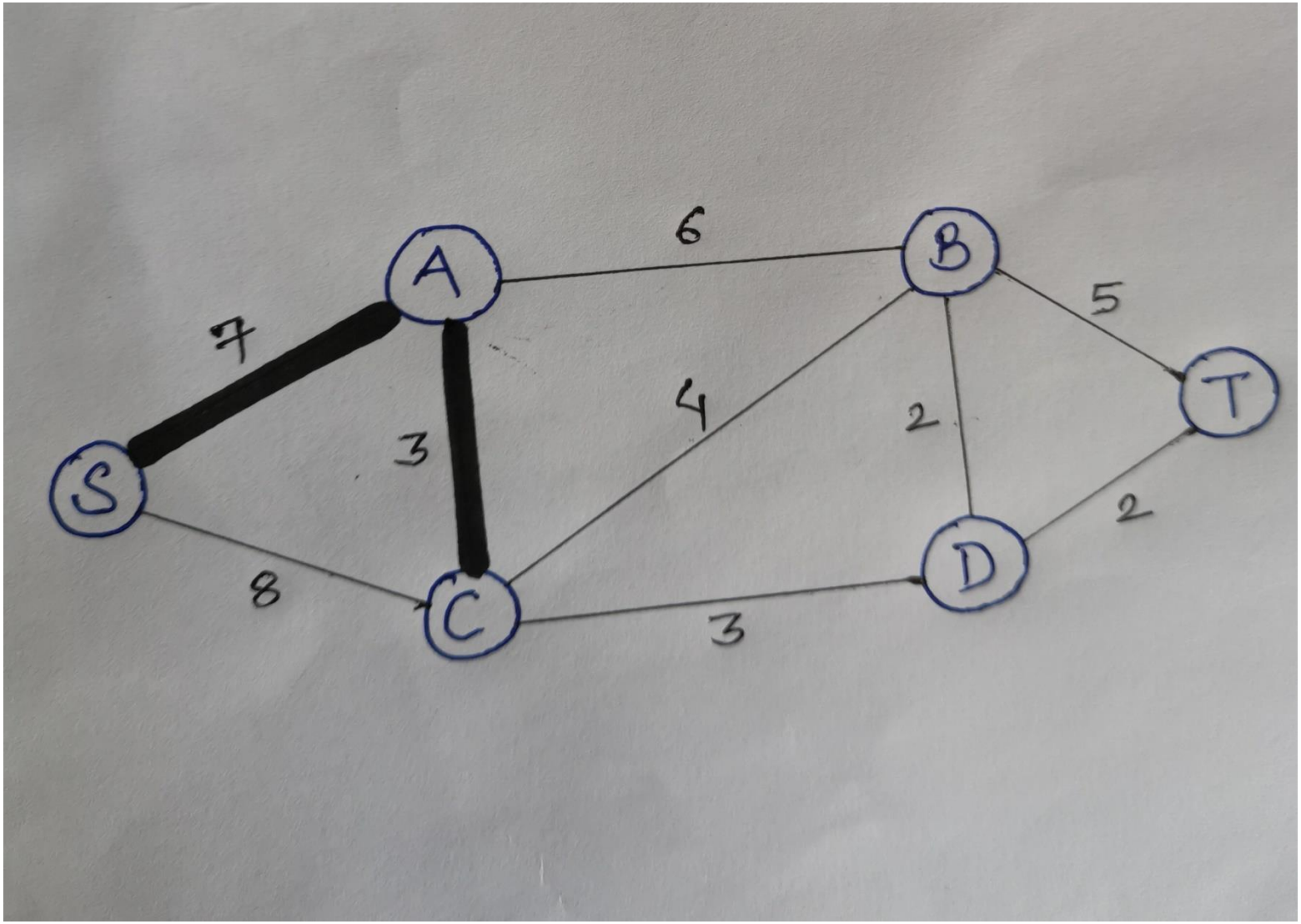
- In this case, we choose **S** node as the root node of Prim's spanning tree. This node is arbitrarily chosen, any node can be the root node.

Step 3: Check outgoing edges and select the one with less cost.

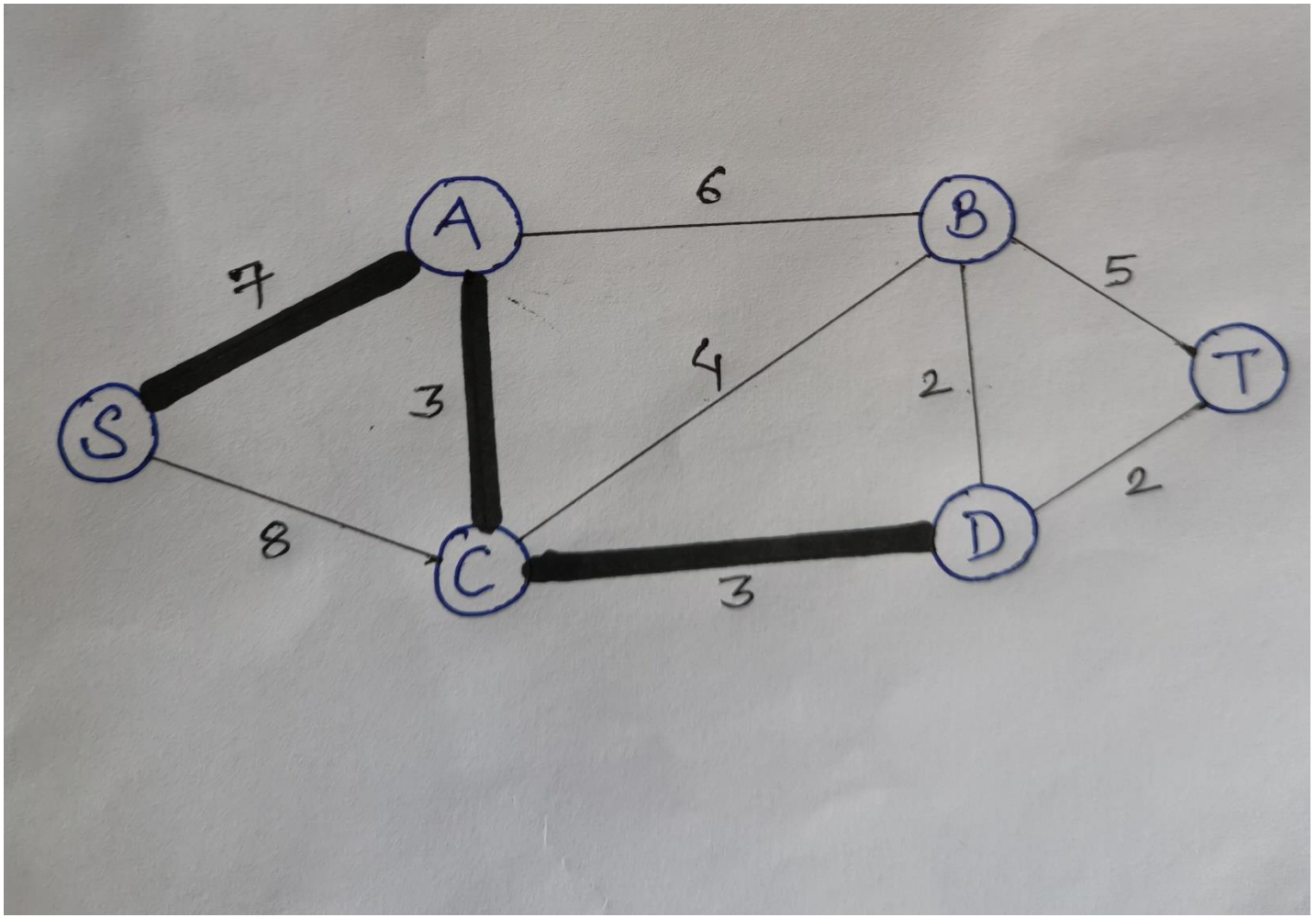
- After choosing the root node **S**, we see that S,A and S,C are two edges with weight 7 and 8, respectively. We choose the edge S,A as it is lesser than the other.


