Q.1 Define graph?

Ans. Graph is a non linear data structure, it contains a set of points known as nodes (or vertices) and set of linkes known as edges (or Arcs) which connets the vertices.

A graph is defined as follows:

- Graph is a collection of vertices and arcs which connects vertices in the graph

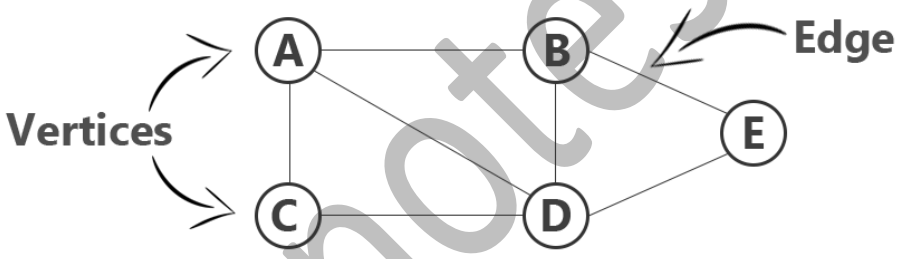
or

- Graph is a collection of nodes and edges which connects nodes in the graph

A Graph 'G' is a collection of two set V and E where vertices are denoted as v0, v1... vn-1 and collection of edges e0, e1... en where V(G) is set of vertices, E(G) is set of edges and graph G can be represented as G(V,E).

Example

The following is a graph with 5 vertices and 6 edges.
This graph G can be defined as G = (V , E)
Where V = {A,B,C,D,E} and E = {(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.



Vertex;

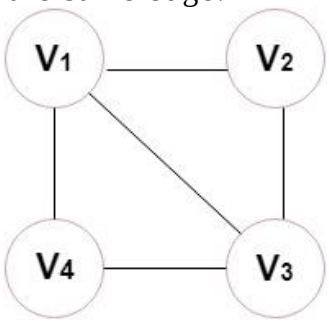
A individual data element of a graph is called as Vertex. Vertex is also known as node. In above example graph, A, B, C, D & E are known as vertices.

Edge:

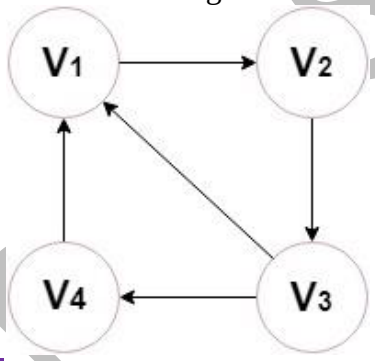
An edge is a connecting link between two vertices. Edge is also known as Arc. An edge is represented as (startingVertex, endingVertex). For example, in above graph, the link between vertices A and B is represented as (A,B). In above example graph, there are 7 edges (i.e., (A,B), (A,C), (A,D), (B,D), (B,E), (C,D), (D,E)).

Q.2 Explain different types of graph and terminology used in graph ?

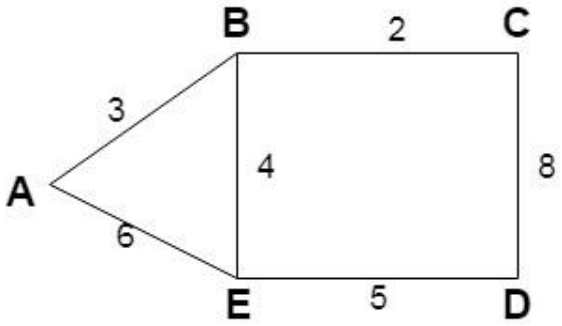
Ans. **Undirected Graph:** If the pair of vertices are unordered i.e if the edges joining 2 vertices are not with an arrow, then the graph is called Undirected i.e (V_1, V_2) or (V_2, V_1) refers the same edge.



Directed Graph: If the pair of vertices are ordered i.e if the edges joining 2 vertices are with an arrow, then the graph is called Directed or Diagraph i.e (V_1, V_2) and (V_2, V_1) refers to the different edge.



Weighted Graph: A Graph is said to be weighted if all the edges in it are labelled with some numbers and this number is known as the weight of the edge.



