MASTER OF COMPUTER APPLICATIONS

SEMESTER 1

DISCRETE MATHEMATICS AND GRAPH THEORY

Unit 13
Basic Graph Theory

Table of Contents

1. INTRODUCTION

Leonhard Euler, a very well-known mathematician, began studying graph theory in 1735. It is a relatively recent field of mathematics. Since then, it has developed into a potent instrument utilised in almost every field of science and is now a hotly debated topic in mathematics.

Graphs are distinct structures made up of vertices and the edges connecting them. Depending on whether edges have orientations, if several edges may link the same set of vertices, and whether loops are permitted, there are various types of graphs. Graph models may be used to tackle issues in practically all disciplines.

1.1. Objectives:

At studying this unit, you should be able to:

❖ *Understand basics of Definitions of graph, and Types of Graphs.*

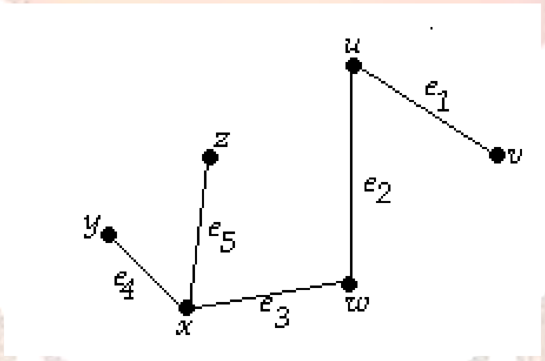
2. DEFINITIONS AND EXAMPLES

2.1. Graph

A graph G is an ordered pair (V, E) where V = {a,b,c,d ,...} is the non-empty finite set of elements called vertices(also called nodes) and E is the set of pairs of distinct vertices called edges. A graph is usually denoted by G = (V, E).

Then G=(V,E) is a graph ,Here V(G) is called Vertex set and E(G) is called an Edge set.

Example: A graph G is defined by the sets V(G) that is vertex set of G and E(G) the edge set of G, where V(G) ={u, v, w, x, y, z} and E(G) = {uv, uw, wx, xy, xz} = {e1, e2, e3, e4, e5 } . R= {(2,2),(2,4), (3,3), (4,4)}



2.2. Order and size of graph

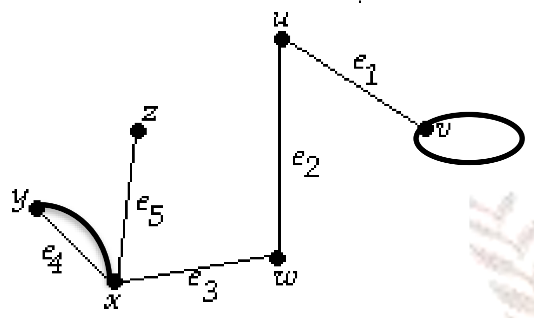
The *number of vertices* in a graph is called **order of the graph** and the *number of edges* in the graph are called its **size.**

Cardinality of V, $|V|$ is the order.

Cardinality of E, $|E|$ is the size.

A graph of order n and size m is called a $(\boldsymbol{n},\ \boldsymbol{m})$ **graph.**

2.3. Loop and Multiple edge



A loop is an edge which begins and ends at the same vertex.

Example: vertex v has loop as the edge is stating ending at the same point

If in a graph there are two or more edges with the same end vertices, the edges are called **multiple edges**.

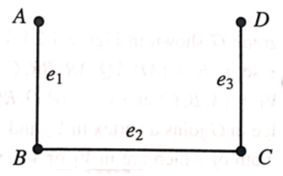
Example: Edges between the vertex y and x

2.4. Adjacent edges and vertices

The two *non-parallel edges* are said to be ***adjacent edges*** if they are *incident on a common vertex.*

Two vertices are said to be ***adjacent vertices*** *if there is an edge joining them.*

Example:



In Fig, *A and B are adjacent vertices* and e_1 *and* e_2 *are adjacent edges*

3. TYPES OF GRAPHS

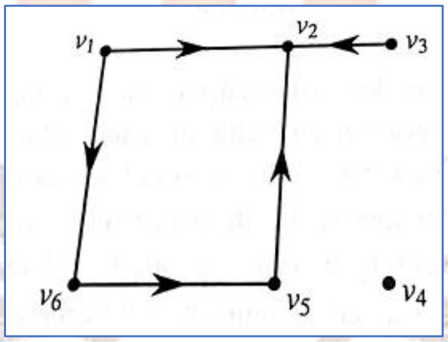
Based on the direction the graphs are divided into two types.

i. Directed Graphs
ii. Undirected Graphs.

