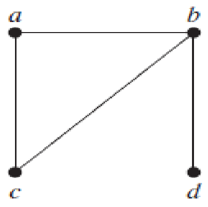
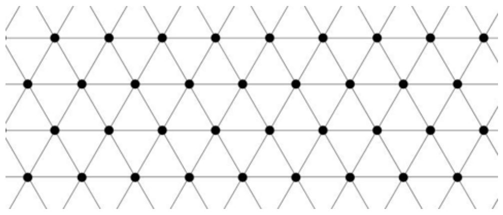
PARUL UNIVERSITY
FACULTY OF ENGINEERING & TECHNOLOGY
Department of Applied Science & Humanities
3rd Semester B. Tech (CSE, IT) 2024-25
Discrete Mathematics (303191202)
UNIT-5 GRAPHS and TREES – Lecture Notes

GRAPHS AND THEIR PROPERTIES

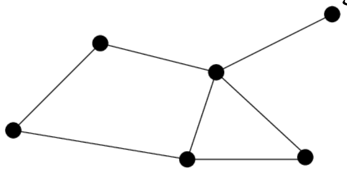
Definition:
Graph:
A ***graph*** *G* = *(V ,E)* consists of *V* , a nonempty set of ***vertices*** **(or *nodes*)** and *E*, a set of ***edges*.** Each edge has either one or two vertices associated with it, called its ***endpoints***. An edge is said to ***connect*** its endpoints.
For example: A graph with vertex set V={a,b,c,d} and with 4 edges (a,b), (b,d), (b,c), (a,c) is as follows:



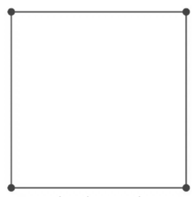
Definition:
Infinite Graph:
The set of vertices *V* of a graph *G* may be infinite. A graph with an infinite vertex set or an infinite number of edges is called an **infinite graph.**
For example:



Definition:
Finite Graph:
A graph with a finite vertex set and a finite edge set is called a **finite graph**.
For example: In following figure finite vertices connected with edges.

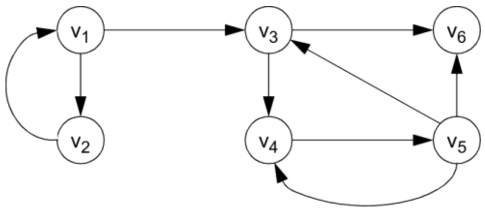


Definition:
Simple Graph:
A graph in which each edge connects two different vertices and where no two edges connect the same pair of vertices is called a **simple graph.**
*For example***:**

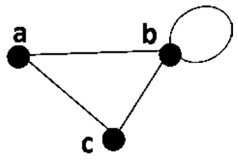

simple graph

Definition:
Multiple Edges:
Graphs that may have **multiple edges** connecting the same vertices are called **multigraphs**.

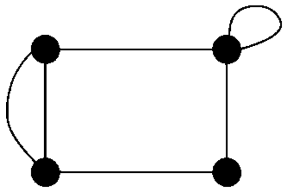
For example:



Definition:
Loops:
The edges that connect a vertex to itself. Such edges are called **loops.**
For example:

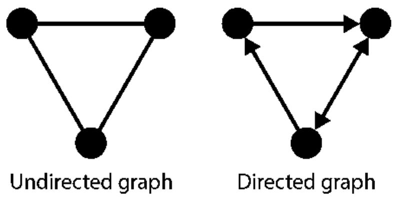


Definition:
Pseudo Graphs
Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself are called **pseudo graphs**.
For example:



Definition:
Directed Graph
A **directed graph (or digraph)** *(V ,E)* consists of a nonempty set of vertices *V* and a set of **directed edges (or arcs)** *E*. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (*u, v*) is said to **start** at *u* and **end** at *v*.

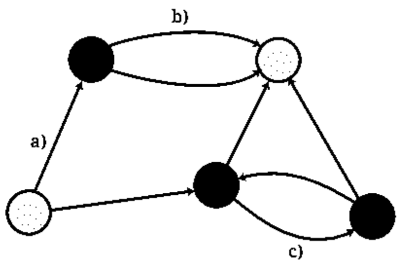
Definition:
Undirected Graph
If direction is not assign to edges in a graph then such graph is call **undirected graph**.
For example:



Definition:
Simple directed Graph:
A directed graph has no loops and has no multiple directed edges, it is called a **simple directed graph**.

Definition:
Multiple Directed Edges:
Directed graphs that may have **multiple directed edges** from a vertex to a second (possibly the same) vertex are used to model the networks.

For example:

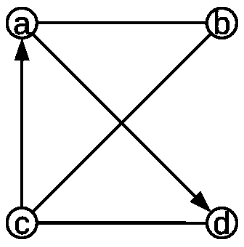


Definition:

Mixed Graph:

A graph with both directed and undirected edges is called a **mixed graph**.

For example:

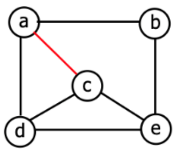
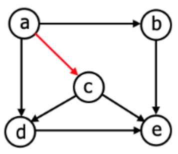
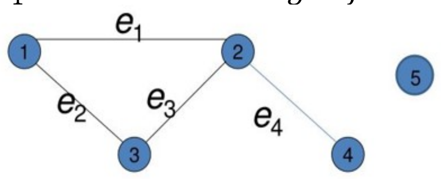


Definition:

Adjacent and incident vertices:

Two vertices u and v in an undirected graph G are called **adjacent (or neighbours)** in G if u and v are endpoints of an edge e of G. Such an edge e is called **incident** with the vertices u and v and e is said to **connect** u and v.

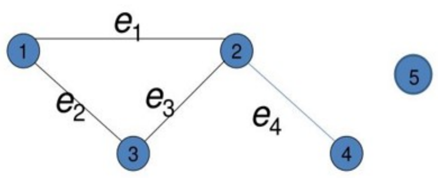
For example:

 Vertex a is adjacent to c and vertex c is adjacent to a **Fig. A**	 Vertex c is adjacent to a, but vertex a is NOT adjacent to c **Fig. B**	e_1 *is the incident edge of* 1  **Fig. C**

Definition:

Degree of a vertex:

The **degree of a vertex** in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex v is denoted by deg(v).



deg(1)=2; deg(2)=3; deg(3)=2; deg(4)=1; deg(5)=0

Definition:

Isolated vertex:

A vertex of degree zero is called **isolated**. **[Summer 2021-22], [Winter 2023-24], [Summer 2023-24]**

For Example.: "In above **fig.-C** vertex-5 is an Isolated Vertex."

Definition:

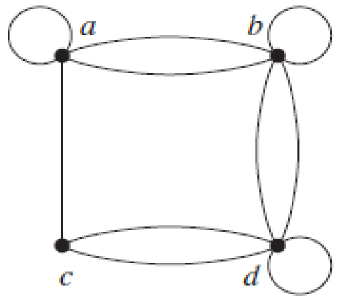
Pendent Vertex:

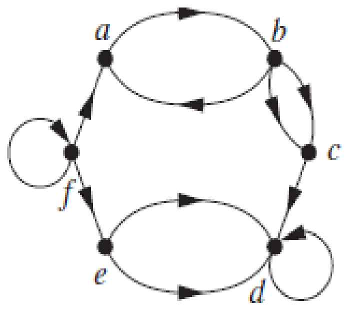
A vertex is **pendant** if and only if it has degree one. **[Winter 2023-24], [Summer 2023-24]**
For Example.: "In above **fig.-C** vertex-4 is an Isolated Vertex."

Short Summary:

Type	*Edges*	*Multiple Edges Allowed?*	*Loops Allowed?*
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed graph	Directed and undirected	Yes	Yes

Example: Determine whether the graph shown has directed or undirected edges, whether it has multiple edges, and whether it has one or more loops. Use your answers to determine the type of graph from above Table.





The Handshaking theorem:

$$Let\ G(V,E) be\ a\ undirected\ graph\ with\ m\ edges. Then\ \mathbf{2m} = \sum_{v\epsilon V} \mathbf{deg\, v}$$

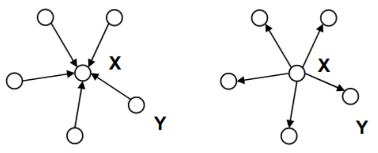
$\rightarrow$ *In any graph $G(V,E)$ the sum of degrees of all the vertices is equal to the twice of no. of edges in that graph.*

Example: How many edges are there in a graph with 10 vertices each of degree six?
***Solution**:* Because the sum of the degrees of the vertices is 6 · 10 = 60, it follows that 2*m* = 60 where *m* is the number of edges. Therefore, *m* = 30.

Example: Give the statement of THE HANDSHAKING THEOREM and use it to find the number of vertices If simple graph G has 24 edges and degree of each vertex is 4. **[Summer 2021-22]**

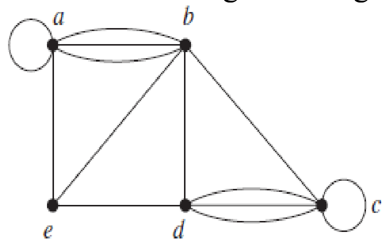
Definition:
Initial and end vertices:
When *(u, v)* is an edge of the graph *G* with directed edges, *u* is said to be ***adjacent*** *to v* and *v* is said to be ***adjacent from*** *u*. The vertex *u* is called the ***initial vertex*** of *(u, v)*, and *v* is called the ***terminal* or *end vertex*** of *(u, v)*. The initial vertex and terminal vertex of a loop are the same.

Definition:
In degree and Out degree:
In a graph with directed edges the ***in-degree*** *of a vertex v*, denoted by deg−*(v)*, is the number of edges with *v* as their terminal vertex. The ***out-degree*** *of v*, denoted by deg+*(v)*, is the number of edges with *v* as their initial vertex.

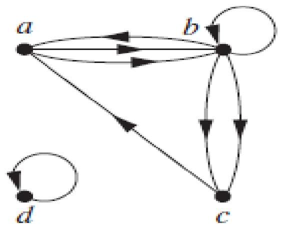


indegree outdegree

Example:
Find the number of vertices, the number of edges, and the degree of each vertex in the given undirected graph. Identify all isolated and pendant vertices. Find the sum of the degrees of the vertices of following graph and verify that it equals twice the number of edges in the graph.