Self Loop: If there is an edge whose starting and ending vertices are same i.e (V_1, V_1) is an edge then it is called Self Loop.

Parallel Edges: If there are more than 1 edge between the same pair of vertices, then they are known as Parallel Edges.

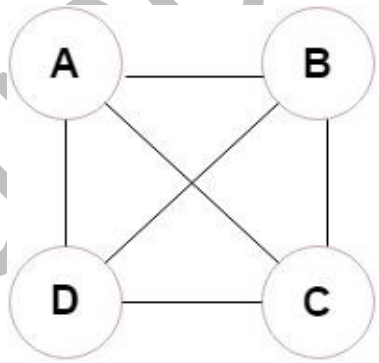
Adjacent Vertices : A vertex 'u' is adjacent to (or the neighbour of) other vertex 'v' if there is an edge from u -> v.

Degree of a Vertex : The degree of a vertex is the number of edges incident on that vertex. In an Undirected Graph, the number of edges connected to a node is called the degree of a node. In a Diagraph, there are 2 types of degrees for every node :

- **Indegree :** It is the number of edges coming to that vertex or in other words edges incident to it.
- **Outdegree :** It is the number of edges going outside from that node or the edges incident from it.

Multigraph : A Graph which has either a self loop or parallel edges or both is called MultiGraph.

Complete Graph :A Graph is said to be complete graph if each vertex is adjacent to every other vertex in graph or we can say that there is an edge between any pair of nodes in the graph. An Undirected complete graph will contain [n(n-1)/2] number of edges, where n is number of vertices.

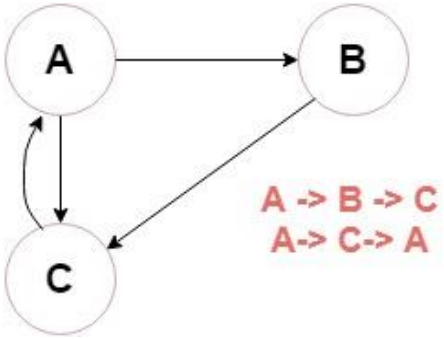


Connected Graph: In a Graph, 2 vertices V_1 and V_2 are said to be both connected if there is a path G from V_1 and V_2 or V_2 and V_1 i.e if there is a path from any node of graph to any other node i.e for every pair of distinct vertices in G there is a path. Example : Complete Graph

Path: A Path is a sequence of distinct vertices each adjacent to the next. The length of a path is the number of edges on it.

Cycle: A Cycle is a path containing atleast one or more edge which starts from a vertex and terminates into the same vertex is called Cyclic Graph.

Bridge : If on removing an edge from the graph, the graph becomes disconnected then that edge is called the Bridge.



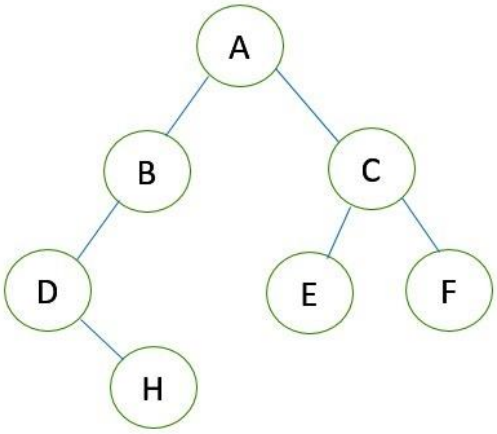
Q.3 Explain the Sequential representation of the graph?

Ans. Sequential representation of Binary Tree:

Let us consider that we have a tree **T**. let our tree **T** is a binary tree that us complete binary tree. Then there is an efficient way of representing **T** in the memory called the sequential representation or array representation of **T**. This representation uses only a linear array TREE as follows:

1. The root **N** of **T** is stored in **TREE [1]**.
2. If a node occupies **TREE [k]** then its left child is stored in **TREE [2 * k]** and its right child is stored into **TREE [2 * k + 1]**.