3.1. Directed Graphs

In a **directed graph or digraph**, each element of E is an ordered pair, and we think of edges as arrows from a **source, head**, or **initial vertex** to a **sink, tail**, or **terminal vertex**; each of these two vertices is called an endpoint of the edge.

Example:



Isolated Vertex: (v_4): A vertex of diagraph which is neither an initial vertex nor a terminal vertex of any directed edge.

Non-isolated vertex happens to be initial vertex or a terminal vertex for some directed edges.

A non-isolated vertex which is not a terminal vertex for any directed edge is called **Source**.

A non-isolated vertex which is not a initial vertex for any directed edge is called **Sink**.

$\boldsymbol{v_1\ and v_3}$ are sources. $\boldsymbol{v_2}$ *is a sink*.

If v is a vertex of a diagraph D, the *number of edges for which v is the initial vertex* is called the **out-degree of v.** $(\boldsymbol{d^+(v)})$

The number of edges for which v is the terminal vertex is called the **in-degree of v**. $(d^-(v))$

$d^+(v_1) = 2$	$d^-(v_1) = 0$
$d^+(v_2) = 0$	$d^-(v_2) = 3$
$d^+(v_3) = 1$	$d^-(v_3) = 0$
$d^+(v_4) = 0$	$d^-(v_4) = 0$
$d^+(v_5) = 1$	$d^-(v_5) = 1$
$d^+(v_6) = 1$	$d^-(v_6) = 1$

3.2. Undirected graphs

In an **undirected graph**, each edge is an undirected pair, which we can

represent as subset of V with one or two elements.

3.3. Finite and Infinite Graph

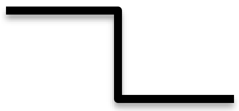
A graph with a finite number of vertices and finite number of edges is called **finite graph** otherwise is called **infinite graph.**

3.4. Simple Graph

A graph does not contain *loops and multiple edges* is called a **simple graph.**

A graph which does not contains a loops is called a **loop-free** graph.

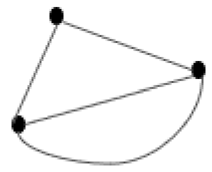
Example:



3.5. Multigraph or Multiple graph

A graph which *contains multiple edges but no loops* is called **multigraphs or Multiple graph**.

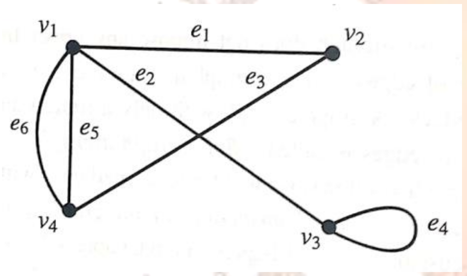
Example:



3.6. General Graph

A graph which *contains multiple edges or loops (or both)* is called **general graphs.**

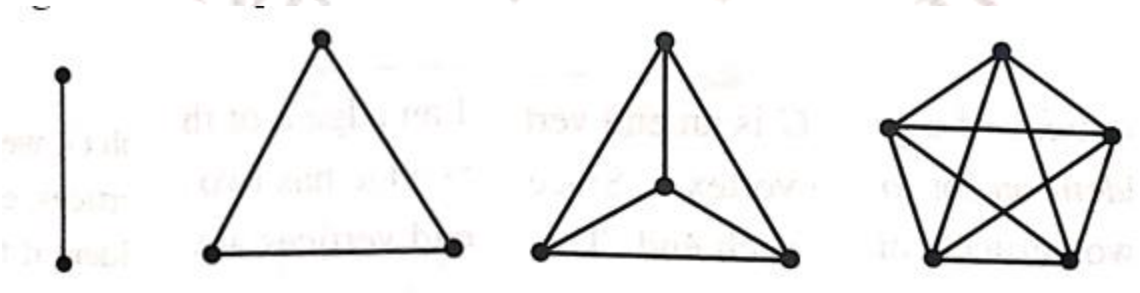
Example:



3.7. Complete Graph

A simple graph of *order* ≥ *2* in which there is an edge between every pair of vertices is called a **complete graph** and is denoted by $\boldsymbol{k_n}$

Following are the complete graph with 2, 3,, 4 and 5 vertices are shown below



k_2 k_3 k_4 k_5

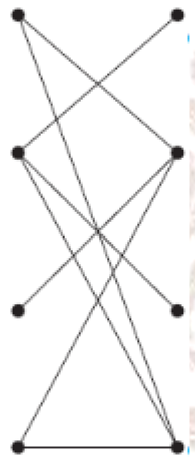
K_5 is called the Kuratowski's first graph

3.8. Bipartite graph

Suppose a simple graph G is such that its vertex set V is the union of two mutually disjoint nonempty sets $V_1 and\ V_2$ which are such that each edges in G joins a vertex in V_1 and a vertex V_2. Then G is called a **bipartite graph.**

If E is the edge set of this graph, the graph is denoted by $G = (V_1,\ V_2; E)$, or $G = G(V_1,\ V_2; E)$. The sets $V_1 and\ V_2$ are called **bipartites** of the vertex set V.

