DrillBit

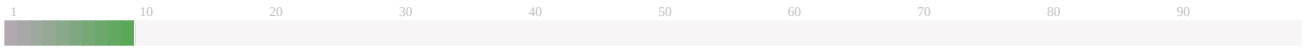
The Report is Generated by DrillBit Plagiarism Detection Software

Submission Information

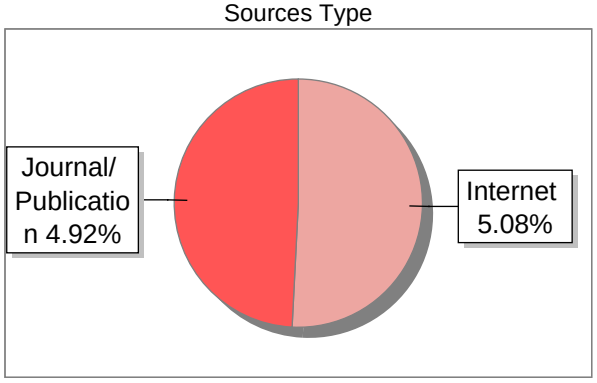
Author Name	TIRTHARAJ SAPKOTA
Title	MATHEMATICAL FOUNDATION OF COMPUTER SCIENCE
Paper/Submission ID	3029593
Submitted by	librarian.adbu@gmail.com
Submission Date	2025-01-24 18:02:26
Total Pages, Total Words	101, 28384
Document type	Others

Result Information

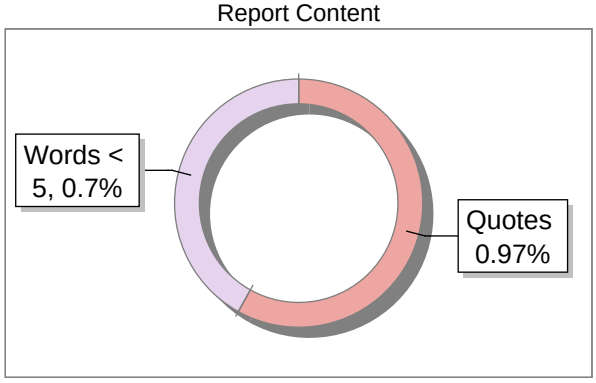
Similarity **10 %**



Sources Type



Report Content



Exclude Information

Quotes	Excluded
References/Bibliography	Excluded
Source: Excluded < 5 Words	Excluded
Excluded Source	**0 %**
Excluded Phrases	Not Excluded

Database Selection

Language	English
Student Papers	Yes
Journals & publishers	Yes
Internet or Web	Yes
Institution Repository	Yes

A Unique QR Code use to View/Download/Share Pdf File

DrillBit

DrillBit Similarity Report

10	95	A
SIMILARITY %	MATCHED SOURCES	GRADE

A-Satisfactory (0-10%)
B-Upgrade (11-40%)
C-Poor (41-60%)
D-Unacceptable (61-100%)

LOCATION	MATCHED DOMAIN	%	SOURCE TYPE
1	pdsaiitm.github.io	<1	Internet Data
2	eceatglance.files.wordpress.com	<1	Publication
3	docplayer.net	<1	Internet Data
4	mu.ac.in	<1	Publication
5	testbook.com	<1	Internet Data
6	pdfcookie.com	<1	Internet Data
7	translate.google.com	<1	Internet Data
8	Handbook of Dynamical Systems Volume 1 Chapter 1 Principal struc	<1	Publication
9	ggu.ac.in	<1	Publication
10	eptcs.web.cse.unsw.edu.au	<1	Publication
11	pdfcookie.com	<1	Internet Data
12	www.thefreelibrary.com	<1	Internet Data
13	www.researchgate.net	<1	Internet Data
14	fdokumen.id	<1	Internet Data

15	repository.up.ac.za	<1	Publication
16	pdfcookie.com	<1	Internet Data
17	users.cs.utah.edu	<1	Publication
18	Student Thesis Published in HAL Archives	<1	Publication
19	moam.info	<1	Internet Data
20	www.arxiv.org	<1	Publication
21	doc.sagemath.org	<1	Publication
22	www.freepatentsonline.com	<1	Internet Data
23	www.arcjournals.org	<1	Publication
24	pdfcookie.com	<1	Internet Data
25	www.geeksforgeeks.org	<1	Internet Data
26	www.nsec.ac.in	<1	Publication
27	moam.info	<1	Internet Data
28	pdfcookie.com	<1	Internet Data
29	www.arxiv.org	<1	Publication
30	seejph.com	<1	Publication
31	docplayer.net	<1	Internet Data
32	moam.info	<1	Internet Data
33	moam.info	<1	Internet Data

34	logicmojo.com	<1	Internet Data
35	moam.info	<1	Internet Data
36	THE EXPECTED NUMBER OF COOPERATORS IN THE N-PERSON PRISONERS DILEMMA WITH RANDO by K-2008	<1	Publication
37	jgaa.info	<1	Publication
38	downloads.hindawi.com	<1	Publication
39	Optimal gathering of oblivious robots in anonymous graphs and its application on by D-2017	<1	Publication
40	pdfcookie.com	<1	Internet Data
41	pgadmissions.iiit.ac.in	<1	Publication
42	Quaternions The hypercomplex number system by Pulver-2008	<1	Publication
43	www.vedantu.com	<1	Internet Data
44	An Extension of the Birthday Problem to Exactly k Matches by Rober-1986	<1	Publication
45	Chip-Firing Games on Directed Graphs by Ander-1992	<1	Publication
46	The duality of option investment strategies for hedge funds by Jos-2007	<1	Publication
47	www.eneuro.org	<1	Internet Data
48	A new hybrid approach to improve the efficiency of homomorphic matching for grap by Kong-2019	<1	Publication
49	dochero.tips	<1	Internet Data
50	fastercapital.com	<1	Internet Data
51	voccedu.org	<1	Publication

52	www-personal.umich.edu	<1	Publication
53	iim-cat-questions-answers.2iim.com	<1	Internet Data
54	moam.info	<1	Internet Data
55	qdoc.tips	<1	Internet Data
56	www.readbag.com	<1	Internet Data
57	www.jec.senate.gov	<1	Publication
58	Hamilton decompositions of regular expanders A proof of Kellys conjecture for by Khn-2013	<1	Publication
59	moam.info	<1	Internet Data
60	IEEE 2015 11th International Conference on Signal-Image Technology by	<1	Publication
61	web.stonehill.edu	<1	Internet Data
62	-intersecting families by Lee-2019	<1	Publication
63	asbmr.onlinelibrary.wiley.com	<1	Internet Data
64	Infinite Hamiltonian paths in Cayley diagraphs of hyperbolic symmetry by Dougla-1995	<1	Publication
65	moam.info	<1	Internet Data
66	www.frontiersin.org	<1	Publication
67	arxiv.org	<1	Publication
68	china-cee.eu	<1	Internet Data
69	Error Corrections in a Sampling Sliding Along a Pulse Envelope Based o by V-2002	<1	Publication

70	moam.info	<1	Internet Data
71	pdfcookie.com	<1	Internet Data
72	pdfcookie.com	<1	Internet Data
73	Provenance analysis for logic and games by Grdel-2020	<1	Publication
74	pub.epsilon.slu.se	<1	Publication
75	IEEE 2011 18th IEEE International Conference on Image Processing (IC	<1	Publication
76	arxiv.org	<1	Publication
77	Feynman Functional Integrals for Systems of Indistinguishable Particles by Laidlaw-1971	<1	Publication
78	moam.info	<1	Internet Data
79	pdfcookie.com	<1	Internet Data
80	repositorio.uchile.cl	<1	Publication
81	www.livoniapublicschools.org	<1	Publication
82	www.proprofs.com	<1	Internet Data
83	aprenderly.com	<1	Internet Data
84	pdfcookie.com	<1	Internet Data
85	Some recent results in the analysis of greedy algorithms for assignmen by Ulric-1994	<1	Publication
86	www.arxiv.org	<1	Publication
87	A preliminary phylogeny for rudist bivalves sifting clades from grades by Skelton-2000	<1	Publication

88	digitalknowledge.cput.ac.za	<1	Publication
89	ejournal.ipdn.ac.id	<1	Internet Data
90	fujipress.jp	<1	Internet Data
91	INFORMATICSwww.jucs.org	<1	Publication
92	runestone.academy	<1	Internet Data
93	www.arxiv.org	<1	Publication
94	www.ijtrd.com	<1	Publication
95	www.jemds.com	<1	Publication

CAOMF0043: MATHEMATICAL FOUNDATION OF COMPUTER SCIENCE

Unit 1: Fundamentals of Mathematical Logic

1.0 Introduction and Unit Objectives: The term logic is derived from the Greek word "logos" which means "reason". Mathematical logic is an area of mathematics that deals with reasoning within mathematical context. It basically consists of three steps i.e. analyzing a statement, derive the conclusion and verify if the conclusion is logically sound. Statements are sentences which may be true or false but cannot be both. Mathematical logic is applicable in many fields like computer science, philosophy, mathematics, linguistics, electrical engineering, artificial intelligence. In the computer science it may be used in designing algorithms, understanding complex data structures, understanding machine learning etc. Greek philosophers like Aristotle were the first to explore the basic principles of reasoning in early days of its evolution.

In this unit, the learners will study about the basic building blocks of mathematical logic like statements, connectives, well defined formulas and truth tables.

Unit Objectives: On completion of this unit, the students will be able to

1. Understand the basics of Logic like statements, notations and connectives.
2. Construct logical expressions.
3. Learn to evaluate the logical statements and tautologies.
4. Enhance their Logical thinking capacity and be creative.

1.1 **Statements and notations, Logical Connectives and Truth Tables**: Statement is a declarative sentence which may be true or may be false, but cannot be both. Statement is also termed as proposition. For example

a. 2+4=9 (False) is a proposition.

b. 5-2=3 (True) is a proposition.

c. " Who are you?" Since the response cannot be stated in True or False, it cannot be a proposition (Response cannot be stated in True/False)

d. X>15. As the given expression depends on the variable x to be true or false it cannot be a proposition.

To simplify expressions, propositions are often represented using propositional variables like X, Y, Z, and so on. The truth values of these propositions are indicated by T for True and F for False. For example, in the statement "X: The sum of two even numbers is always even," X serves as the propositional variable representing the proposition. Using propositional variables helps make concepts easier to comprehend.

Logical notations are symbols used to symbolize operations, relationships in a concise and formal

way. Few notations in mathematical logic:

¬: Negation (NOT)
∧: Conjunction (AND)
∨: Disjunction (OR)
→: Implication (If...Then)
↔: Bi-conditional (If and only if)

These are only few notations used. In the following sections we will discuss more notations with appropriate example.

Connectives and Truth Tables: Connectives are logical operators used to combine simple propositions. When simple propositions are combined they are called compound propositions. At this point, it is better to define truth table because it is fundamental tool in understanding logical reasoning and proposition. A **truth table** is a mathematical table used in logic to display all possible truth values of a logical expression or proposition based on its inputs. We can define truth table as a tool in logic that systematically displays all possible arrangements of truth values for logical statements and their resulting outcomes based on logical operators. The learners will find many examples of truth table in the following sections.

Connectives are divided into two categories as Fundamental and Derived. Fundamental connectives are primary connectors essential for constructing logical expressions where derived connectives are derived from the fundamental connectives that offer more logical operation.

Fundamental Logical Connectives: Following are classified under fundamental connectives:

1. **Conjunction (AND):** Conjunction of two propositions denoted by p ∧ q is a proposition that is true whenever both p and q are true, otherwise it is false.

   Example .11: Consider the propositions p and q given as
   p: Today is Sunday
   q: Today is holiday.
   p ∧ q: Today is Sunday and holiday.
   Table 1.1 shows truth table for conjunction.

Table 1.1: AND

P	q	p ∧ q
T	T	T
T	F	F
F	T	F
F	F	F

2. **Disjunction (OR):** If p and q are two propositions, then the proposition p or q denoted by p ∨ q is called disjunction, which false when both p and q are false, otherwise it is true.
   Example 1.2: If we have two propositions p: He is intelligent and q: He is diligent, then

disjunction of p and q is 'He is intelligent or diligent'. Table 1.2 shows truth table for disjunction.

Table 1.2: OR

p	q	p ∨ q
T	T	T
T	F	T
F	T	T
F	F	F

3. **Negation (NOT):** Let p be a proposition. Then negation of p is denoted by ¬p or ~p (not p). It reverses the truth value of a statement. In propositional logic, the negation of p is true if p is false, and false if p is true. If p represents 'sky is blue' then ¬p represents 'sky is not blue'. Table 1.3 shows truth table for Negation.

Table1.3: NOT

P	¬p
T	F
F	T

4. **Implication (If-then):** Given two propositions p and q, then the implication of p and q read as 'if p then q' or 'p implies q' denoted by p→ q is false when p is true and q is false, otherwise it is true. Here p is called the hypothesis or antecedent or premise and q is called conclusion or consequence. It is also referred to as conditional statement. Table 1.4 shows truth table for Implication.

Table1.4: Implication

P	q	p → q
T	T	T
T	F	F
F	T	T
F	F	T

Example 1.3: Let p be "It is raining " and q be "The ground is wet". The implication p→q means "If it is raining, then the ground is wet". Table 1.4 shows truth table for Implication.

Inverse, Converse and Contra-positive are some of the variation of Implications.

a. The proposition ¬p → ¬q is called the inverse of p→q. For implication "If it is raining, then the road is wet", its inverse would be "if it is not raining, then the road is not wet".
b. The proposition q → p is called the converse of p→q. For implication "If it is raining, then the road is wet", its converse would be "If the road is wet, then it is raining".
c. The proposition ¬q → ¬p is called contrapositive of p→q. For implication "If it is raining, then the road is wet", its contrapositive would be "If the road is not wet, then it is not raining"

5. **Bi-conditional (if and only if):** A bi-conditional statement is a logical statement of the form 'p↔ q' which is read as "p if and only if q. A bi-conditional statement 'p↔ q' is true when both p and q have the same truth values, otherwise it is false. Table 1.5 shows the truth table for Bi-Conditional statement.

Table1.5: Bi-Conditional

p	q	p↔ q
T	T	T
T	F	F
F	T	F
F	F	T

Example:

1. "2 = 3 if and only if 3 = 4" is true since both conditions are false.
2. "A polygon is a hexagon if and only if it has six sides" is true if both conditions are true.

Derived Connectives: Following are classified under derived connectives:

1. NAND (Not AND): It means Negation after AND of two propositions i.e. ¬(A∧B). Assume that p and q be two propositions. Then NAND of p and q is false when both p and q are true, otherwise it is true. It is denoted by ↑. Table 1.6 shows the truth table for NAND.

Table1.6: NAND

p	Q	p↑q
T	T	T
T	F	F
F	T	F
F	F	T

2. NOR (Not OR): It means Negation after OR of wo propositions

i.e.¬(AVB). Assume that p and q be two propositions. Then NOR of p and q is true when both p and q are false, otherwise it is false. It is denoted by ↓. Table 1.7 shows the truth table for NOR

Table1.7: NOR

p	Q	p↓q
T	T	F
T	F	F
F	T	F
F	F	T

3. Exclusive OR (XOR): Given two propositions p and q, the proposition exclusive OR (XOR) of p and q is a proposition that is true whenever p is true or q is true, but not both and vice-versa. It is denoted by ⊕. Table 1.8 shows the truth table for XOR.

Table1.8: XOR

p	q	p⊕q
T	T	F
T	F	T
F	T	T
F	F	F

Solved Examples:

Q1. Consider the propositions" p: He is rich", "q: He is generous". Write propositions which combines the proposition p and q using conjunction, disjunction and negation.

Solution:

a. p ∧ q (Conjunction): He is rich and generous.
b. p ν q (Disjunction): He is rich or generous.
c. ¬p: He is not rich / It is false that he is rich
d. ¬q: He is not generous/ It is false that he is generous.

Q2. Generate the truth table for the following

a. A⊕B⊕C b. A ↑B↑C

Solution: a. A⊕B ⊕C

A	B	C	A⊕B	A⊕B⊕C
T	T	T	F	T
T	T	F	F	F
T	F	T	T	F
T	F	F	T	T
F	T	T	T	F
F	T	F	T	T
F	F	T	F	T
F	F	F	F	F

b. A ↑B↑C

A	B	C	A ↑B	A ↑B↑C
T	T	T	F	T
T	T	F	F	T
T	F	T	T	F
T	F	F	T	T
F	T	T	T	F
F	T	F	T	T
F	F	T	T	F
F	F	F	T	T

Q3. Show that the implication p→q and its contrapositive are logically equivalent.

Solution: We know that the contrapositive of p→q is given by ¬q → ¬p. We will verify the equivalence of these propositions using truth table.

P	q	p→q	¬q	¬p	¬q → ¬p
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

From the above table, if we compare the columns p → q and ¬q → ¬p, we can say that implications and its contrapositive are logically equivalent as they have same values for each combination of propositions.

Note: *While constructing the truth table, the number of unique combinations of propositions is given by the expression* $2^{\text{no of input variables}}$*. For a compound propositions having 2 and 3 variables will have* $2^2=4$ *and* $2^3=8$ *combinations of propositions respectively.*

1.2 **Well-formed formulas, Tautology and Contradiction:** In previous sections, we have studied about propositional variables and logical connectives like AND, OR, NOT. In logic we need expressions to represent complex propositions which are obtained by combining simple propositions. The expressions are going to be called well-formed formulas (wffs). They are syntactically correct expressions constructed using variables, logical connectives and parenthesis that follow some defined rules. Wffs are essential for ensuring that logical statements are meaningful and evaluated for truth.

Components of well-formed formula:

Variables: They are atomic propositions, usually denoted by capital letters like P, Q, R.

Logical Connectives: Operators such as ¬(NOT), ∧ (AND), ∨ (OR), → (IMPLIES), and ↔ (IFF).

Parentheses: Used to group expressions and clarify the order of operations.

Rules to form WFF:

1. A single variable (e.g., P) is a WFF.

2. If P is a WFF, then ¬P is also a WFF.

3. If P and Q are WFFs, then combinations like (P∧Q), (P∨Q), P→Q, P↔Q are WFFs.

4. Parentheses must enclose compound statements to maintain structure.

For example

1. (P ∨ Q →Not a WFF (missing closing parenthesis)
2. ¬(P→Q) → WFF (correctly formed).
3. P ∧ ∨ Q → Not a WFF (misuse of connectives).

Consider the expression

1. P∧Q∨R

2.¬P→Q∧R. These two expressions are not well formed formulas because they are not enclosed by parenthesis. We have to add parentheses to make the following into valid WFFs like P∧ (Q∨R) and (¬P) →(Q∧R).

Tautology: A compound proposition that is always true, regardless of the truth values of simple propositions that are contained in it is called tautology. For example, we can verify that P ∨(¬P) is a tautology using truth table as shown in Table 1.9.

Table 1.9: Tautology

P	¬P	P ∨(¬P)
T	F	T
F	T	T

Table 1.10: Contradiction

A compound proposition that is always false, no matter what the truth values of simple propositions that are contained in it is called **contradiction or absurdity**. The proposition $P \wedge \neg P$ is a contradiction.

P	¬P	P ∧(¬P)
T	F	F
F	T	F

Tautology and contradiction are fundamental concepts in logic, serving important roles in understanding, analyzing, and constructing logical systems. They are essential in ensuring logical consistency and reliability across discipline.

1.3 **Unit Summary:** This unit provides foundational concepts in logic, including statements, notation, connectives, well-formed formulas, and truth tables, enabling effective evaluation of logical expressions and understanding of tautologies. This unit will serve as the foundation for all the upcoming units. Therefore, it is essential for all the learners to understand all the concepts provided in this unit very minutely.

1.4 Check Your Progress:

1. Let p be "It is hot today" and q be "The temperature is 40^0". Write the equivalent simple sentences for the following expression.
   a. ¬P b. ¬ (P∨Q) c. ¬ (P∧ Q) d. ¬ (¬ Q) e. ¬P ∧¬Q.
2. Translate the following into propositional form:
   a. If it is not raining and I have time, then I will go to a movie.
   b. It is raining and I will not go to a movie.
   c. I will go to a movie only if it is not raining.
3. Show that proposition ¬P → ¬ Q is not equivalent to Q → P and ¬P → ¬Q
4. Prove that P ↔ Q is equivalent to (P → Q) ∧ (Q → P).
5. Prove that ((P → Q) ∨ (Q → P)) is a tautology.
6. Prove that (P → Q) ∧ ¬ Q is a contradiction.
7. State the converse, contrapositive for the implications
   a. If it rains tonight, then I will stay at home.
   b. I will go to swimming, whenever it is a sunny day.