For Example:
Consider the following Tree:



Its sequential representation is as follow:

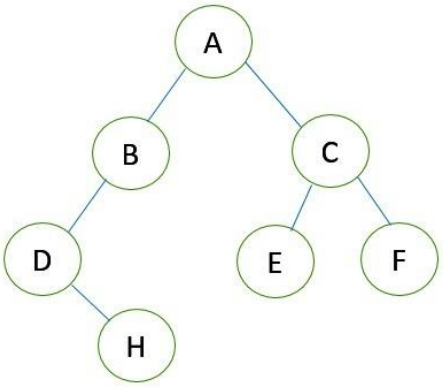
A	B	C	D	-	E	F	-	H		

Q.4 Explain the Link representation of the graph?
Ans.
Consider a Binary Tree **T**. **T** will be maintained in memory by means of a linked list representation which uses three parallel arrays; **INFO**, **LEFT**, and **RIGHT** pointer variable **ROOT** as follows. In Binary Tree each node **N** of **T** will correspond to a location **k** such that

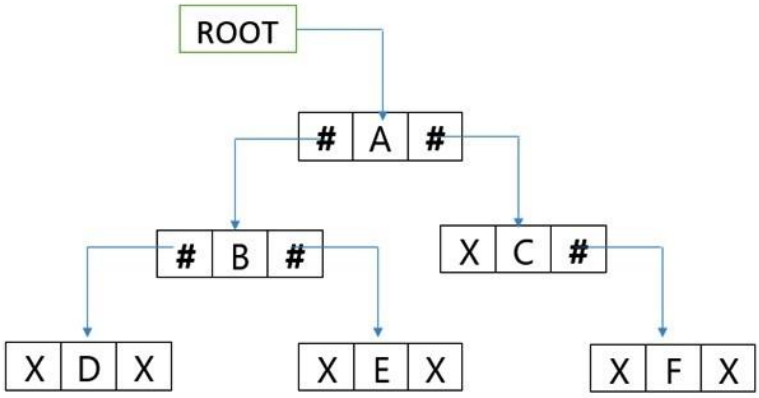
1. **LEFT [k]** contains the location of the left child of node **N**.
2. **INFO [k]** contains the data at the node **N**.
3. **RIGHT [k]** contains the location of right child of node **N**.

Representation of a node:
In this representation of binary tree root will contain the location of the root **R** of **T**. If any one of the subtree is empty, then the corresponding pointer will contain the null value if the tree **T** itself is empty, the **ROOT** will contain the null value.

Example
Consider the binary tree **T** in the figure. A schematic diagram of the linked list representation of **T** appears in the following figure. Observe that each node is pictured with its three fields, and that the empty subtree is pictured by using x for null entries.
Binary Tree



Linked Representation of the Binary Tree

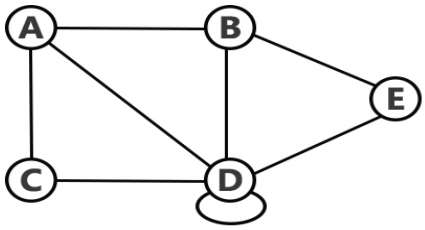


Q.5 Explain the following:
1. Adjacency Matrix
2. Adjacency List

Ans. 1. Adjacency Matrix:

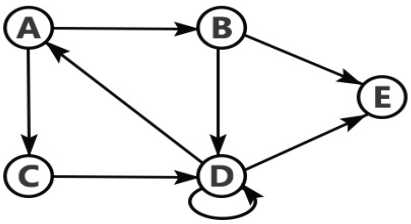
In this representation, graph can be represented using a matrix of size total number of vertices by total number of vertices. That means if a graph with 4 vertices can be represented using a matrix of 4X4 class. In this matrix, rows and columns both represents vertices. This matrix is filled with either 1 or 0. Here, 1 represents there is an edge from row vertex to column vertex and 0 represents there is no edge from row vertex to column vertex.

For example, consider the following undirected graph representation.