Example

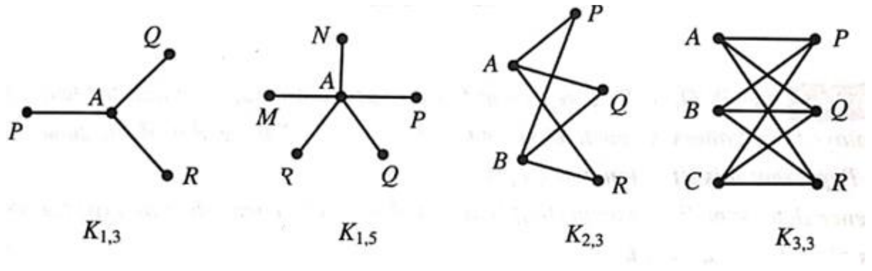


3.9. Complete Bipartite graph

A bipartite graph $G = (V_1,\ V_2; E)$ is called a *complete bipartite graph* if there is an edge between every vertex in $V_1$and every vertex in V_2.

A complete bipartite graph $G = (V_1,\ V_2; E)$ in which the bipartites $V_1 and\ V_2$ contain r and s vertices respectively, with $r \leq s$, is denoted by $K_{r,s}$.

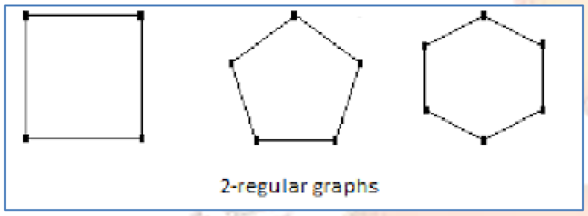
$K_{r,s}$ has $r + s$ vertices and rs edges; that is $K_{r,s}$ is of order $r + s$ and size rs; it is therefore a $(r + s,\ rs)$ graph

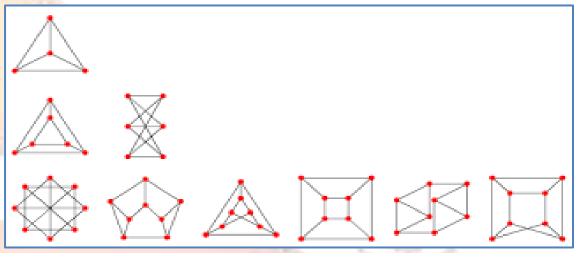


3.10. Regular Graph

A graph in which all the vertices are of the same degree k is called a regular graph of degree k. Or k – regular graph.

Example: Draw 2-regular, and 3-regular graph.



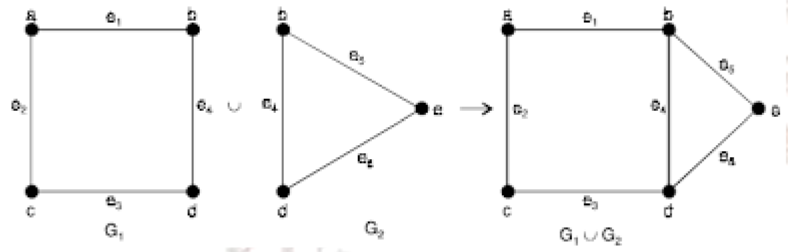


4. OPERATIONS ON GRAPHS

There are many ways of combining two or more graphs to produce new graphs.

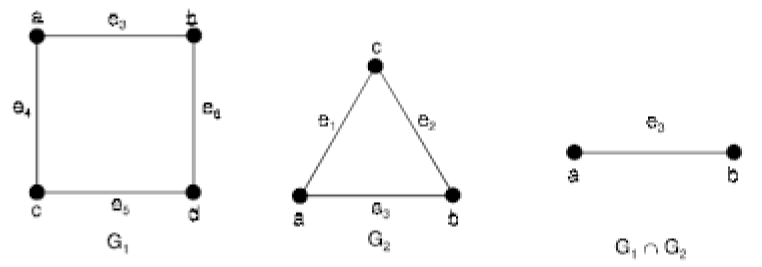
4.1. The Union

The union of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is another graph G_3 as $G3 = G1 \cup G2$ whose vertex set $V3 = V1 \cup V2$ and the edge set $E3 = E1 \cup E2$.



4.2. The Intersection

The intersection $G_1 \cap G_2$ of graphs G_1 and G_2 is a graph consisting only of those vertices and edges that are in both of G_1 and G_2.