Unit 2: Logical Equivalence

2.0 Introduction and Unit Objectives: In previous unit, we have studied about of logical statements, notations and connectors essential for constructing logical expressions and truth evaluations. These topics are the fundamental concept in Mathematical Logic. In this unit, we will learn about foundational principles of Mathematical Logic, focusing on its structure and applications in detail. For comparing statements and determine equivalence using truth values, Logical equivalence is very necessary. Various laws of Logic like De Morgan's Law are used to establish formal rules for reasoning, making sure logical arguments are consistent. Logical implications are very important in establishing formal connection between two statements. Many theorems and algorithms in mathematics are expressed using implications, making things simple for manipulation and evaluation. Normal Forms helps in standardizing various expressions which assist in better analysis and application in domains like designing algorithms. Quantifiers are used in making the logical expression precise which expands the scope of mathematical logic. These concepts are necessary in understanding logic as tool for creative thinking and problem solving. So, this unit will help learners to gain knowledge on several important topics helping them to further extend and improve their problem-solving skills using Mathematical Logic.

Unit Objectives: On completion of this unit, the students will be able to

1. Understand the concept of logical equivalence.
2. Gain knowledge on fundamental laws applied in logical equivalence.
3. Use logical Implications to establish connection between two statements
4. Understand the concept of Normal forms and their application in domains like algorithm design.
5. Use quantifiers in logical expression to make them precise and further enhanced their problem-solving ability.

2.1 Logical Equivalence, Laws of Logic and Logical Implications:

Logical Equivalence: Given two propositions that differ in their syntax but have exactly same semantic are considered as equivalent. We can test the equivalence of these propositions by constructing the truth table for both P and Q and checking every combination of truth values for positional variables.

Table2.1: Logical Equivalence

P	Q	P → Q	¬P	¬P ∨Q
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

If truth value of P is same as truth value of Q, then P and Q are logically equivalent. Such constructs are quite useful in mathematical reasoning where we can substitute one proposition with other to form mathematical arguments. For example, the statements P → Q and ¬P ∨Q is logically equivalent which is shown in Table 2.1. We can observe that the columns P → Q and ¬P ∨ Q has same truth values for all combinations of P and so they are logically Equivalent.

There are few ways to define Logical Equivalence. But it is better to define using tautology because it is more practical and directly establishes the consistency of truth values. The compound propositions P and Q are logically equivalent if the proposition P ↔ Q (the bi-conditional) is a tautology and is denoted as P ≡ Q or P ⇔Q.

Laws of Logic: Some equivalences are useful for deducing other equivalences. We call them identities or laws of logic. These identities can be used to simplify formulas. Following is a list of logical identities which are also referred as laws of logical equivalence.

Identity	**Description**
Identity Law	P ∧ T ⇔ P, P ∨ F ⇔ P
Domination Law	P ∧ F ⇔ F, P ∨ T⇔ T
Idempotent Law	P ∧ P ⇔ P, P ∨ P ⇔ P
Double Negation Law	¬ (¬P) ⇔ P
Commutative Law	P ∧ Q ⇔ Q ∧ P, P ∨ Q ⇔ Q ∨ P
Associative Law	P ∨ (Q ∨ R) ⇔ (P ∨ Q) ∨ R, P ∧ (Q ∧ R) ⇔ (P ∧ Q) ∧ R
Distributive Law	P ∨ (Q ∧ R) ⇔ (P ∨ Q) ∧ (P ∨ R) P ∧ (Q ∨ R) ⇔ (P ∧ Q) ∨ (P ∧ R)
De Morgan's Law	¬ (P ∨ Q) ⇔ ¬ P ∧ ¬ Q, ¬ (P ∧ Q) ⇔ ¬ P ∨ ¬ Q
Absorption Law	P∧ (P ∨ Q) ⇔ P, P ∨ (P ∧ Q) ⇔ P
Law of Implication	P→Q⇔¬P∨Q

Solved Examples:

1. Use logical identities to verify that $(P \wedge Q) \vee (P \wedge \neg Q) \Leftrightarrow P$

   Solution:

   LHS $= (P \wedge Q) \vee (P \wedge \neg Q)$

   $= P \wedge (Q \vee \neg Q)$ (Using distributive law)

   $= P \wedge T$ ($(Q \vee \neg Q)$ is a tautology, so replaced by T)

   $= P$ (Using identity law)

   =RHS.

2. Use logical identities to verify that $(P \rightarrow Q) \wedge (R \rightarrow Q) \Leftrightarrow (P \vee R) \rightarrow Q$

   Solution:

   LHS $= (P \rightarrow Q) \wedge (R \rightarrow Q)$

   $= (\neg P \vee Q) \wedge (\neg R \vee Q)$ (Law of Implication)

   $= (Q \vee \neg P) \wedge (Q \vee \neg R)$ (Commutative Law)

   $= Q \vee (\neg P \wedge \neg R)$ (Distributive law)

   $= Q \vee \neg (P \vee R)$ (De Morgan's Law)

   $= \neg (P \vee R) \vee Q$ (Commutative Law)

   $= (P \vee R) \rightarrow Q$ (Law of Implications)

   =RHS

   Logical Implication: The law represented by $P \rightarrow Q \Leftrightarrow \neg P \vee Q$ is called the Implication Law in logic. It states that a conditional statement $P \rightarrow Q$ (if P, then Q) is logically equivalent to the disjunction $\neg P \vee Q$ (either P is false, or Q is true). The implication $P \rightarrow Q$ is false only when P is true and Q is false. Conversely, $\neg P \vee Q$ also holds in all cases except when P is true and Q is false. So it is clear that these two statements are logically equivalent. This law of implication is quite useful while discussing the normal forms in next section.

2.2 Normal Forms: We have already learned from the previous discussions that using truth table it is possible to check if two statements are logically equivalent. But when the number of input variable increases the complexity of constructing the truth table also increases. So to simplify things and bring clarity, we convert the statements into simpler form which are normally called normal or canonical forms. In mathematical logic, **normal forms** are standardized ways of writing logical formulas that are equivalent to the

original formula but follow specific structural rules. Normal forms are essential in logic because they simplify and standardize logical expressions, enabling easier manipulation, analysis, and computational processing. While discussing the normally forms we will use the term sum for disjunction (OR) and product for conjunction (AND).

At this point, it will be relevant to introduce two new terms which will help the learners in understanding the upcoming concepts. These two concepts are given below:

a. **Elementary product**: A product(AND) of variables and their negations in a formula is called elementary product. Each variable in the expression appears exactly once, either in its original form (P) or its negated form (¬P). For example, ¬ P ∧ P ∧ ¬Q, ¬ Q ∧ ¬ P ∧ P.
b. **Elementary sum**: A sum (OR) of variables and their negations in a formula is called elementary sum. Each variable in the expressions appears exactly once. For example, ¬ P ∨ P ∨ ¬Q, ¬ Q ∨ ¬ P ∨ P.

There are two types of Normal Forms

a. Disjunctive Normal Form (DNF)
b. Conjunctive Normal Form (CNF)

Disjunctive Normal Form (DNF): A formula which is equivalent to the given formula and consist of sum of elementary products is called disjunctive normal form of the given formula. For example, the formula (¬ P ∧ Q) ∨ (P ∧ Q) is in disjunctive normal form.

Conjunctive Normal Form (CNF): A formula which is **equivalent** to the given formula and consist of product of elementary sums is called conjunctive normal form of the given formula. For example, the formula (¬ P ∨ Q) ∧ (P ∨ Q) is in conjunctive normal form.

Solved Examples:

1. Obtain the disjunction normal form of the formula P ∧ (P→Q)

Solution: Given, P ∧ (P→Q)

= P ∧ (¬P ∨Q) (Implication Law)

= (P ∧ ¬P) ∨ (P ∧Q) (Distributive Law)

This is in the form of sum of elementary products. Hence it is in disjunctive normal form.

2. Obtain the conjunctive normal form of the formula (P→Q) ∧ (Q ∨ (P ∧R)).

Solution: Given, (P→Q) ∧ (Q ∨ (P ∧R))

$= (\neg P \vee Q) \wedge (Q \vee (P \wedge R))$ (Implication Law)

$= (\neg P \vee Q) \wedge (Q \vee P) \wedge (Q \vee R)$ (Distributive Law)

This is in the form of product of elementary sum. Hence it is in conjunctive normal form.

2.3 Quantifier, Universal and Existential Quantifiers:

Quantifiers: Quantifiers are symbols in logic that indicate the scope of a statement by specifying how many elements in a domain satisfy a given property. They refer to quantities like some or all and indicates how frequently certain statement is true. There are two types of quantifiers namely universal and existential quantifier.

Universal quantifier: The phrase "for all" denoted by ∀, is called universal quantifier.
Example: ∀x, P(x) means "P(x) is true for all x."

Existential quantifier: The phrase "there exist" denoted by ∃, is called existential quantifier.
Example: ∃x, P(x) means "There is at least one x for which P(x) is true."
In the next unit, while discussing predicates we will study in detail about these quantifiers.

2.4 Unit Summary:

This unit introduces key principles of logical equivalence, laws of logic, implications, normal forms, and quantifiers, equipping students to analyze and construct logically valid expressions. These concepts will further extend the abilities of the learner in the domain of mathematical logic and enhance their analytical and problem solving skills.

2.5 Check Your Progress:

1. Use De Morgan's laws to rewrite $\neg (P \wedge Q)$ and $\neg (P \vee Q)$ in an equivalent form.
2. Verify the equivalence $(P \vee Q) \wedge (P \vee R)$ and $P \vee (Q \wedge R)$ using the distributive law.
3. Show that $P \rightarrow Q$ is logically equivalent to $\neg P \vee Q$.
4. If $P \rightarrow Q$ and $Q \rightarrow R$, prove that $P \rightarrow R$.
5. Simplify $(P \rightarrow Q) \wedge (\neg Q \rightarrow R)$.
6. Convert $\neg(P \vee Q) \wedge (P \Longrightarrow R)$ into Conjunctive Normal Form (CNF).
7. Express $(P \vee Q) \wedge (Q \vee \neg R)$ in Disjunctive Normal Form (DNF).
8. Identify whether the formula $(P \vee \neg Q) \wedge (R \vee S)$ is in CNF or DNF. Justify your answer.
9. Prove De Morgan's Law using truth table.
10. What is the significance of quantifiers in Logic? Explain.

Unit 3: Predicates

3.0 Introduction and Unit Objectives: The logic on the basis of its representation capacity are of two types: propositional logic and predicate logic. **Propositional logic** also known as sentential logic is the branch of logic that studies way of joining or modifying propositions to form complex propositions as well as logical relationship. **Predicate logic** is an extension of propositional logic that involves predicates, variables, and quantifiers to express more complex statements. It allows us to make statements about specific objects within a domain, using quantifiers. **Propositional logic** deals with simple statements that are either true or false, without any internal structure (e.g., "The sky is blue"). **Predicate logic** adds depth by allowing statements about elements in a domain, expressed with predicates and variables (e.g., "For all x, if x is a student, then x studies"). Thus, predicate logic is more expressive and can handle more complex reasoning than propositional logic.

In this unit, the learners will be able understand predicates and their use in propositional logic, Identify free and bound variables within predicate logic expressions, Learn how quantifiers (∀,∃) impact variable binding, Use inference rules to construct valid arguments and proofs, Understand the concept of consistency in logical systems and Explore methods to ensure logical consistency in argumentation, Understand the steps involved in proving statements by contradiction and Analyze the validity of arguments derived through contradiction proofs.

Unit Objective: On completion of this unit, the students will be able to,

1. Understand how predicates are used in propositional logic for complex statements.
2. Distinguish between free and bound variables and learn the role of quantifiers.
3. Apply logical rules to derive valid conclusions from given premises.
4. Use proof by contradiction to establish the validity of statements.
5. To gain a comprehensive foundation for understanding and applying predicate logic effectively.

3.1 Predicative logic: It is a way of making logical statements about things, where we use **predicates** to describe properties or actions of objects. It's a more powerful tool than simple true or false statements (like in propositional logic) because it allows us to talk about specific objects and their characteristics.

A **predicate** is a declarative statement that involves variables and is commonly found in mathematical assertions and computer programs. It cannot be classified as true or false until the values of the variables are specified. Let us take an example- '12<23', this is a

propositional statement because it is true. But the statement ' y>23' is not a proposition because we can't say anything about it until a value for variable y is specified. We can say that any statement having a variable is not a proposition. The question is can we convert such a statement to proposition? The answer is yes.

In the above statement y>23 there are two parts one is the variable part called "**subject**" and another is relation part ">23" called "**predicate**". We can denote the statement "y>23" by P(y) where P is predicate ">23" and y is the variable. We also call P as a propositional function where P(y) gives value of P at y. Once value is assigned to the propositional function then we can tell whether it is true or false i.e. a proposition. For e.g. if we put the value of y as 3 and 45 then we can conclude that P (3) is false since 3 is not greater than 23 and p (45) is true since 45 is greater than 23.

We also can denote a statements with more than one variable using predicate like for the statement "x=y', we can write P (x,y) such that P is relation 'equals to'. Similarly, the statements with higher number of variables can also be expressed. Thus a predicate is a sentence that contains a finite number of variables and becomes a proposition when specific values are substituted for the variables.

Note: The logic involving predicates is called Predicate logic or Predicate calculus similar to logic involving propositions is Propositional logic or propositional calculus.

Solved Example:

1. If P(x): x+2<10, find the truth value of P(5) and P(9).

Solution: Given, P(x): x+2<10

When x=5, P (5):5+2<10, 7<10 (True)

When x=9, P (9): 9+2<10, 11<10 (false).

2. Let F(x,y): x=y+6. Find the truth value of F(1,5) and F(6,0)

Solution: Given F(x,y): x=y+6

For F (1,5) using x=1 and y=5 we get,

F (1,5): 1=5+6

1=11(false)

For F (5,0) using x=6 and y=0 we get,

F (6,0): 6=0+6

6=6 (true)

Therefore F (1,5) is false and F (6,0) is true for F(x,y): x=y+6.

Quantifiers: At this point it is relevant to revise the concept of Quantifiers with some additional information. Quantifiers are the tools that change the propositional function into a proposition. Construction of propositions from the predicates using quantifiers is called

quantification. The variables that appear in the statement can take different possible values which are collectively known as "Universe of Discourse" or "Universal set".

In the previous unit we already discussed about two types of quantifiers- universal and existential quantifier. The process of converting predicate into proposition using universal quantifier is called **universal quantification.** So, the universal quantification of P(x), denoted by ∀x, P(x) is a proposition where "P(x) is true for all the values of x in the universe of discourse". The process of converting predicate into proposition using existential quantifier is called **existential quantification.** So, the existential quantification of P(x), denoted by ∃x P(x) is a proposition where "P(x) is true for some values of x in the universe of discourse".

The universal quantification is conjunction of all the propositions that are obtained by assigning the value of the variable in the predicate whereas existential quantification is the disjunction of all the propositions that are obtained by assigning the values of the variable from the universe of discourse.

Solved Example:

1. Let Z, the set of integers, be the universe of discourse, find the truth values of the following statements.

   i. $\forall x \in Z, x^2=x$ ii. $\exists x \in Z, x^2=x$

Solution: Let P(x): $x^2=x$ then

∀x P(x) is false because for $2 \in Z$, P (2): $2^2 =2$ is false.

∃x P(x) is true because at least one proposition is true i.e. $1 \in Z$, P (1): $1^2=1$ is true.

Translating English Sentences into Logical Expression:

2. Translate 'not every integer is even' where the universe of discourse is set of integers.

Solution: Let P(x) denotes x is even. Then ¬ ∀x P(x) represents the given sentence "not every integer is even".

3.2 Free and bound variables: When the variable is assigned a value or is quantified, it is called bound variable. If the variable is not assigned a value or not bounded, then it is called free variable. Bound variables are restricted to specific values or conditions imposed by quantifiers, summation, or integration operators. They contrast with free variables, which are not constrained and can take any value within their domain.

Following are few examples of free and bound variables

a. P (x, z) has two free variables x and z.
b. In the expression: ∀x (x+y=z), x is a bound variable because it is within the scope of the quantifier ∀x. y and z are free variables since they are not bound by any quantifier.
c. In the expression P(x)∧Q(y), both x and y are free variables as they are not bounded by any quantifier.

Free and bound variables play a crucial role in understanding and interpreting mathematical and logical expressions.

3.3 Rules of Inference: In logical reasoning a certain number of propositions are assumed to be true and based on that assumptions some other propositions are derived or deduced or inferred. The proposition that are assumed to be true are called hypothesis or premises and that are derived are called conclusion. The process of deriving a conclusion based on the assumption of the premise is called a **valid argument.**

To draw conclusion from the given premise we must be able to apply some well-defined steps that helps reaching the conclusion. These steps of reaching the conclusion are provided by the rules of inference. These rules of inference are simply tautologies in the form of implications i.e. $P \rightarrow Q$. For example, $P \rightarrow (P \vee Q)$ is such a tautology. We write this as

$$\frac{P \text{ (Premise/Hypothesis)}}{\therefore P \vee Q \text{ (Conclusion)}}$$

This is read as if we know that P is true then $P \vee Q$ is also true.

Here some of the rules of inferences are given below:

Rule 1: Modus Ponens (or Law of Detachment):
Whenever two propositions p and $p \rightarrow q$ are both true then we confirm that q is true. We write this rule as

$$\frac{\begin{array}{l}P \\ P \rightarrow Q\end{array}}{\therefore Q}$$

This rule is valid rule of inference because the implication $[p \wedge (p \rightarrow q)] \rightarrow q$ is a tautology.

Rule 2: Hypothetical Syllogism (Transitive Rule):
Whenever two propositions $P \rightarrow Q$ and $Q \rightarrow R$ is both true then we confirm that implication $P \rightarrow R$ is true. We write this rule as:

$$\frac{\begin{array}{l}P \rightarrow Q \\ Q \rightarrow R\end{array}}{\therefore P \rightarrow R}$$

This rule is valid rule of inference because the implication $[(p \rightarrow q) \wedge (q \rightarrow r)] \rightarrow (p \rightarrow r)$ is a tautology.

Rule 3: Addition: Due to the tautology $p \rightarrow (P \vee Q)$, following is a valid rule of inference.

$$\frac{P}{\therefore (P \vee Q)}$$

Rule 4: Simplification: Due to the tautology $(P \wedge Q) \rightarrow P$, following is a valid rule of inference.

$$\frac{(P \wedge Q)}{\therefore P}$$

Rule 5: Conjunction:

Due to the tautology $[(P) \wedge (Q)] \rightarrow (P \wedge Q)$, the following is a valid rule of inference.

$$\begin{array}{l} P \\ Q \\ \hline \therefore (P \wedge Q) \end{array}$$

Rule 6: Modus Tollens: Due to the tautology $[\neg Q \wedge (P \rightarrow Q)] \rightarrow \neg P$, the following is a valid rule of inference.

$$\begin{array}{l} \neg Q \\ P \rightarrow Q \\ \hline \therefore \neg P \end{array}$$

Rule 7: Disjunctive Syllogism: Due to the tautology $[(P \vee Q) \wedge \neg P] \rightarrow Q$, the following is a valid rule of inference

$$\begin{array}{l} P \vee Q \\ \neg P \\ \hline \therefore Q \end{array}$$

Rule 8: Constructive Dilemma:

Due to the tautology $[(P \rightarrow Q) \wedge (R \rightarrow S)] \wedge (P \vee R) \rightarrow (Q \vee S)$, then following is a valid rule of inference.

$$\begin{array}{l} (P \rightarrow Q) \wedge (R \rightarrow S)] \\ P \vee R \\ \hline \therefore Q \vee S \end{array}$$

Rule 9: Destructive Dilemma:

Due to the tautology $[(P \rightarrow Q) \wedge (R \rightarrow S) \wedge (\neg Q \vee \neg S)] \rightarrow (\neg P \vee \neg R)$, then following is a valid rule of inference.

$$\begin{array}{l} (P \rightarrow Q) \wedge (R \rightarrow S) \\ \neg Q \vee \neg S \\ \hline \therefore (\neg P \vee \neg R) \end{array}$$

Rule 10: Resolution:

Due to the tautology $[(P \vee Q) \wedge (\neg P \vee R)] \rightarrow (Q \vee R)$, then following is a valid rule of inference.

$$\begin{array}{l} (P \vee Q) \\ (\neg P \vee R) \\ \hline \therefore (Q \vee R) \end{array}$$

Solved Example:

1. Check the validity of the following arguments.

   If Mark has completed M.Tech or PhD then he is assured of a good position. If Mark is assured a good position, then he is happy. Mark is not happy. So Mark has not completed PhD.