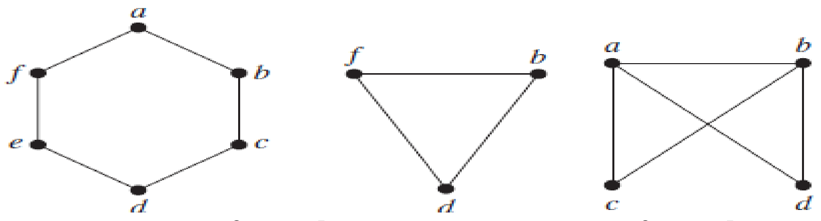


Example:
Determine the number of vertices and edges and find the in-degree and out-degree of each vertex for the given directed multigraph.



<u>Types of Graphs:</u>

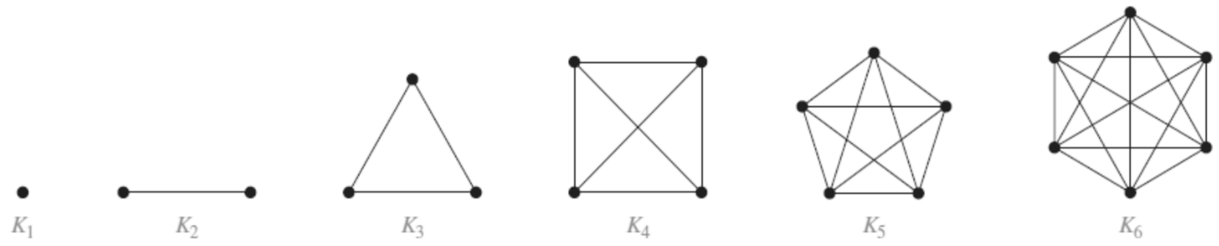
Definition:
Regular graphs: A simple graph is called **regular** if every vertex of this graph has the same degree. A regular graph is called ***n*-regular** if every vertex in this graph has degree *n*.



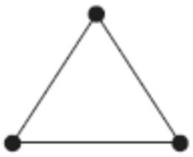
2-regular **3-regular**

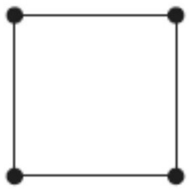
Example: For which values of n are the following graphs regular? (i) K_n (ii) C_n

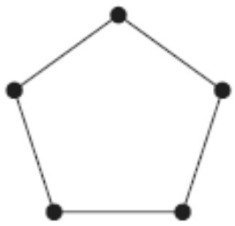
Definition:
Complete Graphs:
A **complete graph on *n* vertices**, denoted by K_n, is a simple graph that contains exactly one edge between each pair of distinct vertices.

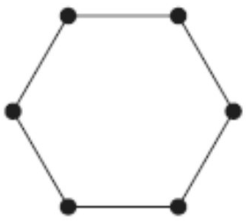


Definition:
Cycles :
A **cycle *Cn***, $n \geq 3$, consists of *n* vertices *v*1*, v*2*, . . . , vn* and edges {*v*1*, v*2),{*v*2*, v*3}*, . . . ,* {*vn*−1*, vn*}, and {*vn, v*1}. The cycles C_3, C_4, C_5 & C_6 are displayed in figure

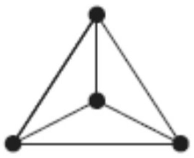

C_3

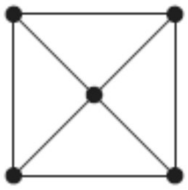

C_4

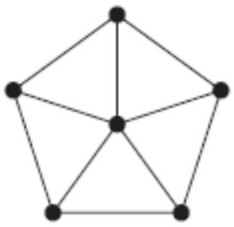

C_5

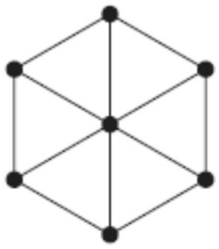

C_6

Definition:
Wheels:
We obtain a **wheel** W_n when we add an additional vertex to a cycle C_n, for n ≥ 3, and connect this new vertex to each of the *n* vertices in C_n, by new edges. The wheels W_3, W_4, W_5 & W_6 are displayed in Figure

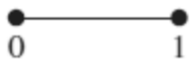

W_3


W_4

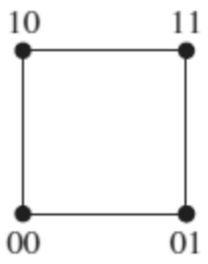

W_5


W_6

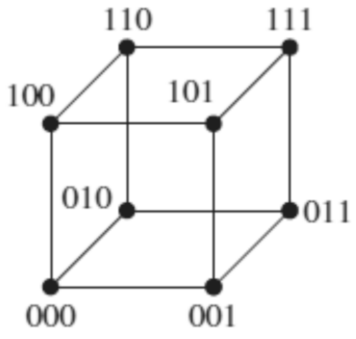
Definition:
***n*-Cubes:**
An ***n*-dimensional hypercube**, or ***n*-cube**, denoted by Q_n, is a graph that has vertices representing the 2*n* bit strings of length *n*. Two vertices are adjacent if and only if the bit strings that they represent differ in exactly one bit position.



Q_1



Q_2

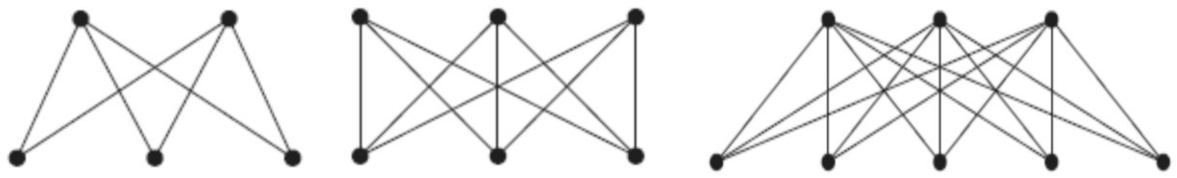


Q_3

Definition:
Bipartite Graph:
A simple graph *G* is called ***bipartite*** if its vertex set *V* can be partitioned into two disjoint sets V_1 and V_2 such that every edge in the graph connects a vertex in V_1 and a vertex in V_2 (so that no edge in *G* connects either two vertices in V_1 or two vertices in V_2). When this condition holds, we call the pair (V_1, V_2) a ***bipartition*** of the vertex set *V* of *G*.

Example: Is C_6 and K_3 a bipartite graph or not?

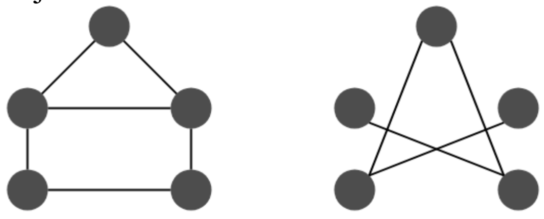
Definition:
Complete Bipartite Graphs:
A **complete bipartite graph *Km,n*** is a graph that has its vertex set partitioned into two subsets of *m* and *n* vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset. The complete bipartite graphs $K_{2,3}$, $K_{3,3}$, $K_{3,5}$, and $K_{2,6}$ are displayed in Figure



$K_{2,3}$ $K_{3,3}$ $K_{3,5}$

Example:[Summer-2021-22]
Define Complete Bipartite Graphs also draw $K_{2,3}$, $K_{3,3}$ and $K_{3,5}$

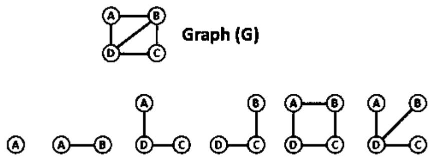
Definition:
Complementary graph:
The **complementary graph** $\overline{G}$ of a simple graph *G* has the same vertices as *G*. Two vertices are adjacent in $\overline{G}$ if and only if they are not adjacent in *G*.



Graph G Complement Graph $\overline{G}$

Example: Describe each of these graphs. (i) $\overline{K_n}$ (ii) $\overline{C_n}$

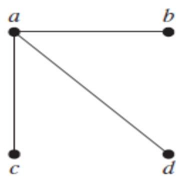
Definition:
Subgraph:
A **subgraph** of a graph *G = (V ,E)* is a graph *H = (W, F)*, where *W* ⊆ *V* and *F* ⊆ *E*. A subgraph *H* of *G* is a **proper subgraph** of *G* if *H* = *G*.



Graph (G)

Subgraphs of Graph (G)

Example: Draw all subgraphs of this graph.



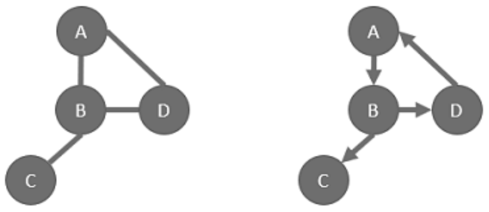
Definition:
Adjacency Matrix (for undirected graph):
Suppose that *G = (V ,E)* is a simple graph where |*V*| = *n*. The **adjacency matrix A** (or A_G) of *G*, with respect to this listing of the vertices, is the *n* x *n* zero–one matrix with 1 as its (i, j)th entry when v_i and

v_j are adjacent, and 0 as its (i, j)th entry when they are not adjacent. In other words, if its adjacency matrix is $\mathbf{A} = [a_{ij}]$, then

$$a_{ij} = \begin{cases} 1, & if\,\{v_i, v_j\} is\ an\ edge\ of\ G \\ 0, & otherwise \end{cases}$$

Definition:
Adjacency Matrix (for Directed graph):
An **adjacency matrix** is defined as follows: Let G be a graph with "n" vertices that are assumed to be ordered from v_1 to v_n. The n x n matrix A, in which a_{ij}= 1 if there exists a path from v_i to v_j , a_{ij} = 0 otherwise is called an adjacency matrix.
For example:

1.