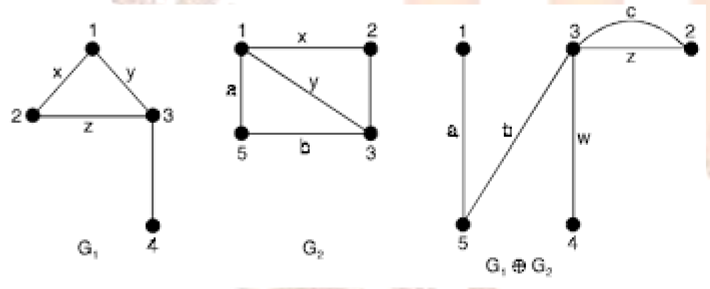


4.3. The ring sum

The ring sum of two graphs G1 and G2 (written G1⊕ G2) is a graph consisting of the vertex set $V_1 \cup V_2$ and of edges that are either in G1 or G2 but not in both.

The above three operations are commutative.

$G_1 \cup G_2 = G_2 \cup G_2$, $G_1 \cap G_2 = G_2 \cap G_1$ and $G_1 \oplus G_2 = G_2 \oplus G_1$



If G1 and G2 are edge disjoint, then $G_1 \cap G_2$ is a null graph (empty graph) and $G_1 \oplus G_2 = G_1 \cup G_1$. If they are vertex disjoint then $G_1 \cap G_2$ is empty.

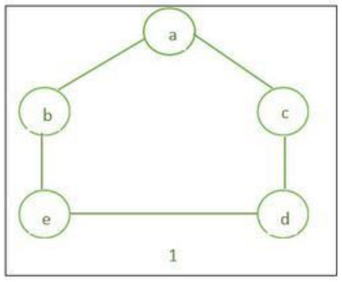
For any graph G, $G \cup G = G \cap G = G$. $G \oplus G = a$ null graph (empty graph), G $G \oplus g$ is written as G – g whenever $g \subseteq G$ therefore $G \oplus g$ = G – g.

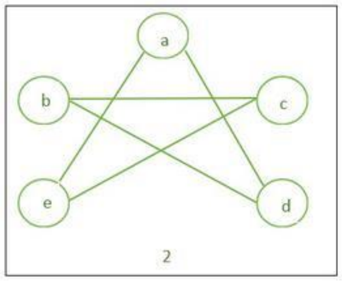
4.4. The Decomposition

A graph G is said to have been decomposed into two sub-graphs g_1 *and* g_2 if $g_1 \cup g_2 = G$ *and* $g_1 \cap g_2 = a$ null graph.

4.5. Complement of a Graph

The complement $\overline{G}$ of a graph G *is* a graph having the same vertex set as G where two vertices u and v are adjacent if and only if, they are not adjacent in G. A graph is *Self-complementary* if it is isomorphic to its complement.





<u>Note</u> :

For any graph G, the component of $\overline{G}$ is G itself. That is $G = \overline{(\overline{G})}$.

If a graph is connected, then its complement $\overline{G}$ may be connected or disconnected.

5. TREES AND THEIR BASIC PROPERTIES

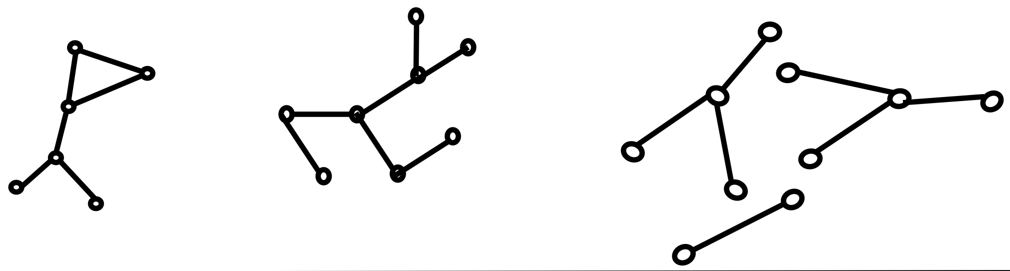
A graph is a **tree**, if it is connected and has no cycle.

Tree must be simple graph as loops and parallel edges form cycles.

A vertex of degree 1 is called a terminal vertex or a leaf, the other vertices are called internal nodes.

A tree is denoted by T .

Example: Only second figure is Tree as It does not have any cycle and no breakages



Note:

Any tree with more than one vertex has at least one vertex of degree 1.

Any tree with n vertices has n – 1 edges.

That is every tree T = (V, E), | V | = | E | + 1 (or | E | = | V | - 1).

