# UNIT 3: Algebraic Structure

1. **What are the properties of Abelian Group?**

   An **Abelian Group** (or commutative group) is a set $G$ with a binary operation $*$ that satisfies the following five properties:

   - **Closure:** For all $a,b \in G$, the result of the operation $a*b$ is also in $G$.
   - **Associativity:** For all $a,b,c \in G$, $(a*b)*c=a*(b*c)$.
   - **Identity Element:** There exists an element $e \in G$ (called the identity element) such that for every $a \in G$, $a*e=e*a=a$.
   - **Inverse Element:** For every element $a \in G$, there exists an element $a-1 \in G$ (called the inverse of a) such that $a*a-1=a-1*a=e$.
   - **Commutativity:** For all $a,b \in G$, $a*b=b*a$.
2. **Design Composition table for (Z7,+7),(Z7,∗7)**

   Here, $Z7=\{0,1,2,3,4,5,6\}$. **Composition Table for (Z7,+7) (Addition Modulo 7):**

   | +7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
   |:---:|:---:|:---:|:---:|:---:|:---:|:---:|:---:|
   | **0** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
   | **1** | 1 | 2 | 3 | 4 | 5 | 6 | 0 |
   | **2** | 2 | 3 | 4 | 5 | 6 | 0 | 1 |
   | **3** | 3 | 4 | 5 | 6 | 0 | 1 | 2 |
   | **4** | 4 | 5 | 6 | 0 | 1 | 2 | 3 |
   | **5** | 5 | 6 | 0 | 1 | 2 | 3 | 4 |
   | **6** | 6 | 0 | 1 | 2 | 3 | 4 | 5 |

   **Composition Table for (Z7,∗7) (Multiplication Modulo 7):**

   | ∗7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
   |:---:|:---:|:---:|:---:|:---:|:---:|:---:|:---:|
   | **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
   | **1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
   | **2** | 0 | 2 | 4 | 6 | 1 | 3 | 5 |
   | **3** | 0 | 3 | 6 | 2 | 5 | 1 | 4 |
   | **4** | 0 | 4 | 1 | 5 | 2 | 6 | 3 |
   | **5** | 0 | 5 | 3 | 1 | 6 | 4 | 2 |
   | **6** | 0 | 6 | 5 | 4 | 3 | 2 | 1 |

3. **What is Coset & Lagrange Theorem?**
   - **Coset:** Let $\langle G,*\rangle$ be a group, and $H$ be a subgroup of $G$. For any element $a \in G$:
     - The **Left Coset** of $H$ with respect to $a$ is the set $a*H=\{a*h \mid h \in H\}$.
     - The **Right Coset** of $H$ with respect to $a$ is the set $H*a=\{h*a \mid h \in H\}$. The set of all left (or right) cosets partitions the group $G$.
   - **Lagrange's Theorem:** This fundamental theorem states that for any **finite group** $\langle G,*\rangle$, the order (number of elements) of any subgroup $H$ of $G$ must be a divisor of the order of $G$. Mathematically, if $|G|$ is the order of $G$ and $|H|$ is the order of $H$, then $|H|$ divides $|G|$. The quotient $|G|/|H|$ is the number of distinct cosets of $H$ in $G$.
4. **What are the properties of algebraic structure?**

   An **Algebraic Structure** consists of a non-empty set (or carrier set) $S$ together with one or more binary operations defined on $S$. The properties commonly associated with algebraic structures, which define different types of structures (like semi-groups, monoids, groups, etc.), include:

   - **Closure:** The operation applied to any two elements in the set yields an element in the set.

- **Associativity:** The order in which three or more elements are grouped for the operation does not affect the result.
- **Identity Element:** An element that, when combined with any other element using the operation, leaves the other element unchanged.
- **Inverse Element:** For each element, there is a corresponding element that, when combined using the operation, yields the identity element.
- **Commutativity:** The order of the two elements in the operation does not affect the result.

---

# UNIT 5: Trees

13. **Elaborate the properties and application of trees.**

## Properties of Trees

A **Tree** is an undirected graph in which any two vertices are connected by exactly one path. Key properties include:

- A graph G with n vertices is a tree if and only if it is **acyclic** (contains no cycles) and has **$n-1$ edges**.
- A graph is a tree if and only if there is a unique path between any two of its vertices.
- Every tree with at least two vertices has at least two **leaves** (vertices of degree 1).
- Removing any single edge from a tree disconnects it, forming two smaller trees.
- Adding a single edge between any two non-adjacent vertices in a tree creates exactly one cycle.