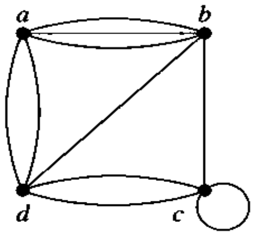


Ending Points

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 0 & 1 \\ B & 1 & 0 & 1 & 1 \\ C & 0 & 1 & 0 & 0 \\ D & 1 & 1 & 0 & 0 \end{array} \qquad \text{Starting Points} \quad \begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 0 & 0 \\ B & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 0 & 0 \\ D & 1 & 0 & 0 & 0 \end{array}$$

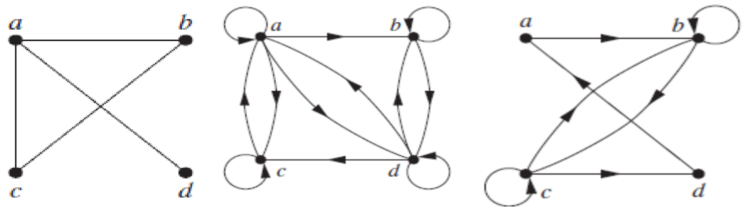
Undirected Directed

2.



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

Example: Use an adjacency matrix to represent the graph shown in Figure



Example: Draw a graph with the adjacency matrix with respect to the ordering of vertices *a, b, c, d*.

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

Definition:
Incidence matrix (for undirected graph):

Let $G = (V, E)$ be an undirected graph. Suppose that $v_1, v_2, \ldots, v_n$ are the vertices and $e_1, e_2, \ldots, e_m$ are the edges of G. Then the incidence matrix with respect to this ordering of V and E is the $n \times m$ matrix **M** = [*mij*], where

$$m_{ij} = \begin{cases} 1, & \text{when } e_i \text{ is incident with vertex } v_i \\ 0, & \text{otherwise} \end{cases}$$

Definition:

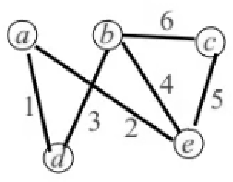
Incidence matrix (for directed graph): The **incidence matrix** of a **directed graph** is a n × m matrix B where n and m are the number of vertices and edges respectively such that,

$$b_{ii} = \begin{cases} -1, & \text{when the edge } e_i \text{ leaves vertex } v_i \\ 1, & \text{if it enters vertex } v_i \\ 0, & \text{otherwise} \end{cases}$$

Example: Represent the graph shown in below Figure with an incidence matrix.

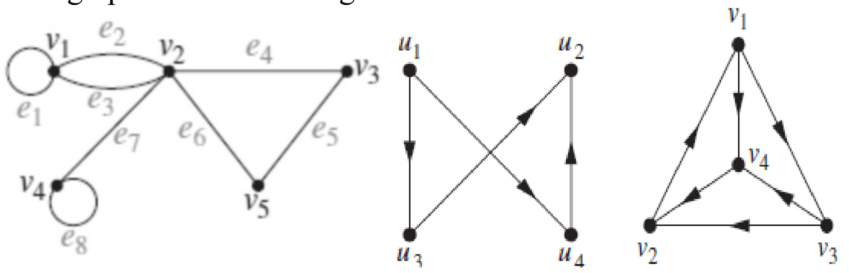
Solution:

The Incidence Matrix of given figure is



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

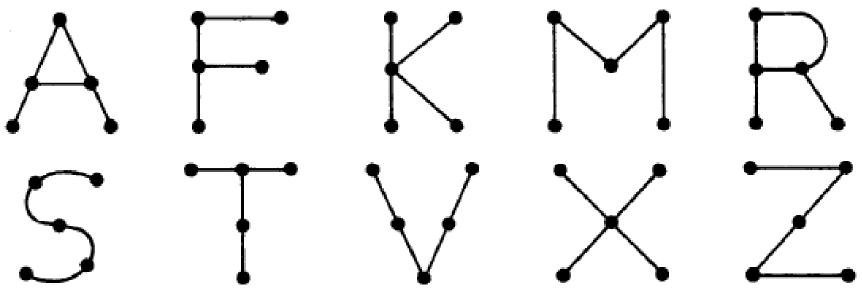
Example: Represent the graph shown in the figure with an incidence matrix



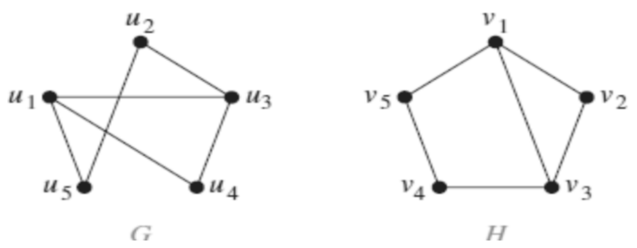
Definition:

Isomorphism

Graphs $G(V, E)$ and $G(V^*, E^*)$ are said to be *isomorphic* if there exists a one-to-one correspondence $f: V \to V^*$ such that $\{u, v\}$ is an edge of G if and only if $\{f(u), f(v)\}$ is an edge of G^*. Normally, we do not distinguish between isomorphic graphs (even though their diagrams may "look different"). Figure 8-6 gives ten graphs pictured as letters. We note that *A* and *R* are isomorphic graphs. Also, *F* and *T* are isomorphic graphs, *K* and *X* are isomorphic graphs and *M*, *S*, *V* , and *Z* are isomorphic graphs.



Example: Determine whether the graphs *G* and *H* shown in Figure are isomorphic.



Solution:
Both *G* and *H* have five vertices and six edges, both have two vertices of degree three and three vertices of degree two, and both have a simple circuit of length three, a simple circuit of length four, and a simple circuit of length five. Because all these isomorphic invariants agree, *G* and *H* may be isomorphic.

Example: Show that the following graphs displayed in Figures a and b are isomorphic in both the cases.

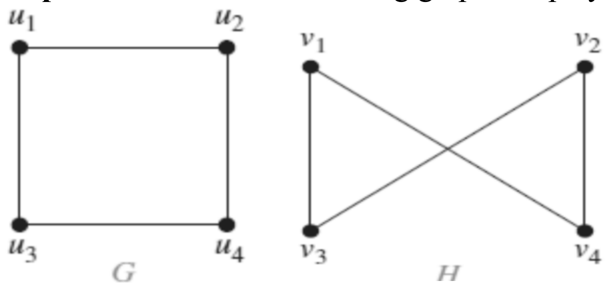


Figure-a

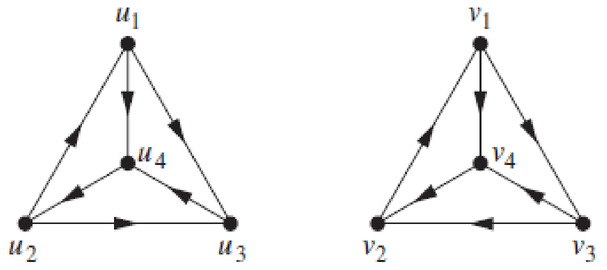
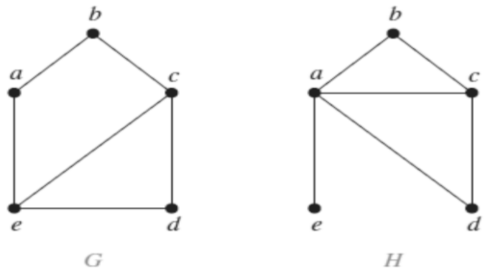
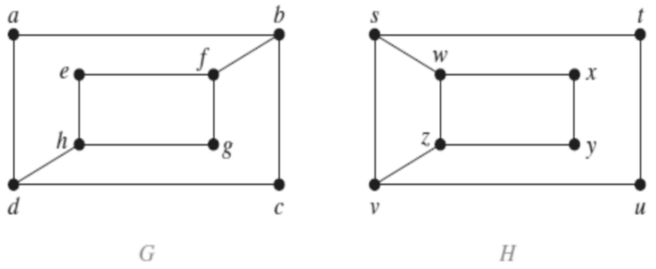


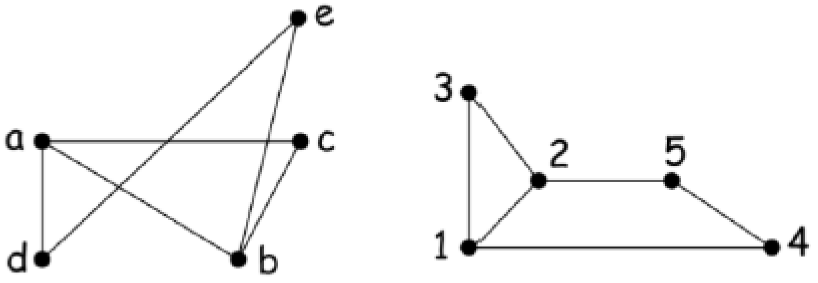
Figure-b

Example: Show that the following graphs G=(V,E) and H=(W, F) are not isomorphic**.**





Example: Determine whether the following pairs of graphs are isomorphic or not? **[Summer-2021-22]**

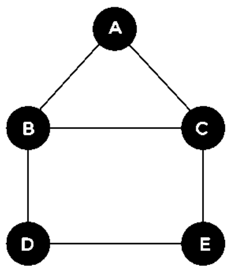


Definitions: Walk, Path, Circuit and Cycle:

Walk:
A walk can be defined as a sequence of edges and vertices of a graph. When we have a graph and traverse it, then that traverse will be known as a walk. In a walk, there can be repeated edges and vertices. The number of edges which is covered in a walk will be known as the Length of the walk. In a graph, there can be more than one walk.
So for a walk, the following two points are important, which are described as follows:

- Edges can be repeated
- Vertex can be repeated

For example: In this example, we have a graph, which is described as follows:



There can be many walks, but some of them are described as follows:

1. A, B, C, E, D (length of walk = 4)
2. D, B, A, C, E, D, C (length of walk = 7)
3. E, C, B, A, C, E, D (length of walk= 6)

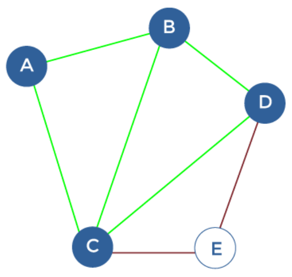
Types of Walks
There are two types of the walk, which are described as follows:

1. Open walk
2. Closed walk

Definitions:

Open Walk:
A walk will be known as an open walk in the graph theory if the vertices at which the walk starts and ends are different. That means for an open walk, the starting vertex and ending vertex must be different. In an open walk, the length of the walk must be more than 0.