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

Directed graph representation:

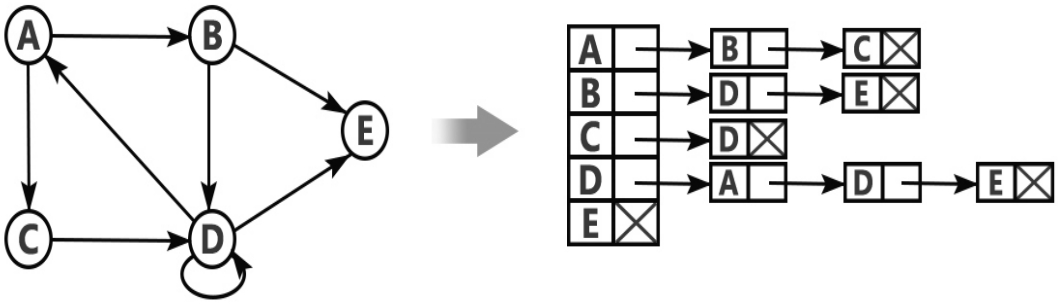


	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

2. Adjacency List :

In this representation, every vertex of graph contains list of its adjacent vertices.

For example, consider the following directed graph representation implemented using linked list.

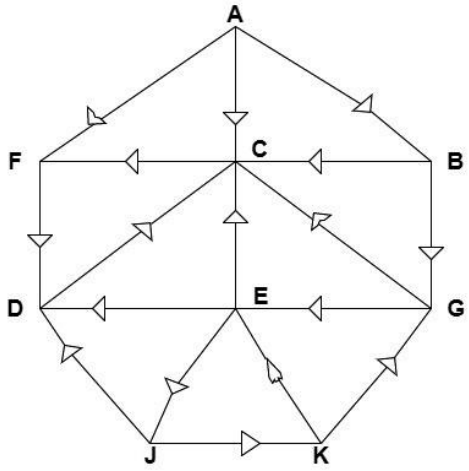


Q. 6 Explain the DFS algorithm with the help of example

Ans.It is the traversal technique which is used whenever it is possible to search the graph deeper. Given as input graph G = (V, E) and a source vertex S from where the searching start.

- First we travel or visit the starting node then we travel through each node along a path which begins at Stack 'S' i.e we visit a neighbour vertex of S and again a neighbour os S and so on.
- DFS works on both directed and undirected graph.
-

DFS(vertex i)
1. Initialize a stack 'S' of vertex 'w'
2. Push 'i' in the stack 'S'
3. Repeat step 4 to 5 while(stack S is not empty)
4. i = pop(S)
5. if(!visited[i]) then
   a) visited[i] = 1
   b) for each w adjacent to i
      if(!visited[w]) then push w in the stack S
6. Return



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

1. Initially, push J onto the stack as follows :
STACK : J

2. Pop J and push onto stack all the neighbours of J
STACK : D, K

3. Pop top element K and push onto stack all the neighbours of K
STACK : D, E, G

4. Pop top element G and push onto stack all the neighbours of G
STACK : D, E, C

5. Pop top element C and push onto stack all the neighbours of C
STACK : D, E, F

6. Pop top element F and push onto stack all the neighbours of F
STACK : D, E

7. Pop top element E and push onto stack all the neighbours of E
STACK : D

8. Pop D and push onto stack all the neighbours of D
STACK : ∅

The nodes which were printed are: J,K,G,C,F,E,D

Q.7 Explain BFS algorithm with the help of example?

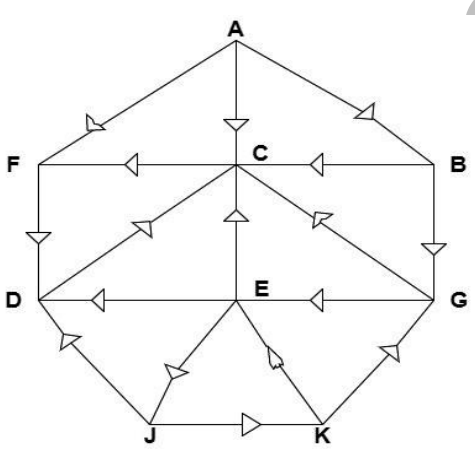
Ans. It is the technique which uses queue for traversing all the nodes of the graph.

- In this, first we take any node as the starting node then we take all the adjacent nodes to that starting node. Similarly, we take for all other adjacent nodes and so on.

- We maintain the status of visited nodes in an array so that no node can be traversed again. BFS also works on directed and undirected graphs.