A
B
S
C
D
T
6
7
5
4
3
2
2
8
3

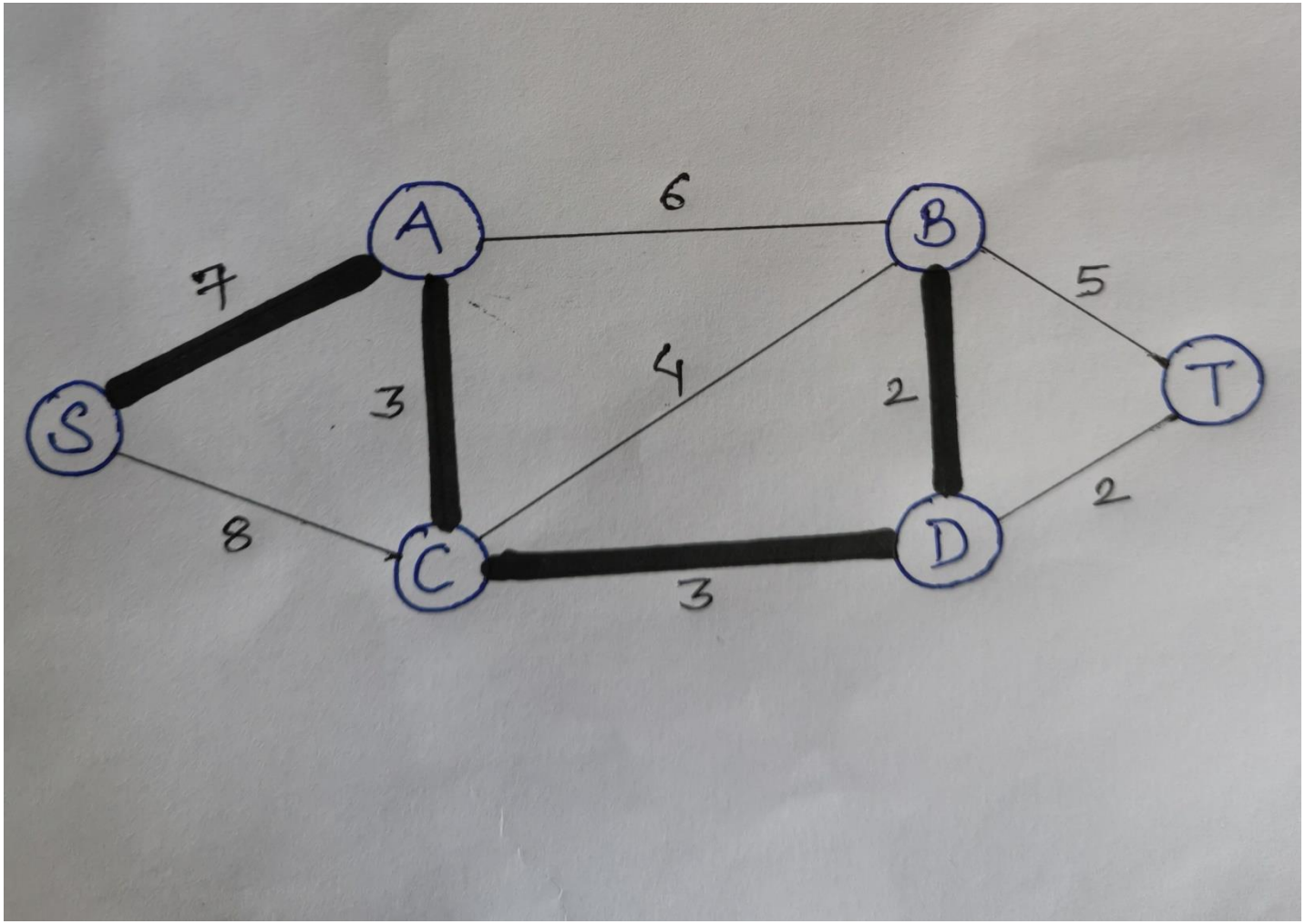
Now, the tree S-7-A is treated as one node and we check for all edges going out from it. We select the one which has the lowest cost and include it in the tree.



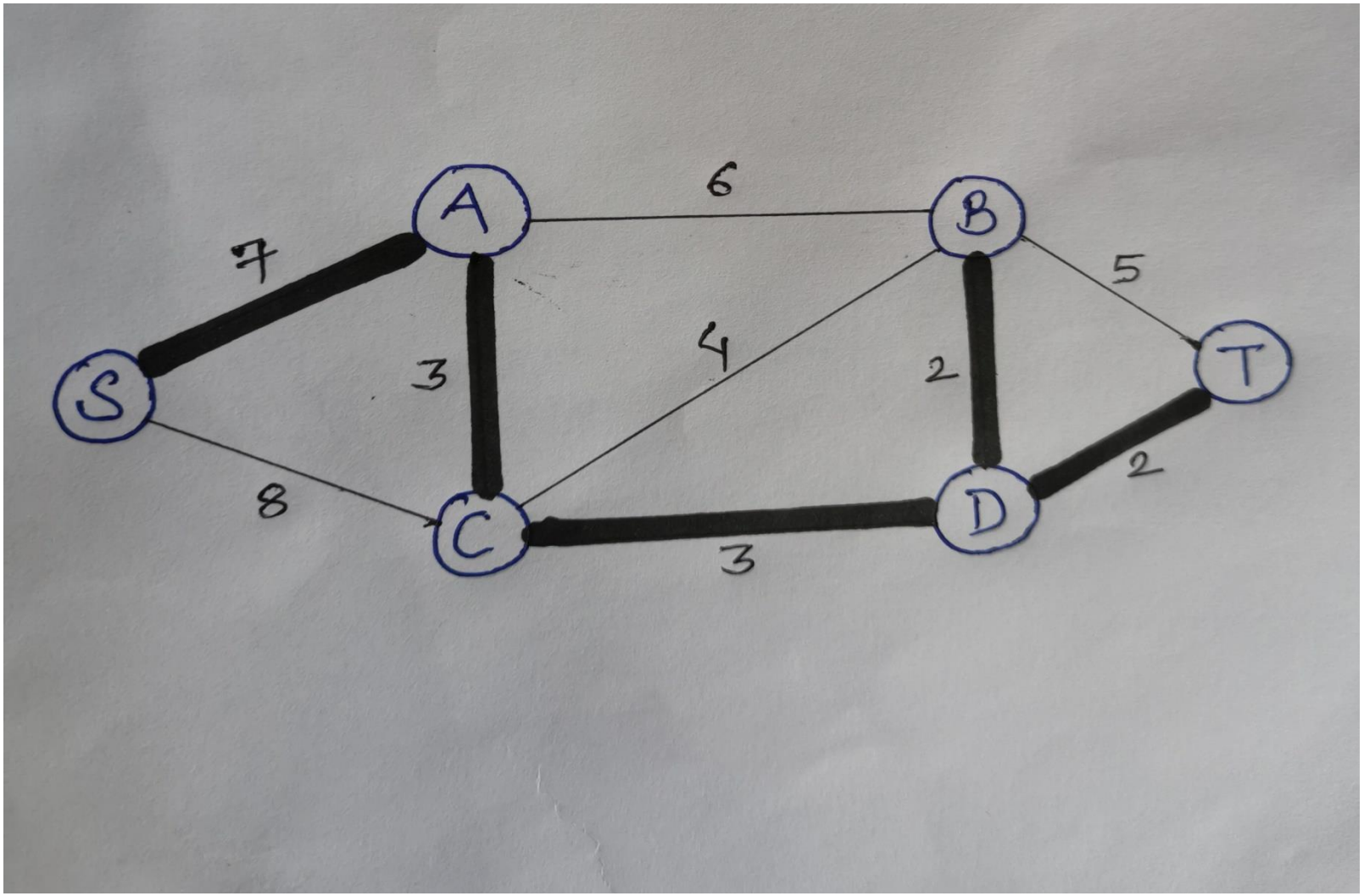
After this step, S-7-A-3-C tree is formed. Now we'll again treat it as a node and will check all the edges again. However, we will choose only the least cost edge. In this case, C-3-D is the new edge, which is less than other edges' cost 8, 6, 4, etc.



After adding node **D** to the spanning tree, we now have two edges going out of it having the same cost, i.e. D-2-T and D-2-B. so choose any one edge.

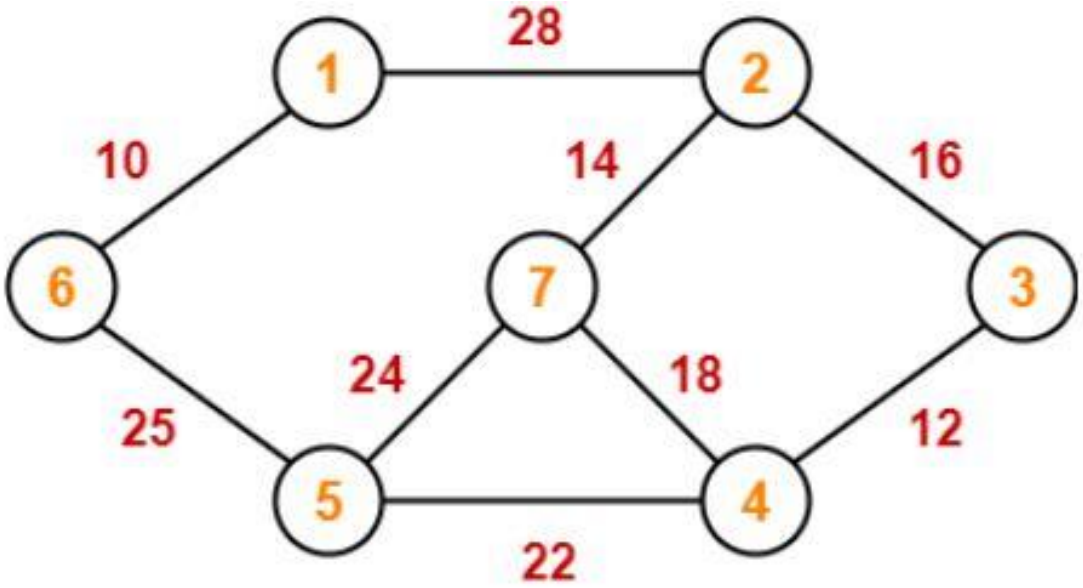


So the final minimum spanning tree is as follows:



Example 2:

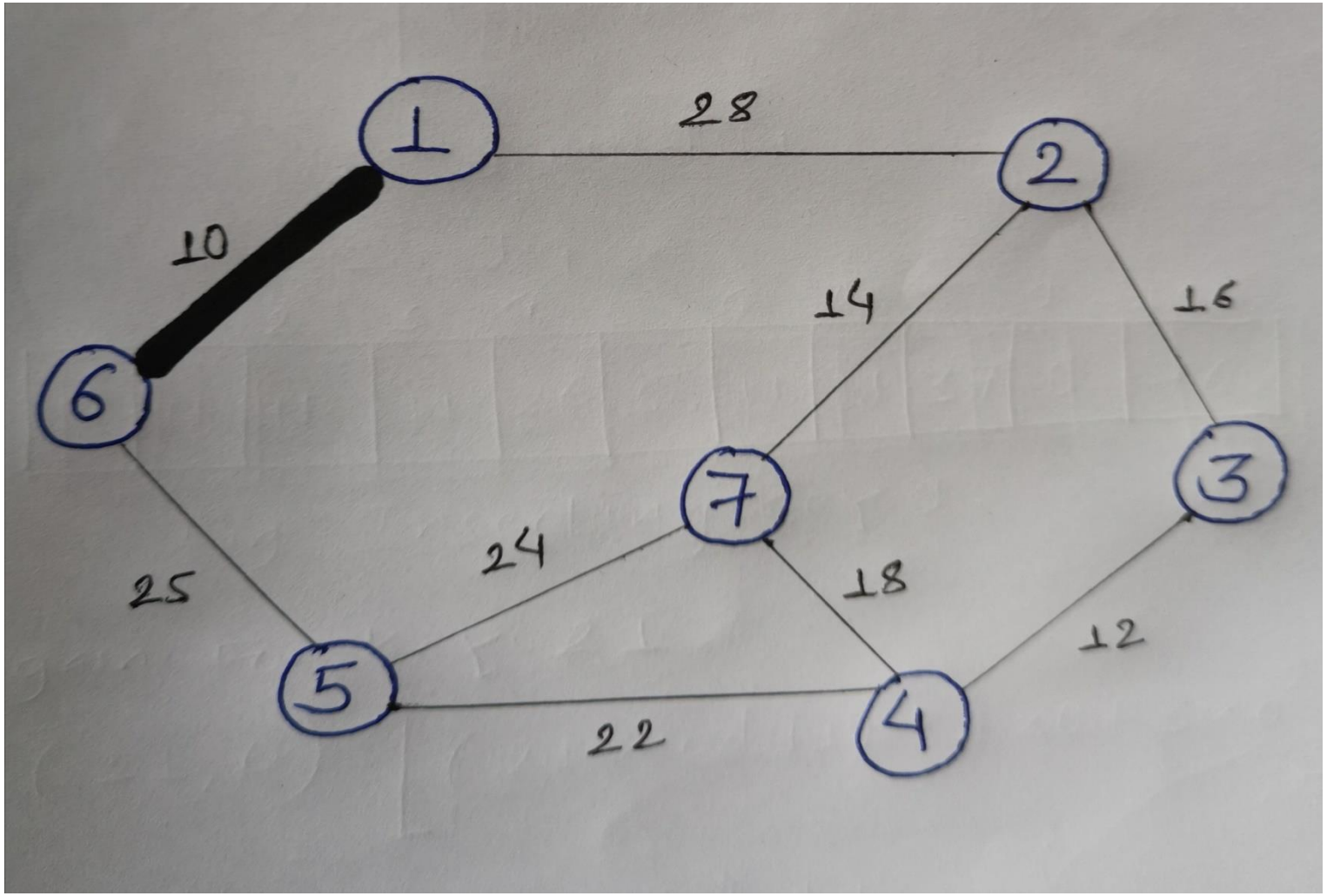
Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm.



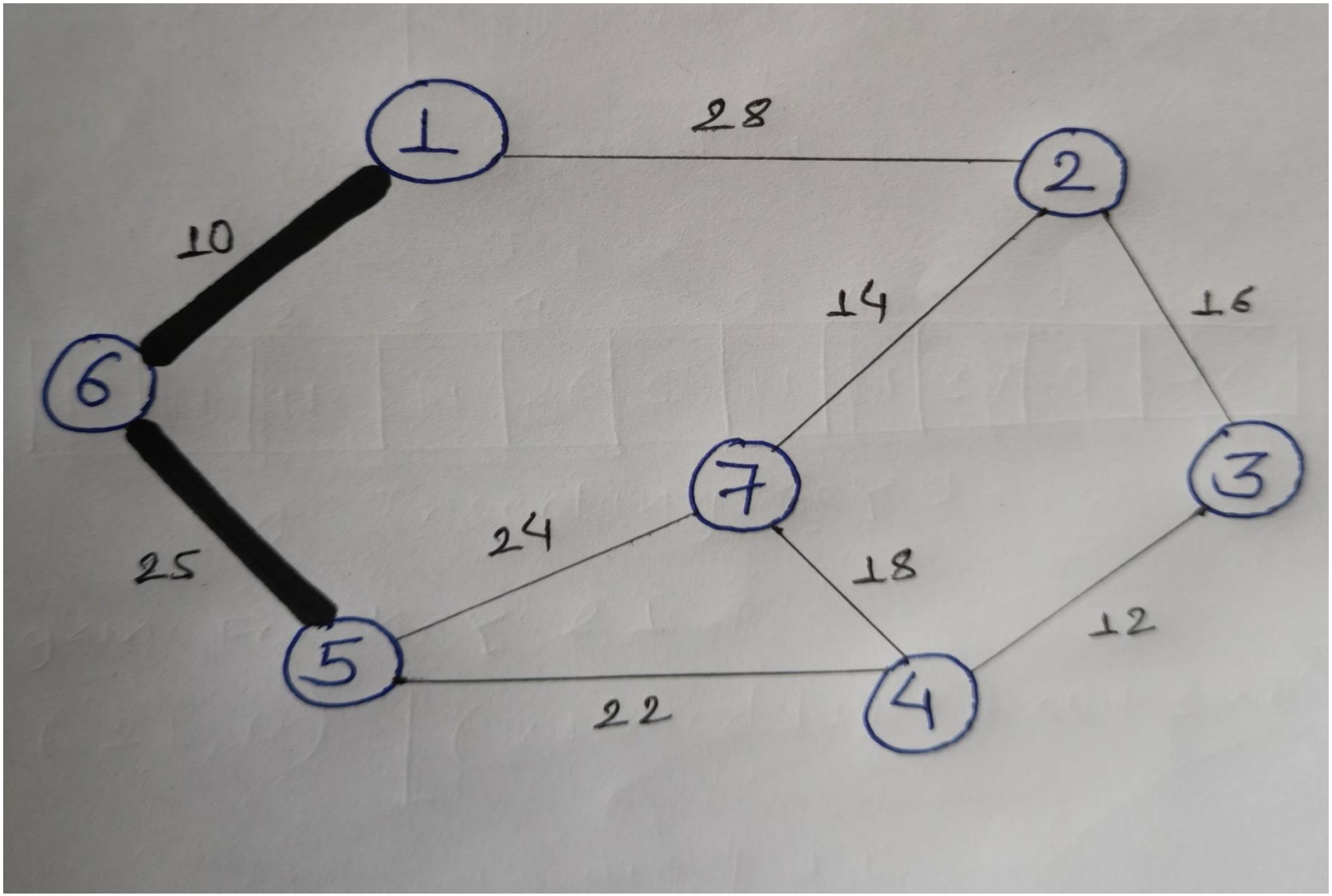
Following are the steps to find the minimum spanning tree of above graph using prim algorithm.

Step 1:

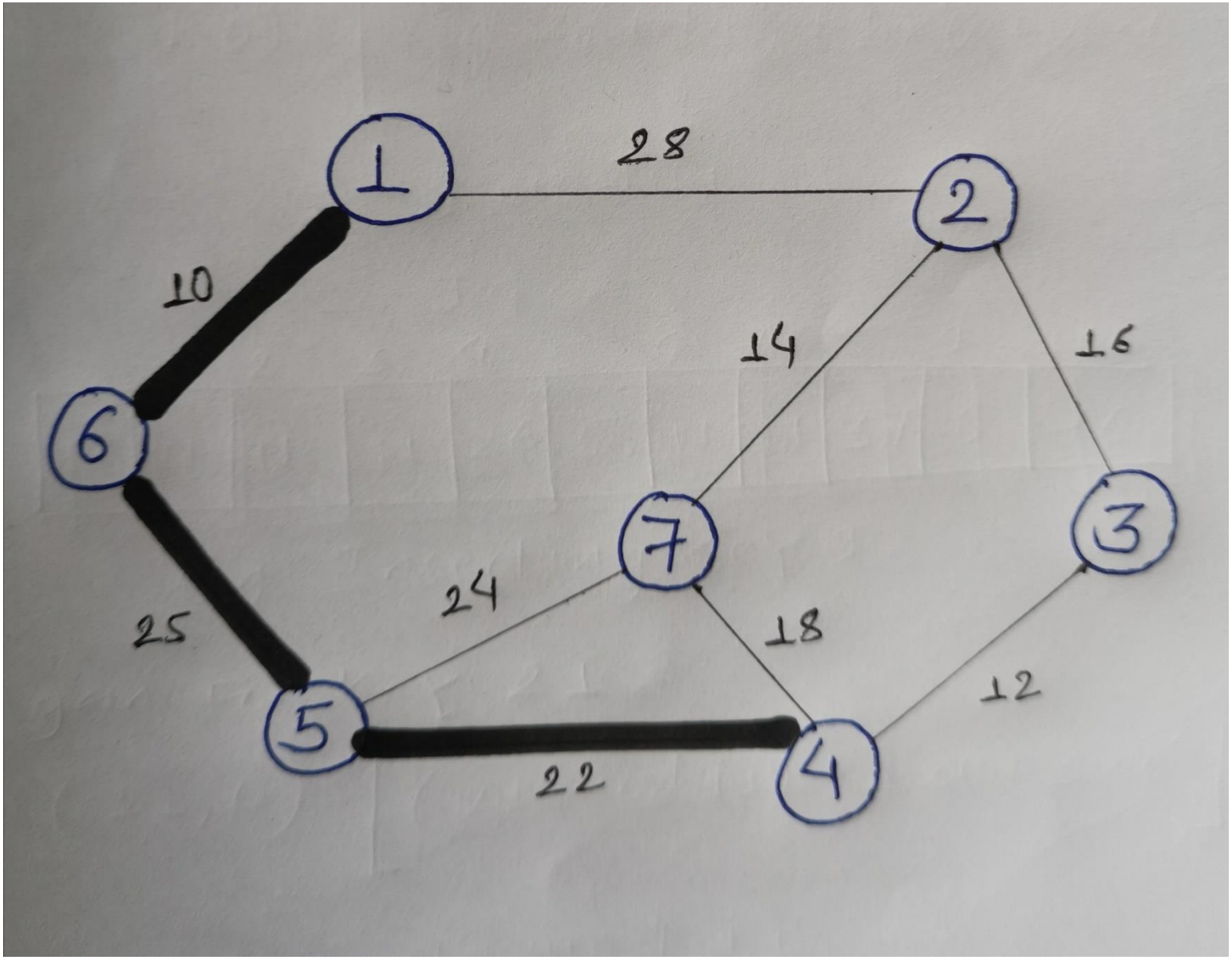
- Choose any arbitrary node as root node. Here root node is 6