## Applications of Trees

Trees are widely used in Computer Science and mathematics:

- **Hierarchical Data Representation:** Trees are ideal for representing hierarchical relationships, such as the **file system structure** in an operating system (directories and files) or the structure of an organization chart.
- **Parsing and Expression Evaluation: Syntax trees** (or parse trees) are used in compilers to represent the structure of a program's code, and **expression trees** are used to represent and evaluate mathematical expressions.
- **Searching and Sorting: Binary Search Trees (BSTs)** are data structures designed for efficient searching, insertion, and deletion of data, ensuring $O(\log n)$ average time complexity.
- **Network Routing and Data Compression: Spanning trees** are used in network protocols to prevent loops, and **Huffman coding** trees are used for data compression.

   - **Decision Making: Decision trees** are used in machine learning and data mining as a predictive model for classification and regression.

14. **Describe binary search tree with suitable example**

    A **Binary Search Tree (BST)** is a specific type of binary tree that maintains a special ordering property on its nodes. For every node X in the tree:

    1. All values in the **left subtree** of X are **less than** the value in X's node.
    2. All values in the **right subtree** of X are **greater than** the value in X's node.
    3. Both the left and right subtrees are themselves binary search trees.

    This property makes searching for a specific value highly efficient.

    **Example:** Construct a BST for the sequence: 15,6,18,3,7,17,20.

    1. **Start with the Root:** Insert 15.
    2. **Insert 6:** 6<15, so 6 becomes the left child of 15.
    3. **Insert 18:** 18>15, so 18 becomes the right child of 15.
    4. **Insert 3:** 3<15, go left. 3<6, so 3 becomes the left child of 6.
    5. **Insert 7:** 7<15, go left. 7>6, so 7 becomes the right child of 6.
    6. **Insert 17:** 17>15, go right. 17<18, so 17 becomes the left child of 18.
    7. **Insert 20:** 20>15, go right. 20>18, so 20 becomes the right child of 18.

    *(A diagram would show 15 at the top, with 6 to its left and 18 to its right. 3 and 7 would be children of 6. 17 and 20 would be children of 18.)*

14. What is ordered rooted tree that represents the expression ((x+y)↑2)+((x−4)/3)? And Write these expressions in prefix and postfix notation of the given tree.

    An **Ordered Rooted Tree** for an expression represents the operations and operands, where the root is the last operation performed (which is ∗∗+** in this case), and the left and right subtrees represent the left and right sub-expressions.

    - **Root:** +
    - **Left Subtree (for (x+y)↑2):** Root is ↑ (exponentiation), Left child is (x+y), Right child is 2.
      - Left child of ↑: Root is +, Left child is x, Right child is y.
    - **Right Subtree (for (x−4)/3):** Root is /, Left child is (x−4), Right child is 3.
      - Left child of /: Root is −, Left child is x, Right child is 4.

    *(A diagram would show + as the root. Its left child is ↑, and its right child is /. The subtrees branch out to the operands.)*

    **Notation from the Tree:**

    - **Prefix Notation (Preorder Traversal: Root, Left, Right):**

+↑+xy2/−x43

- **Postfix Notation (Postorder Traversal: Left, Right, Root):**

  xy+2↑x4−3/+

15. Describe Huffman coding and use huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

## Huffman Coding

**Huffman Coding** is a lossless data compression algorithm that uses a variable-length code table for encoding a source message. It works by assigning shorter bit codes to characters (symbols) that appear more frequently and longer bit codes to those that appear less frequently, resulting in an overall reduction in the message size. The process builds an optimal tree, known as the Huffman Tree, which is a **full binary tree** where each leaf represents a source symbol, and the weight of a node is the sum of the frequencies of the leaves in its subtree.

## Huffman Encoding Example

**Symbols and Frequencies (P):** A: 0.08, B: 0.10, C: 0.12, D: 0.15, E: 0.20, F: 0.35.

**Steps (Simplified):**

1. **Combine the two lowest frequencies (A and B):** 0.08+0.10=0.18.
   - New set of frequencies: C: 0.12, D: 0.15, **(A, B): 0.18**, E: 0.20, F: 0.35.
2. **Combine the two lowest: C and D:** 0.12+0.15=0.27.
   - New set: **(C, D): 0.27**, (A, B): 0.18, E: 0.20, F: 0.35. (Error in order: must combine (A,B) and E next as they are the smallest, or (A, B) and (C, D) if we sort by weight)
3. **Combine next two lowest: (A, B) and E:** 0.18+0.20=0.38.
   - New set: (C, D): 0.27, F: 0.35, **((A, B), E): 0.38**.
4. **Combine next two lowest: (C, D) and F:** 0.27+0.35=0.62.
   - New set: **((C, D), F): 0.62**, ((A, B), E): 0.38.
5. **Combine the last two:** 0.62+0.38=1.00 (Root).