BFS(vertex V)
1. Initialize a Queue Q
2. Add V to Queue Q
3. Set visited[V] = 1
4. Repeat step 5 to 6 while(Q is not empty)
5. Delete the element V from Q
6. for all vertices w adjacent from V
 a) if(!visited[w]) then
 set visited[w] = 1 and add vertex w to the Q
7. Return

Example of BFS



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

1. Consider the above figure. Initially, add A to Queue and add Null to ORIG array as follows :
QUEUE : A
ORIG : Ø

2. Remove A from Queue and add neighbours of A to Queue
QUEUE : F, C, B

ORIG : A

3. Remove first element F rom Queue and add neighbours of F to Queue
QUEUE : C, B, D
ORIG : A, F

4. Remove first element C rom Queue and add neighbours of C to Queue
QUEUE : B, D
ORIG : A, F, C

5. Remove first element B rom Queue and add neighbours of B to Queue
QUEUE : D, G
ORIG : A, F, C, B

6. Remove D rom Queue and add neighbours of D to Queue
QUEUE : G
ORIG : A, F, C, B, D

7. Remove G rom Queue and add neighbours of G to Queue
QUEUE : E
ORIG : A, F, C, B, D, G

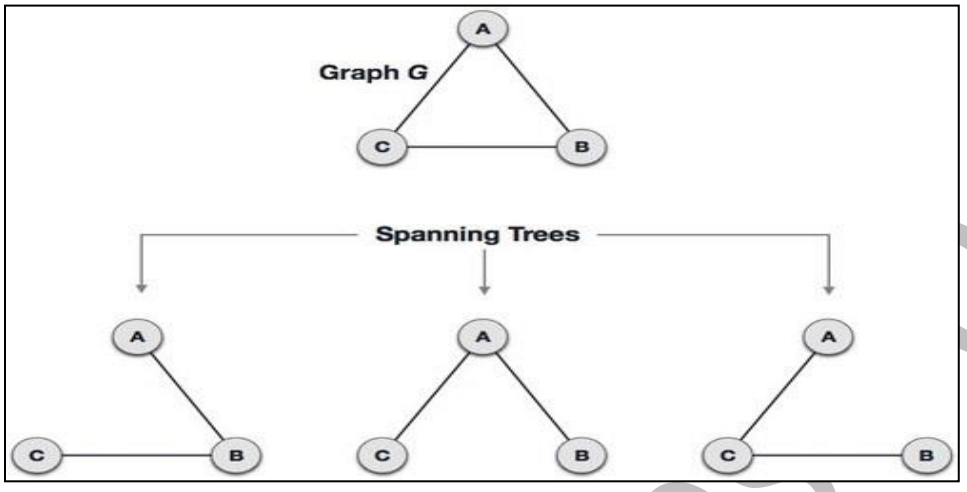
8. Remove E rom Queue and add neighbours of E to Queue
QUEUE : J
ORIG : A, F, C, B, D, G, E

9. Remove J rom Queue and add neighbours of J to Queue
QUEUE : Ø
ORIG : A, F, C, B, D, G, E, J

Q.7 Define the term spanning tree?

Ans. A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. Hence, a spanning tree does not have cycles and it cannot be disconnected..

By this definition, we can draw a conclusion that every connected and undirected Graph G has at least one spanning tree. A disconnected graph does not have any spanning tree, as it cannot be spanned to all its vertices.



We found three spanning trees off one complete graph. A complete undirected graph can have maximum nn-2 number of spanning trees, where n is the number of nodes. In the above addressed example, n is 3, hence 33−2 = 3 spanning trees are possible.

Q.8 Explain Minimum Spanning tree?

Ans. 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 small as possible. Such a tree which is formed with least possible cost is called Minimum Cost Spanning Tree and connects all the nodes in G.

Some of the methods for creating a minimum cost spanning tree for a weighted graph is -

- Prim's Algorithm
- Kruskal's Algorithm

Q.9 Write down the algorithm for Prim's spanning tree with the help of example.
Ans.
The following are the main 3 steps of the Prim's Algorithm:

1. Begin with any vertex which you think would be suitable and add it to the tree.

2. Find an edge that connects any vertex in the tree to any vertex that is not in the tree. Note that, we don't have to form cycles.
3. Stop when n - 1 edges have been added to the tree.