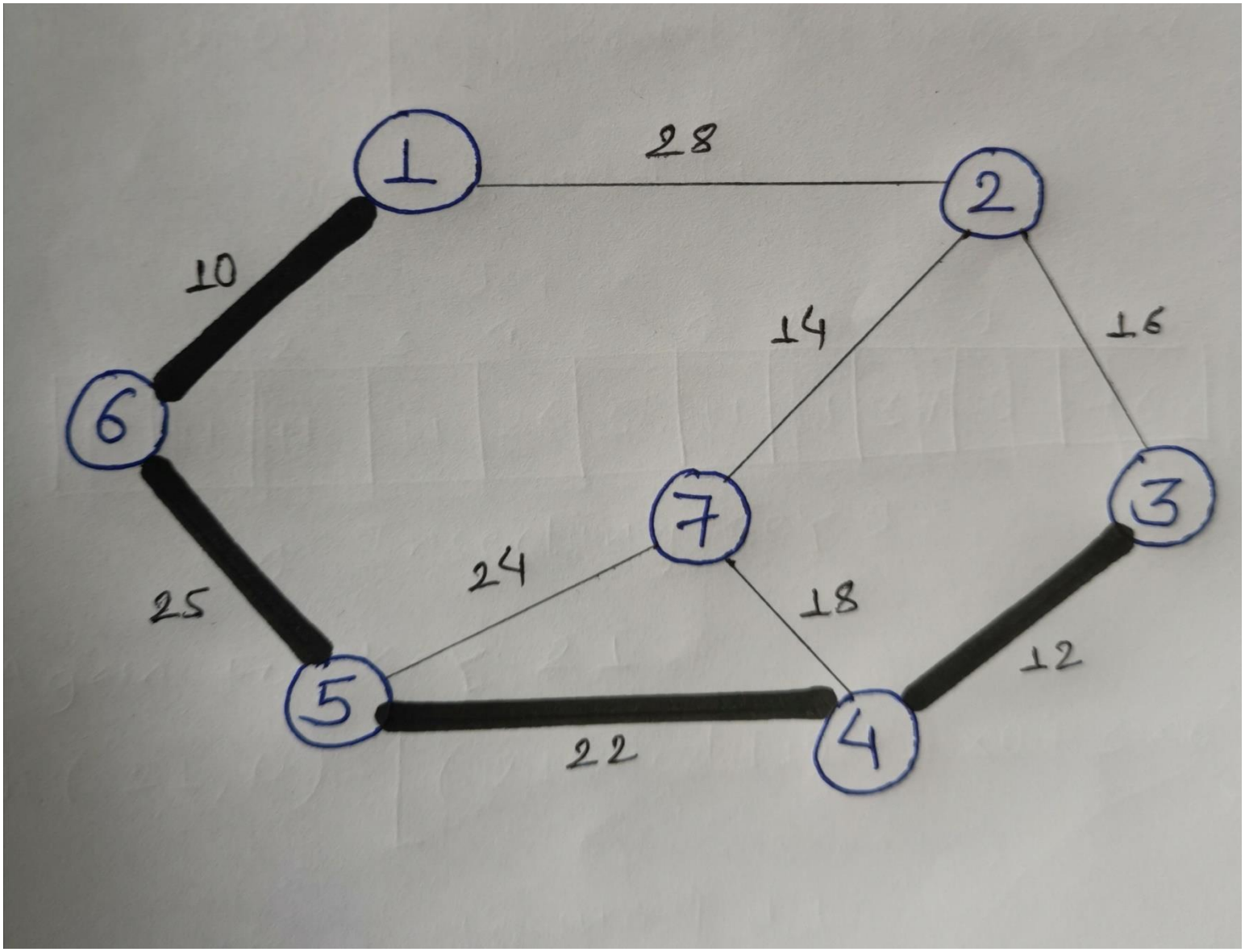
Step 2:



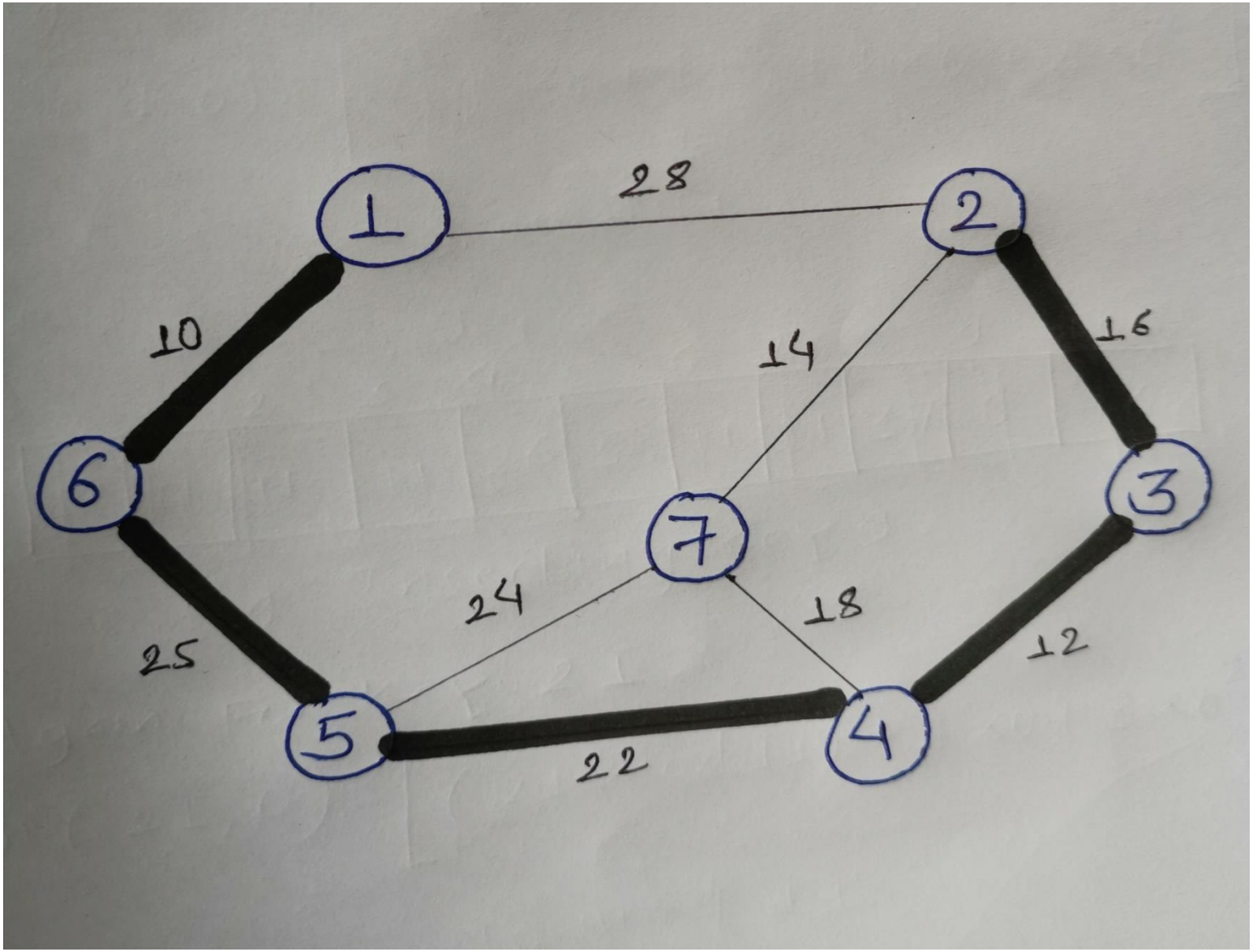
Step 3:



Step 4:



Step 5:



Step 6:

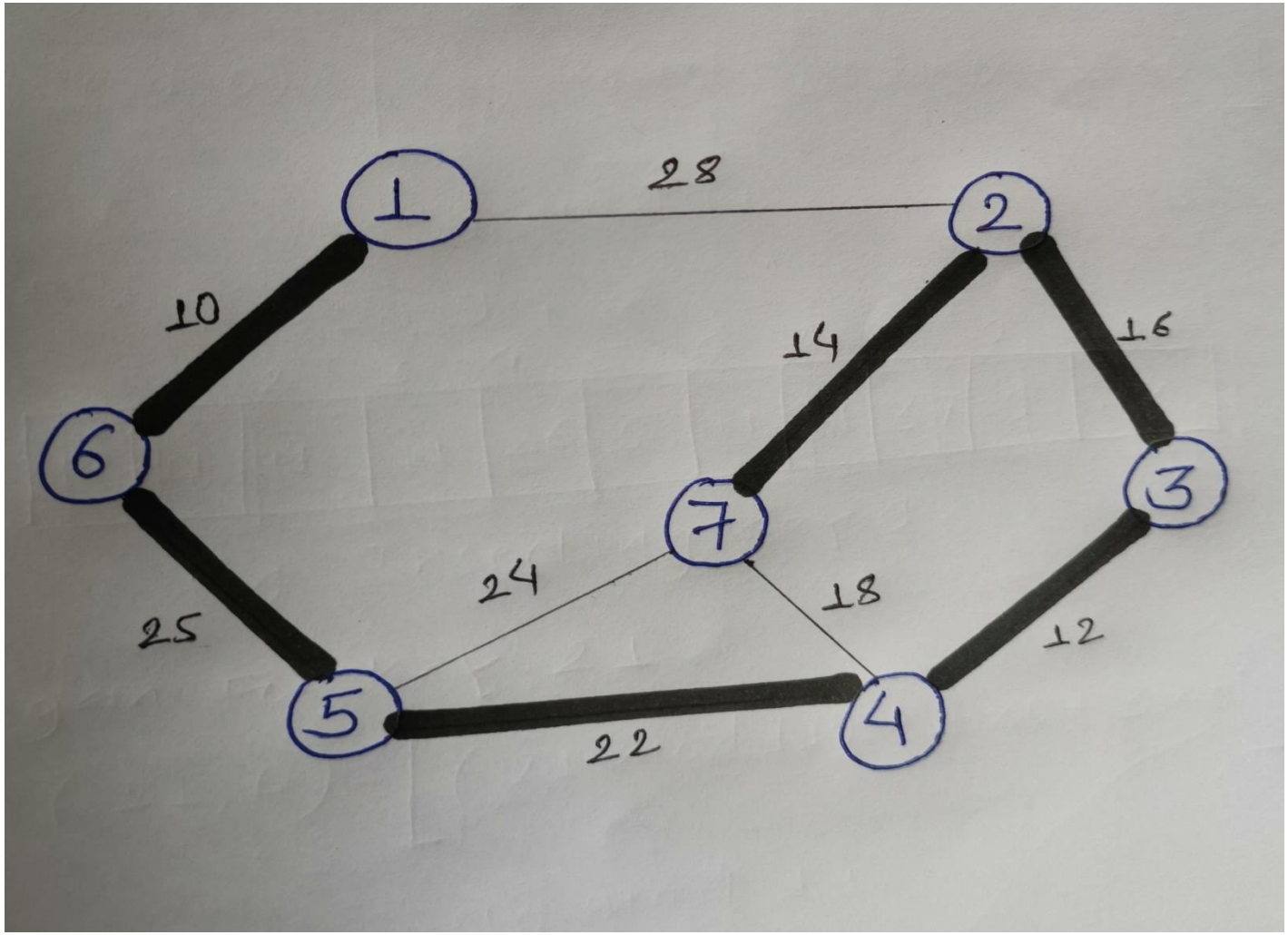
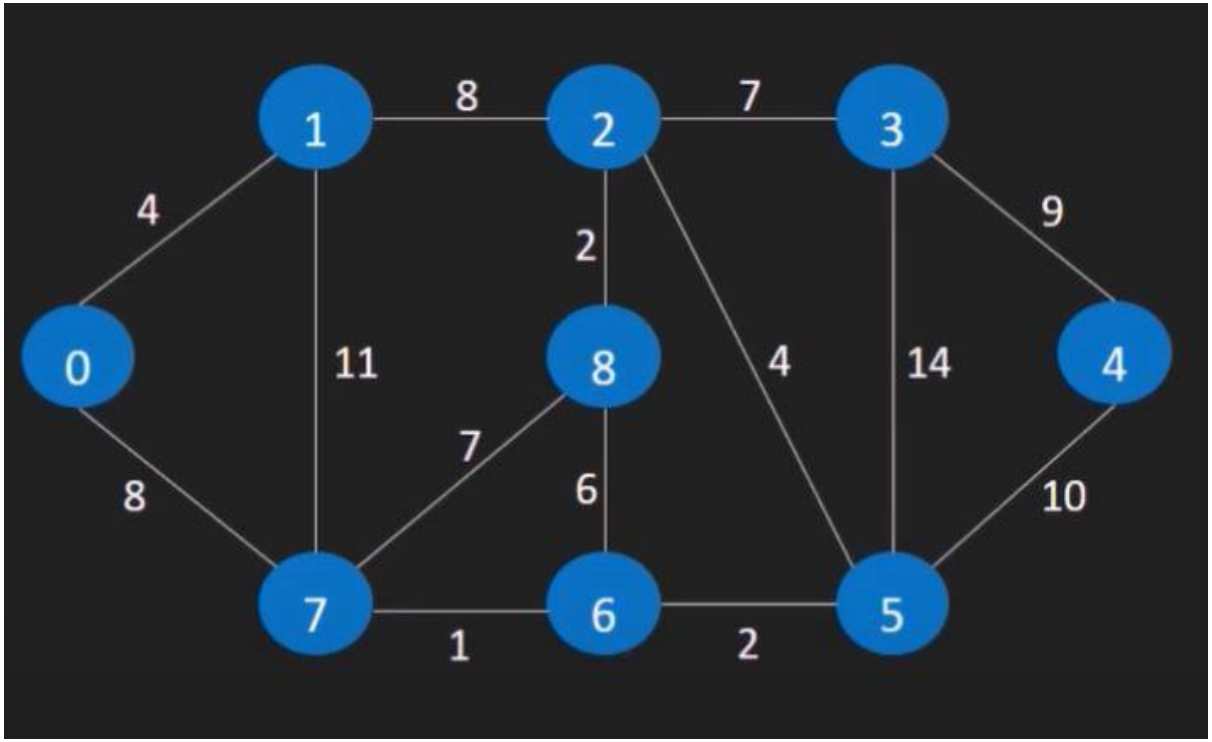


fig: final minimum spanning tree

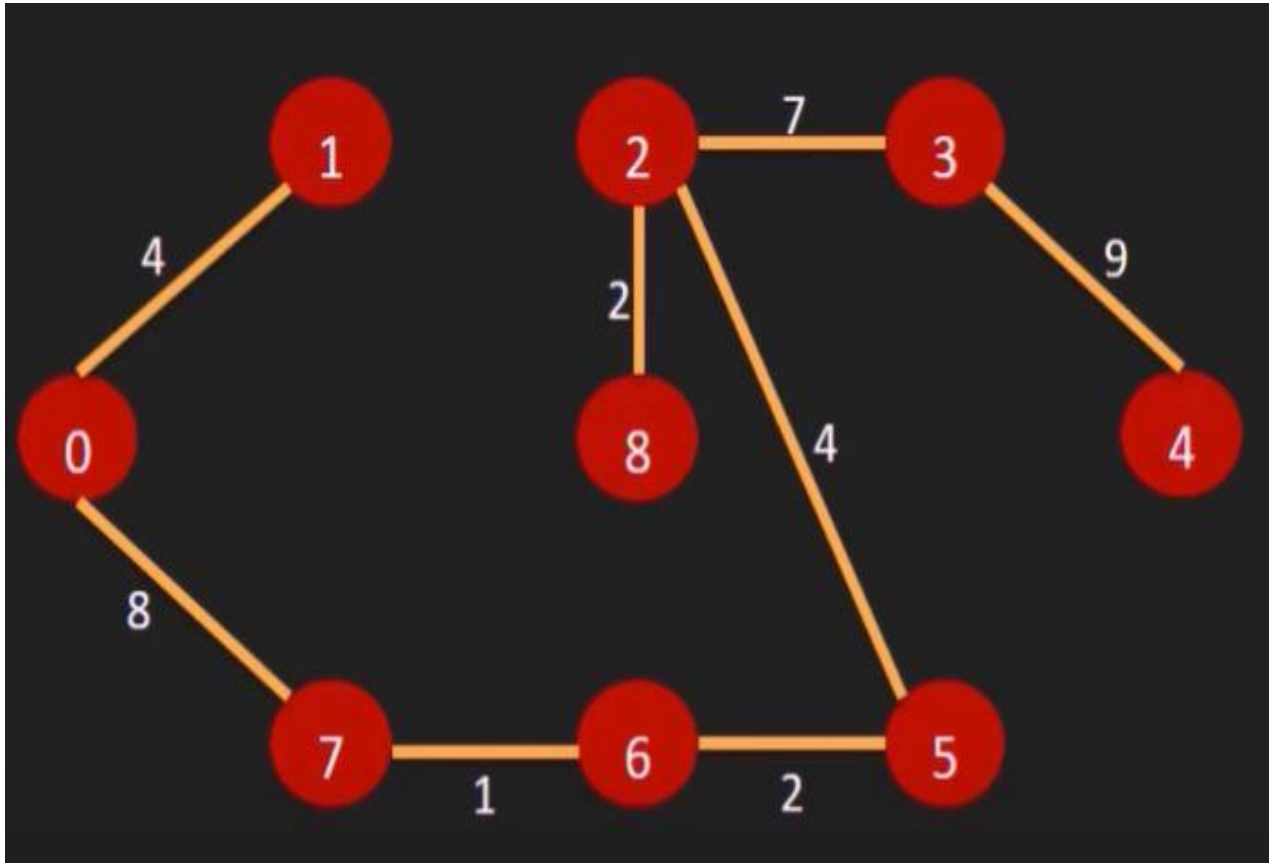
DO THIS QUESTION BY YOURSELF

Example 3:

Construct the minimal spanning tree for the graph shown below using Prim's algorithm. (show all necessary steps).