Closed Walk:
A walk will be known as a closed walk in the graph theory if the vertices at which the walk starts and ends are identical. That means for a closed walk, the starting vertex and ending vertex must be the same. In a closed walk, the length of the walk must be more than 0.



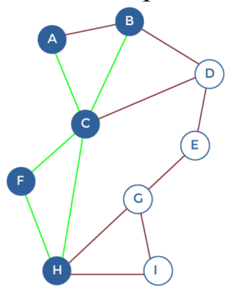
In this graph, there is also a closed walk and an open walk, which are described as follows:

Closed walk = A, B, C, D, E, C, A

Open walk = A, B, C, D, E, C

Trails
A trail can be described as an open walk where no edge is allowed to repeat. In the trails, the vertex can be repeated.
So for a trail, the following point is important, which is described as follows:

- Vertex can be repeated



In this graph, there is a tail and closed tail, which is described as follows:

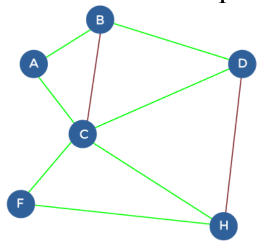
Tail = A, C, H, F, C, B

Closed tail = A, C, H, F, C, B, A

Circuit
A circuit can be described as a closed walk where no edge is allowed to repeat. In the circuit, the vertex can be repeated. A closed trail in the graph theory is also known as a circuit.
So for a circuit, the following two points are important, which are described as follows:

- Edges cannot be repeated

- Vertex can be repeated

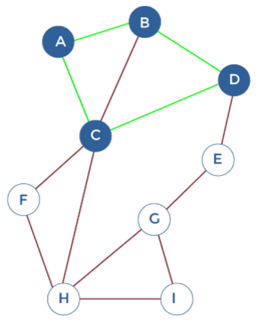


In the above graph, there is a circuit, which is described as follows:

Circuit: A, B, D, C, F, H, C, A

Cycle:
A closed path in the graph theory is also known as a Cycle. A cycle is a type of closed walk where neither edges nor vertices are allowed to repeat. There is a possibility that only the starting vertex and ending vertex are the same in a cycle.
So for a cycle, the following two points are important, which are described as follows:

- Edges cannot be repeated
- Vertex cannot be repeated

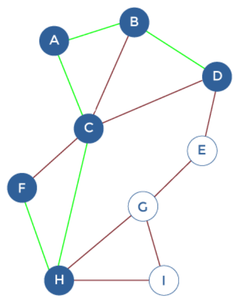


In the above graph, there is a cycle, which is described as follows:

Cycle: A, B, D, C, A

Path:
A path is a type of open walk where neither edges nor vertices are allowed to repeat. There is a possibility that only the starting vertex and ending vertex are the same in a path. In an open walk, the length of the walk must be more than 0.
So for a path, the following two points are important, which are described as follows:

- Edges cannot be repeated
- Vertex cannot be repeated



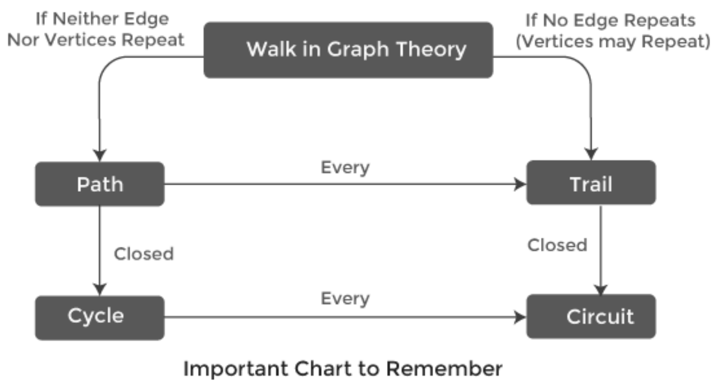
In the above graph, there is a path, which is described as follows:

Path: F, H, C, A, B, D

Remarks

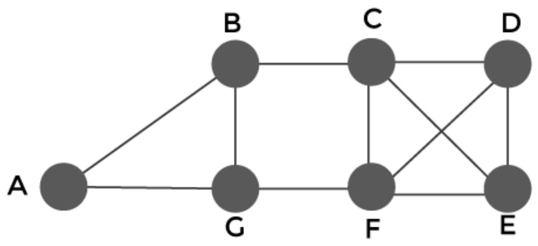
- Every path can be a trail, but it is not possible that every trail is a path.
- Every cycle can be a circuit, but it is not important that every circuit is a cycle.

The above definitions can be easily remembered with the help of following chart:


If Neither Edge Nor Vertices Repeat
Walk in Graph Theory
If No Edge Repeats (Vertices may Repeat)
Path
Every
Trail
Closed
Closed
Cycle
Every
Circuit
Important Chart to Remember

]

Example: Find all possible sequence of vertices determining walks.



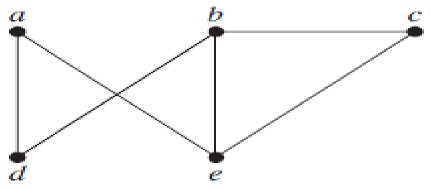
Solution:

The sequence is described below

1. A, B, G, F, C, D
2. B, G, F, C, B, G, A
3. C, E, F, C
4. C, E, F, C, E
5. A, B, F, A
6. F, D, E, C, B

Example: Does each of these lists of vertices form a path in the following graph? Which paths are simple? Which are circuits? What are the lengths of those that are paths?

a) *a, e, b, c, b* **b)** *a, e, a, d, b, c, a* **c)** *e, b, a, d, b, e* **d)** *c, b, d, a, e, c*



Definition:

- An undirected graph is called **connected** if there is a path between every pair of distinct vertices of the graph. An undirected graph that is not **connected** is called **disconnected.**
- A **connected component** of a graph *G* is a connected subgraph of *G* that is not a proper subgraph of another connected subgraph of *G*.

- Sometimes the removal from a graph of a vertex and all incident edges produces a subgraph with more connected components. Such vertices are called **cut vertices** (or **articulation points**).
- The removal of a cut vertex from a connected graph produces a subgraph that is not connected. Analogously, an edge whose removal produces a graph with more connected components than in the original graph is called a **cut edge** or **bridge**.

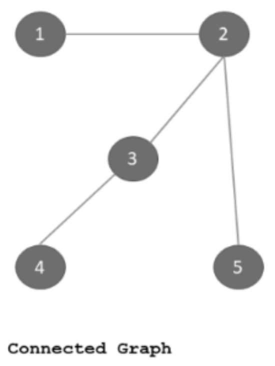


Figure-1

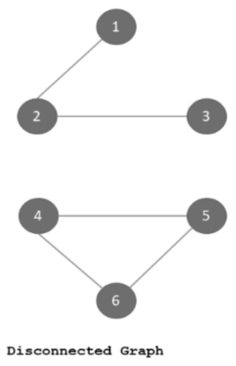
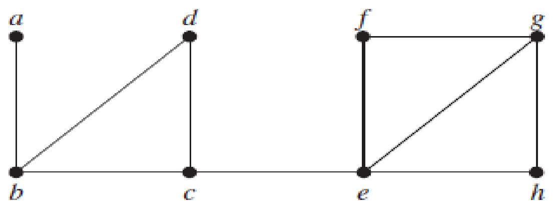


Figure-2

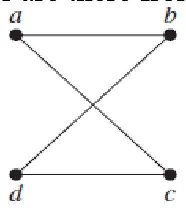
Example: Find the cut vertices and cut edges in the graph *G* shown in Figure.



Solution:
The cut vertices of *G*1 are *b*, *c*, and *e*. The removal of one of these vertices (and its adjacent edges) disconnects the graph. The cut edges are {*a, b*} and {*c, e*}. Removing either one of these edges disconnects *G*1.

- A directed graph is **strongly connected** if there is a path from *a* to *b* and from *b* to *a* whenever *a* and *b* are vertices in the graph.
- A directed graph is **weakly connected** if there is a path between every two vertices in the underlying undirected graph.

Example: How many paths of length four are there from *a* to *d* in the simple graph *G* in Figure?



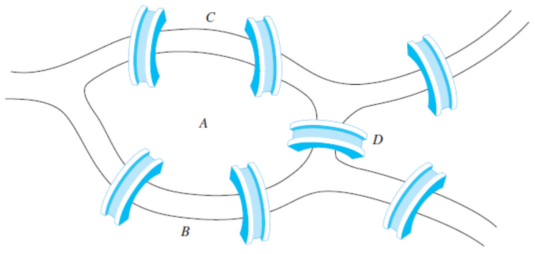
Solution:
The adjacency matrix of *G* (ordering the vertices as *a, b, c, d*) is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

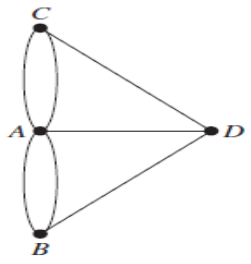
The adjacency matrix of *G* (ordering the vertices as *a, b, c, d*) is there are exactly eight paths of length four from *a* to *d*. By inspection of the graph, we see that *a, b, a, b, d*; *a, b, a, c, d*; *a, b, d, b, d*; *a, b, d, c, d*; *a, c, a, b, d*; *a, c, a, c, d*; *a, c, d, b, d*; and *a, c, d, c, d* are the eight paths of length four from *a* to *d*.

$$A^4 = \begin{bmatrix} 8 & 0 & 0 & 8 \\ 0 & 8 & 8 & 0 \\ 0 & 8 & 8 & 0 \\ 8 & 0 & 0 & 8 \end{bmatrix}$$