Solution: Let P denotes "Mark has completed M. Tech"

Let Q denotes "Mark has completed PhD"

Let R denotes "Mark is assured of a good position"

Let S denoted "Mark is happy"

The given premises can be written as

1. $P \vee Q \rightarrow R$
2. $R \rightarrow S$
3. $\neg S$

The conclusion is $\neg Q$

Now we need to perform the derivation as follows

1. $P \vee Q \rightarrow R$ (Premise 1)
2. $R \rightarrow S$ (Premise 2)
3. $P \vee Q \rightarrow S$ (Applying Transitive Rule on line 1 and 2)
4. $\neg S$ (Premise 3)
5. $\neg(P \vee Q)$ (Applying Modus Tollens on line 3 and 4)
6. $\neg P \wedge \neg Q$ (Applying De Morgan's law on line 5)
7. $\neg Q$ (Simplification)

means mark has not completed PhD which is required conclusion.

2. If today is Sunday, then today is rainy day and if today is rainy day, then it is wet today.

By transitivity rule we can verify that It is wet today.

3.4 Consistency and Proof of Contradiction:

Consistency: In mathematical logic, consistency means that a system has no contradictions. A contradiction happens when a statement p and its opposite ¬p (not p) can both be proven. If a system is consistent, this will never happen. Consistency is important because it confirms that the system is reliable for reasoning. Let us consider two statements " All dogs are mammals" and "All mammals are warm-blooded". They don't contradict each other, so the whole system can be considered consistent.

Consider two more statements- "All man are tall" and "Some man are short". These two statements contradict each other so they are not consistent. If a system is inconsistent, then it cannot be trusted for validity of statements.

Proof of contradiction: It is a logical method used to prove the truth of a proposition by assuming the opposite of the statement is true and demonstrating that this assumption leads to a contradiction. When a contradiction occurs, it implies that the original assumption (the negation of the statement) is false, thereby proving that the statement itself must be true.

Steps of Proof by Contradiction:

1. Assume the negation (¬P) of the proposition (P).
2. Use logical reasoning and existing facts to derive consequences of ¬P.
3. Arrive at a contradiction (a statement that is both true and false, or contradicts established facts).
4. Conclude that the original proposition (P) must be true.

Solved example:

Prove that if n^3+5 is odd, then n is even

Solution: Let $P \rightarrow Q$: if n^3+5 is odd, then n is even

Suppose n^3+5 is odd and n is also odd.

Now n can be expressed as $n=2k+1$ for some positive k

$=8k^3 + 12\,k^2 + 6k + 1 + 5$

$=8k^3 + 12\,k^2 + 6k + 6$

$=2(4k^3 + 6k^2 + 3)$, which is even

This contradicts our assumption that n^3+5 is odd.

3.5 Unit Summary: This unit introduces the fundamental concepts of predicate logic, a key area of formal logic that underpins much of mathematics and computer science. Predicate logic helps in formulating complex statements more precisely. Distinction between free and bound variables were discussed which will help the learner to understand the transformation of the logical expression. The rules of inferences were dealt with which will help to guide the reasoning process in formal system. Consistency and Proof of contradiction were discussed that will help the learner in confirming the truth of the original proposition. the unit provides a detailed exploration of these logical tools, which are vital for constructing valid arguments and proofs in various fields, including mathematics, computer science, and philosophy.

3.5 Check Your Progress

1. What is a predicate, and how does it differ from a propositional logic statement?
2. What are free and bound variables in predicate logic, and why are they important?
3. What are the rules of inference in predicate logic, and how are they applied?
4. Express the following statements using predicates and quantifiers.
    a. Every student in the class has studied discrete mathematics
    b. Some numbers are not Prime.
    c. There is a student who likes DBMS but not Java.
    d. There is a student in our university who can speak Hindi and who is good in C++.
5. Prove that "If a number is divisible by 6, then it is divisible by both 2 and 3" using appropriate rules of inference.
6. Prove that "If it is either Monday or Tuesday, then today is not Wednesday" using Disjunctive Syllogism.
7. Validate the argument: If I go to swimming then I will stay in the sun too long. If I stay in the sun too long, then I will sunburn. Therefore, If I go swimming I will be sunburn.
8. If John play football too much he will score low in exams. John did not score low. Therefore, John did not play football.
9. Prove that Product of two odd integers is odd.
10. Prove that if n is an even integer and 3n+2 is also even integer, then n is even integer.

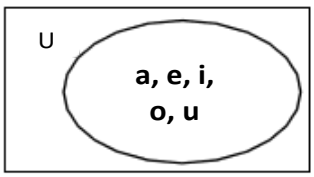
Unit 4: Introduction to set theory and discrete structures

4.0 Introduction and Unit Objectives: Mathematical foundations serve as the backbone of computer science and related disciplines. In this unit, we will delve into **Set Theory**, a critical component of discrete mathematics, which is fundamental to understanding abstract concepts and computational logic. Sets are the building blocks for defining relations, functions, graphs, and other discrete structures essential in computer science.

This introduction provides a basis for exploring how mathematical structures underpin algorithms, data structures, and reasoning systems. The unit emphasizes practical applications of sets and discrete structures in problem-solving and programming.

Unit Objective: On completion of this unit, the students will be able to

1. Understand the foundational concepts and terminology of Set Theory
2. Explore the laws and operations of sets, such as union, intersection, and complement.
3. Develop skills for representing and analyzing discrete structures like relations, graphs, and functions
4. Apply the knowledge of sets and discrete structures in solving computational problems



4.1 Concept of set theory: Set theory deals with the study of sets, which are collection of unique objects like numbers or letters. Set is a very essential concept in mathematics that is required in computer science and its applications. For e.g. if we are dealing with relations in database then they are sets ordered collection of elements, similarly we can view graph as a set. Set is a collection of zero or more objects (or elements or members), the elements need not be ordered. If we denote set by S and some element from the set by e then we say "e belongs to S" or "S contains e" or in symbol we can write e ∈ S. for e.g. V = {a, e, i, o, u} is a set of vowels and i ∈ V, if some object doesn't belong to the set we write it as "does not belong to" i.e. say x ∉ V.

Representations of Sets: Some of the ways of representing a set are:

a. **Listing of elements:** In the method all the elements of the set are listed. For example, set of vowels is {a, e, i, o, u}.

b. **Set builder form:** In this method set is described using properties of the members of the set is described for e.g. R = {x | x is a real number}

c. **Venn Diagram:** Venn diagram is the graphical representation of the set. For example, set of vowels can be represented as

In the above diagram the circle represents the set of vowels whereas the enclosed rectangle represents the "universe of discourse" or "universe".

Some Definitions

Subset: Set A is called subset of set B if every element of A is also an element of B.

A is **a proper subset** of B if A is a subset of B and there is at least one element in B that does not belongs to A. Symbolically, a subset is represented as $A \subseteq B$ and proper subset is denoted by $A \subset B$.

Superset: A superset refers to a set having all the elements of another set. If all the elements of set B are also the elements of set A, then A is superset of B is written as $A \supseteq B$.

Proper Superset: A proper superset (denoted by $A \supset B$) means that A contains all the elements of B, but A has additional elements that are not in B.

Equal Sets: If $A \subseteq B$ and $B \subseteq A$, then $A=B$. For example, $A=\{1,2,3,4,5\}$ and $B=\{2,5,4,1,3\}$ are equal sets.

Empty set: The set containing zero element is called empty set and denoted by $\varnothing$. It is also called null set. We have $\varnothing = \{\}$ but $\varnothing \neq \{\varnothing\}$.

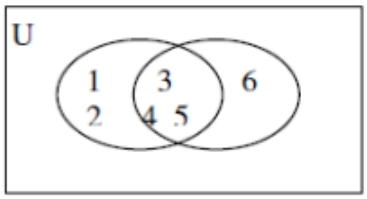
Power Set: For a set S, power set denoted by P(S) is the set that contains all the subsets of the set S. Symbolically we can write $P(S) = \{x \mid x \subseteq S\}$. For e.g. power set for the set $\{2, 3\}$ is $\{\varnothing, \{2\}, \{3\}, \{2,3\}\}$. The total elements in the power set of set containing n elements is $2_{|n|}$. Note that $\varnothing$ is member of all power set.

Set Operations

Union Operator: Union of set A and B is the set containing all elements that are either in A, in B, or in both. This is denoted by $A \cup B$.

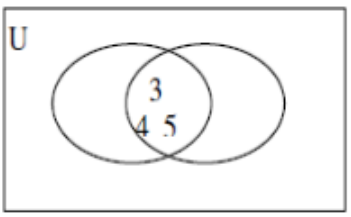
Symbolically, $A \cup B = \{x | x \in A \text{ or } x \in B\}$

Example: Let $A = \{1,2,3,4,5\}$ and $B= \{3,4,5,6,7\}$ Then $A \cup B = \{1,2, 3, 4, 5, 6,7\}$. This can be shown in Venn diagram as



Intersection Operator: The intersection of set A and B is the set having the common elements which is denoted by A∩B.

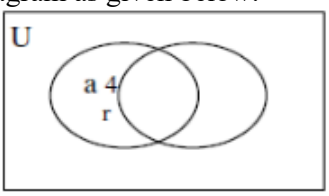
Symbolically, A∩B={x|x∈A and x∈B}. It can be represented as Venn diagram as the overlapping region between the two sets



Disjoint Set: If the intersection of the two sets is a null set then these sets are called disjoint (i.e., they do not have common) elements.

Set Difference: The difference of A and B is the set having all the elements that are in A but not in B, denoted by A−B. Symbolically, A−B= {x|x∈A and x∉B}

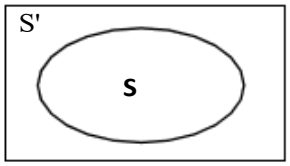
Example: A = {2, 4, a, r} and B = {1, 2, a, s, t} then A – B = {4, r}. This can be represented by a Venn diagram as given below:



Complement: The complement of a set S is denoted by U−S or S′, where U is the universal set. It is the set difference between the universal set U and the set S.

Symbolically, the complement is written as: S′={x|x∉S}

S′ is illustrated in the Venn diagram as the region outside of set S, within the universal set U.



Set Identities

The set identities that we learn here is similar to that of propositional logic.

$A \cap U = A$	Identity law
$A \cup \varnothing = A$	Identity law
$A \cap \varnothing = \varnothing$	Domination law
$A \cup U = U$	Domination law
$A \cap A = A$	Idempotent law
$A \cup A = A$	Idempotent law
$(A')' = A$	Complementation law
$A \cap B = B \cap A$	Commutative law
$A \cup B = B \cup A$	Commutative law
$(A \cap B) \cap C = A \cap (B \cap C)$	Associative law
$(A \cup B) \cup C = A \cup (B \cup C)$	Associative law
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$	Distributive law
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$	Distributive law
$(A \cap B)' = A' \cup B'$	De Morgan's law
$(A \cup B)' = A' \cap B'$	De Morgan's law

Example: Show $(A \cup B)' = A' \cap B'$.

We can denote $(A \cup B)' = \{x \mid x \notin A \cup B\} = \{x \mid \neg(x \in A \cup B)\}$

$= \{x \mid \neg(x \in A \vee x \in B)\} = \{x \mid x \notin A \wedge x \notin B)\}$

$= \{x \mid x \in A' \wedge x \in B')\} = \{x \mid x \in A' \cap B'\}$

$= A' \cap B'$

Cartesian Product: The Cartesian product of set A and B is a set containing all ordered pairs (a,b) where $a \in A$ and $b \in B$.

Symbolically, we can write as: $A \times B = \{(a,b) \mid a \in A \text{ and } b \in B\}$.

4.2 Representation of Discrete Structures: Discrete structures forms the basis or mathematical frameworks used to study various objects and their relationships that are distinct and countable. Unlike continuous systems, discrete structures deal with finite or countable sets of elements and their interactions. These structures are essential in designing various algorithms and computational systems, ensuring complex reasoning and problem-solving. Some of the discrete structures and the way they are represented in computer science are discussed below

Sets: They are collections of distinct objects, including concepts like subsets, unions, intersections, and complements. As discussed in the previous section, they are represented as a list of elements within curly braces, e.g., {1, 2, 3,4,5,…} or using Visualization tool such as Venn diagrams to depict relationships between multiple sets

Relations: Connections between elements of two sets, including equivalence and partial order relations. They are represented as a set of ordered pairs, e.g., {(x, y)}. They can also be represented as **matrices**, where rows and columns indicate the relationship, or **directed graphs (digraphs)** where nodes and edges represent elements and relationships.

Functions: Mappings from one set to another, such as bijections, injections, and surjections. Functions are represented as mappings between sets, e.g., f: A → B. This is read as f is function from set A to B.

Logic: Propositional and predicate logic, including logical operators, equivalence, and inference. They may be represented using truth table.

Graphs: Representation of objects (nodes) and connections (edges), with topics like graph traversal, trees, and network flows. They can be represented through adjacency lists, adjacency matrices, or edge lists.

Combinatorics: Counting techniques, permutations, combinations, and the principle of inclusion-exclusion. They are represented using factorials, tree diagrams for combinatorial expressions.

Boolean Algebra: Algebraic structure used in digital logic design, including operations like AND, OR, NOT. They are represented using truth tables, logic gates, or symbolic expressions (e.g., A ∧ B).

Algebraic Structures: Groups, rings, and fields relevant to coding theory and cryptography. They are represented using a combination of a set and a binary operation. For example, the set Q of rational numbers along with the binary operation "addition" is represented as algebraic structure (Q, +).

We will discuss all these discrete structures in detail throughout the course. They are the basic building blocks that are required to build a strong foundation in the domain of computer science for developing analytical skills and problem solving.

4.3 Unit Summary: In this unit we have discussed the basic concepts of set theory and discrete structures, essential tools in computer science and mathematics. The unit starts with the definitions and principles of set theory, including operations, properties, and practical examples. The unit also explores the representation of discrete structures like graphs, relations, and functions using various methods such as Venn diagrams, adjacency matrices, and directed graphs. These concepts are fundamental in problem-solving, algorithm design,

and data organization.

4.4 Check Your Progress:

1. Define a set. Give examples of finite, infinite, and null sets.
2. If a set A= {1,2,3} and B= {3,4,5}, find A∪B, A∩B, and A−B. Represent these operations using a Venn diagram.
3. If a set C has 5 elements, how many subsets and proper subsets does C have? Explain your calculations.
4. Prove that A∩(B∪C) =(A∩B) ∪(A∩C) using set laws.
5. Write a real-world example where Venn diagrams are used for data classification.
6. What are discrete structure? Write their importance in Computer science.

Unit 5: Relations and Partial ordered sets

5.0 **Introduction and unit objectives:** In discrete mathematics and computer science, relations and partial order sets (posets) are foundational concepts. These concepts are important as they form the backbone of structures used in database systems, graph theory, and decision-making algorithms. Relations and partial order sets (posets) are used to represent connections and orderings between elements of a set. A relation defines a connection between elements of two sets, while ordering introduces the concept of arrangement or hierarchy. This unit explores various ways to define and represent relations, such as matrix and graphical forms, and then move into the properties and applications of partial order sets. Posets are widely used in computer science for representing hierarchies and dependencies, such as task scheduling and data organization. These ideas are helpful

in studying complex systems and solving computational problems.

Unit Objectives: On completion of this unit, the students will be able to

1. Define and explain relations, their properties, and their role in ordering.
2. Represent relations using matrices and directed graphs.
3. Understand and identify partial order sets and their properties.
4. Analyze and interpret the graphical representation of relations.
5. Establish connections between relations and functions in mathematical modeling.

5.1 Relations and Ordering: The word relation is used to show a relationship between two objects. It is a connection or association between elements of two different sets. For example, 'greater than' denoted by >, may be relation between two sets of integers, where the elements to the left of > are greater than the element to the right of >. To study relation in detail, the idea of ordered pair and Cartesian product is important.

Ordered pair: It is a set of two elements arranged in a specific sequence, written as (a,b). The order is significant, meaning (a,b)≠(b, unless a=b. For example, in the ordered pair (5,7), 5 is the first element called abscissa or x-coordinate in a Cartesian plane, whereas 7 is the second element called ordinate or y-coordinate.

Cartesian Product: Let A and B be two non-empty sets. The set of all ordered pairs (x,y) where x∈A and y∈B is called the Cartesian product of set A and B. It is denoted by A×B.

Symbolically, the Cartesian product is written as:
$A \times B = \{(x,y) | x \in A \text{ and } y \in B\}$.

For example, if A= {1,2,5} and B={6,7} then A×B={(1,6), (1,7), (2,6), (2,7), (5,6), (5,7)}

Formal definition of Relation: A relation R from a non-empty set A to a set B is any subset of the Cartesian Product A × B satisfying the given condition. i.e. $R \subseteq A \times B$.

Domain and Range of a relation: Suppose R is a relation from A to B then, domain of the relation R is the set of all first elements of ordered pairs which belongs to R. It is donated by Dom (R). Mathematically Dom (R) = $\{a \in A: (a,b) \in R$, for some $b \in B\}$.
Similarly, the range of the relation R is the set of all second elements of the ordered pair belonging to the relation R. Mathematically Range(R) = $\{b \in B:(a,b) \in R$, for some $a \in A\}$.

Solved Example:

1. If R= {(2,3), (2,4), (3,4)}, then Dom(R)= {2,3} and Range(R)= {3,4}.
2. Let A= {4,5,6}. Find the relation on set A for the condition x+y<10. Also find the domain and range of the relation R.

Solution:

A × A= {(4,4), (4,5), (4,6), (5,4), (5,5), (5,6), (6,4), (6,5), (6,6)}

Using the given condition R= {(4,4), (4,5), (5,4)}

Dom (R)= {4,5} and Range (R)= {4,5}

Types of Relations:

Complementary Relation: Let R be a relation from set A to set B. The complementary relation of R is denoted by R′ which consist of those ordered pairs which are not in R, that is R′ = {(a,b) ∈ A × B : (a,b) ∉R }

Example: Let R be a relation on a set A = {1,2,6} defined as R= {(x,y): x<y}. Find the complement relation of R.

Solution: A × A= {(1,1), (1,2), (1,6), (2,1), (2,2), (2,6), (6,1), (6,2), (6,6)}

R= {(1,2), (1,6), (2,6)}, R'= {(A × A)-R} = {(1,1), (2,1), (2,2), (6,1), (6,2), (6,6)}

Inverse Relation: Let R be a relation from A to B, the inverse of R denoted by R^{-1}, is the relation from B to A which consist of those ordered pairs which when reversed belongs to R. i.e. R^{-1}={(b,a): (a,b) ∈R}

Solved Example:

Let A= {1,2,3,4} and B={a,b,c}

Let R= {(1, a),(1,b),(2,b),(2,c)} and S={(1,b),(2,c),(3,b),(4,b)}

Compute a. R' b. R^{-1} c. R ∪ S

Solution: First we have to find AX B

A X B= {(1, a), (1, b), (1, c), (2, a), (2, b), (2, c), (3, a), (3, b), (3, c), (4, a), (4, b), (4, c)}

a. R'= {(1, c), (2, a), (3, a), (3, c), (4, b), (4, c)}

b. R^{-1}={(a,1), (b,1), (b,2), (c,2), (b,3), (a,4)}

c. RUS= {(1, a), (1, b), (2, b), (2, c), (3, b), (4, a), (4, b)}

Identity Relation: A relation R in a set A i.e. a relation R from set A to A is said to be an identity relation, denoted by I_A, if I_A= {(x, x): x∈A}. For example, let A= {1,2,3} then I_A= {(1,1), (1,2), (1,3)}.

Void Relation: A relation R from set A to B is said to be void, if there is no any order pair in the relation R that belongs to A×B satisfying the given condition. If R=Φ, then R is called a void or empty relation.

Universe Relation: A relation R from set A to B is called universe relation if R=A×B.