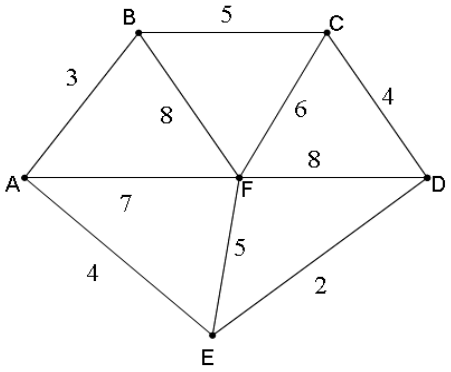
Q.10 Write down the algorithm for Kruskal's spanning tree with the help of example.

Ans Kruskal's algorithms is one of the types of greedy algorithms which is used to solve a minimal spanning tree (MST) by choosing the best possible choice at each step and then, this decision leads to the best over all solution.

The main steps of the Kruskal's Algorithm are as follows:

1. Arrange the edges by weight: least weight first and heaviest last.
2. Choose the lightest not examined edge from the diagram. Add this chosen edge to the tree, only if doing so will not make a cycle.
3. Stop the process whenever n - 1 edges have been added to the tree.

For example :

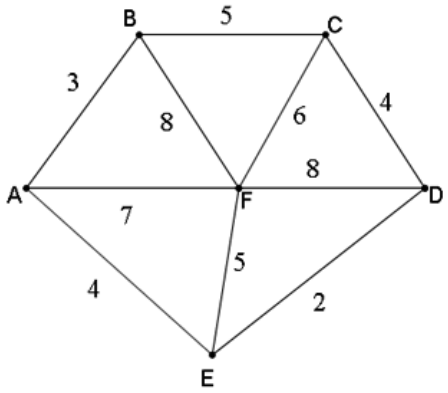


Solution:
The list of edges in order of their weights or sizes would be as follows:

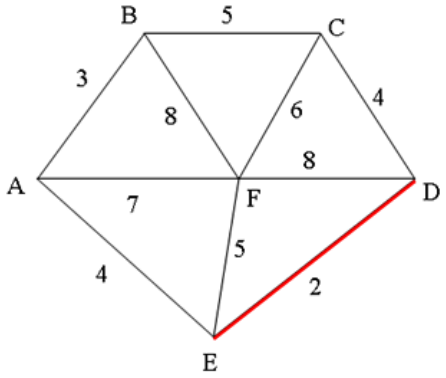
Edge	Weight	Edge	Weight
ED	2	**EF**	5
AB	3	**CF**	6
AE	4	**AF**	7
CD	4	**BF**	8
BC	5	**CF**	6

The various iterations would be as follows:

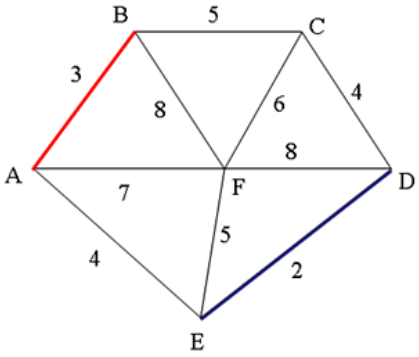
Iteration 0:



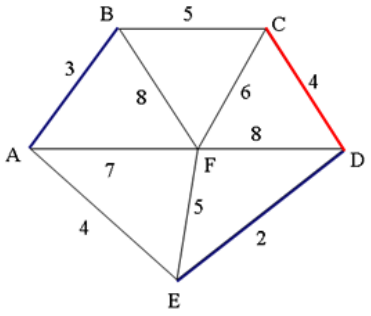
Iteration 1:



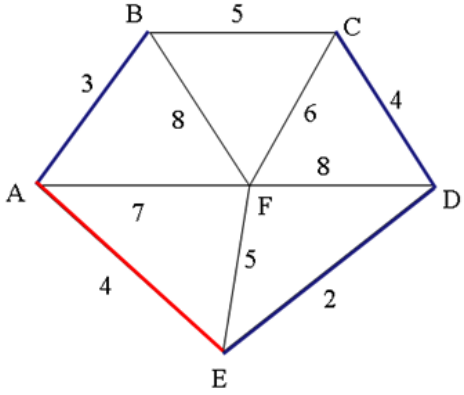
Iteration 2:



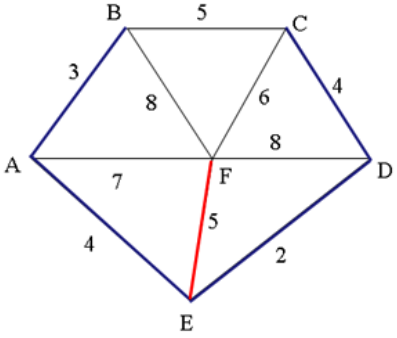
Iteration 3:



Iteration 4:



Iteration 5:



All the vertices have been connected now, hence, the last iteration number 5, gives us the optimal solution, and the minimum length would be the sum of all weights given to these edges, as 2 + 3 + 4 + 4 + 5 = 18 is the length of the shortest path by applying Kruskal's Algorithm.

That is, the solution is

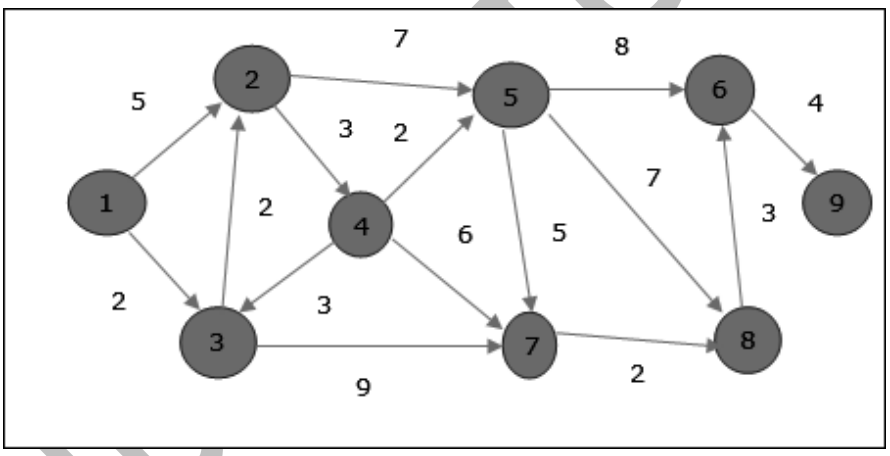
ED 2
AB 3
CD 4
AE 4
EF 5,
with Total weight of tree equal to 18

Q.11 Write down the Dijikstra algorithm for finding out the shortest path in the graph.

Ans. Dijkstra's algorithm solves the single-source shortest-paths problem on a directed weighted graph $G = (V, E)$, where all the edges are non-negative (i.e., $w(u, v) \geq 0$ for each edge $(u, v) \in E$).

The complexity of this algorithm is fully dependent on the implementation of Extract-Min function. If extract min function is implemented using linear search, the complexity of this algorithm is $O(V^2 + E)$.

Let us consider vertex ***1*** and ***9*** as the start and destination vertex respectively. Initially, all the vertices except the start vertex are marked by ∞ and the start vertex is marked by ***0***.



Vertex	Initial	Step1 V_1	Step2 V_3	Step3 V_2	Step4 V_4	Step5 V_5	Step6 V_7	Step7 V_8	Step8 V_6
1	0	0	0	0	0	0	0	0	0
2	∞	5	4	4	4	4	4	4	4
3	∞	2	2	2	2	2	2	2	2
4	∞	∞	∞	7	7	7	7	7	7
5	∞	∞	∞	11	9	9	9	9	9
6	∞	∞	∞	∞	∞	17	17	16	16
7	∞	∞	11	11	11	11	11	11	11
8	∞	∞	∞	∞	∞	16	13	13	13
9	∞	∞	∞	∞	∞	∞	∞	∞	20

Hence, the minimum distance of vertex ***9*** from vertex ***1*** is ***20***. And the path is

1→ 3→ 7→ 8→ 6→ 9