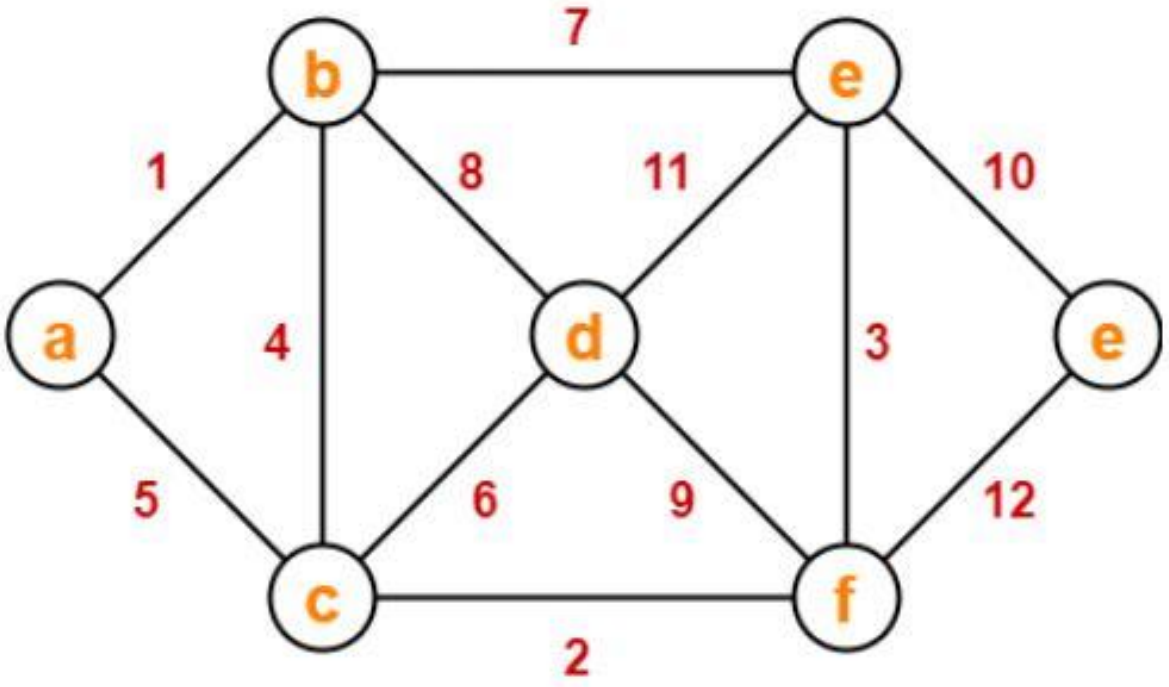
The final minimum spanning tree should look like this



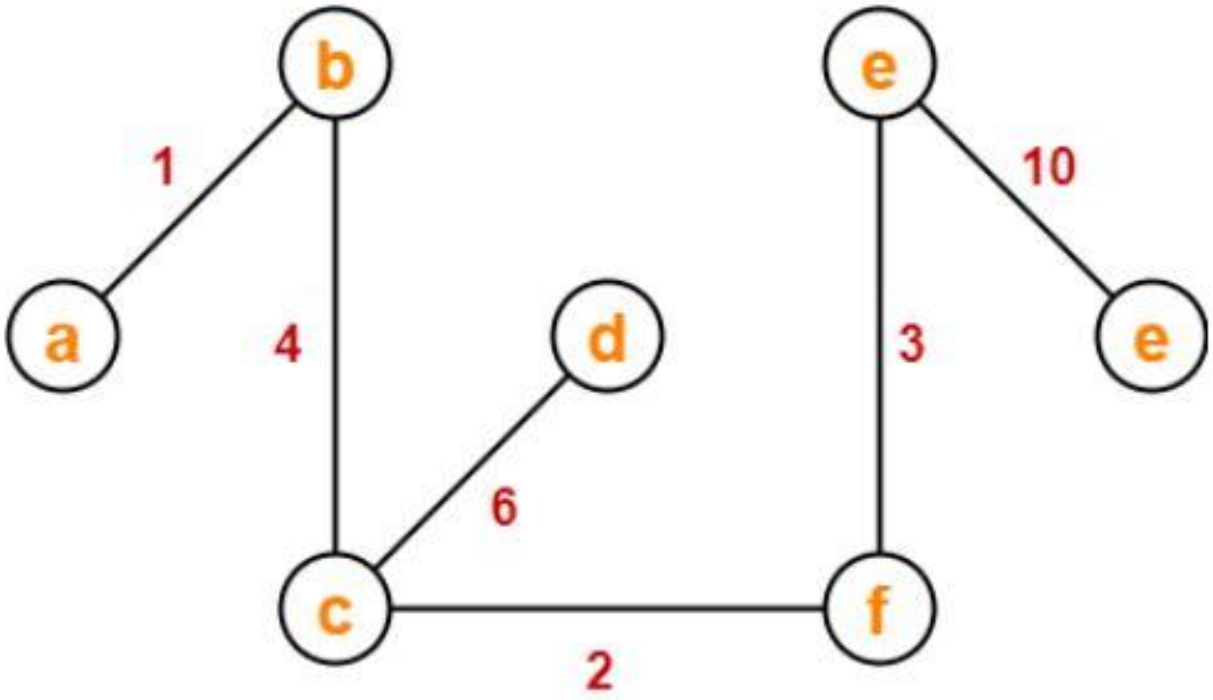
DO THIS QUESTION BY YOURSELF

Example 4:

Using Prim's Algorithm, find the cost of minimum spanning tree (MST) of the given graph.(show all necessary steps).



The final minimum spanning tree should look like this:



Cost of Minimum Spanning Tree

= Sum of all edge weights

= 1 + 4 + 2 + 6 + 3 + 10

= 26 units

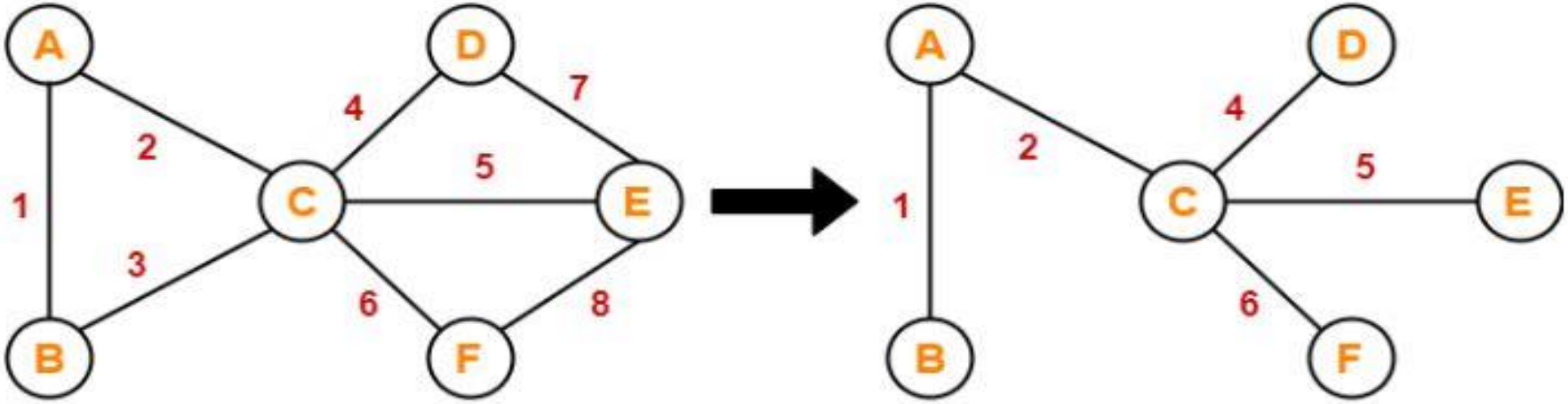
Difference Between Prim's and Kruskal's Algorithm:

Some important concepts based on them are

Concept-01: If all the edge weights are distinct, then both the algorithms are guaranteed to find the same MST.