5.2 Matrix Representation of Relations: The matrix representation of a relation is a method of expressing a relation R from a set A to a set B in the form of a two-dimensional binary matrix. Let A= $\{a_1, a_2,...,a_m\}$ and B=$\{b_1,b_2,...,b_n\}$ be finite sets containing m and n elements respectively. Let R be a relation from A to B, then R can be represented by mn matrix $M_R=[m_{ij}]_{mn}$ where m_{ij} is given by

$$\begin{cases} 1, & \text{if } (a_i, b_j) \in R \\ 0, & \text{otherwise.} \end{cases}$$

Then the matrix M_R is called matrix of relation R.

If R and S are relations on set A, then using operations on Boolean matrices we can show that $M_{RUS}= M_R \vee M_S$, $M_{R\cap S}=M_R \wedge M_S$

Solved Examples:
Example1: Represent the relation {(1,1), (1,2), (1,3), (2,2), (2,3), (3,2), (3,3)} on the set {1,2,3) using matrix.
Solution:

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

,

Since A is a set containing 3 elements so M_R contains 3 rows and 3 columns.

Example 2: Let A= $\{a_1, a_2, a_3\}$ and B= $\{b_1, b_2, b_3, b_4\}$. Find which ordered pairs are in the relation R given by the matrix.

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

Solution: Let us find A X B= $\{(a_1, b_1), (a_1, b_2), (a_1, b_3), (a_1, b_4), (a_2, b_1), (a_2, b_2), (a_2, b_3), (a_2, b_4), (a_3, b_1), (a_3, b_2), (a_3, b_3), (a_3, b_4)\}$

Now the relation R consist of those ordered pairs (a_i,b_j) with $m_{ij}=1$. Therefore

R= $\{(a_1, b_1), (a_1, b_2), (a_2, b_1), (a_2, b_3), (a_2, b_4), (a_3, b_1), (a_3, b_3)\}$

Boolean Matrix: Let $A=[a_{ij}]$ be a matrix whose elements are either 1 or 0 is called Boolean Matrix. We define two Boolean operations $\vee$ and $\wedge$ as

`

$b_1 \wedge b_2$ { = 1, if $b_1=b_2=1$
=0, otherwise

$b_1 \vee b_2$ { = 1, if $b_1=1$ or $b_2=1$

$=0$, otherwise

Boolean Product of two matrices: Let $A=[a_{ij}]_{mn}$ and $B=[b_{ij}]_{np}$ be two Boolean matrices. Then the Boolean product of A and B is defined as $C=[c_{ij}]_{mp}$, a Boolean matrix where

$$C_{ij}= (a_{i1} \wedge b_1) \vee (a_{i2} \wedge b_2) \vee (a_{i3} \wedge b_3) \vee \ldots\ldots \vee (a_{in} \wedge b_n)$$

Properties of Relation:

Reflexive: A relation R on a set A is **reflexive** if every element of A is related to itself i.e. (a, a) $\in$R for all a $\in$ A. A relation R on a set is called irreflexive if (a, a) $\notin$ R for all a$\in$ A.

If A= {1,2,3} and R= {(1,1, (1,2), (2,2), (2,3), (3,3)}. Then R is a reflexive relation since for all a$\in$ A, (a, a) $\in$R.

Symmetric: A relation R on set A is called symmetric if (a, b) $\in$ R then (b, a) $\in$ R for all a, b $\in$ A. A relation R on a set A is called asymmetric if (a, b) $\in$ R then (b, a) $\notin$R for all a, b $\in$ A.

Antisymmetric: A relation R on a set A is called antisymmetric if a=b whenever (a, b) $\in$ R and (b, a) $\in$ R.

Transitive: A relation R on a set A is called transitive if whenever (a, b) $\in$ R and (b, c) $\in$ R then (a, c) $\in$ R for all a, b, c a$\in$ A. Let A= {1,2,3} and R= {(1,2), (3,2), (2,3), (1,3), (2,2), (3,3)}. Then R is transitive.

5.3 Partial Ordered Sets: A relation R on a set A is called partial order if it is reflexive, antisymmetric and transitive. A set A together with a partial order relation R is called **partially ordered set** or POSET and is denoted by (A, R).

Solved Example:

1. Prove that the greater than or equal to ($\geq$) relation is a partial ordering on Z, the set of integers.

Solution: To prove that $\geq$ is partial ordering, we have to show that R satisfies reflexive, antisymmetric and transitive property.

Reflexive: Since a$\geq$a for every integer a$\in$Z, so $\geq$ is reflexive.

Antisymmetric: let a$\geq$b and b$\geq$a. Then clearly a=b. So $\geq$ is antisymmetric.

Transitive: Let a$\geq$b and b$\geq$c. This implies a$\geq$c. So $\geq$ is transitive.

Therefore, the given relation $\geq$ o set of integers is a partial order.

2. Show that a relation $\leq$ "less than or equal" is partial order on the set of integers.

Solution: We know $\forall$a$\in$ Z, a $\leq$ a, hence $\leq$ is reflexive. If a $\leq$ b and b $\leq$ a, then a = b, hence $\leq$ is antisymmetric, and if a $\leq$ b and b $\leq$ c, then a $\leq$ c, hence $\leq$ is transitive. It follows that $\leq$ is a partial ordering on the set of integers Z and (Z, $\leq$) is a poset.

In a poset (S, R), the elements a and b of a poset are **comparable** if either aRb or bRa ie either (a,b) ∈ R or (b,a) ∈R.When neither aRb nor bRa, then a and b are incomparable.

If every two elements of a set S are comparable, then S is called a **totally ordered** or **linearly ordered set** or a **chain**, and R is called **total order** or a linear order.

(S, R) is a **well ordered set** if it is a poset such that R is a total order and such that every nonempty subset of S has a least element.

External Elements of POSET:

External Elements of POSETs refer to specific elements of a partially ordered set (POSET) that are defined based on their relationships within the set. These include *maximal elements*, *minimal elements*, *greatest elements*, and *least elements*:

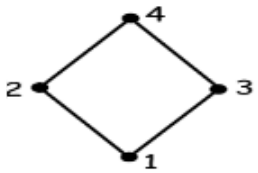
Maximal Element: An element a ∈ A is maximal if there is no element c ∈ A such that a≤c, except for a itself. In the set A= {1,2,3} with the divisibility relation (x|y), 3 is maximal because there is no element y∈A such that 3|y.

Minimal Element: An element b∈A is minimal if there is no element c∈A such that c≤b, except for b itself. In the same set A= {1,2,3}, 1 is minimal as no element divides 1 other than itself.

Greatest Element: An element g is the greatest if g≥x for all x∈A. It is unique if it exists. In A= {1,2,4,8} under divisibility, 8 is the greatest element.

Least Element: An element l is the least if l≤x for all x∈A. It is also unique if it exists. In A= {1,2,4,8}, 1 is the least element.

For the relation R= {(1,1), (1,2),(1,3),(1,4),(2,2),(2,4),(3,3),(3,4),(4,4)}, the hasse digram and the maximal, minimal, greatest and the least elements are shown as in the figure.



Maximal element:4
Minimal element:1
Greatest element:4
Least element:1

Some other important terms in lattice:

In a **lattice**, which is a partially ordered set, the following terms are defined:

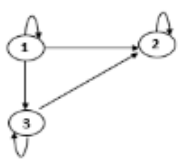
1. **Upper Bound**: For elements a and b, an element u is an upper bound if a≤u and b≤u. For example, in the set {1,2,3,4,6} with divisibility as the order, 6 is an upper bound of 2 and 3 because 2|6 (read as 2 divides 6) and 3|6.

2. **Lower Bound**: For elements a and b an element l is a lower bound if l≤ and l≤b. For example, in the same set, 1 is a lower bound of 2 and 3 because 1|2 and 1|3.
3. **Least Upper Bound (Join)**: Also called the **supremum**, it is the smallest element among all upper bounds of a and b. For example, in the divisibility set above, the least upper bound of 2 and 3 is 6 because it is the smallest number divisible by both.
4. **Greatest Lower Bound (Meet)**: Also called the **infimum**, it is the largest element among all lower bounds of a and b. For example, the greatest lower bound of 2 and 3 in the same set is 1 because it is the largest number dividing both.

5.4 Graphical representation of Relations: Relations can be represented with the help of directed graphs. A directed graph, or digraph is a set of vertices V together with the set of edges. The vertex a is called initial vertex of the edge (a, b), and the vertex b is called the terminal vertex of this edge. We will discuss more about graphs and digraphs in Module IV.

Example: Represent the relation R= {(1,1), (1,2), (1,3), (2,2), (3,2), (3,3)} on the set A= {1,2,3) with the help of a digraph.

Solution: For drawing the digraph of a relation, one node or vertex is created for each element of set A. For each of order pair (a, b) in R, we draw one edge starting from a and ending at b. Since each of the graph has directions, so they are called directed graph. An edge starting and ending at the same node is called a loop. The following is the graphical representation of R.



Hasse Diagrams: We have already discussed about partially ordered set in the previous section. Hasse diagrams are used to represent partial ordering pictorially.

A partial ordering on a finite set can be represented using the pictorial notation as follows:

a. Construct the directed graph of a relation.
b. Remove all the loops (since it is clear that partial order is reflexive so every vertex has a loop)
c. Remove all the edges that are there due to the transitivity property.
d. Arrange each edge so that its initial vertex is below its terminal vertex.

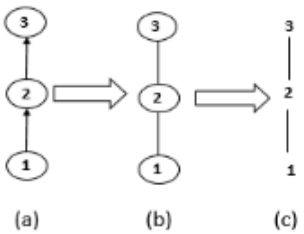
e. Remove all arrows from the edges since edges point in upward direction only.

The diagram formed using above steps contains sufficient information to find the partial ordering. This diagram is called **Hasse diagram**.

Example: Draw a Hasse diagram using the set P = {1, 2, 3} with the order defined by "less than or equal to" (≤).

Solution: Let us first find the relation represented by the relation ≤.

P={(1,1), (1,2), (1,3), (2,2), (2,3), (3,3)}. Now let us remove the ordered pairs formed by reflexivity and transitivity. Then R contains only (1,2) and (2,3). SP the Hasse diagram for the relation can be represented by



The figure in (a) represent the Hasse diagram for the given relation which can be simplified by removing the arrows as shown in (b). The arrows can be removed because we know that in a Hasse diagram, the arrows points upwards only. The diagram can be further simplified by removing the circles as shown in (c).

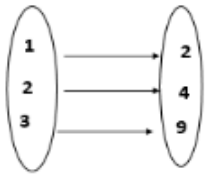
5.5 Functions: Functions: Let A and B be two non-empty sets. A function f from A to B is a set of ordered pairs having the property that, for every element x in A, there exists a unique element y in B. The set A is called the domain of the function, and the set B is called the co-domain.

If (x,y)∈f, it is customary to say that y=f(x), where y is called the image of x, and x is the pre-image of y.

The set consisting of all the images of the elements of A under the function f is called the range of f. The range of f is a subset of B (the co-domain), which may or may not be equal to B.

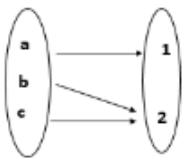
Types of Function:

1. **One-to-one or Injective function:** A function f from A to B is called one-to-one if for all $x_1, x_2 \in A$ such that $f(x_1) = f(x_2)$ implies $x_1 = x_2$.



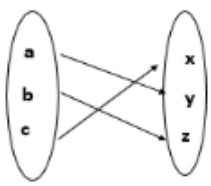
One-to-one

2. **Onto or surjective function:** A function f from A to B is an onto function if every element of B is the image of some element in A.



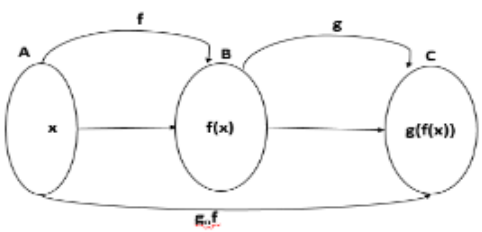
onto

3. **One-to-one correspondence or bijective function:** A function f from A to B is said to be bijective if it is both injective or surjective.



One-to-one correspondence

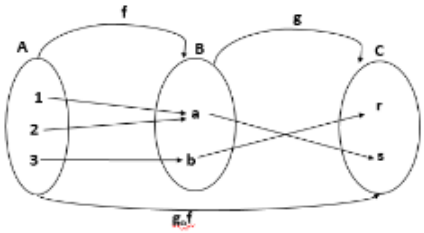
Composition of function: Let f: A→B and g: B → C be two functions. The composition of f and g, denoted by $g_o f$, is a new function from A to C defined by $(g_o f)(x)$, is a new function from A to C defined by $(g_o f)(x)=g(f(x))$, for all $x \in A$



Solved examples:

Example: Let A= {1,2,3}, B={a, b}, C={r, s} and f:A→B defined by f(1)=a, f(2)=a, f(3)=b and g:B→C defined by g(a)=s, g(b)=r. Find the composition function $(g_o f)$: A→C.

Solution: Given f:A→B and g: B→C, then



Using the definition of composition of functions,

$g_of(1) = g(f(1)) = g(a) = s$

$g_of(2) = g(f(2)) = g(a) = s$

$g_of(3) = g(f(3)) = g(b) = r.$

Example: Let the function f and g be defined by $f(x)=2x+1$ and $g(x)=x^2-2$. Find the formula for defining the composition function g_of.

Solution: We have, $f(x)=2x+1$ and $g(x)=x^2-2$

Then, $gof(x)=g(f(x))=g(2x+1)$

$=(2x+1)^2 -2$

$=4x^2 + 4x+ 1-2$

$=4x^2 + 4x-1$

Equality of function: Two functions f and g defined on the same domain are said to be equal iff $f(x)=g(x)$, for every $x \in D$, then we write $f=g$.

Identity Function: A function $f:A \rightarrow A$ defined by $f(x) = x$ for every $x \in A$ is called the identity of A and is denoted by I_A.

Inverse of a function: Let $f:A \rightarrow B$ be a function which is bijective, then there exists a function from B to A called an inverse function of f. Thus $f^{-1}:B \rightarrow A$ is an inverse function of $f:A \rightarrow B$.

A function: $A \rightarrow B$ is invertible if its inverse relation f^{-1} is a function from B to A. In general, the inverse relation f^{-1} may not be a function.

Example: Let A= {1,2, 3, 4}, B={a, b, c, d} and let f={(1,a),(2,a),(3,d),(4,c)}. Show that f is a function and state with reason, whether f is invertible.

Solution: Here we have f(1)=a, f(2)=a, f(3)=d,f(4)=c. Clearly f is a function. Now f^{-1} ={(a,1), (a,2), (d,3),(c,4)}, which is not function becaue for some element of B there are multiple element of A. So f is not invertible since $f^{-1}(a)=\{1,2\}$.

Floor and Ceiling Function: Let x be a real number then ⌊x⌋ called the floor function of x. This function rounds x into greatest integer less than or equal to x. It is often called the

greatest integer function.

Let x be a real number then ⌈x⌉ called the ceiling function of x. This function rounds x into smallest integer greater than or equal to x.

For example, ⌊3.7⌋=3 and ⌈3.7⌉=4.

The floor function is used to round down to the nearest integer, while the ceiling function rounds up. This is essential in applications requiring integer values from floating-point data, such as data normalization and discretization in machine learning.

5.5 Unit Summary: In this chapter, we explored the fundamental concepts of relations and partial ordered sets, which form the backbone of discrete mathematics and computer science. Starting with the basics of relations and ordering, we examined how different properties like reflexivity, symmetry, and transitivity shape the structure of relations and allow us to classify them into specific types.

The matrix representation of relations provided a powerful and efficient way to visualize and compute relations, particularly in algorithmic and computational contexts. This representation is widely used in graph theory and other domains for analyzing connections between entities.

Exploring into partial ordered sets (posets), we studied their defining properties—reflexivity, antisymmetric, and transitivity. The hierarchical nature of posets has significant implications in areas such as scheduling, dependency resolution, and database design.

Additionally, we introduced the graphical representation of relations, offering a visual perspective that aids in comprehending complex structures and their interconnections. This approach is particularly useful in graph theory and optimization problems. Finally, the concept of functions connected the chapter to practical applications by establishing the mathematical foundations of mappings, a critical tool in algorithms, data structures, and software development.

By combining these concepts, this chapter equips students with a complete understanding of relations and their applications, empowering them to approach advanced topics in computer science and mathematics with confidence.

5.6 Check Your Progress.

1. Let A= {4,5,6}. Find the relation in A×A under the conditions
   i. X+y<10.
   ii. x/y is an integer.
   Also find the domain of the relation.
2. If R and S are two relations on set A= {1,2,3,4} defined by R={(1,1), (1,2), (3,4),(4,2)}

and S={(1,1),(2,1),(3,1),(4,4),(2,2)}. Find $R \cup S$ and $S \cap R$.

3. For the following relations, state with reason if they are reflexive, symmetric, antisymmetric or transitive.
    1. {(2,2),(2,3),(2,4),(3,2),(3,3),(3,4)}
    2. {(1,1),(1,2),(2,1),(2,2),(3,3),(4,4)}
4. The relation R on set A={1,2,3,4} is defined by R={(1,1),(1,2),(1,4),(2,2),(2,1),(2,4),(3,3),(3,4),(3,2),(4,3),(4,2),(4,1)}. Draw the digraph of R and hence find R^{-1}.
5. Display the ordered pairs in the relation R={(a,b): a divides b}on the set {1,2,3,4,5,6}.
    1. Display this relation graphically
    2. Display the relation in matrix form.
6. Explain the concept of a Hasse diagram and how it represents a poset.
7. Prove that the power set of a set, ordered by inclusion ⊆), forms a poset.
8. If $f(x)=x^2-2$ and the domain of the function is {-1,0,1,2,3}. Find the range of the function. Is it one-to-one?
9. Let f:R→R and g:R→R be two functions defined as f(x)=2x+1 and $g(x)=x^2-2$, then find the composite functions.
    1. f_og b. f_of c. g_of d. g_og e. f_og_of
10. What are equivalence relations? Give an example.

Unit 6: Lattices and Boolean Algebra.

6.0 Introduction and Unit Objectives Lattices and Boolean Algebra are the foundational concepts and pivotal areas in discrete mathematics and theoretical computer science. In the previous we learnt about the partially ordered sets. Lattices, as a special class of partially ordered sets, are fundamental in structuring data and analyzing relationships. Boolean Algebra, on the other hand, provides the mathematical framework for binary systems, laying the foundation for digital logic design, switching theory, and computer architecture.

Together, lattices and Boolean algebra contribute to solving practical problems in computer science, telecommunications, and mathematics, offering tools for structuring data, building circuits, and creating efficient logical systems.

The unit aims to build a clear understanding of how mathematical principles underpin computational frameworks. Learners will explore the properties of lattices, their representation, and applications in set theory, logic, and computer science. The introduction to Boolean Algebra emphasizes its axiomatic basis and its role in logical operations, truth tables, and circuit design.

Unit Objectives: On completion of the unit, the students will be able to

1. Understand the concept of partially ordered sets and identify lattices within them.
2. Learn the properties and types of lattices, such as bounded, distributive, and modular lattices.
3. Apply lattice theory to solve problems in discrete mathematics and computer applications.
4. Understand the laws of Boolean Algebra.
5. Analyze and simplify Boolean expressions using algebraic methods.

6.1 Lattice, Lattices as Partially ordered set:

A lattice is a poset in (L,≤) in which every subset {a, b} consisting of two elements has a least upper bound and a greatest lower bound.

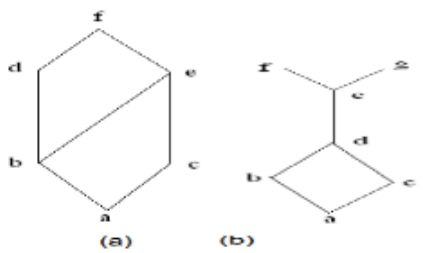
LUB ({a, b}) is denoted by **a v b** and is called the join of a and b.

GLB ({a, b}) is denoted by a Λ b and is called the meet of a and b.

Consider a lattice L= {1,2,3,6} with the divisibility relation (|): a|b means a divides b. Now to find the LUB {2,3}, we first find elements greater than or equal to both 2 and 3 is 6. So 6 is the smallest among these elements, since there is only e element. Hence LUB {2,3} =6.

To find the GLB {2,3}, we first find elements less than or equal to both 2 and 3 is {1}. Now the largest among these is 1 since there is only one element. So GLB {2,3}=1

Example: consider the following two hasse diagrams and determine if they are lattice.



Solutions: The Hasse diagram in (a) is a lattice because every subset {a, b} consisting of two elements has a least upper bound and a greatest lower bound. The Hasse diagram in (b) is not a lattice because LUB {f, g} does not exists.