EULERIAN AND HAMILTONIAN PATH:
The Swiss mathematician Leonhard Euler solved this problem. His solution, published in 1736, may be the first use of graph theory. Euler studied this problem using the multigraph obtained when the four regions are represented by vertices and the bridges by edges. This multigraph is shown in Figure



Seven bridges of koinsberg



Multigraph Model of the Town of Königsberg

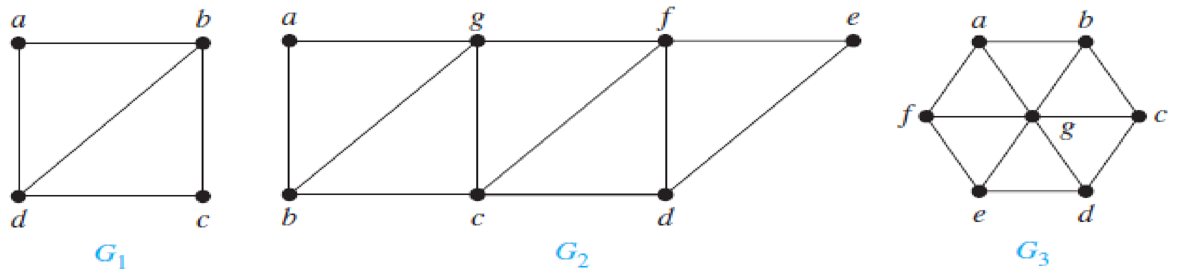
The problem of traveling across every bridge without crossing any bridge more than once can be rephrased in terms of this model. The question becomes: Is there a simple circuit in this multigraph that contains every edge?

Definition:
Euler Circuit
An ***Euler circuit*** in a graph *G* is a simple circuit containing every edge of *G*. An ***Euler path*** in *G* is a simple path containing every edge of *G*.
-In other words: A path that travels through every edge of a connected graph once and only once and starts and ends at different vertices. **An Euler path that starts and ends at the same vertex**
For example:

Theorem: [Without Proof] A connected multigraph with at least two vertices has an Euler circuit if and only if each of its vertices has even degree.

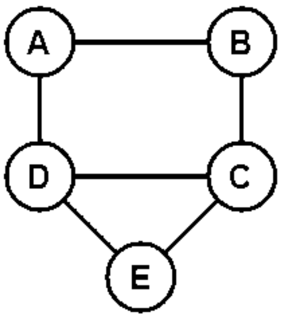
Theorem: [Without Proof] A connected multigraph has an Euler path but not an Euler circuit if and only if it has exactly two vertices of odd degree.

Example: Which graphs shown in Figure have an Euler path?



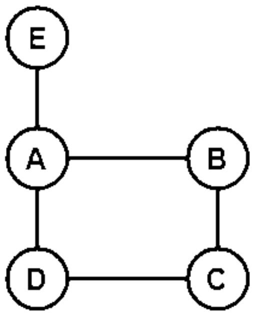
Definition:
Hamiltonian Path
A simple path in a graph *G* that passes through every vertex exactly once is called a ***Hamilton path***, and a simple circuit in a graph *G* that passes through every vertex exactly once is called a ***Hamilton circuit.***

Example : In the following graph with 5 nodes, determine whether this graph contains a Hamiltonian path or not?



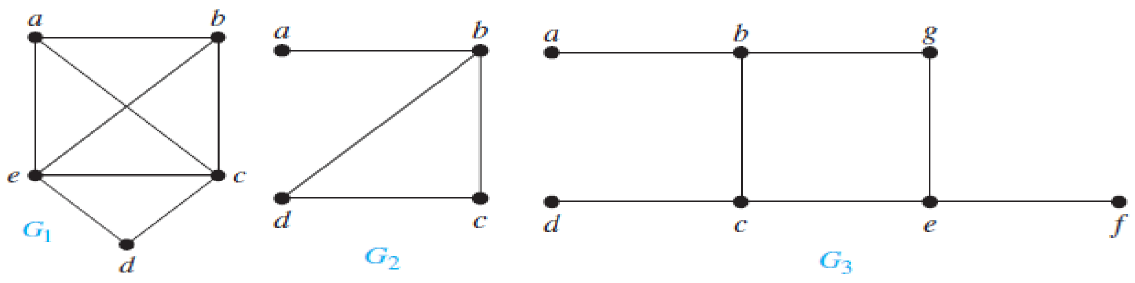
Solution:
In the above graph, we can see that when we start from A, then we can go to B, C, D, and then E. So this is the path that contains all the vertices (A, B, C, D, and E) only once, and there is no repeating edge. That's why we can say that this graph has a Hamiltonian path, which is described as follows:
Hamiltonian path = ABCDE

Example: In the following graph with 5 nodes, determine whether this graph contains a Hamiltonian path.



Solution:
In the above graph, we can see that when we start from the E, then we can go to A, B, C, and then D. So this is the path that contains all the vertices (A, B, C, D, and E) only once, and there is no repeating edge. That's why we can say that this graph has a Hamiltonian path, which is described as follows:
Hamiltonian path = EABCD

Example: Which of the simple graphs in Figure have a Hamilton circuit or, if not, a Hamilton path?

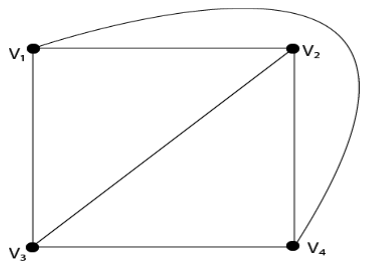


Remarks:

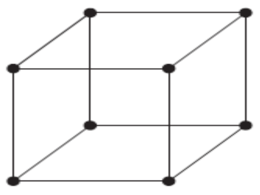
- **DIRAC'S THEOREM** [Without Proof]: If G is a simple graph with n vertices with $n \geq 3$ such that the degree of every vertex in G is at least $n/2$, then G has a Hamilton circuit.
- **ORE'S THEOREM** [Without Proof]: If G is a simple graph with n vertices with $n \geq 3$ such that $\deg(u) + \deg(v) \geq n$ for every pair of nonadjacent vertices u and v in G, then G has a Hamilton circuit.

Definition:
Planar Graphs: A graph is called ***planar*** if it can be drawn in the plane without any edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a *planar representation* of the graph.

Example: The graph shown in fig is planar graph.



Example: Is Q_3, shown in Figure, planar?



Definition:

Graph Colouring

A ***coloring*** of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.

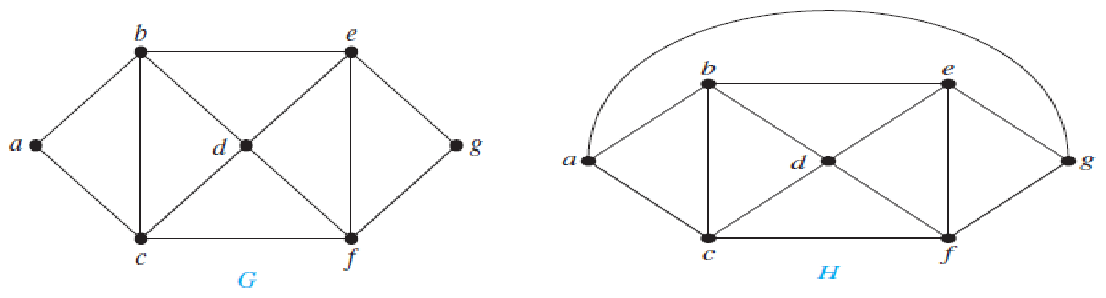
Definition:

Chromatic Number

The ***chromatic number*** of a graph is the least number of colors needed for a coloring of this graph. The chromatic number of a graph *G* is denoted by $\chi(G)$. (Here χ is the Greek letter *chi*.)

THE FOUR-COLOR THEOREM: [Without Proof] The chromatic number of a planar graph is no greater than four.

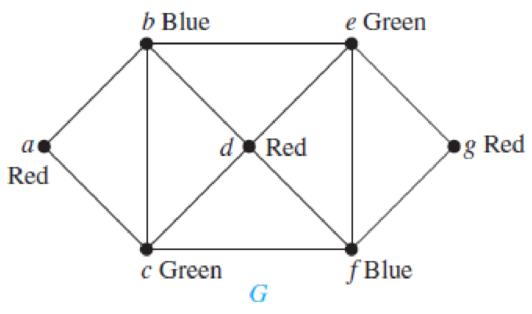
Example: What are the chromatic numbers of the graphs *G* and *H* shown in Figure?

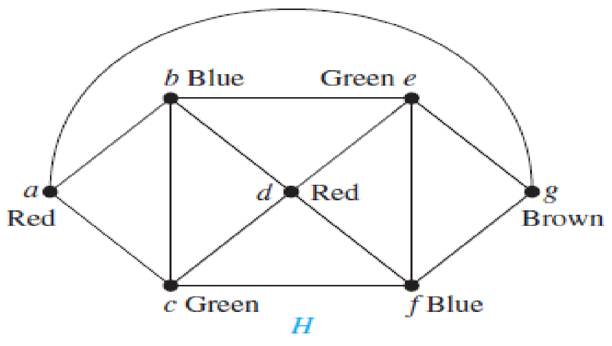


Solution:

The chromatic number of *G* is at least three, because the vertices *a, b,* and *c* must be assigned different colors. To see if *G* can be colored with three colors, assign red to *a*, blue to *b*, and green to *c*. Then, *d* can (and must) be colored red because it is adjacent to *b* and *c*. Furthermore, *e* can (and must) be colored green because it is adjacent only to vertices colored red and blue, and *f* can (and must) be colored blue because it is adjacent only to vertices colored red and green. Finally, *g* can (and must) be colored red because it is adjacent only to vertices colored blue and green. This produces a coloring of *G* using exactly three colors. Below Figure displays such a coloring.