Example-

Consider the following example



Given Graph

Minimum Spanning Tree (MST)

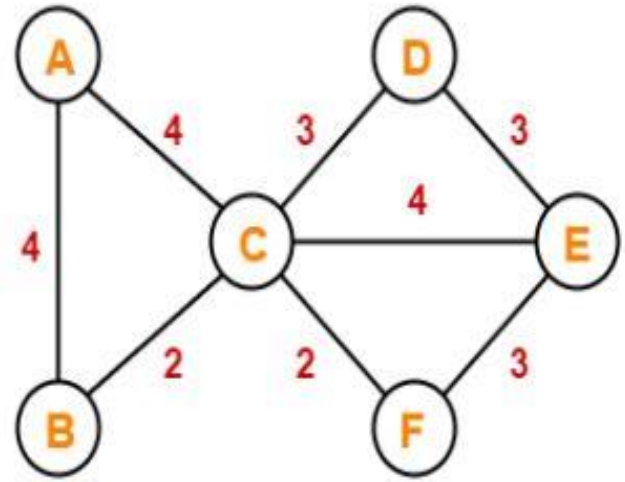
(Cost = 18 units)

Here, both the algorithms on the above given graph produces the same MST as shown.

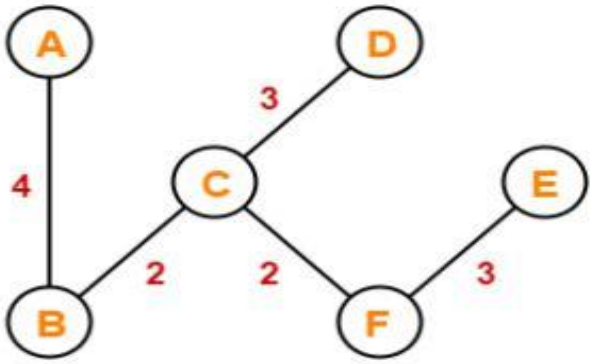
Concept-02: If all the edge weights are not distinct, then both the algorithms may not always produce the same MST. However, cost of both the MST would always be same in both the cases.

Example-

Consider the following example

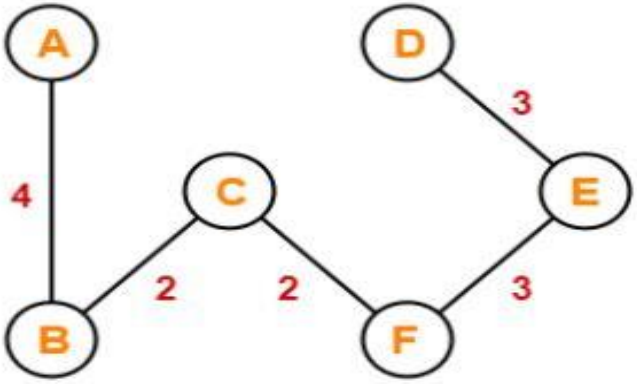


Given Graph



Result from Prim's Algorithm

(Cost = 14 units)



Result from Kruskal's Algorithm

(Cost = 14 units)

Here, both the algorithms on the given graph produces different MST as shown but the cost is same in both the cases.

Traversing a Graph/Graph Traversal Algorithm:

- Traversing the graph means examining/visiting all the nodes or vertices of the graph.

- There are two standard methods by using which, we can traverse the graphs i.e.

1. Breadth First Search/Traversal (BFS)

2. Depth First Search/Traversal (DFS)

During the execution of these algorithms , each node N of G will be in one of three states , called the *status* of N, as follows:

a. STATUS = 1 (Ready state):

The initial state of the node N.

b. STATUS = 2 (Waiting state):

The node N is on the QUEUE or STACK, waiting to be processed.

c. STATUS = 3 (Processed state):

The node N has been processed.

1. Breadth First Search/Traversal (BFS):

- There are many ways to traverse graphs. BFS is the most commonly used approach.

- BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbor nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.

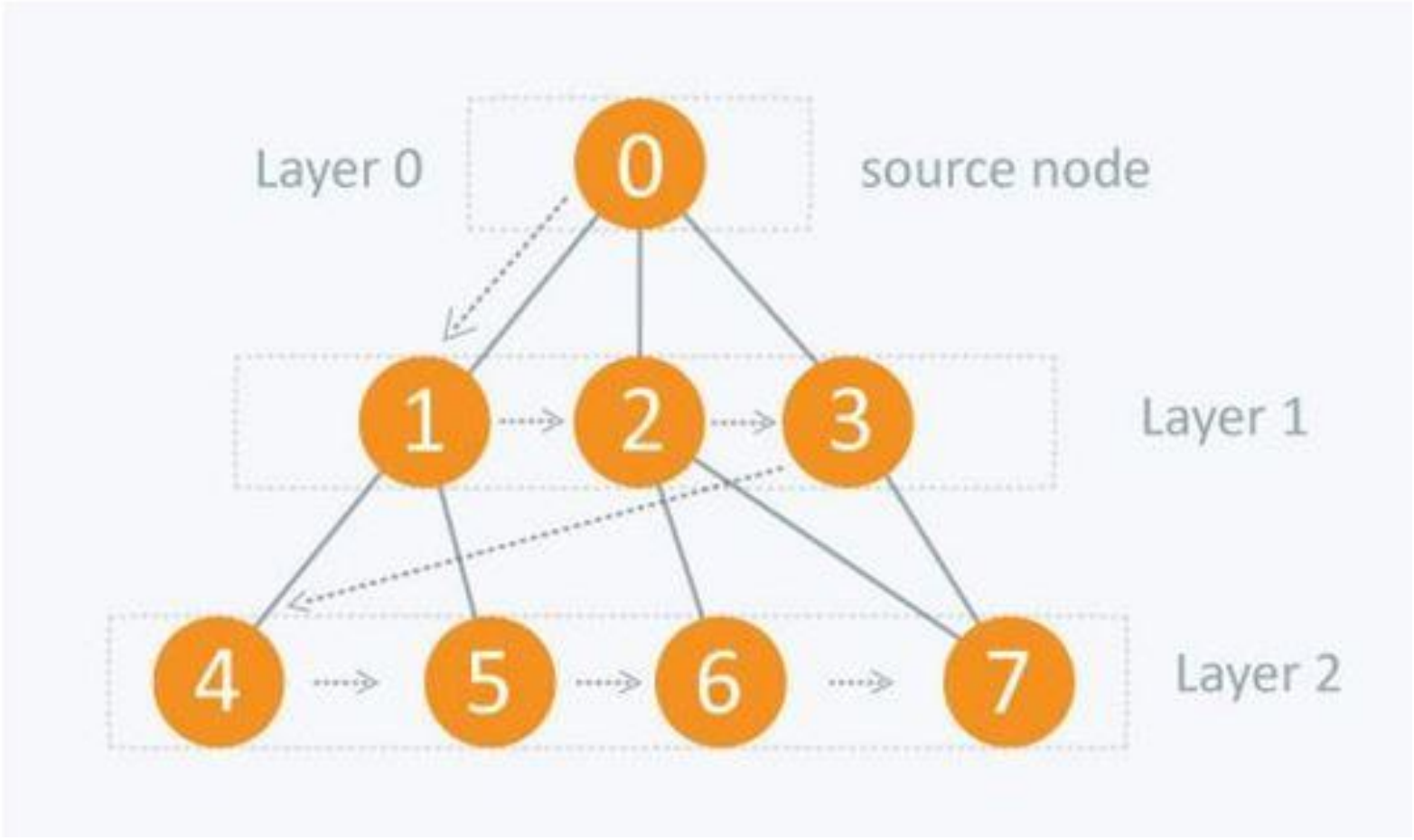
- As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

  a. First move horizontally and visit all the nodes of the current layer

  b. Move to the next layer

- The BFT will use a queue as an auxiliary structure to hold nodes for future processing.

- Consider the following diagram.



Breadth first traversal algorithm:

This algorithm executes a BFT on graph G beginning at a starting node A.