Properties of Lattices:

1. Idempotent Properties	a) $a \vee a = a$	b) $a \wedge a = a$
2. Commutative Properties	a) $a \vee b = b \vee a$	b) $a \wedge b = b \wedge a$
3. Associative Properties	a) $a \vee (b \vee c) = (a \vee b) \vee c$	b) $a \wedge (b \wedge c) = (a \wedge b) \wedge c$
4. Absorption Properties	a) $a \vee (a \wedge b) = a$	b) $a \wedge (a \vee b) = a$

Type of Lattice: There are several types of lattice which are listed below

a. **Complete Lattice:** A lattice P is called complete iff every non-empty subset of P has GLB and LUB. Consider the set of integers Z with the relation $\leq$ i.e. $(Z, \leq)$. Let us consider a subset S of integers $S=\{x \in S \mid x>0\}$. We can see that least upper bound does not exist for this subset. Therefore, it is not a complete lattice. If the set is finite, then it will be a complete lattice but for infinite sets it will not be a complete lattice.

b. **Bounded Lattice:** A lattice having least and a greatest element denoted by 0 and 1 respectively is called bounded lattice. The set Z^+ under the partial order of divisibility is not a bounded lattice since it has a least element, the number 1, no greatest element. The lattice P(S) of all subsets of a set S is bounded as the greatest element is S and least element is Φ

c. **Isomorphic Lattice:** An isomorphic lattice refers to two lattices L1 and L2 that are structurally identical in terms of their ordering relationships. This is established through a bijective function f: L1→L2 such that:

$$f(a \wedge b) = f(a) \wedge f(b) \text{ and } f(a \vee b) = f(a) \vee f(b)$$

where $\wedge$ represents the meet (greatest lower bound) and $\vee$ represents the join (least upper bound).

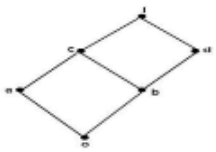
d. **Distributive Lattice:** A lattice L is considered a distributive lattice if for any elements a, b, and c of L, it satisfies the following distributive properties:

1. $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
2. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

If the lattice L does not satisfy these properties, it is known as non-distributive lattice.

e. **Complemented Lattice:** Let L be a bounded lattice with a lower bound 0 and an upper bound 1. Let a be an element of L. An element x in L is called a complement of a if $a \vee x = 1$ and $a \wedge x = 0$. A lattice L is said to be complemented if L is bounded and every element in L has a complement.

Example: Determine the complement of a and c in figure given below.



Solution: The complement of a is d, as a ∨ d = 1 and a ∧ d = 0. The complement of c does not exist because there is no element c such that c ∨ c' = 1 and c ∧ c' = 0.

6.2 **Boolean Algebra:** Boolean algebra can be viewed as a lattice that satisfies additional properties. A **lattice** is an algebraic structure defined by a partially ordered set (poset) in which any two elements have a unique least upper bound (join, denoted by ∨) and greatest lower bound (meet, denoted by ∧).

Boolean Algebra as a Lattice

A **Boolean algebra** is a special type of lattice with the following properties:

1. **Bounded**: It has a least element 0 (bottom) and a greatest element 1 (top).
2. **Complemented:** Every element x has a complement x′ such that x ∧ x′=0aand x ∨ x′=1
3. **Distributive**: The lattice operations meet (∧) and join (∨) are distributive over each other:

    1. a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c)
    2. a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c)

Definition: A complemented distributive lattice A12 is referred to as a Boolean Algebra. It is represented as (B, ∧, ∨, ', 0, 1), where B is a set with two binary operations, ∧ (*) and ∨ (+), and a unary operation (complement) defined. Here, 0 and 1 are two distinct elements of B. Since (B, ∧, ∨) is a complemented distributive lattice, every element in B has a unique complement.

Properties of Boolean Algebra:

1. **Commutative Properties:**
    1. a+b = b+a
    2. a*b=b *a
2. **Distributive Properties**
    1. a+(b*c)=(a+ b)*(a +c)
    2. a*(b+ c) =(a*b) +(a*c)
3. **Identity Properties**
    1. a+0=a
    2. a *1=a
4. **Complemented Laws:**

1. a+ a'=1
2. a * a'=0

Example: Prove that the set {0,1}, with operations ∧ (AND), ∨ (OR), and complement x′, forms a Boolean algebra.

Solution:

Bounded Lattice: 0 is the least element, and 1 is the greatest element

Distributive law: a ∧ (b ∨ c) = (a ∧ b) ∨ (a ∧ c) and a ∨ (b ∧ c) = (a ∨ b) ∧ (a ∨ c) .

Complement Law: a ∨ a' = I and a ∧ a' = 0

Since all conditions are satisfied, this is a Boolean algebra.

Example: Prove that the power set P(S) of S={a, b}, with union (∪) as join, intersection (∩) as meet, and set complement as the complement, forms a Boolean algebra.

Solution:

Bounded Lattice: ∅ (least element), S={a, b} (greatest element)

Distributive Law: Union and intersection satisfy distributive property.

Complement: A∪A′=S and A∩A′=∅

All conditions of Boolean algebra are met.

Example: Prove that the set {0,1,2,3} under the divisibility relation ∧ (GCD), ∨ (LCM), and complements forms a Boolean algebra.

Solutions:

Elements: 0 (least element), 3 (greatest element)

Join and Meet: ∧=GCD, ∨=LCM.

Complement: Elements have unique complements in this structure.

hence, this forms a Boolean algebra.

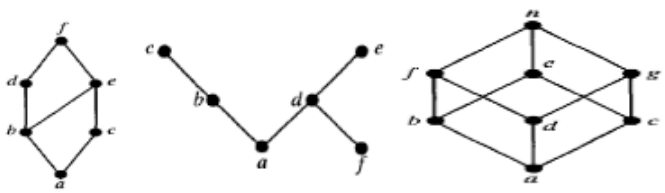
6.3 Unit Summary: This unit introduces foundational concepts of lattices and Boolean algebra, which are critical in understanding abstract algebra and logical structures used in computer science and mathematics. It begins with defining lattices as algebraic structures derived from partially ordered sets (posets), emphasizing the ordering relationships between elements. A lattice is a set in which any two elements have a unique least upper bound (join) and greatest lower bound (meet). This section explores their properties and how they extend poset concepts to algebraic frameworks, forming the basis for understanding hierarchical structures, decision systems, and optimization problems.

The second part delves into Boolean algebra, a specialized type of lattice that is distributive and complemented. Boolean algebra is a mathematical system where elements represent truth values (1/0 or true/false) and operations like AND, OR, and NOT are defined. It is integral to fields like digital logic design, programming, and data structures. By combining lattice theory with logic, this section provides a comprehensive understanding of how Boolean operations adhere to

algebraic laws, enabling applications in simplifying logical expressions and building efficient computational systems.

6.3 Check Your Progress.

1. Define a lattice. Prove that every lattice is a partially ordered set.
2. Prove that the set of integers under divisibility forms a lattice.
3. Show that in a lattice, the meet and join operations satisfy the associative law.
4. Verify that the set {0, 1} with operations AND, OR, and NOT is a Boolean algebra.
5. Prove that every Boolean algebra is a distributive lattice.
6. Determine whether the following structure forms a Boolean algebra: Set of subsets of a given set under union, intersection, and complement operations.
7. Determine whether the poset represented by the hasse diagram is a Boolean algebra.



Unit 7: Foundations of Algebraic Structures.

7.0 Introduction Unit Objectives: This unit focuses on algebraic structures, which are the building blocks of modern mathematics and computer science. They involve a set of elements and operations like addition, multiplication, or combinations of these operations. They help us understand and solve problems in a systematic way. First, we will study semigroups and monoids. A semigroup is a set where you can combine two elements using a specific rule (or operation), and the rule always works the same way, no matter how the elements are grouped. If a semigroup also has a special element called the identity element, which does not change other elements when used in the operation, it becomes a monoid. These concepts have practical

applications in computer science, such as understanding processes or systems that involve sequences of operations.

Next, we will look at groups, which are advanced algebraic structures. A group has even more structure than a monoid, and it includes the ability to "undo" an operation using something called an inverse. Groups are widely used in areas like cryptography and network security. Finally, we will explore congruence relations, which help us simplify complicated structures by grouping similar elements together. This leads to the concept of quotient structures, which are simpler versions of the original systems. The unit also covers permutation groups, which deal with rearranging objects and have applications in solving puzzles, analyzing patterns, and computer algorithms. By the end of this unit, learners will have a good understanding of these algebraic systems and their significance in both theoretical and practical contexts.

Unit Objectives: On completion of this unit, learners will be able to

1. Understand the Concept of Algebraic Systems.
2. Explore Group Theory.
3. Introduce Congruence Relations and Quotient Structures
4. Analyze Permutation Groups
5. Develop Problem-Solving Skills.

7.1 Concept of Algebraic System: Semigroups and Monoids

An **algebraic system** in discrete mathematics refers to a structured set equipped with one or more operations that satisfy certain rules. These systems provide a framework to study relationships and operations within a defined mathematical structure.

Key Components of an Algebraic System:

1. **Set (Domain):** A non-empty set of elements forms the foundation of the system.
2. **Operations:** Binary operations (like addition or multiplication) define how elements in the set interact. For example, combining two elements from the set must result in another element from the same set (closure property).
3. **Properties:** The system may satisfy properties like associativity, commutativity, and the existence of identity or inverse elements.

Some common types of Algebraic Systems are Semigroups (A set with an associative binary operation), **Monoids (**A semigroup with an identity element), **Groups (**A monoid where every element has an inverse, **Rings and Fields (**Structures involving two operations (e.g., addition and multiplication) with more complex properties). These systems are crucial for understanding discrete structures and are widely used in areas like cryptography, automata theory, and computer algorithms.

So, algebraic system or structure is a type of non-empty set G which is equipped with one or more than one binary operation. Let us assume that * describes the binary operation on non-empty set G. In this case, (G, *) will be known as the algebraic structure. (1, -), (1, +), (N, *) all are algebraic structures. (R, +, .) is a type of algebraic structure, which is equipped with two operations (+ and .)

Binary Operation: A **binary operation** is a function that combines two elements of a set to produce another element of the same set. Mathematically, a binary operation $*$ on a set A is a function $*: A \times A \rightarrow A$, meaning it takes two inputs from A and returns an output in A.

Common Binary can be:

Addition (+): Combines two numbers to produce their sum.
Multiplication (·): Produces the product of two numbers.
Subtraction (−): Produces the difference between two numbers.
Division (/): Produces the quotient of two numbers.
Modulus (%): Gives the remainder after division.
Boolean Operations: Binary operations such as AND ($\wedge$), OR ($\vee$), and XOR ($\oplus$) are used in logic.

Properties of Binary Operations:

1. **Closure Property:**
   If a, b $\in$ A then a $*$b $\in$ A.
   Example: Addition of integers is closed because a + b $\in$ Z for all a, b $\in$ Z.

2. **Commutative Property:**
   A binary operation * is commutative if a $*$ b=b $*$ a for all a, b $\in$ A.
   Example: Multiplication of real numbers is commutative because a· b=b· a.

3. **Associate Property:** A binary operation * is associative if (a*b) *c= a* (b * c) for all a, b, c $\in$ A.
   Example: Addition of real numbers is associative because (a+ b) +c=a+ (b +c).
4. **Identity:** An element e $\in$ A is an identity element if a $*$e=a=e$*$ a for all a$\in$A. Example: 0 is the identity for addition in integers (a+0=a).

5. **Inverse Element:** An element b∈A is the inverse of a∈A if a∗b=e, where e is the identity element. Example: For addition in integers, the inverse of a is –a because a + (-a)=0.
6. A binary operation ∗ is distributive over another operation ⊕ if a* (b⊕c)=(a∗b)⊕(a∗c).

Semi Group:

A non-empty set S, (S,*) is called a semigroup if it follows the following axiom:
Closure: (a*b) belongs to S for all a, b ∈ S.
Associativity: a*(b*c) = (a*b)*c where a, b ,c belongs to S.

Monoid: A non-empty set S, (S, *) is called a monoid if it follows the following axiom:
Closure: (a*b) belongs to S for all a, b ∈ S.
Associativity: a*(b*c) = (a*b) *c for all a, b, c ∈S.
Identity Element: There exists e ∈ S such that a*e = e*a = a for a ∈ S,

Example: Determine whether the following set Z, the set of integers, where a ∗ b= a+ b- ab together with the binary operation is a semigroup, a monoid or neither. If it is a monoid, specify the identity.

Solution:

1. Closure Property : - a, b∈ z then a + b - ab ∈ z ∀ a,b ∈ Z so * is closure
2. Associative Property: Let a, b, c ∈ Z, then

(a∗b) ∗c=(a + b – ab) ∗ c
= (a + b – ab) + c - (a + b – ab) c
= a+b-ab+c-ac-bc+abc
=a+b+c-ab-bc-ca+abc

a∗(b ∗c)= a ∗ (b +c-bc)
= a+b+c-ab-bc-ca+abc

Hence (a∗b) ∗c= a∗(b ∗c)

Therefore ∗ is associative.

∴ (z, ∗) is a semigroup.

Existence of Identity:

Let e be the identity element a * e = q

a + e - q.e = a
a + e - a.e = a
e (1-a) = 0
e = 0 or a = 1
But a ≠1
E = 0

$\therefore$ O $\in$Z is the identity element.
$\therefore$(Z, *) is monoid.

7.2 Groups: A non-empty set G and a binary operation $*$ defined on it is called a group if
(i) binary operation $*$ is close.
(ii) binary operation $*$ is associative.
(iii) (G, $*$) has an identity.
(iv) every element in G has inverse in G, We denote the group by (G, $*$).

Example: Prove that Z (set of integers) with the binary operation of addition is a group.

Solution: We need to show that the following four properties are satisfied

Closure: Addition of two integers results in another integer
Identity: There is an identity element, 0, such that for any integer n, n+0=0+n =n
Inverses: Every integer n, there exist an integer –n such that n+(-n)=(-n)+n=0 (identity).
Associativity: Addition of integers is associative.
Since all properties are satisfied, therefore set of integers together with binary operation addition is a group.

7.3 Congruence Relations, Quotient Structures, Permutation Groups.
Congruence Modulo relation: Let a and b two integers then a is a congruence to b modulo m if m divides a-b. It is generally written as **a≡b(mod m).**

Let a=20 , b=10 ,m=5
Then 20≡ 10(mod 5)
5| (20-10) or 20-10=5*2

Example: Let m be a positive integer with m>1. Show that the relation R={(a,b): a≡b(mod m)} is an equivalence relation on the set of positive integers.
(Note: A relation is called equivalence relation if it is reflexive, symmetric and transitive)
Solution: To prove that R is equivalence we have to show that R is reflexive, symmetric and transitive properties.

i. **Reflexive:** $\forall a \in Z^+$, a≡a(mod m) $\Rightarrow$m|(a-a) $\Rightarrow$m|0 (which is true)
So (a,a) $\in$ R, hence R is reflexive..

ii. **Symmetric:** $\forall a,b \in Z^+$, if (a,b) $\in$ R then a≡b(mod m) $\Rightarrow$m|(a-b)
$\Rightarrow$a-b=m*k $\Rightarrow$b-a=m*(-k) $\Rightarrow$b≡ a(mod m) $\Rightarrow$m|(b-a) $\Rightarrow$ (b,a) $\in$ R
Hence R is Symmetric

iii. **Transitive:** $\forall a, b, c \in Z^+$
Let (a,b) $\in$ R
$\Rightarrow$ a≡b(mod m) $\Rightarrow$m|(a-b)
$\Rightarrow$a-b=m*k1 ------(1)
Again let (b,c) $\in$ R
$\Rightarrow$ b≡c(mod m) $\Rightarrow$m|(b-c)
$\Rightarrow$b-c=m*k2 -------(2)

Adding (1) and (2), we get
a-b+b-c=m8K1+m*k2
⟹(a-c)=m(K1+K2)
⟹(a-c)=m*k3
⟹m|(a-c)
⟹a≡c(mod m)
⟹(a,c) ∈ R
Thus R us transitive.
Hence the given relation R is an equivalence relation.

Quotient Structures:

A **quotient structure** in mathematics, particularly in group theory, refers to a structure that results from dividing a given group by one of its subgroups. The quotient is the set of cosets of the subgroup in the group, and it forms a new group, known as the **quotient group**.

For example, consider the **additive group of integers** G=Z under addition and let N=3Z the set of multiples of 3. The quotient group G/N is formed by the cosets of N in G, which are:

$$G/N=\{N,1+N,2+N\}$$

This quotient group represents the structure of the integers modulo 3, where the elements {0,1,2} form the quotient group, and the operation is addition modulo 3. The quotient group essentially abstracts the idea of "grouping" elements in G based on the subgroup N.

Permutation and Permutation Group:

Definition 1. Let A be a nonempty set. A permutation of A is a bijective function(mapping) of A onto itself.

Definition 2. A group G is called a permutation group on a nonempty set A if the elements of G are some permutations of A and the group operation is the composition of two functions.

Let us now consider permutation of a finite set. Suppose for any positive integer n, In denotes the finite set {1, 2, 3, . . . , n}. For example, *I*3 = {1, 2, 3}. Now any permutation on *I*n is a bijective function on {1, 2, 3, . . . , n}. The set of all permutations on *I*n forms a group under the binary operation 'composition of two functions'. This permutation group is called the symmetric group on n elements and is denoted by Sn. Let α ∈ Sn. Generally, we demonstrate α in the following way:

$$\alpha : \begin{array}{l} 1 \longrightarrow \alpha(1) \\ 2 \longrightarrow \alpha(2) \\ 3 \longrightarrow \alpha(3) \\ \vdots \\ n \longrightarrow \alpha(n) \end{array}$$

But it is sometimes convenient to describe this permutation by means of the following notational device:

$$\alpha = \begin{pmatrix} 1 & 2 & 3 & \dots & n \\ \alpha(1) & \alpha(2) & \alpha(3) & \dots & \alpha(n) \end{pmatrix}.$$

This notation is called the two-row notation. In the upper row, we list all the elements of ***In*** and in the lower row under each element i ∈ ***In***, we write the image of the element, i.e., α(i). It is easy to compute the number of elements of Sn. The two-row notation of permutations is quite convenient while doing computations, such as determining the composition of permutations.

Example: In this example we consider the group S3, the elements of this group being all the permutations on I3 = {1, 2, 3}. As the number of bijective functions of I3 onto itself is 3! = 6, we have |S3| = 6. We now enlist below the permutations on I3 = {1, 2, 3}.

$$e = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} \quad \alpha = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \end{pmatrix} \quad \beta = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$$

$$\gamma = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} \quad \delta = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} \quad \sigma = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$$

Definition: A permutation σ on In = {1, 2, . . . , n} is called a k-cycle or cycle of length k if there exist distinct elements $i_1, i_2, \dots, i_k$ in **In** such that $\sigma(i_1) = i_2, \sigma(i_2) = i_3, \sigma(i_3) = i_4, \dots, \sigma(i_{k-1}) = i_k, \sigma(i_k) = i_1$ and σ(x) = x for all x ∈ In.

A k-cycle with k = 2 is called a **transposition**

A permutation is called **even** if it can be expressed as a product of even number of transpositions. A permutation is called **odd** if it can be expressed as a product of odd number of transpositions.

7.4 Unit Summary: In this unit, the focus was to discuss and make the students understand about the algebraic structures. Algebraic structures are foundational to understanding how mathematical systems

work and are defined by sets with operations that obey certain rules. A semigroup is an algebraic system with a set and an associative binary operation, while a monoid is a semigroup with an identity element. These structures provide the basic building blocks for more complex systems. Groups further extend these ideas by requiring that every element has an inverse, along with the properties of associativity and the presence of an identity element. Groups play a crucial role in studying symmetry and transformations, forming the basis of numerous applications in mathematics and computer science.

Congruence relations help simplify algebraic systems by partitioning sets into equivalence classes, leading to quotient structures that retain essential properties while reducing complexity. Permutation groups, which represent the set of all possible rearrangements of elements, are another key structure that models symmetry and order. These concepts are interconnected and widely applied in fields ranging from cryptography to combinatorics, demonstrating the significance of algebraic structures in both theoretical and practical contexts.