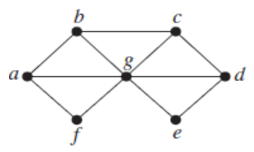
The graph *H* is made up of the graph *G* with an edge connecting *a* and *g*. Any attempt to color *H* using three colors must follow the same reasoning as that used to color *G*, except at the last stage, when all vertices other than *g* have been colored. Then, because *g* is adjacent (in *H*) to vertices colored red, blue, and green, a fourth color, say brown, needs to be used. Hence, *H* has a chromatic number equal to 4. A coloring of *H* is shown in Figure





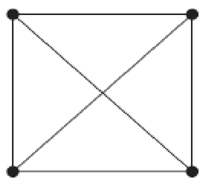
Definition:
Edge coloring:
An **edge coloring** of a graph is an assignment of colors to edges so that edges incident with a common vertex are assigned different colors. The **edge chromatic number** of a graph is the smallest number of colors that can be used in an edge coloring of the graph. The edge chromatic number of a graph *G* is denoted by $\chi'(G)$.

Example: Find the edge chromatic number of each of the graphs in figure.



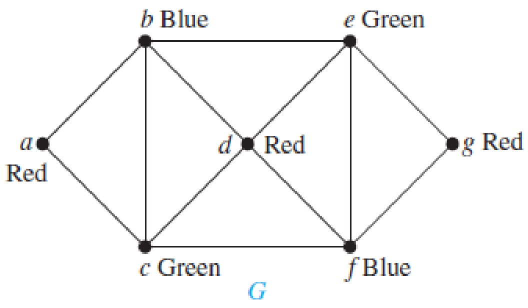
Definition:
Clique:
Let G be a graph, Z be a subset of its vertex set. If every pair of vertices in Z are adjacent then Z is called a clique of G.

Example: The following graph is adjacent because every vertex is connected to every other vertex and its clique number is 4.
For example:



Definition:
Clique Number:
The cardinal number of the largest clique in a graph. G is called the clique number of G. It is denoted by $\omega(G)$.

Definition:
Perfect Graphs:
A **perfect graph** is a **graph** in which the chromatic number of every induced subgraph equals the size of the largest clique of that subgraph.
For example:



Here chromatic number of the above graph is 3 i.e. $\chi(G) = 3$ and clique number $\omega(G)$=3. Therefore perfect graph.

Definition:
Weighted Graphs:
Graphs that have a number assigned to each edge are called **weighted graphs.**

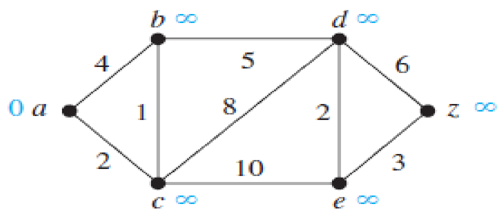
Dijkstra's Algorithm:

- This algorithm maintains a set of vertices whose shortest paths from source is already known.
- The graph is represented by its cost adjacency matrix, where cost is the weight of the edge.
- In the cost adjacency matrix of the graph, all the diagonal values are zero.
- If there is no path from source vertex V_s to any other vertex V_i then it is represented by $+\infty$. .
- In this algorithm, we have assumed all weights are positive.
- Following are the steps of Dijkstra's Algorithm.

1. Initially, there is no vertex in sets.
2. Include the source vertex V_s in S. Determine all the paths from V_s to all other vertices without going through any other vertex.
3. Now, include that vertex in S which is nearest to V_s and find the shortest paths to all the vertices through this vertex and update the values.
4. Repeat the step until n-1 vertices are not included in S if there are n vertices in the graph.

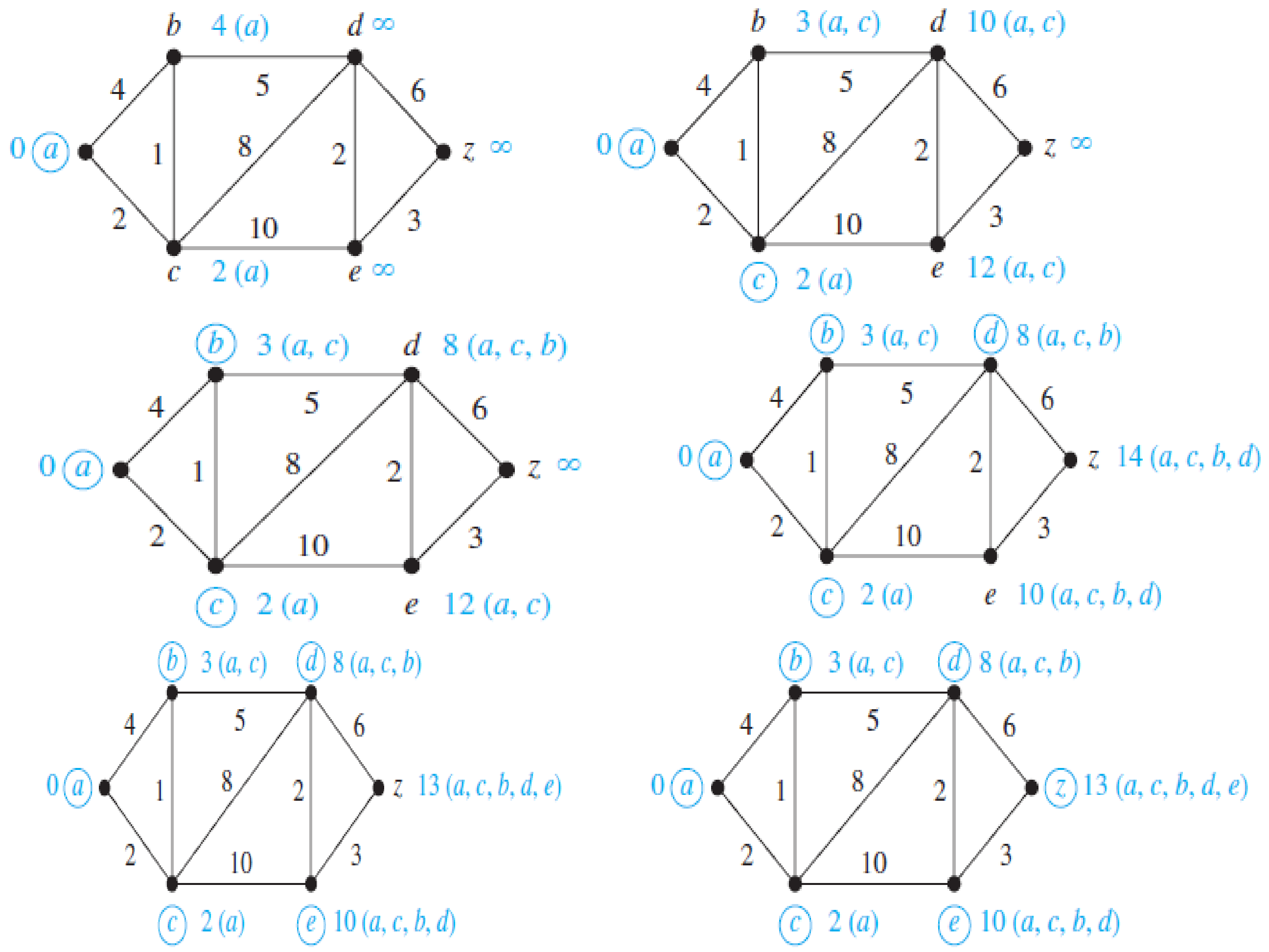
After completion of the process, we got the shortest paths to all the vertices from the source vertex.

Example: Use Dijkstra's algorithm to find the length of a shortest path between the vertices *a* and *z* in the weighted graph displayed in the beside Figure



Solution:

- The steps used by Dijkstra's algorithm to find a shortest path between *a* and *z* are shown in below Figures.
- At each iteration of the algorithm the vertices of the set *Sk* are circled.
- A shortest path from *a* to each vertex containing only vertices in *Sk* is indicated for each iteration.
- The algorithm terminates when *z* is circled.
- We find that a shortest path from *a* to *z* is *a, c, b, d, e, z*, with length 13.



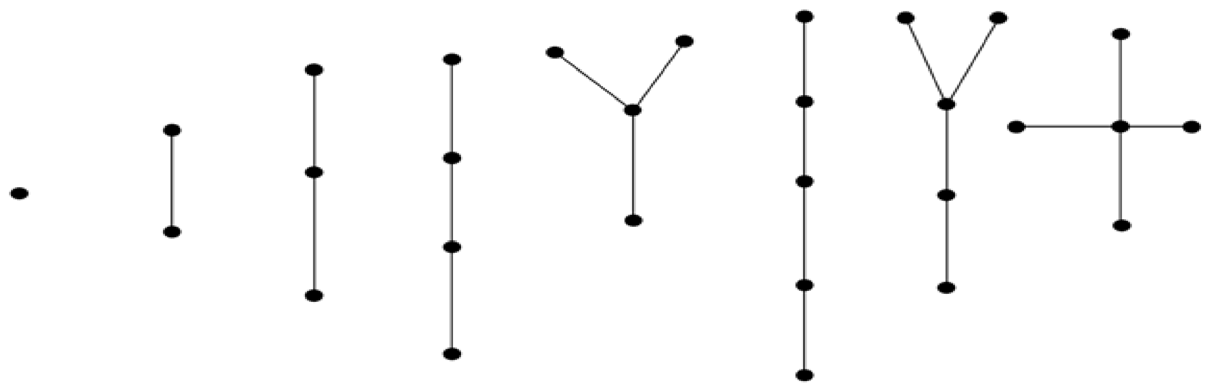
Definition:

Tree:

A tree is a connected undirected graph with no simple circuits.