If a connected graph with n vertices has n – 1 edges, then it is a tree

Example:

If a tree T has 3 vertices of degree 2, 2 vertices of degree 3 and 2 vertices of degree 4. Find the number of pendant vertices in T.

Solution:

Let N be the number of pendant vertices in T.

The total number of vertices in the tree

T = N + 3 + 2 + 2 = N + 7

Therefore the number of edges in the tree

T = N + 7 – 1 = N + 6

By handshaking property sum of all the degrees of the vertices: (By handshaking lemma $\sum_{i=1}^{p} deg(v_i) = 2q$)

(N ×1) + (3 × 2) + (2 × 3) + (2 × 4) = 2(N + 6)

N + 6 + 6 + 8 = 2N + 12

2N – N = 20 – 12

N = 8.

5.1. Application of Trees

The unique properties of trees—being connected, acyclic graphs with a specific relationship between their vertices and edges—make them incredibly versatile for a wide array of applications. Here are some notable applications of trees:

1. Data Structure Implementation

Trees are foundational in implementing various data structures, such as:

- Binary Search Trees (BSTs): Used in databases and search engines for efficient searching and sorting operations. BSTs allow for average-case time complexity of O(logn) for search, insert, and delete operations.
- Heaps: A specialized tree-based data structure that satisfies the heap property, used in implementing priority queues, scheduling algorithms, and for efficient sorting (heap sort).
- Trie (Prefix Tree): Utilized in auto-complete features in search engines and dictionaries, where the tree structure allows for the fast retrieval of words or phrases starting with a given prefix.

2. Network Routing Algorithms

In computer networks, trees play a crucial role in the design of routing algorithms. Spanning trees, such as those generated by the Spanning Tree Protocol (STP), ensure that a network is loop-free. Algorithms like Dijkstra's and Prim's utilize tree structures to find the shortest path and minimum spanning tree, respectively, which are crucial for efficient data transmission.

3. Graphical User Interfaces (GUIs)

The hierarchical structure of trees makes them ideal for representing the organization of graphical user interfaces in software applications. The Document Object Model (DOM) in web development is a tree structure that represents the layout of web pages, allowing for the systematic manipulation of elements.

4. File Systems

In operating systems, trees are used to organize file systems. Directories and subdirectories are structured as trees, where each node represents a file or directory. This hierarchical structure allows for efficient file storage, retrieval, and management.

5. Decision Trees

In machine learning and data mining, decision trees are used for classification and regression tasks. They model decisions and their possible consequences as a tree, including chance event outcomes, resource costs, and utility. It's a straightforward yet powerful tool for predictive analytics.

6. Abstract Syntax Trees (ASTs)

In compilers, ASTs are used to represent the syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code, facilitating the compiler's understanding and manipulation of the program's structure.

7. Game Theory

In game theory, trees represent game scenarios, where each node corresponds to the state of the game at a certain point, and the edges represent possible moves or decisions by the players. This structure is pivotal in analysing and strategizing games, especially in AI for games like chess or go.

8. Organizational Structures

In business and management, tree diagrams are used to represent organizational structures, showing the hierarchy of roles, responsibilities, and relationships within an organization. This helps in understanding the reporting structure and the distribution of tasks.

Trees, with their diverse properties and applications, are integral to solving complex problems across various domains, highlighting the interconnectedness of discrete mathematics and practical applications in technology and beyond.

6. TREE TRAVERSAL

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree.

There are three ways which we use to traverse a tree

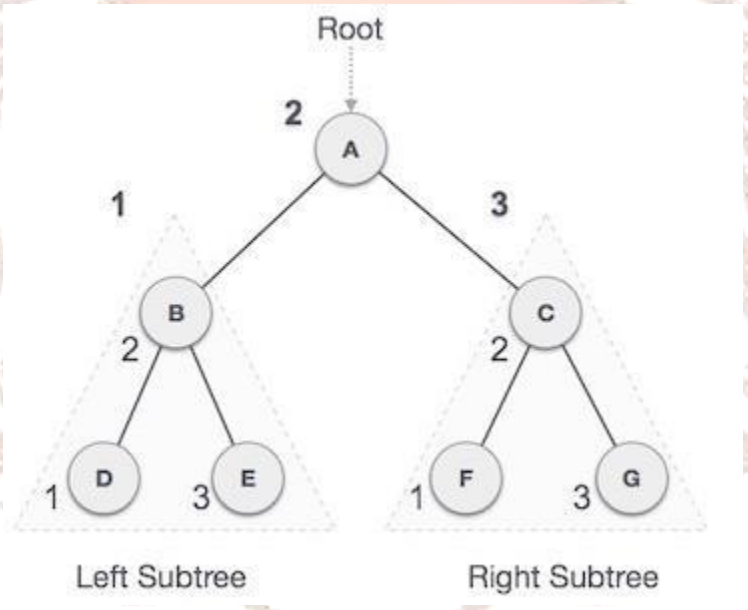
- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

6.1. In-order Traversal

In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.