7.4 Check your Progress.

1. Define a binary operation * on the set of real numbers as a*b=a+b+1.
   Determine if there exist an identity element e and find it.
2. Prove that set of Positive integers under addition including zero is a monoid.
3. **Positive integers under addition** is semi group
4. Prove that set of non-zero real numbers under multiplication is a group
5. Given the set S={1,2,3}S = \{1, 2, 3\}S={1,2,3}, list all possible permutations and calculate the total number of permutations.
6. Determine if the permutation σ=(1 3)(2 4 5)\sigma = (1 \, 3)(2 \, 4 \, 5)σ=(13)(245) is even or odd.
7. Show that the relation R={(x,y): x≡y(mod 4)} defined on set A={1,2,3,4,5,6,……,15} is equivalence.

Unit 8: Advanced Algebraic Structures.

8.0 Introduction and Unit Objectives: This unit explores advanced algebraic structures in the field of abstract algebra, focusing on their significance in Mathematical Foundations of Computer Science. These structures form the backbone of various algorithms, encryption systems, error correction codes, and data processing techniques in computing. They are fundamental in understanding how to manipulate and transform data efficiently. In this unit, we will explore Lagrange's Theorem and its connection to group theory. This unit also cover important concepts on algebraic structures like the role of normal subgroups in group theory, Algebraic structures that involve two binary operations, namely rings and fields. The application of integral domains in computing contexts will also be discussed briefly.

Unit Objectives: On completion of this unit, the learners will be able to

1. Understand the basics of **groups**, **subgroups**, and **normal subgroups**.
2. Learn about **Lagrange's Theorem** and its role in analyzing the scope of subgroups.
3. Examine **algebraic structures** with two binary operations, focusing on **rings** and **fields**.
4. Define and explore **integral domains** and **fields**, with examples and basic properties.
5. Gain understandings into the application of these structures in computer science, particularly in cryptography, coding theory, and data structures.

8.1 Lagrange's Theorem and Normal Subgroups:

Lagrange's Theorem:

Lagrange's Theorem is a fundamental result in group theory that relates the size of a subgroup to the size of the group. It states that:

If G is a finite group and H is a subgroup of G, then the order (i.e., the number of elements) of H divides the order of G.

Example: Consider the group G={e,a,b,c}, where e is the identity element. Let H={e,a} be a subgroup of G. According to Lagrange's Theorem:

|G|=4 (since the group has 4 elements).
|H|=2 (since the subgroup has 2 elements).

4 is divisible by 2, so the theorem holds.

Normal Subgroups:

A **normal subgroup** N of a group G is a subgroup that is invariant under conjugation. This means for all elements $g \in G$ and $n \in N$ the element gng^{-1} is also in N.

Example: Let $G=S_3$ (the symmetric group on 3 elements), and let $N=\{e,(12)\}$ be a subgroup of G. If N is normal, we should have for all $g \in G$, $gng^{-1} \in N$. For instance:

$(13)(12)(13)^{-1}=(132)$, which is not in N. Thus, N is not normal in S_3.

Properties of Normal Subgroups:

1. Every subgroup of an abelian group is normal.
2. Normal subgroups are key in defining quotient groups.

Example: In the group $G=S_3$ (the symmetric group of degree 3), the subgroup $A3=\{(),(123),(132)\}$ is normal.

7.2 Algebraic Structures with Two Binary Operations:

Definition of Algebraic Structures: An algebraic structure consists of a set and one or more binary operations that satisfy certain axioms. For example, a group is a set with a single binary operation satisfying closure, associativity, identity, and invariability.

An algebraic structure with two binary operations consists of a set which is equipped with two operations, typically denoted + and ·. These structures include rings and fields.

Ring is an algebraic structure that has two binary operations: addition and multiplication. The set must satisfy the following:

1. **Additive Group**: The set is an abelian group under addition.
2. **Distributive**: Multiplication distributes over addition.

8.3 Rings, Integral Domains, Fields:

Definition of a Ring:

A ring is a set R equipped with two binary operations: addition (+) and multiplication (·), satisfying:

1. R is an abelian group under addition.
2. Multiplication is associative.
3. Distributive laws hold:
    1. $a\cdot(b+c)=a\cdot b+a\cdot c$
    2. $(a+b)\cdot c=a\cdot c+b\cdot c$

Examples: Z: The set of integers under addition and multiplication i.e (Z,+,.)

Integral Domains

An integral domain is a commutative ring with unity $1 \neq 0$, where there are no zero divisors.
Properties:

1. ab=0 implies a=0 or b=0.

2. The cancellation law holds: $ac=bc \Longrightarrow a=b$ (if $c\neq 0$)

Examples:

1. Z: Integers under addition and multiplication.
2. Q[x]: Polynomials over rational numbers.

Fields:

A field is a commutative ring with unity in which every nonzero element has a multiplicative inverse.

The following properties must be satisfied by a Field:

3. Associative, commutative, and distributive laws apply.
4. Division is defined for nonzero elements.

Example: Prove that Q forms a field under usual addition and multiplication.

Solution: We can solve this problem in two steps.

1. First Verify that it satisfies additive group properties.
2. Then verify the multiplicative closure and existence of inverses element.

8.4 Unit Summary: This unit provided a structured exploration of advanced algebraic structures. From the theoretical foundation of Lagrange's Theorem to practical examples of rings, integral domains, and fields, we have established critical concepts that pave the way for further mathematical studies. Mastery of these topics is essential for advancing in abstract algebra and related fields. In this unit, we discussed concepts of algebraic structures, building upon the foundational principles of groups, rings, and fields. This unit aims to Provide a deeper understanding of Lagrange's Theorem and the role of normal subgroups. Also we Explored algebraic structures with two binary operations and their properties. Finally, the unit concluded with a brief discussion on rings, integral domains, and fields, including definitions, basic properties, and examples. These structures are fundamental in understanding how to manipulate and transform data efficiently.

8.5 Check Your Progress:

1.What does Lagrange's Theorem state about the order of subgroups in a finite group?

2.Define a normal subgroup. How is it different from a regular subgroup?

3.Define a ring. Give examples of commutative and non-commutative rings.

4.Prove that the set of 2×2 matrices with real entries under matrix addition and multiplication forms a ring.

5.What is an integral domain? How does it differ from a general ring?

6.Give an example of a finite field and verify its field properties.

Unit 9: Counting techniques

9.0 **Introduction and Unit Objectives**: Counting techniques form a foundational part of discrete mathematics and are essential in understanding the principles of combinatorics, probability, and optimization. These techniques provide methods to quantify possibilities, arrangements, and selections systematically, enabling their use in diverse fields such as data

analysis, computer algorithms, and cryptography. In this unit, learners will explore basic counting principles and advanced techniques to solve complex problems involving arrangements and selections. Combinatory is that branch of discrete mathematics which concerns with counting problems. Techniques for counting are important in mathematics and computer science, especially in probability theory and in analysis of algorithms. For example, counting are required to determine whether there are enough telephone numbers or internet protocol addresses to meet the demand. It is also used to determine the complexity of algorithms and also in probability theory. So counting problems arises throughout mathematics and computer science.

Unit Objectives: By the end of this unit, learners will be able to:

1. Understand and apply the fundamental principles of counting.
2. Differentiate between permutations and combinations and use them to solve problems.
3. Employ the Pigeonhole Principle to derive conclusions in problem-solving scenarios.
4. Utilize the Principle of Inclusion-Exclusion for calculating the union of multiple sets.
5. Apply these techniques in real-world problem-solving and theoretical applications.

9.1 Basics of Counting:

There are two basic counting principles that can be used to solve the counting problems. We define those two principles below.

Sum rule: The principle of disjunctive counting: If the first task can be completed in m ways and the second task can be completed in n ways and if both the tasks cannot be completed at a time, then there are m + n ways to do one of the task. We can generalize this rule as, if a set X is union of disjoint nonempty subsets $S_1, S_2, \ldots, S_n$, then $|X| = |S_1| + |S_2| + \ldots + |S_n|$.

Example 1: In how many ways we can draw a heart or a diamond from an ordinary deck of playing cards?
Solution: There are total 13 cards of heart and 13 card of diamond. So, by sum rule total number of ways of picking heart or diamond is 13 + 13 = 26.

Example 2: How many ways we can get a sum of 4 or of 8 when two distinguishable dice (say one die is red and the other is white) are rolled?
Solution: Since dice are distinguishable outcome (1, 3) is different form (3, 1) so to get 4 as sum we have the pairs (1, 3), (3, 1), (2, 2), so total of 3 ways. And similarly getting 8 can be from pairs (2, 6), (6, 2), (3, 5), (5, 3), (4, 4), so total 5 ways. Hence getting sum of 4 or 8 is 3 + 5 = 8.

Product Rule: Principle of sequential counting.

If a work can be done in m ways and another work can be done after the completion of first work in n ways, then there is m × n ways to do the task that consists both the work. Generally, suppose an event E1 can occur in m ways and following E1, a second event E2 can occur in n ways, the following E2, a third event can occur in r ways and so on. Then all of these events can occur simultaneously in mnr ways.

Example 1: An office building contains 27 floors and has 37 offices on each floor. How many offices are there are in the building?
Solution: Using the product rule there are 27.37 = 999 offices in the building.

Example 2: How many different three-letter initials with none of the letters can be repeated can people have?

Solution: Here the first letter can be chosen in 26 ways, since the first letter is assigned we can choose second letter in 25 ways and in the same manner we can choose third letter in 24 ways. So by product rule number of different three-letter initials are 26.25.24 = 15600.

More Examples on Basics:

Example 1: How many strings are there of four lowercase letters that have the letter x in them?

Solution: There are total 26.26.26.26 strings of four lowercase letters, by product rule. In the same way we can say that there are 25.25.25.25 strings of four lowercase letters without x, since without x there will be a set of 25 characters only. So there are total of 26.26.26.26 - 25.25.25.25 = 66351 four lowercase letter strings with x in them. This is true because we are decrementing total numbers of strings with the number of strings that do not contain x in them so at least one x will be in the strings.

9.2 Combinations: A combination refers to choosing items from a larger set where there is no any specific of choosing items. It calculates ways to select a group of items without accounting for their arrangement. The formula for combinations of n items taken r at a time is:

$$C(n,r) = \frac{n!}{r! \cdot (n-r)!}$$

Example1: Calculate C(12,10).

$C(12,10) = \frac{12!}{10!(12-10)!} = \frac{12!}{10!2!} = \frac{12X11X10!}{10!X2\ X1} = 66$

Example2: In how many ways can a hand of 4 cards be dealt from a ordinary pack of 52 playing cards

Solution: We need to consider combination, since the order in which the cards are dealt with is not important.

C(52,4)= $\frac{52!}{4!(52-4)!}=\frac{52!}{4!48!}=\frac{52X\ 51\ X\ 50X\ 49X\ 48!}{4X3X2X1}$=270275 ways

Example 3: If there are four students, find the number of unique committees that can be made from these four students using three students?

Solution: We determine the number of subsets containing three elements from a set of four students in order to solve this problem. There are four such subsets, which correspond to the four students that can be eliminated, because choosing three students is the same as omitting one. Therefore, ignoring the sequence of selection, there are four options to select three students for the committee.

.

Example 4: A cricket team has to be formed consisting of two wicket keepers, 4 bowlers and 5 batsmen from a group of players containing 4 wicket keepers, 8 bowlers and 11 bats man. In how many ways a cricket team can be formed.

Solution: Number of ways to form a cricket team is

=C(4,2) × C(8,4) × C(11,5)= 194040 ways.

Exmaple5: A committee is to be formed of 12 men and 8 women and consist of 3 men and 2 women. How many different committees can be formed?

Solution: 3 men can be chosen from 12 men in

C(12,3) ways=$\frac{12!}{3!(12-3)!}$= 220 ways.

2 women can be chosen from 8 women in

C(8,2) ways=$\frac{8!}{2!(8-2)!}$= 28 ways.

Hence total number of different committees possible is 220 ×28=6160

Example 6: A bag contains six white flowers and five red flowers. Determine the number of ways to select four flowers under the following conditions:

i. The flowers can be of any color.

ii. Two flowers must be white, and two must be red.

iii. All four flowers must be of the same color.

Solution:

i. Here total number of flowers is 11. Four flowers can of any color from 11 marbles can be chosen in C (11,4) ways=330 ways.

ii. Two white flowers can be chosen in C (6,2) ways and two red flowers can be chosen in C(5,2) ways. Thus there are C (6,2) × C (5,2) = 150 ways.

iii. There are C (6,4) =15 ways of drawing four white flowers and C (5,4)=5 ways of drawing 4 red flowers. Thus there are 15+5=20 ways of drawing four flowers of the same color.

9.3 Permutations: A **permutation** is an arrangement of objects in a specific order. The order of arrangement is important in permutations. It calculates the number of ways to arrange n objects taken r at a time.

The formula for permutations is:

$$P(n,r)= \frac{n!}{(n-r)!} \ (r \leq n)$$

Permutation for the three letters A, B and C taking two at a time can be : AB, BA, AC, CA, BC, CB.

The total number of arrangements can be calculated as

$$P(3,2) = \frac{3!}{(3-2)!} = \frac{3 \times 2 \times 1}{1} = 6$$

Permutation can be classified in three different categories:

i. Permutation when repetition is not allowed
ii. Permutation when repetition is allowed
iii. Permutation when there are repeated elements.

Example1: Determine the number of ways in which letters of the word "APPLE" be arranged?

Solution: The word "APPLE" has 5 letters, where "P" appears twice. So the total permutations is:

$$\frac{n!}{p_1! \cdot p_2! \cdot \ldots \cdot p_k!} = \frac{5!}{2!} = \frac{120}{2} = 60$$

Example2: Determine the number of ways in which 5 people be seated in a row?
Solution: Here, n=5n = 5n=5. The total number of arrangements is:

$$P(5,5) = 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

Thus, there are **120 ways** to arrange the 5 people.

Example3: Determine the number of permutations where 4 out of 6 objects be arranged?

Solution: Use the formula for permutation P(n,r):

$$P(6,4) = \frac{6!}{(6-4)!} = \frac{6 \times 5 \times 4 \times 3 \times 2 \times 1}{2 \times 1} = 360$$

Thus, there are **360 ways** to arrange 4 objects out of 6

Example 4: How many number of three digits can be formed from the digits 3,4,5,6,7,8? How many of these are divisible by 5.

Solution: Here n=6 and r=3.

So number of three digits that can be formed is

$P(6,3) = \frac{6!}{(6-3)!} = 120$

To find the number divisible by 5, we fix the digit 5 on unit place.

Now n=6-1=5 and r=3-1=2

$P(5,2) = \frac{5!}{(5-2)!} = 20$

Permutation with repetitions:

The permutation of n objects taken all at a time, when there are P objects of one kind, q objects are of other kind, r objects are of third kind is

$$\frac{n!}{p!q!r!}$$

Example 1: How many distinct seven-letter words can be created using the letters of the word BENZENE**?**

Solution: There are seven objects (letters) of which there are three Es and two Ns. Therefore, total number of permutation is

$$\frac{7!}{3!2!} = \frac{7\times6\times5\times4\times3!}{3!2} = 420$$

Example 2:

a. Find the distinct arrangements that can be done using the letters of the word **ARRANGE**?

b. Find the number of arrangements so that the letters of the word **ARRANGE** be arranged such that the Rs are always together?

c. Find the total number of ways of so that the letter of the word **ARRANGE** be arranged such that the Rs are never together?

Solution:

a. There are 7 letters out of which two As and Rs.

So the total number of alphabets to arrange (n)=7

Number of repetition of letter A =2

Number of repetition of letter R =2

Total number of arrangements= $\frac{7!}{2!2!}$ =1260 ways.

b. For the case when two Rs are always together, we consider two Rs as a single letter. Then the word ARRANGE becomes ARANGE. So there are six letters including two As.

Therefore total number of arrangements = $\frac{6!}{2!}$ =360 ways.

c. The total number of arrangements of letters of word ARRANGE is 1260 and total words when two Rs always come together is 360. So the number of ways where two Rs never come together is (1260-360) =900 ways.

9.4Pigeonhole Principle and Principles of Inclusion-Exclusion:

Pigeonhole principle: The pigeonhole principle asserts that if the number of pigeons exceeds the number of pigeonholes by at least two, then at the minimum one pigeonhole must contain more than one pigeon.

Theorem 1: The Pigeonhole Principle

The Pigeonhole Principle states that if more than k objects are distributed into k boxes, then at least one box will contain more than one object.

Example: Prove that if a class has 30 students, at least two of them have last names starting with the same letter.

Solution: There are 30 students in the class and we have 26 letters in English alphabet that can be used in beginning of the last name. Since there are only 26 letters and 30 students, by pigeonhole principle at least two students have the last name that begins with the same letter.

Theorem 2: The generalized pigeonhole principle: If N objects are placed into k boxes, then there is at least one box containing at least $\lceil N/k \rceil$ objects.

Example: If a class has 24 students, what is the maximum number of possible grading that must be done to ensure that there at least two students with the same grade.

Solution:

There are total 24 students and the class and at least two students must have same grade.

If the number of possible grades is k then by pigeonhole principle we have $\lceil 24/k \rceil = 2$.

Here the largest value that k can have is 23 since 24 = 23.1 + 1. So the maximum number

of possible grading to ensure that at least two of the students have same grading is 23.

Example:

How many numbers must be selected from the set {1, 3, 5, 7, 9, 11, 13, 15} to guarantee that at least one pair of these numbers add up to 16?

Solution:

The pairs of numbers that sum 16 are (1,15), (3, 13), (5, 11), (7, 9) i.e. 4 pairs of numbers are there that add to 16. If we select 5 numbers, then by pigeonhole principle there are at least $\lceil 5/4 \rceil = 2$ numbers, that are from the set of selected 5 numbers, that constitute a pair. Hence 5 numbers must be selected.

Example 2: Find the least number of cables required to connect eight computers to four printers to guarantee that four computers can directly access four different printers. Justify your answer.

Solution: If we connect first 4 computers directly to each of the 4 printers and the other 4 computers are connected to all the printers, then the number of connection required is 4 + 4.4 = 20. To verify that 20 is the least number of cables required we have if there may be less than 20 cables then we would have 19 cables, then some printers would be connected by at most $\lceil 19/4 \rceil = 4$ cables to the computers. Then the other 3 printers would have to connect the other 4 computers here all the computers cannot simultaneously access.

Inclusion and Exclusion and Applications

In the counting problems where the sets are not disjoint we extensively use inclusion exclusion principle. Given set A and set B the union of A and B is given by the formula $|A \cup B| = |A| + |B| - |A \cap B|$.

Example 1:

There are 345 students at a college who have taken a course in calculus, 212 who have taken a course in discrete mathematics, and 188 who have taken course in both calculus and discrete mathematics. How many students have taken the course in either calculus or discrete mathematics?

Solution:

Here we have $|C| = 345$ (students taking the calculus course), $|D| = 212$ (students taking the discrete mathematics course), and $|C \cap D| = 188$ (students taking both discrete

mathematics and calculus courses). Number of students taking either discrete mathematics or calculus, $|C \cup D| = |C| + |D| - |C \cap D| = 345 + 212 - 188 = 369$.

Theorem 8: The Principle of Inclusion – Exclusion

Let $A_1, A_2, \ldots, A_n$ be finite sets. Then $|A_1 \cup A_2 \cup \ldots \cup A_n|$

$$= \sum_{1 \le i \le n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| - \ldots + (-1)^{n+1} |A_1 \cap A_2 \cap \cdots \cap A_n|.$$

Example:
How many elements are in the union of four sets if each of the set has 100 elements, each pair of the set shares 50 elements, each three of the sets share 25 elements, and there are 5 elements in all four sets?

Solution: Try Yourself

9.5 Unit Summary: This unit provides a structured approach to **counting techniques**, essential in solving complex combinatorial problems in discrete mathematics. It begins with an introduction to the **basics of counting**, including the fundamental principles such as the addition and multiplication rules, which form the foundation for more advanced methods. The unit explores **combinations** and **permutations**, emphasizing their differences and applications. Combinations focus on selecting items without considering the order, while permutations deal with arrangements where order matters. Practical examples like seating arrangements or choosing committees illustrate these principles.

The unit also covers the **Pigeonhole Principle**, which states that if n+1tems are placed into n containers, at least one container must hold more than one item. This is extended into **the Principle of Inclusion-Exclusion**, a technique to compute the number of elements in the union of multiple sets by accounting for overlaps systematically. These advanced methods are particularly useful in solving problems involving overlapping subsets and optimization scenarios. On completion of this unit the students should understand how to apply these counting techniques to a variety of mathematical and real-world problems.