*(A diagram would show the tree structure, assigning '0' to left branches and '1' to right branches.)*

**Resulting Codes and Lengths (L):**

- **F (0.35):** Code **01** (Length 2)
- **E (0.20):** Code **11** (Length 2)

- **D (0.15):** Code **001** (Length 3)
- **C (0.12):** Code **000** (Length 3)
- **B (0.10):** Code **101** (Length 3)
- **A (0.08):** Code **100** (Length 3)

### Average Number of Bits

The average number of bits per character is calculated by the formula: $\sum(Pi \times Li)$, where Pi is the frequency and Li is the code length.

Average Bits=(0.35×2)+(0.20×2)+(0.15×3)+(0.12×3)+(0.10×3)+(0.08×3)

=0.70+0.40+0.45+0.36+0.30+0.24=2.45 bits/character

---

# UNIT 6: Graphs

21. **Identify the degrees and neighborhoods of the vertices in the graphs G and H displayed below in figure?**

*(Assuming G is the square on the left, u1,u2,u3,u4, and H is the hourglass shape on the right, v1,v2,v3,v4)*

### Graph G (u1,u2,u3,u4)

- **Degrees:** The degree of every vertex is 2, as each vertex is connected to exactly two other vertices.
  - deg(u1)=2, deg(u2)=2, deg(u3)=2, deg(u4)=2.
- **Neighborhood (N(v)):** The set of vertices adjacent to v.
  - N(u1)={u2,u3}
  - N(u2)={u1,u4}
  - N(u3)={u1,u4}
  - N(u4)={u2,u3}

### Graph H (v1,v2,v3,v4)

- **Degrees:** The degree of every vertex is 2, as each vertex is connected to exactly two other vertices.
  - deg(v1)=2, deg(v2)=2, deg(v3)=2, deg(v4)=2.
- **Neighborhood (N(v)):**
  - N(v1)={v3,v4}
  - N(v2)={v3,v4}
  - N(v3)={v1,v2}
  - N(v4)={v1,v2}

22. **Use adjacency lists to describe the simple graph given in figure below.**

    *(Assuming the figure is the graph with vertices a,b,c,d,e from the image on Page 4.)*

    An **Adjacency List** represents a graph by listing all the vertices adjacent to each vertex.

    | Vertex | Adjacency List (Neighbors) |
    |---|---|
    | **a** | {b,c,e} |
    | **b** | {a} |
    | **c** | {a,d,e} |
    | **d** | {c,e} |
    | **e** | {a,c,d} |

    Export to Sheets

23. Illustrate planar graph suitable example also explain euler's formula. Identify which of the following are planer graphs.

    ## Planar Graph and Example

    A graph is **Planar** if it can be drawn on a plane (a flat surface) such that no two edges cross each other except possibly at a vertex.

    **Example:** Any simple cycle graph (like a square or a triangle) is planar. The graph G from question 21 (the square) is planar.

    ## Euler's Formula for Planar Graphs

    For any connected planar graph with V vertices, E edges, and R regions (or faces, including the exterior region), Euler's formula states:

    $V - E + R = 2$

    This formula is a fundamental property of connected planar graphs and can be used to prove that certain graphs (like K5 or K3,3) are non-planar.

    ## Identification

    - **The Cube Graph (Figure on Page 5):** This graph is **Planar**. Even though it is drawn in 3D, it can be redrawn in 2D without any edges crossing. *Example: Draw the outer square, then the inner square, and connect corresponding vertices.*

24. What is graph? Explain different types of graph with example.

    ## What is a Graph?

A **Graph** G is a mathematical structure consisting of a non-empty set of **vertices** (or nodes) V and a set of **edges** E, which connect pairs of vertices. It is formally defined as G=(V,E).

## Different Types of Graphs (with Examples)

1. **Simple Graph:** An undirected graph with no loops (edges connecting a vertex to itself) and no multiple edges (more than one edge between the same pair of vertices).
   - *Example:* The graph G1 (a square with a diagonal) shown on Page 6 is a simple graph.
2. **Directed Graph (Digraph):** A graph where the edges have a direction, usually indicated by an arrow. The edges are ordered pairs of vertices.
   - *Example:* A street map with one-way roads.
3. **Weighted Graph:** A graph where each edge is assigned a numerical value called a **weight** (or cost), often representing distance, time, or capacity.
   - *Example:* The graphs used for Minimum Spanning Tree problems (Questions 11 and 12).
4. **Complete Graph (Kn):** A simple undirected graph where every distinct pair of vertices is connected by a unique edge.
   - *Example:* K4 is a graph with 4 vertices where every vertex is connected to the other 3.
5. **Bipartite Graph:** A graph whose vertices can be divided into two disjoint and independent sets, V1 and V2, such that every edge connects a vertex in V1 to one in V2. No edges exist within V1 or V2.
   - *Example:* The graph H (the hourglass shape) from question 21 is a bipartite graph (v1,v2 in one set, v3,v4 in the other).