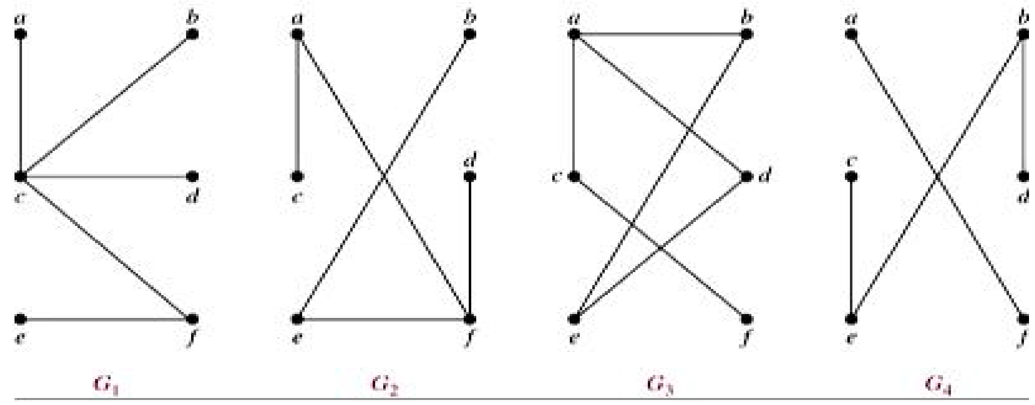
- A *tree* is a connected undirected graph with
  – No simple circuits
  – No multiple edges
  – No loops
- An undirected graph is a tree if and only if there is a unique simple path between any two of its vertices

Example



In the above example, all are trees with fewer than 6 vertices.

Example: Which graphs are trees among the below drawn graphs?



Solution: G_1, G_2

Definition:
Rooted trees:
A *rooted tree* is a tree in which one vertex has been designated as the root and every edge is directed away from the root.

Parent vertex:
T is a rooted tree. If *v* is a vertex in *T* other than the root, the **parent** of *v* is the unique vertex *u* such that there is a directed edge from *u* to *v* (the reader should show that such a vertex is unique).

Child vertex:
When *u* is the parent of *v*, *v* is called a **child** of *u*.

Siblings:
Vertices with the same parent are called **siblings**.

Ancestors:
The **ancestors** of a vertex other than the root are the vertices in the path from the root to this vertex, excluding the vertex itself and including the root (that is, its parent, its parent's parent, and so on, until the root is reached).

Descendants:
The **descendants** of a vertex *v* are those vertices that have *v* as an ancestor.

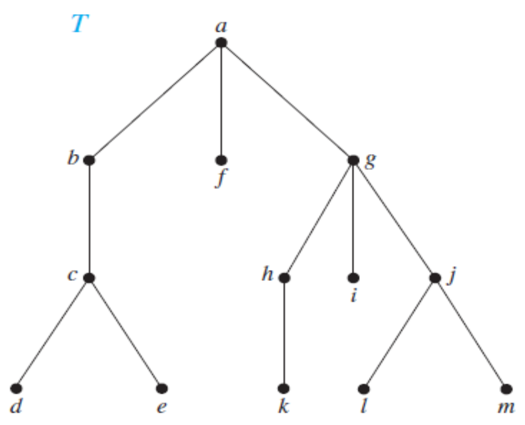
Leaf vertex:
A vertex of a rooted tree is called a **leaf** if it has no children.

Internal Vertices:
Vertices that have children are called **internal vertices**.

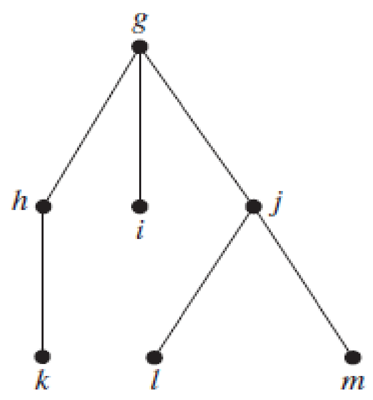
- The root is an internal vertex unless it is the only vertex in the graph, in which case it is a leaf.

Subtree of a Tree:
If *a* is a vertex in a tree, the **subtree** with *a* as its root is the subgraph of the tree consisting of *a* and its descendants and all edges incident to these descendants.

Example: In the rooted tree *T* (with root *a*) shown in Figure, find the parent of *c*, the children of *g*, the siblings of *h*, all ancestors of *e*, all descendants of *b*, all internal vertices, and all leaves. What is the subtree rooted at *g*?



Rooted Tree T



Subtree roooted at g

Solution:
The parent of *c* is *b*. The children of *g* are *h*, *i*, and *j* . The siblings of *h* are *i* and *j* . The ancestors of *e* are *c*, *b*, and *a*. The descendants of *b* are *c*, *d*, and *e*. The internal vertices are *a*, *b*, *c*, *g*, *h*, and *j* . The leaves are *d*, *e*, *f* , *i*, *k*, *l*, and *m*. The subtree rooted at *g* is shown in Figure.

Definition:
m-ary tree:
A rooted tree is called an *m-ary tree* if every internal vertex has no more than *m* children.
Definition:
Full *m*-ary tree :
The tree is called a *full m-ary tree* if every internal vertex has exactly *m* children.

Definition:
Binary Tree:
An *m*-ary tree with $m = 2$ is called a *binary tree*.

Properties of Tree:

1. Every tree which has at least two vertices should have at least two leaves.
2. Trees have many characterizations:

   Let T be a graph with n vertices, then the following statements are equivalent:

   - T is a tree.
   - T contains no cycles and has n-1 edges.
   - T is connected and has (n -1) edge.
   - T is connected graph, and every edge is a cut-edge.
   - Any two vertices of graph T are connected by exactly one path.
   - T contains no cycles, and for any new edge e, the graph T+ e has exactly one cycle.
3. Every edge of a tree is cut -edge.
4. Adding one edge to a tree defines exactly one cycle.
5. Every connected graph contains a spanning tree.
6. Every tree has at least two vertices of degree two.

Definition:
Complete m-ary Tree:
A **complete *m*-ary tree** is a full *m*-ary tree in which every leaf is at the same level.

Definition:

Eccentricity of a vertex:
The **eccentricity** of a vertex in an unrooted tree is the length of the longest simple path beginning at this vertex.

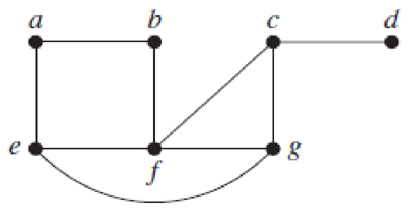
Definition:
Centre of a vertex:
A vertex is called a **center** if no vertex in the tree has smaller eccentricity than this vertex.

Definition:
Prefix codes:
One way to ensure that no bit string corresponds to more than one sequence of letters is to encode letters so that the bit string for a letter never occurs as the first part of the bit string for another letter. Codes with this property are called **prefix codes**. For instance, the encoding of *e* as 0, *a* as 10, and *t* as 11 is a prefix code.

For example: The string 10110 is the encoding of *ate*. To see this, note that the initial 1 does not represent a character, but 10 does represent *a* (and could not be the first part of the bit string of another letter). Then, the next 1 does not represent a character, but 11 does represent *t*. The final bit, 0, represents *e*.

Definition:
Spanning Tree:
Let *G* be a simple graph. A ***spanning tree*** of *G* is a subgraph of *G* that is a tree containing every vertex of *G*.

Example: Find a spanning tree of the simple graph shown in below Figure

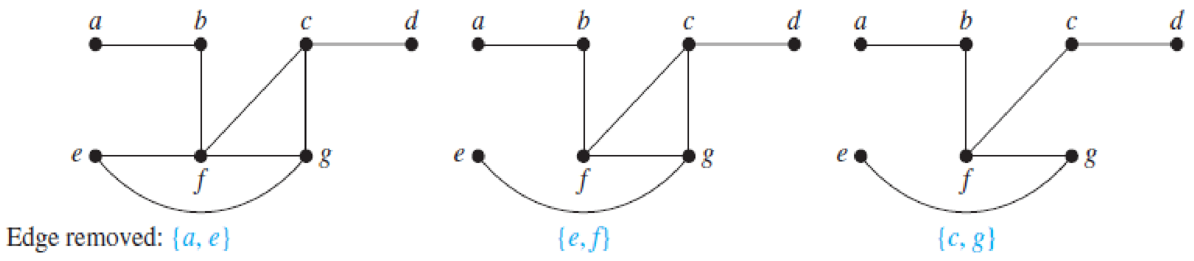


G

Solution:

- The graph *G* is connected, but it is not a tree because it contains simple circuits.
- Remove the edge {*a, e*}.
- This eliminates one simple circuit, and the resulting subgraph is still connected and still contains every vertex of *G*.
- Next remove the edge {*e, f*} to eliminate a second simple circuit. Finally, remove edge {*c, g*} to produce a simple graph with no simple circuits.
- This subgraph is a spanning tree, because it is a tree that contains every vertex of *G*.

The sequence of edge removals used to produce the spanning tree is illustrated in Figure

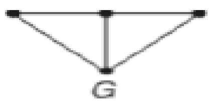


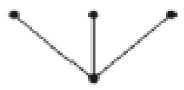
Remark: A simple graph is connected if and only if it has a spanning tree.

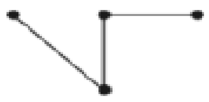
Example: Spanning tree of the simple graph is unique(True/False) **[Summer 2021-22]**

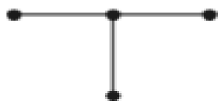
Example: [Winter 2023-24]

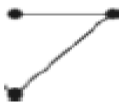
Which of the following is not spanning tree of graph



A) 

B) 

C) 

D) 

Definition:
Rooted Spanning Tree:
A **rooted spanning tree** of a directed graph is a rooted tree containing edges of the graph such that every vertex of the graph is an endpoint of one of the edges in the tree.

Definition:
Minimum Spanning Tree:
A ***minimum spanning tree*** in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

PRIM'S ALGORITHM

- **Prim's Algorithm** is a greedy algorithm that is used to find the minimum spanning tree from a graph.
- Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.
- Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step.
- The edges with the minimal weights causing no cycles in the graph got selected.

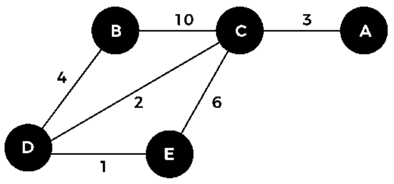
Steps to be followed for Prim's Algorithm:
Prim's algorithm is a greedy algorithm that starts from one vertex and continue to add the edges with the smallest weight until the goal is reached. The steps to implement the prim's algorithm are given as follows

- First, we have to initialize an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
- Repeat step 2 until the minimum spanning tree is formed.