9.5 Check Your Progress.

1. Describe and clarify the addition and multiplication rules, providing examples for each.
2. How many 5-digit numbers can be made by the digits 0-9 if repetition is allowed?
3. In how many different ways can a team of 3 be selected from a group of 10 individuals?
4. After shuffling a deck of 52 cards, how many different ways can you draw 5 cards, all of which are hearts?
5. How many unique ways can the letters of the word "COMPUTE" be arranged?
6. Demonstrate that in any group of 13 individuals, at least two must share the same birth month.
7. How many integers between 1 and 100 are divisible by either 3 or 5?

Unit 10: Binomial and Multinomial Theorems.

10.0 Introduction and Unit Objectives: The Binomial and Multinomial Theorems are fundamental components of combinatorial mathematics, providing efficient methods for expanding expressions raised to a power. Historically, the binomial theorem has been known for centuries, with early instances appearing in the works of mathematicians such as Al-Karaji and Omar Khayyam. It has profound applications in various fields, including algebra, calculus, and probability theory. In probability, for instance, the binomial theorem underpins the binomial distribution, which models the number of successes in a fixed number of independent Bernoulli trials. Additionally, the theorem is instrumental in combinatorics for calculating combinations and in calculus for series expansions. Its utility extends to computational fields where polynomial expressions are prevalent, making it a fundamental tool in both theoretical and applied mathematics. The multinomial theorem extends the binomial theorem to expressions involving sums of multiple terms. It provides a method to expand any power of a sum of several variables into a sum of products of these variables, each multiplied by

a specific coefficient. These theorems have extensive applications across various fields, including algebra, calculus, probability theory, and computer science.

Unit Objectives: On completion of this unit, the learner will be able

1. Understand Binomial Coefficients: Learn the definition and calculation of binomial coefficients.
2. Comprehend the Binomial Theorem: Grasp the statement and proof of the Binomial Theorem.
3. Explore the Multinomial Theorem: Extend the understanding of the Binomial Theorem.
4. Apply Theorems to Problem-Solving: Develop the ability to apply both the Binomial and Multinomial Theorems
5. Analyze Theoretical Implications: Investigate the theoretical significance of these theorems in mathematical proofs and their role in the development of more advanced mathematical concepts.

10.1 Binomial Coefficients and Binomial Theorem:

Statement: According to the binomial theorem, it is possible to expand any non-negative power of binomial (x + y) into a sum of the form.

$$(a+b)^n = {}^nC_0\, a^n b^0 + {}^nC_1\, a^{n-1} b^1 + {}^nC_2\, a^{n-2} b^2 + \ldots + {}^nC_{n-1}\, a^1 b^{n-1} + {}^nC_n\, a^0 b^n$$

General term:
If $\mathbf{(r+1)^{th}}$ term in the expansion of $(x+y)^n$ is usually called the general term which is denoted by t_{r+1} .
In the expansion of $(x+y)^n$
$t_1 = 1^{st}$ term $= C(n,0)x^n$
$t_2 = 2^{nd}$ term $= C(n,1)x^{n-1}\ y^1$
$t_3 = 3^{rd}$ term $= C(n,2)x^{n-2}\ y^2$
$t_{r+1} = (r+1)^{th}$ term $= C(n,r)x^{n-r}\ y^r$

SO the General term is $C(n,r)x^{n-r}\ y^r$

Example: Expand $(x+3)^5$ using the binomial theorem.

Solution: Here y = 3 and n = 5. Substituting and expanding, we get:

$$(x+3)^5 = {}^5C_0\, x^5 3^0 + {}^5C_1\, x^{5-1} 3^1 + {}^5C_2\, x^{5-2}\, 3^2 + {}^5C_3\, x^{5-3}\, 3^3 + {}^5C_4\, x^{5-4}\, 3^4 + {}^5C_5\, x^{5-5}\, 3^5$$

$$= x^5 + 5\,x^4 . 3 + 10\,x^3 . 9 + 10\,x^2 . 27 + 5x . 81 + 3^5$$

$$= x^5 + 15\,x^4 + 90x^3 + 270\,x^2 + 405\,x + 243$$

Properties of Binomial Theorem:

1. The number of coefficients in the binomial expansion of $(x + y)^n$ is equal to $(n + 1)$.
2. There are $(n+1)$ terms in the expansion of $(x+y)^n$.
3. The first and the last terms are x^n and y^n respectively.

4. From the beginning of the expansion of $(x + a)^n$, the powers of x, decrease from n up to 0, and the powers of a, increase from 0 up to n.

Example: Find the 6th term in the expansion of $(3x + 4)^8$
Solution: The general term in the expansion of $(3x + 4)^8$ is,

$T_{r+1} = {}^8C_r(3x)^{8-r}(4)=$

For 6th term r = 5

$T_6 = T_{5+1} = {}^8C_5(3x)^{8-5}(4)^5$

Simplifying the above term, we get our answer,

$T_6 = {}^8C_5(3x)^{8-5}(4)^5$

$= (8.7.6/3.2.1)\ (27)(1024)x_3$

$= 1593648x^3$

Example: Find the value of $(x+y)^2$ and $(x+y)^3$ using Binomial expansion.
Solution:

$(x+y)^2 = {}^2C_0\ x^2y^0 + {}^2C_1\ x^{2-1}y^1 + {}^2C_2\ x^{2-2}\ y^2$
$\Rightarrow (x+y)^2 = x=+\ 2xy + y^2$

And $(x+y)^3 = {}^3C_0\ x^3y^0 + {}^3C_1\ x^{3-1}y^1 + {}^3C_2\ x^{3-2}\ y^2 + {}^3C_3\ x^{3-3}\ y^3$
$\Rightarrow (x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$
$\Rightarrow (x+y)^3 = x^3 + 3xy(x+y) + y^3$

10.2 Multinomial Theorem: The multinomial theorem is a generalization of the binomial theorem to polynomials with more than two terms. While the binomial theorem provides a method to expand expressions of the form $(x+y)^n$, the multinomial theorem extends this to expressions like $(x_1+x_1+\cdots+x_m)^n$, where m is the number of terms and n is a non-negative integer exponent.

For example, the expansion of $(x_1 + x_2 + x_3)^3$ is
$x_1^3 + 3x_1^2x_2 + 3x_1^2x_3 + 3x_1x_2^2 + 3x_1x_3^2 + 6x_1x_2x_3 + x_2^3 + 3x_2^2x_3 + 3x_2x_3^2 + x_3^3$.

The General term of Multinomial Expansion:

$$(x_1 + x_2 + \cdots + x_m)^n = \sum_{\substack{k_1+k_2+\cdots+k_m=n \\ k_1,k_2,\cdots,k_m \geq 0}} \binom{n}{k_1, k_2, \ldots, k_m} x_1^{k_1} \cdot x_2^{k_2} \cdots x_m^{k_m}$$

Number of terms in Multinomial Expansion: The number of terms in the multinomial theorem is given by:

$${}^{n+k-1}C_{k-1}$$

Consider the expansion of $(a + b+ c)^3$ Here, n=3 and k= 3.

The number of distinct terms

$${}^{3+3-1}C_{4-1}$$

$${}^{5}C_{2} = 10$$

Example: Expand $(x + 2y - z)^3$ using the Multinomial Theorem

Solution:

$(x + 2y - z)^3$

$= x^3 + 8y^3 - z^3 + 6x^2y - 3x^2z + 12xy^2 - 6xyz - 3xz^2 + 12y^2z - 6yz^2$

Example: Find the term containing x^2y^2z in the expansion of $(2x - 3y + z)^5$

Solution: The term is $(5! / (2!2!1!)) * 2^2 * (-3)^2 * 1^1 * x^2y^2z = 180x^2y^2z$

10.3 **Unit Summary:** Both the Binomial and Multinomial Theorems are essential tools in mathematics, providing efficient methods for expanding expressions raised to a power. Their applications span various disciplines, including algebra, probability, and statistics, highlighting their significance in both theoretical and applied contexts. The Multinomial Theorem is particularly useful in Combinatorics for Calculating probabilities and outcomes involving multiple categories. In statistical Distributions it is used Understanding distributions like the multinomial distribution, which generalizes the binomial distribution to more than two outcomes.

10.4 Check Your Progress:

1.Expand the expression $(2x-3)^6$ using the Binomial Theorem

2. Find the 7th term in the expansion of $(x+2)^{10}$

3. State the Multinomial Theorem and explain its significance in combinatorics and algebra.

4. Expand $(x + y + z)^4$ with the help of Multinomial Theorem.

5. In how many ways can 10 identical objects be distributed among 4 distinct boxes? Use the Multinomial Theorem to justify your answer.

6. Prove that the sum of all multinomial coefficients for a given n and k is equal to kn.

Unit 11: Introduction to Graph Theory

11.0 Introduction and Unit Objectives: Graph theory is a foundational concept in mathematics and computer science that studies the properties, structures, and relationships of graphs. A **graph** consists of a set of **vertices (nodes)** connected by **edges (lines)**, representing relationships or connections. Graphs are widely used to model networks in diverse fields such as communication, social networks, transportation, and computer algorithms. Graphs can be used to represent almost any problem involving discrete arrangements of objects.

This unit introduces the fundamental concepts of **graphs and multigraphs**, including **subgraphs** and various types of graphs such as directed, undirected, weighted, and unweighted graphs. It also covers **graph representation methods**, such as adjacency matrices and lists and the concept of **graph isomorphism**, which explores structural similarity between graphs.

Unit Objectives: By the end of this unit, students will:

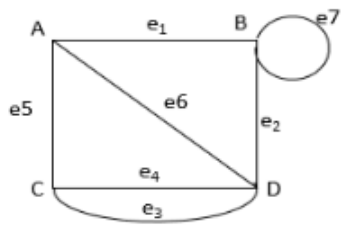
1. Understand the basic concepts and terminologies of graphs, multigraphs, and subgraphs.

2. Identify and differentiate between types of graphs.
3. Learn methods to represent graphs and analyze their structures.
4. Understand the concept of isomorphism and its application in comparing graph structures.

This knowledge forms the basis for advanced topics like graph algorithms, graph coloring, and network analysis.

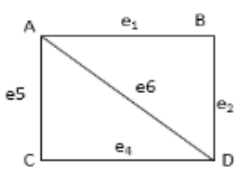
11.1 Basic Concepts of Graph and multigraph, Subgraphs, types of graph.

Graph is a discrete structure with vertices and edges connecting the vertices. A graph G= (V, E) is a mathematical model consist of two non-empty sets V= $\{v_1, v_2, v_3, \ldots, v_n\}$called set of vertices and E=$\{e_1, e_2, e_3, \ldots, e_n\}$ called set of edges. We write it as G=(V, E).

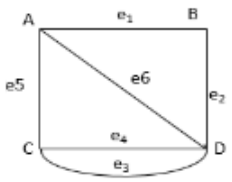


The above figure represents a graph G (V, E) where set of vertices V consist of {A, B, C, D} and the set of edges E consist of $\{e_1, e_2, e_3, e_4, e_5\}$. The edge e_1 represents {A, B} and so on.
In the graph the edge e_7 is called a **loop** as it has the same vertex B as starting and ending vertex. Also the edges e_3 and e_4 are called **parallel edges**. Parallel edges are those that starts at one vertex and ends at the other.

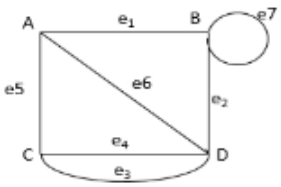
Simple Graphs: A graph G=(V,E) is said to be simple if G has no loops and parallel edges. For example



Multi graph: A computer network may contain multiple links between data centers, to model such networks we need graphs that have more than one edge connecting the same pair of vertices. Two or more edges between same pair of vertices are called parallel edges. A graph G=(V,E) is called multi-graph if contains parallel edges. The following graph is a multigraph as it contains parallel edges e_3 and e_4.



Pseudo graph: A graph G=(V,E) is called pseudograph if G contains both loops and parallel or it contains only loops. The following is a pseudograph that contains a loop (edge e_7) and parallel or multiple edges (e_3 and e_4)



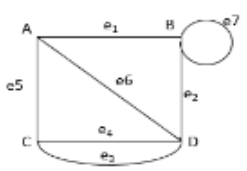
Terminologies:

In a finite graph G(V,E) , two vertices u and v are termed adjacent if there exists an edge {u,v}∈E. An edge e is said to be incident on the vertices u and v if e={u,v}, thereby connecting u and v, which are referred to as the endpoints of the edge.

The degree of a vertex v in an undirected graph, denoted as deg (v), represents the total number of edges incident on v. For a loop at a vertex, it contributes twice to the degree. A vertex with no incident edges (deg(v)=0) is classified as an isolated vertex, while a vertex with exactly one incident edge (deg(v)=1) is referred to as a pendant vertex.

Additionally, the total number of vertices in G is defined as the graph's order, while the total number of edges is referred to as its size.

Example: Find the degrees of the vertices in the following graph.



Solution: deg (A)=3, deg(B)=4 (as loop contribute two to the degree of a vertex),
deg (C)=3, deg(D)=4.

Theorem: The Handshaking Theorem
Let G = (V, E) be an undirected graph with e edges. Then $2e = \sum \deg(v_i)$ i.e the sum of degrees of all the vertices is equal to the twice the number of edges in the graph.

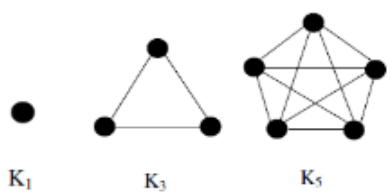
Theorem: The number of odd vertices in a graph is always even.

Special Classes of Graphs:

Trivial Graph: A graph is classified as trivial if it contains precisely one vertex and no edges.

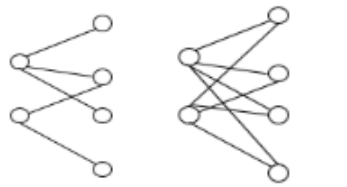
Connected Graph: A graph G is connected if there exists a path between every pair of vertices, ensuring no vertex is isolated.

Complete Graph: A graph G is termed complete if every pair of distinct vertices in G is joined by exactly one edge. The complete graph with n vertices is symbolized as Kn, where n denotes the total number of vertices. By definition, a complete graph is always connected due to its fully interconnected structure.



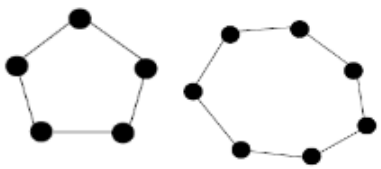
Regular Graph: A graph having vertices of the same degree is known as a regular graph. If each vertex has a degree of 'r', the graph is referred as an r-regular graph. For instance, K_1 is 1-regular, K_3 is 3-regular, and K_5 is 5-regular. In a regular graph with n vertices, the degree of each vertex is one less than the total number of vertices in the graph.

Bipartite graph: A simple graph G is considered bipartite if its vertex set V can be divided into two disjoint subsets, V1 and V2, such that every edge in the graph connects a vertex from V1 to a vertex in V2. No edges connect vertices within the same subset.
Complete Bipartite Graphs: A graph denoted by $K_{m,n}$ is complete bipartite if each vertex of the first set is connected to all vertices of the second bipartite set. The given below complete bipartite graph can be denoted as $k_{2,4}$.



Bipartite Graph Complete Bipartite

Cycles: The cycle C_n, $n \geq 3$, consists of n vertices $v_1, v_2, \ldots, v_n$ and edges $\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{n-1}, v_n\}$, and $\{v_n, v_1\}$.



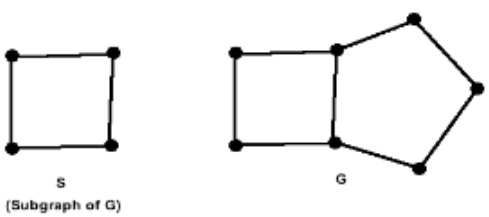
C_5 C_7

Subgraphs: Let G= (V, E) be a graph with vertex set V (G) and edge set E(G). Let H=(V, E) be a graph with vertex set V(H) and edge set E(H). Then H is referred as subgraph of G written as $H \subseteq G$ if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. H satisfies the following properties

a.Every vertices of H belongs to G

b.Every edges of H are in G.

c.Each edge has the same end points in H and G.



S
(Subgraph of G)

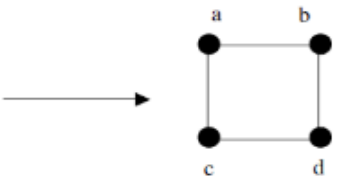
G

11.2 Representation of Graphs and Isomorphism:

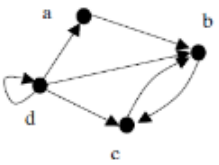
Adjacency List:

This type of representation is suitable for the undirected graphs without multiple edges, and directed graphs. This representation looks as in the tables below.

Edge List for Simple Graph	
Vertex	Adjacent Vertices
a	b, c
b	a, d
c	a, d
d	b, c



Edge List for Directed Graph	
Initial Vertex	End Vertices
a	b
b	c
c	b
d	a, b, c, d



If we try to apply the algorithms of graph using the representation of graphs by lists of edges, or adjacency lists it can be tedious and time taking if there are large number of edges.

Representing a graph using adjacency matrix:

Given a simple graph G =(V, E) with |V| = n. Let v1, v2, …, vn are the vertices of the graph in order. Adjacency matrix A of G, with respect to the order of vertex is n-by-n zero-one matrix (A = [a_{ij}]) with the condition,

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

In case of the directed graph we can extend the same concept as in undirected graph as dictated by the relation.

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

Incidence matrix Representation:

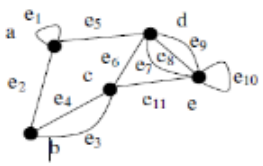
This is another way of representing graph. Given an undirected graph G = (V, E). Assume that the vertices of the graph are $v_1, v_2, \ldots, v_n$ and the edges of the graph are $e_1, e_2, \ldots, e_m$. the incidence matrix of a graph with respect to the above ordering of V and E is n-by-m matrix M = [m_{ij}],

$$\text{where } m_{ij} = \begin{cases} 1 & \text{when edge } e_j \text{ is incident with } v_i, \\ 0 & \text{otherwise.} \end{cases}$$

When the graph is not simple then also the graph can be represented by using incidence

matrix where multiple edges corresponds to two different columns with exactly same entries. Loops are represented with column with only one entry.

Example: Use adjacency matrix and incidence matrix to represent the following graph.



Solution:
Let the order of the vertices be a, b, c, d, e, f and edges order be e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}.

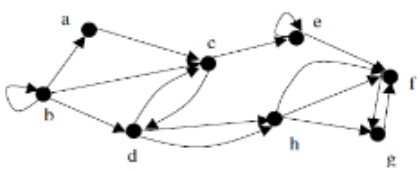
$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 2 & 0 & 0 \\ 0 & 2 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 3 & 1 \end{bmatrix}$$

Adjacency Matrix

$$\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 & e_{10} & e_{11} \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Incidence Matrix

Example: Represent the following directed graph using adjacency matrix.



Solution: The required adjacency matrix is

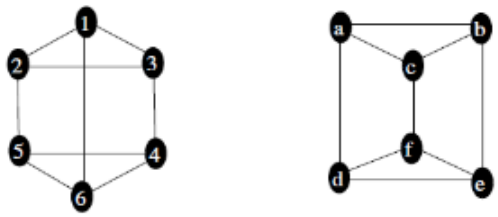
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 1 & 0 \end{bmatrix}$$

Graph Isomorphism Graph isomorphism is a concept in graph theory where two graphs are considered isomorphic if there exists a one-to-one correspondence between their vertices and edges, preserving adjacency relationships. Essentially, isomorphic graphs have identical structures but may differ in their vertex or edge labels.

Isomorphic simple graphs have same number of vertices (one-to-one correspondence between vertices of two graphs is required). Isomorphic simple graphs have same number of edges (due to adjacency preservation). Degrees of the vertices in the isomorphic graphs must be same because the number of edges from that vertex is determined by degree.

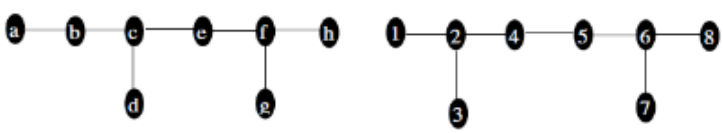
Existence of a simple circuit of length n, where n is a positive integer greater than 2, in both the graphs is an invariant. The subgraphs formed by connecting the edges from the vertex with same degree in both the graphs must be isomorphic.

Example: Determine whether the given two graphs are isomorphic or not?