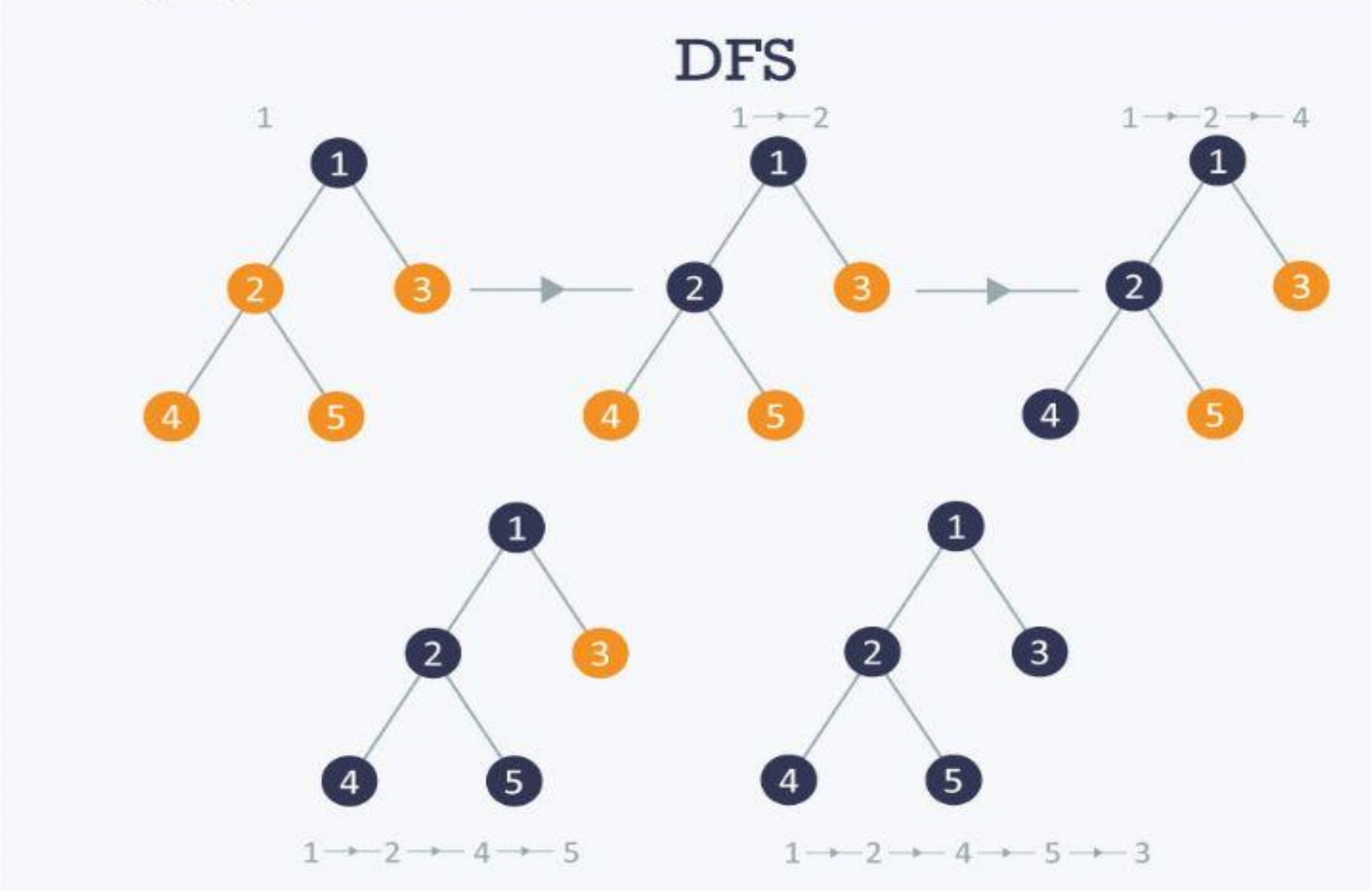
1. Initialize all nodes to the ready state (STATUS = 1).
2. Put the starting node A in QUEUE and change its status to the waiting state (STATUS = 2).
3. Repeat the following steps until QUEUE is empty:
   a. Remove the front node N of QUEUE. Process N and change the status of N to the processed state (STATUS = 3).
   b. Add to the rear of QUEUE all the neighbors of N that are in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2).
4. Exit.

2. Depth First Search/Traversal (DFS):

- Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.

- The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

The following image shows how DFS works.



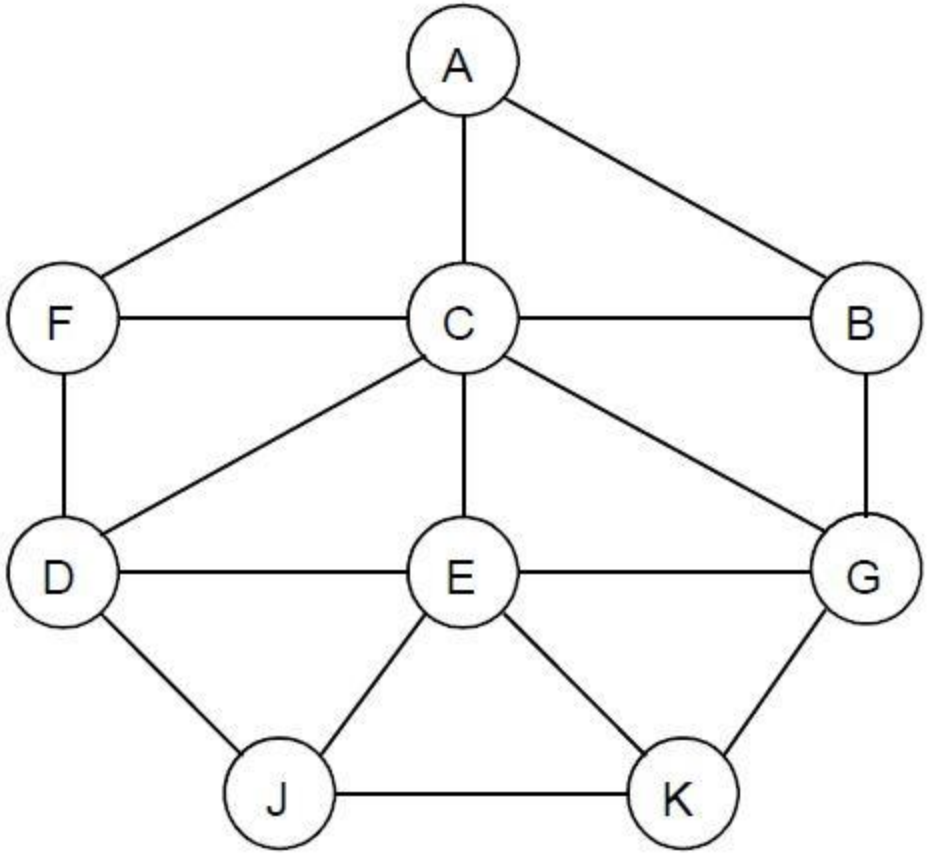
Depth first traversal algorithm:

This algorithm executes a DFT on graph G beginning at a starting node A.

1. Initialize all nodes to the ready state (STATUS = 1).
2. Push the starting node A into STACK and change its status to the waiting state (STATUS = 2).
3. Repeat the following steps until STACK is empty:
   a. Pop the top node N from STACK. Process N and change the status of N to the processed state (STATUS = 3).
   b. Push all the neighbors of N that are in the ready state (STATUS = 1) and change their status to the waiting state (STATUS = 2).
4. Exit.

Example:

1. **Consider the graph shown below. Traverse the graph shown below in breadth first order and depth first order.**



A Graph G

Node	*Adjacency List*
A	F, C, B
B	A, C, G
C	A, B, D, E, F, G
D	C, F, E, J
E	C, D, G, J, K
F	A, C, D
G	B, C, E, K
J	D, E, K
K	E, G, J

The steps involved in breadth first traversal are as follows:

Current Node	QUEUE	Processed Nodes	Status								
			A	*B*	*C*	*D*	*E*	*F*	*G*	*J*	*K*
			1	1	1	1	1	1	1	1	1
	A		2	1	1	1	1	1	1	1	1
A	F C B	A	3	2	2	1	1	2	1	1	1
F	C B D	A F	3	2	2	2	1	3	1	1	1
C	B D E G	A F C	3	2	3	2	2	3	2	1	1
B	D E G	A F C B	3	3	3	2	2	3	2	1	1
D	E G J	A F C B D	3	3	3	3	2	3	2	2	1
E	G J K	A F C B D E	3	3	3	3	3	3	2	2	2
G	J K	A F C B D E G	3	3	3	3	3	3	3	2	2
J	K	A F C B D E G J	3	3	3	3	3	3	3	3	2
K	EMPTY	A F C B D E G J K	3	3	3	3	3	3	3	3	3

For the above graph the Breadth first traversal sequence is: ***A F C B D E G J K***.

The steps involved in depth first traversal are as follows:

Current Node	Stack	Processed Nodes	Status								
			A	*B*	*C*	*D*	*E*	*F*	*G*	*J*	*K*
			1	1	1	1	1	1	1	1	1
	A		2	1	1	1	1	1	1	1	1
A	B C F	A	3	2	2	1	1	2	1	1	1
F	B C D	A F	3	2	2	2	1	3	1	1	1
D	B C E J	A F D	3	2	2	3	2	3	1	2	1
J	B C E K	A F D J	3	2	2	3	2	3	1	3	2
K	B C E G	A F D J K	3	2	2	3	2	3	2	3	3
G	B C E	A F D J K G	3	2	2	3	2	3	3	3	3
E	B C	A F D J K G E	3	2	2	3	3	3	3	3	3
C	B	A F D J K G E C	3	2	3	3	3	3	3	3	3
B	EMPTY	A F D J K G E C B	3	3	3	3	3	3	3	3	3

For the above graph the Depth first traversal sequence is: ***A F D J K G E C B***.

Applications of Graph Theory

Graph theory has its applications in diverse fields of engineering −

Electrical Engineering − The concepts of graph theory is used extensively in designing circuit connections. The types or organization of connections are named as topologies. Some examples for topologies are star, bridge, series, and parallel topologies.

Computer Science − Graph theory is used for the study of algorithms. For example,

- Kruskal's Algorithm
- Prim's Algorithm
- Dijkstra's Algorithm

Computer Network − The relationships among interconnected computers in the network follows the principles of graph theory.

Science − The molecular structure and chemical structure of a substance, the DNA structure of an organism, etc., are represented by graphs.

Linguistics − The parsing tree of a language and grammar of a language uses graphs.

General − Routes between the cities can be represented using graphs. Depicting hierarchical ordered information such as family tree can be used as a special type of graph called tree.