We start from A, and following in-order traversal, we move to its left subtree B.B is also traversed in-order. The process goes on until all the nodes are visited. The output of in-order traversal of this tree will be −

D → B → E → A → F → C → G

Algorithm

Until all nodes are traversed −

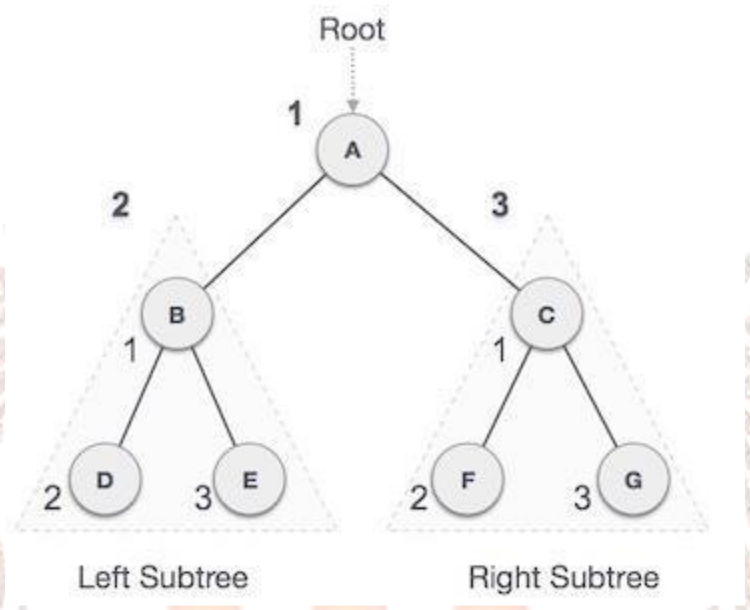
Step 1 − Recursively traverse left subtree.

Step 2 − Visit root node.

Step 3 − Recursively traverse right subtree.

6.2. Pre-order Traversal

In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.



We start from A, and following pre-order traversal, we first visit A itself and then move to its left subtree B. B is also traversed pre-order. The process goes on until all the nodes are visited. The output of pre-order traversal of this tree will be –

A → B → D → E → C → F → G

Algorithm

Until all nodes are traversed –

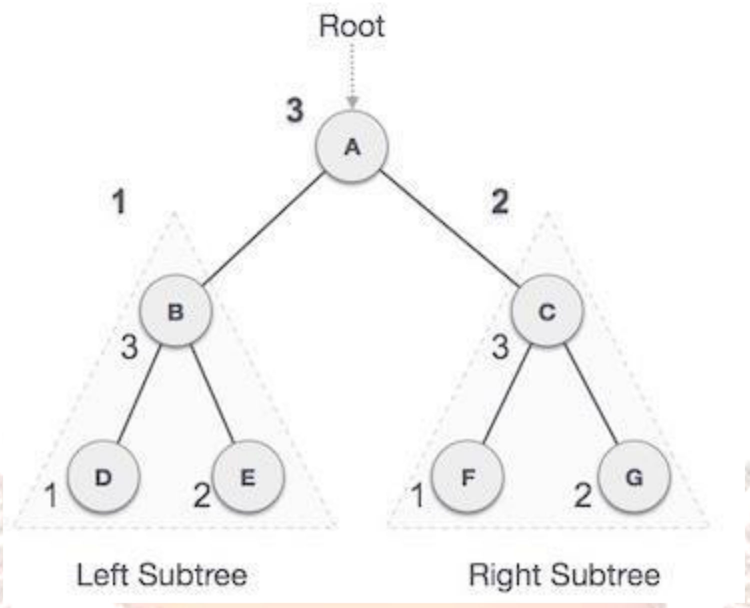
Step 1 – Visit root node.

Step 2 – Recursively traverse left subtree.

Step 3 – Recursively traverse right subtree.

6.3. Post-order Traversal

In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.



We start from A, and following pre-order traversal, we first visit the left subtree B. B is also traversed post-order. The process goes on until all the nodes are visited. The output of post-order traversal of this tree will be –

D → E → B → F → G → C → A

Algorithm

Until all nodes are traversed –

Step 1 – Recursively traverse left subtree.

Step 2 – Recursively traverse right subtree.

Step 3 – Visit root node.