The applications of prim's algorithm are -

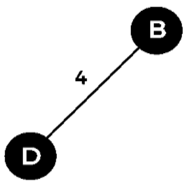
- Prim's algorithm can be used in network designing.
- It can be used to make network cycles.
- It can also be used to lay down electrical wiring cables.

Example
Suppose, a weighted graph is as shown below:

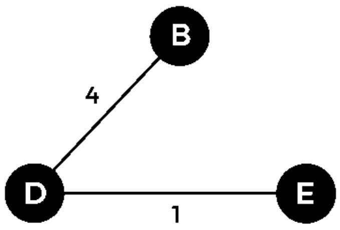


Step 1 - First, we must choose a vertex from the above graph. Let us choose B.

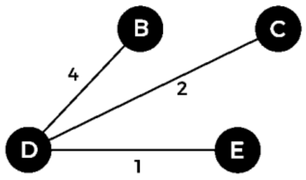
Step 2 - Now, we must choose and add the shortest edge from vertex B.
There are two edges from vertex B that are B to C with weight 10 and edge B to D with weight 4. Among the edges, the edge BD has the minimum weight. So, add it to the MST.(MST-minimum spanning tree)



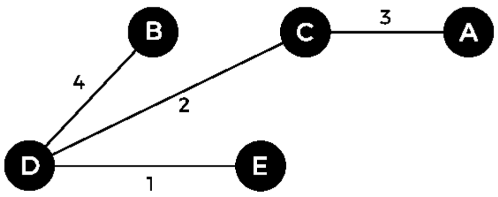
Step 3 - Now, again, choose the edge with the minimum weight among all the other edges. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C, i.e., E and A. So, select the edge DE and add it to the MST.



Step 4 - Now, select the edge CD, and add it to the MST.

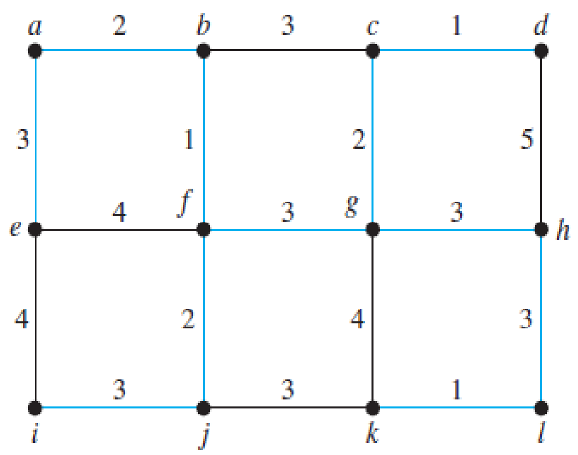


Step 5 - Now, choose the edge CA. Here, we cannot select the edge CE as it would create a cycle to the graph. So, choose the edge CA and add it to the MST.



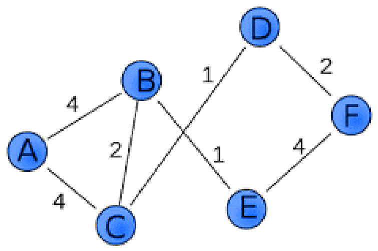
So, the graph produced in step 5 is the minimum spanning tree of the given graph. The cost of the MST is given below -
Cost of MST = 4 + 2 + 1 + 3 = 10 units.

Example: Use Prim’s algorithm to find a minimum spanning tree in the graph shown in below Figure **[Summer 2021-22]**
Solution:
A minimum spanning tree constructed using Prim’s algorithm is shown in Figure. The successive edges chosen are displayed.



Choice	Edge	Weight
1	$\{b, f\}$	1
2	$\{a, b\}$	2
3	$\{f, j\}$	2
4	$\{a, e\}$	3
5	$\{i, j\}$	3
6	$\{f, g\}$	3
7	$\{c, g\}$	2
8	$\{c, d\}$	1
9	$\{g, h\}$	3
10	$\{h, l\}$	3
11	$\{k, l\}$	1
	Total:	24

Example: Use Prim's algorithm to find the minimum spanning tree for the following weighted graph. **[Winter 2023-24]**



KRUSKAL'S ALGORITHM

- **Kruskal's Algorithm** is used to find the minimum spanning tree for a connected weighted graph.
- The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph.
- It follows the greedy approach that finds an optimum solution at every stage instead of focusing on a global optimum.

Steps to be followed in Kruskal Algorithm:

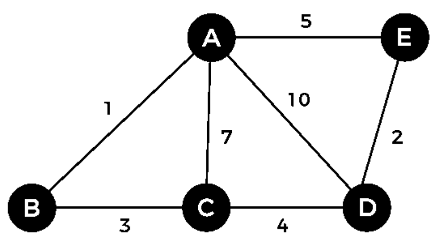
- In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached.
- The steps to implement Kruskal's algorithm are listed as follows -
- First, sort all the edges from low weight to high.
- Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
- Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

The applications of Kruskal's algorithm are -

- Kruskal's algorithm can be used to layout electrical wiring among cities.
- It can be used to lay down LAN connections.

Example:

Suppose a weighted graph is given as shown in below figure:



The weight of the edges of the above graph is given in the below table -

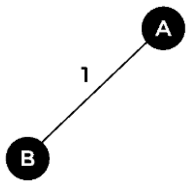
Edge	AB	AC	AD	AE	BC	CD	DE
Weight	1	7	10	5	3	4	2

Now, sort the edges given above in the ascending order of their weights.

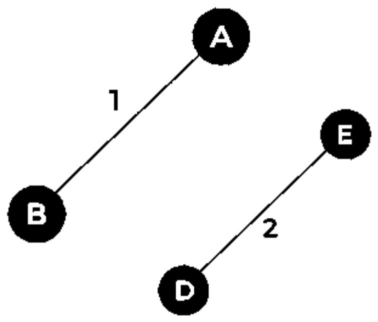
Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

Now, let us start constructing the minimum spanning tree.

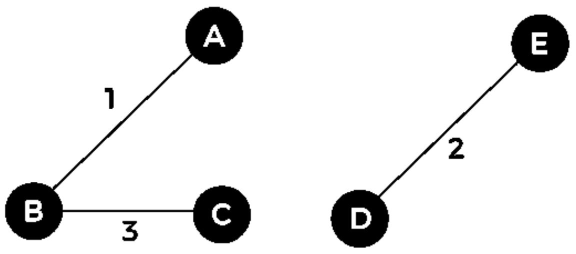
Step 1 - First, add the edge **AB** with weight **1** to the MST.



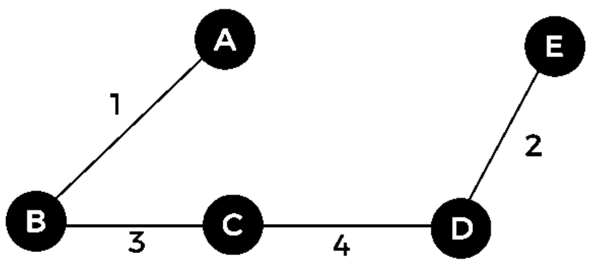
Step 2 - Add the edge **DE** with weight **2** to the MST as it is not creating the cycle.



Step 3 - Add the edge **BC** with weight **3** to the MST, as it is not creating any cycle or loop.



Step 4 - Now, pick the edge **CD** with weight **4** to the MST, as it is not forming the cycle.

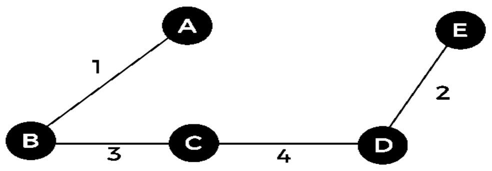


Step 5 - After that, pick the edge **AE** with weight **5.** Including this edge will create the cycle, so discard it.

Step 6 - Pick the edge **AC** with weight **7.** Including this edge will create the cycle, so discard it.

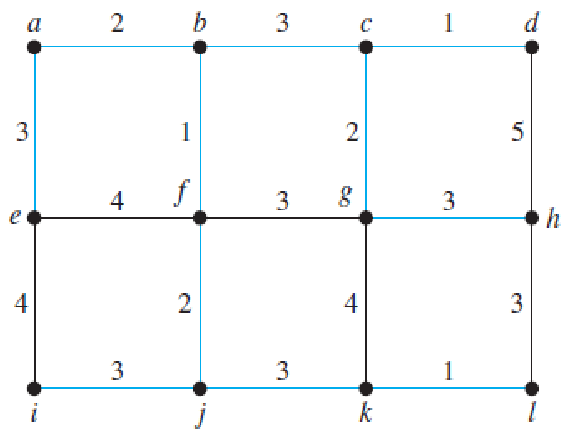
Step 7 - Pick the edge **AD** with weight **10.** Including this edge will also create the cycle, so discard it.

So, the final minimum spanning tree obtained from the given weighted graph by using Kruskal's algorithm is -



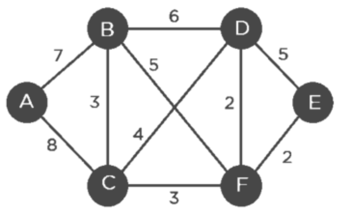
The cost of the MST is = AB + DE + BC + CD = 1 + 2 + 3 + 4 = 10.

Example: Use Kruskal's algorithm to find a minimum spanning tree in the weighted graph shown in below Figure. **[Summer 2021-22]**
Solution:
A minimum spanning tree and the choices of edges at each stage of Kruskal's algorithm are shown in below Figure



Choice	Edge	Weight
1	$\{c, d\}$	1
2	$\{k, l\}$	1
3	$\{b, f\}$	1
4	$\{c, g\}$	2
5	$\{a, b\}$	2
6	$\{f, j\}$	2
7	$\{b, c\}$	3
8	$\{j, k\}$	3
9	$\{g, h\}$	3
10	$\{i, j\}$	3
11	$\{a, e\}$	3
	Total:	24

Example: Use Kruskal's algorithm to find the minimum spanning tree for the following weighted graph


B
6
D
7
5
5
A
3
2
E
4
8
2
C
3
F