6.4. Spanning Trees

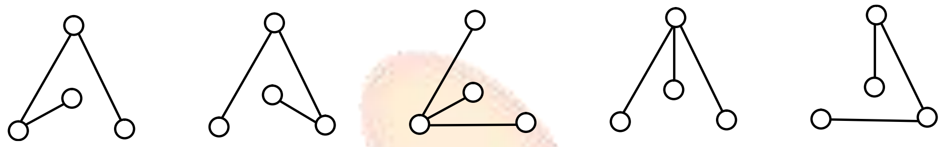
A Spanning Tree of a graph is a subgraph that includes all the vertices of the original graph with the minimum possible number of edges to keep the subgraph connected. Importantly, a spanning tree must meet two key criteria:

It must be a subgraph of the original graph, meaning it can only use the vertices and edges that exist in the original graph.

It must be a tree, which means it must be connected (there's a path between every pair of vertices) and acyclic (it contains no cycles).

As a tree with n vertices has exactly n-1 edges, a spanning tree of a graph with n vertices will also have n-1 edges

Following are some examples of spanning tree:



Typically, we are provided with an initial graph to use as a reference. Naturally, an arbitrary spanning tree does not meet our desired criteria. We desire to obtain the minimal cost spanning tree (MCST).

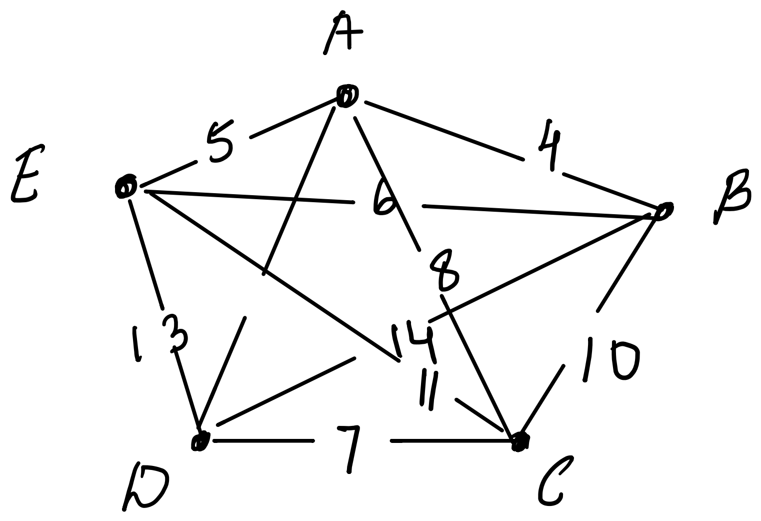
The Minimum Cost Spanning Tree is the Spanning Tree with the smallest total edge weight. A nearest neighbour methodology is not applicable in this context as a circuit is not required. Instead, we shall adopt a strategy akin to sorted edges. For this, we use Kruskal's Algorithm:

Steps:

1. Select the cheapest unused edge in the graph.
2. Repeat step 1, adding the cheapest unused edge, unless:
    a. Adding the edge would create a circuit
3. Repeat until a spanning tree is formed

Example:

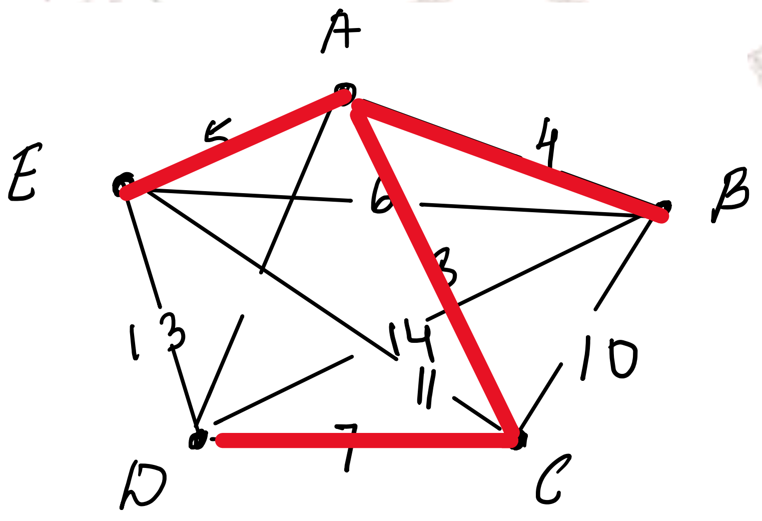
A company requires reliable internet and phone connectivity between their five offices (named A, B, C, D, and E for simplicity) in Jaipur, so they decide to lease dedicated lines from the phone company. The phone company will charge for each link made. The costs, in thousands of rupees per year, are shown in the graph.



Using the Phone Line graph, Let's Start Adding the Edges:

1. AB ➔ Rs. 4 OK
2. AE ➔ Rs. 5 OK
3. BE ➔ Rs. 6 Reject-closes circuit ABEA
4. DC ➔ Rs. 7 OK
5. AC ➔ Rs. 8 OK

We stop at this point, as every vertex is now connected, and have formed a spanning tree with a cost of Rs. 24 Thousand a year.



7. SELF-ASSESSMENT QUESTIONS

1. Draw a diagram of the graph $G = G(V, E)$ in each of the following cases.

   (i) $\boldsymbol{V = \{A, B, C, D\},\ E = \{(A, B),\ (A, C),\ (A, D),\ (C, D)\}}$

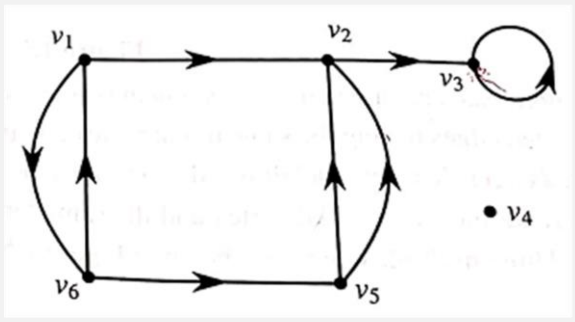
   (ii) $\boldsymbol{V = \{v_1, v_2, v_3, v_4, v_5\},\ E = \{(v_1, v_2), (v_1, v_3), (v_2, v_3), (v_4, v_5)\}}$

   (iii) $\boldsymbol{V = \{P, Q, R, S, T\},\ E = \{(P, S),\ (Q, R),\ (Q, S)\}}$

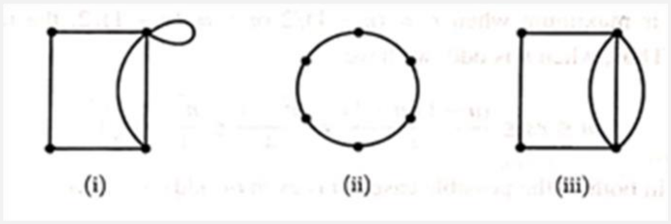
   (iv) $\boldsymbol{V = \{v_1, v_2, v_3, v_4, v_5, v_6\},\ E =}$
   $\boldsymbol{\{(v_1, v_4), (v_1, v_6), (v_3, v_2), (v_4, v_6), (v_3, v_5), (v_2, v_5)\}}$

2. How many vertices and how many edges are there in the complete bipartite graph $K_{5,7}$ and $K_{7,7}$?

3. Find the indegree and outdegree of the following graph.



4. Which of the following is simple graph? A multi graph? A general Graph?

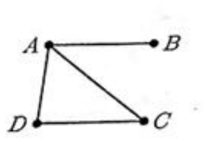


8. SUMMARY

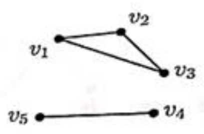
In this chapter we studied about graphs and various types of graphs with examples also we discussed different kinds of graphs, depending on whether edges have directions, whether multiple edges can connect the same pair of vertices, and whether loops are allowed.

9. ANSWERS TO SELF ASSESSMENT QUESTIONS

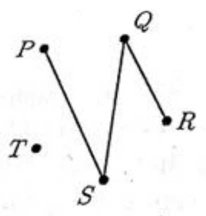
1.



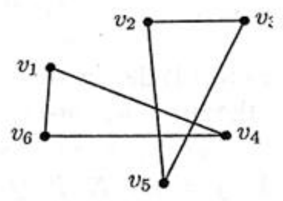
(i)



(ii)



(iii)



(iv)

2. 12 vertices and 35 edges

   14 vertices and 49 edges.

3.

Out Degree	**In Degree**
$d^+(v_1) = 2$	$d^-(v_1) = 1$
$d^+(v_2) = 1$	$d^-(v_2) = 3$
$d^+(v_3) = 1$	$d^-(v_3) = 2$
$d^+(v_4) = 0$	$d^-(v_4) = 0$
$d^+(v_5) = 2$	$d^-(v_5) = 1$
$d^+(v_6) = 2$	$d^-(v_6) = 1$

4.

(i) General Graph.

(ii) Simple graph

(iii) Multi graph

10. REFERENCES

1. Ralph P. Grimaldi (2019) Discrete and Combinatorial Mathematics: An Applied Introduction (5ed.) Pearson Publication.
2. Kenneth H. Rosen (2012) Discrete Mathematics and Its Applications (7 the Edition) McGraw-Hill Publication.
3. Dr. DSC (2016) Discrete Mathematical Structures (5th Edition), PRISM publication.
4. https://www.geeksforgeeks.org/elements-of-poset/
5. https://ncert.nic.in