Solution: In the above two graphs number of vertices in both graphs is same (i.e. 6), number of edges equal to 9 in both the graphs and all the vertices in both the graphs have degree 3. Since the invariants agree in both the graphs, we can try out to find the function that is isomorphism. Take the sequence of vertices from the first graph as 1, 2, 3, 4, 5, and 6. Now define f(1) = c, f(2) = a here there is adjacency preservation since we have {1, 2} as and edge in the first graph whereas {f(1), f(2)} ={c, a} is an edge in the second graph. Similarly we can assign f(3) = b, f(4) = e, f(5) = d, f(6) = f. Since we found one to one correspondence between vertices of two graphs persevering the adjacency, the above two graphs are isomorphic.

Example: Determine whether the given two graphs are isomorphic or not?



Solution:
In the above two graphs number of vertices in both graphs is 8, the number of edges equal to 7 in both the graphs, in both graphs two vertices have degree 3, 4 vertices have degree 1 and the remaining 2 vertices have degree 2. Since the invariants agree in both the graphs, we can continue to get the function such that it is isomorphism. However, in case of first graph the subgraph containing the vertex c (degree 3), with vertices a, b, c, d, and e is not isomorphic with any of the subgraph formed by connecting edges with vertex 2 or 6 (both of degree 3). Hence the two above graphs are not isomorphic.

11.3 Unit Summary: Graph Theory is a branch of discrete mathematics that studies graphs as mathematical structures to model pairwise relationships between objects. A graph consists of vertices (or nodes) connected by edges (or lines), and its applications range from computer science

and biology to social networks and logistics. The unit begins by introducing foundational concepts, setting objectives to familiarize learners with different types of graphs, their representations, and isomorphism. It aims to provide a strong theoretical framework for understanding the properties and uses of graphs in various domains.

The unit covers essential topics such as basic graph concepts, multigraphs, subgraphs, and types of graphs like simple graphs, pseudo graphs, directed graphs, and bipartite graphs. It then explores methods for graph representation using adjacency matrices, incidence matrices, and adjacency lists, highlighting their efficiency in computational applications.The unit concludes with a summary that ties together the key principles, ensuring a clear understanding of graphs' theoretical and practical aspects

11.4Check Your progress

1. What is a graph in graph theory, and how is it defined?
2. What is the difference between a simple graph and a multigraph?
3. Explain the concept of vertices and edges in a graph.
4. What is a subgraph, and how is it related to a larger graph?
5. List and explain at least five types of graphs with examples.
6. How can graphs be represented using adjacency matrices and adjacency lists?
7. What is graph isomorphism, and how can we determine if two graphs are isomorphic?
8. Describe the properties of a bipartite graph and give an example.
9. How do directed and undirected graphs differ in terms of their edges?

Unit 12: Graph Traversal.

12.0 Introduction and Unit Objectives: Graph traversal is a key concept in graph theory that helps in exploring the structure of a graph. A graph consists of vertices (nodes) and edges (connections between nodes). Traversing a graph involves visiting each vertex at least once, in a specific order. There are several methods to explore graphs, including exploring paths, circuits, and using traversal algorithms like Depth-First Search (DFS) and Breadth-First Search (BFS). These concepts include Graph Traversal which involves systematically visiting each vertex and edge in a graph. DFS and BFS are popular algorithms for this. Depth-First Search (DFS) which is a traversal method that explores as far as possible along a branch before backtracking. **Breadth-First Search (BFS)**. A traversal method that explores all neighbors of a vertex before moving on to the next level of vertices.

Unit Objectives: On completion of this unit, the students will be able to

1. Understand the basic concepts of paths, circuits, and graph traversal.
2. Learn the difference between DFS and BFS.
3. Implement and apply DFS and BFS algorithms to solve graph-related problems.
4. Explore the significance of traversal algorithms in real-world applications such as searching, networking, and pathfinding.

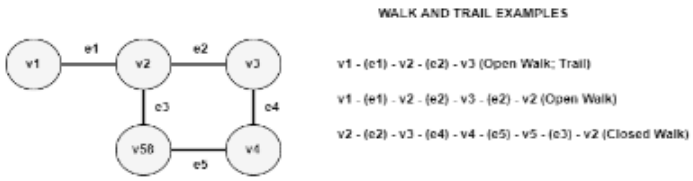
12.1 Path and Circuit, Graph Traversal:

Walk: If we traverse a graph then we get a walk. Edge and Vertices can both be repeated.

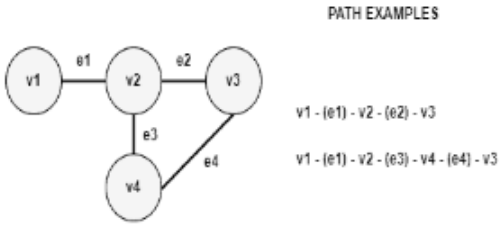
Open walk- A walk is open walk if the starting and ending vertices are different i.e. the origin vertex and terminal vertex are different.

Closed walk- A walk is closed walk if the starting and ending vertices are identical i.e. if a walk starts and ends at the same vertex, then it is said to be a closed walk.

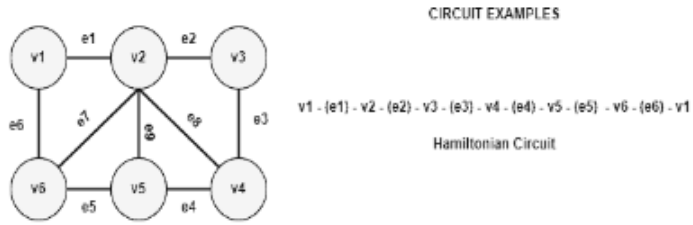
Trail: Trail is an open walk in which no edge is repeated, and vertex can be repeated. There are two types of trails: Open trail and closed trail. The trail whose starting and ending vertex is same is called closed trail. The trail whose starting and ending vertex is different is called open trail.



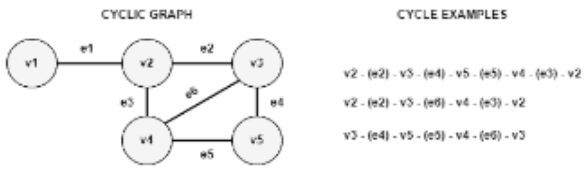
Path: It is a trail in which neither vertices nor edges are repeated i.e. if we traverse a graph such that we do not repeat a vertex and nor we repeat an edge. As path is also a trail, thus it is also an open walk.



Circuit: Traversing a graph such that not an edge is repeated but vertex can be repeated, and it is closed also i.e. it is a closed trail.



Cycle: Traversing a graph such that we do not repeat a vertex, nor we repeat an edge but the starting and ending vertex must be same i.e. we can repeat starting and ending vertex only then we get a cycle. In other words, cycle can be defined as the closed path.



The table below represents the repetition of edges and vertices in walk, trail and path.

Category	**Edges**	**Vertices**
Walk	Can be Repeated	Can be Repeated

Trail	Cannot be Repeated	Can be Repeated
Path	Cannot be Repeated	Cannot be Repeated

Graph Traversal Algorithms:

It is a systematic method for visiting all the vertices and edges in a graph. It ensures that every vertex is visited once to explore the graph structure, extract useful information, or solve problems like finding paths, detecting cycles, or analyzing connectivity.

There are two primary types of graph traversal algorithms:

A. **Breadth-First Search (BFS)**:

a. Traverses the graph level by level.
b. Uses a queue to store of vertices to visit.
c. Useful for finding shortest path in unweighted graphs and exploring connected components.

B. **Depth-First Search (DFS)**:

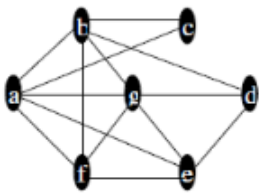
a. Explores as far as possible along each branch before backtracking.
b. Uses a stack (or recursion) for tracking.
c. Useful for tasks like detecting cycles, topological sorting, and solving puzzles like mazes.

Graph traversal is fundamental for understanding the relationships between nodes in graphs and solving real-world problems like network routing or social network analysis.

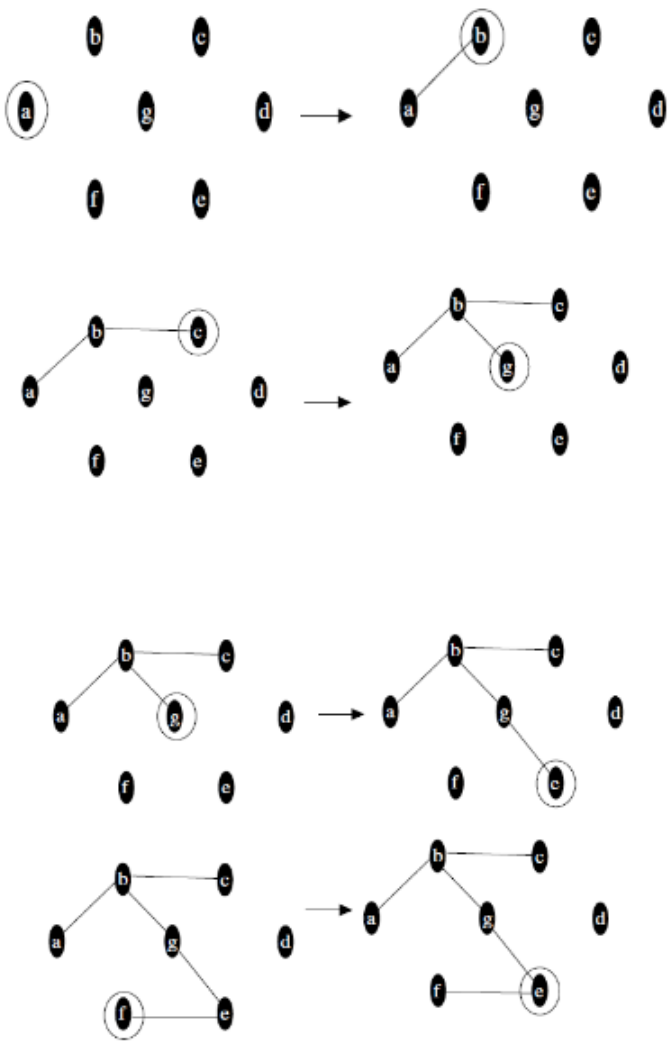
12.2 Depth-First Search (DFS): Depth-First Search (DFS) is a method used to find the spanning tree of a connected graph. The process begins by selecting an arbitrary vertex as the root. From the root, we create a path by consecutively adding vertices and edges, ensuring that the added edge connects to the last vertex in the path and has not been traversed before. This continues until no further path can be formed. If the path includes all the vertices, then the tree formed by this path is a spanning tree. If not, additional vertices and edges must be added. To do this, we backtrack from the last vertex in the current path and check if a new path can be formed starting from this vertex. If a new path is found, the process repeats. If no new path can be found, we backtrack to another vertex and continue the process. This continues until all

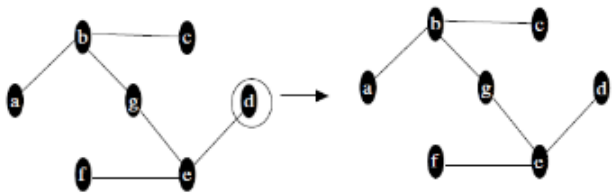
vertices are included, and since the graph is finite, the process eventually terminates, providing a spanning tree. The graph formed by the edges of this rooted tree is the spanning tree of the original graph.

Example: Use the depth -First Search algorithm to traverse through the graph.



Solution: Choose a as initial vertex and following the steps mentioned above do the traversal.





12.3 Breadth-First Search (BFS):

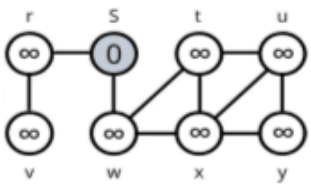
It is a graph traversal algorithm that explores nodes layer by layer. It starts from a given source node, visits all its adjacent nodes, and then moves to the next level of neighbors. We can start traversing from any node as root node. BFS uses a queue data structure to ensure that nodes are visited in the correct order.

The pseudocode for BFS is given below

BreadthFirstSearch(graph, startNode):

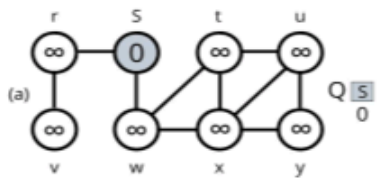
1. *Create a queue Q*
2. *Mark startNode as visited and enqueue it into Q*
3. *While Q is not empty:*
    a. *Dequeue the front element, currentNode, from Q*
    b. *Process currentNode (e.g., print it or add to result list)*
    c. *For each neighbor of currentNode:*
        i. *If the neighbor is not visited:*
            - *Mark it as visited*
            - *Enqueue it into Q*

Example: Use the Breadth-First Search algorithm to traverse through the graph. Consider s as the initial vertex.

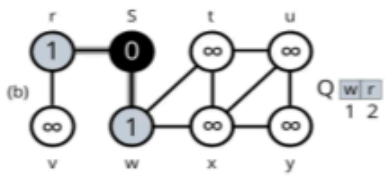


Solution: The BFS algorithm begins with all roots unvisited first and select one node (S) as

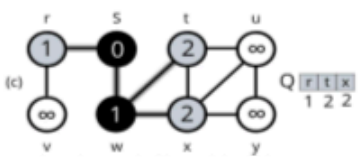
source node. The vertex s is hence added to the queue.



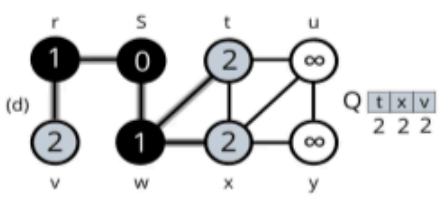
The algorithm begins to traverse in breadth first manner to discover the neighboring nodes. Thus it visits vertices w and r and mark them visited by changing the color to gray. So r and w are enqueued. The vertex s is dequequed because it is visited.



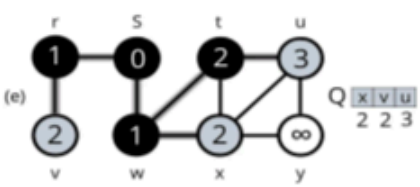
Next the neighboring nodes of recently discovered vertices (w and r) are visited and marked as visited by changing their color. The algorithm dequeue the node to visit its neighbors. In this case w is dequeued and its neighboring vertices t and x are queued. Once all the neighboring nodes to the specific vertex are discovered the algorithm mark it as black.



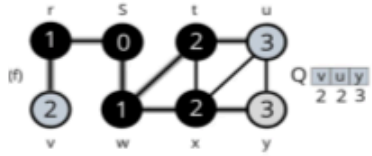
Similarly, vertex r is dequeued and is turned to black once its neighbor v is visited by the algorithm. Vertex v is enqueued next.



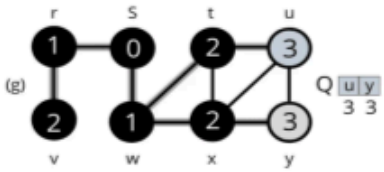
Next t is dequeued to visit its neighbor i.e u. Once u is visited it is added to the queue and the color is changed to black.



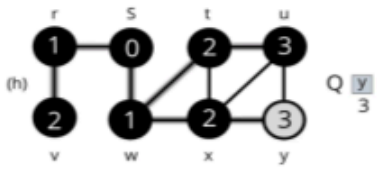
Vertex x being the next in queue is dequeued and its neighbor vertex y is visited. Once visited changes to black and vertex y changes to gray.



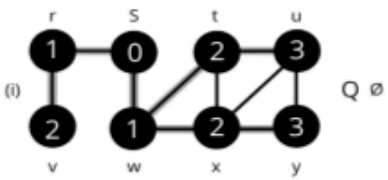
Next in queue vertex v is dequeued and is changed to black as it does not have any neighbor.



Similarly, vertex u is dequeued and is changed to black as it does not have any neighbor.



Similarly, vertex y is dequeued and is changed to black as it does not have any neighbor.

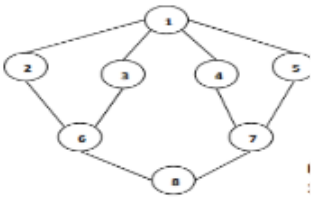


Once all the nodes are visited the algorithm computes the shortest path between the root s and the linking nodes.

12.4 Unit Summary: This unit covers graph traversal techniques, focusing on Depth-First Search (DFS) and Breadth-First Search (BFS) for systematically visiting vertices. DFS explores deeper paths using recursion or a stack, making it ideal for detecting cycles and topological sorting. BFS uses a queue for level-order traversal, suited for finding the shortest path in unweighted graphs, with applications in routing and network analysis. This unit equips learners with theoretical and practical insights into these traversal techniques, emphasizing their applications across graph problems.

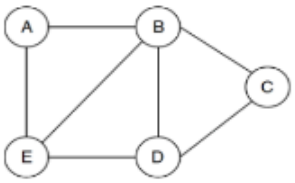
12.5 Check Your Progress.

1. Explain the practical applications of graph traversal techniques in computer science.
2. Define a path in a graph. How does it differ from a circuit?
3. Write the pseudocode for the Breadth-First Search (BFS) algorithm.
4. What is the difference between a simple path and a closed circuit in a graph?
5. Perform Depth First Search Traversal on the following graph



Graph G

6. Perform Depth First Search Traversal on the following graph



Unit 13: Advanced Graph Theory Concepts

13.0 Introduction and Unit Objectives: Graph theory is a critical branch of discrete mathematics with applications in computer science, operations research, and engineering. This unit explores advanced concepts in graph theory, including Eulerian and Hamiltonian graphs, planarity, and graph coloring. Eulerian and Hamiltonian graphs deal with paths and circuits that traverse edges or vertices under specific conditions. Planar graphs focus on graph embedding's in two-dimensional spaces without edge crossings. Graph coloring addresses the assignment of colors to vertices, edges, or regions of a graph, with chromatic numbers providing the minimum colors required. These concepts are essential for solving real-world problems like network routing, scheduling, and optimization.

Unit Objectives: By the end of this unit, learners will:

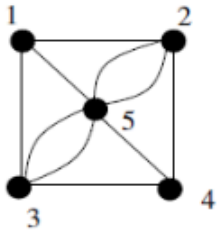
1. Understand the concepts of Eulerian paths and circuits, and their properties.
2. Analyze Hamiltonian paths and circuits and differentiate them from Eulerian graphs.
3. Explore planar graphs and determine conditions for graph planarity.
4. Learn about graph coloring techniques and calculate the chromatic number of graphs.
5. Apply these advanced graph theory concepts to practical problems in various domains.

This unit provides both theoretical insights and practical applications, ensuring a comprehensive understanding of advanced graph theory topics.

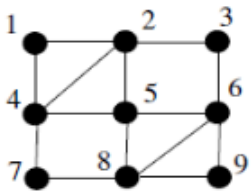
13.1 Euler Graph: Path and Circuit:

A Euler circuit is a specific type of Euler path that begins and ends at the same vertex while traversing every edge exactly once. To summarize, a Euler path covers all edges exactly once, starting and ending at different vertices, whereas a Euler circuit also covers all edges exactly once but starts and ends at the same vertex, with all vertices having an even degree.

Example: Find the Euler path or circuit in the following graphs.



For this graph --No Euler circuit exist, but the Euler path is (1,2,5,1,3,5,4,3,5,2,4)



For this graph --The Euler circuit is (1,2,3,6,9,8,7,4,5,8,6,5,2,4,1).

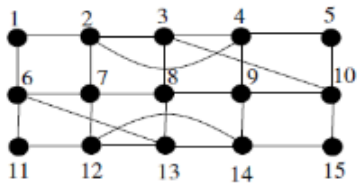
Conditions for existence of Euler Circuits and Paths

Theorem 1:

a. If a graph G contains a vertex with an odd degree, then G cannot have an Euler circuit.
b. If G is a connected graph and every vertex has an even degree, then G has a Euler circuit.

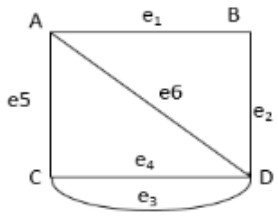
Theorem 2: a. If a graph G has more than two vertices with odd degrees, then it cannot have an Euler path.
b. If G is connected and has exactly two vertices with odd degrees, then there exists an Euler path in G.
Any Euler path in G must begin at one vertex with an odd degree and end at the other.

4o mini
Example: Consider the graph given below and state whether it has Euler circuit or path. Give reason.



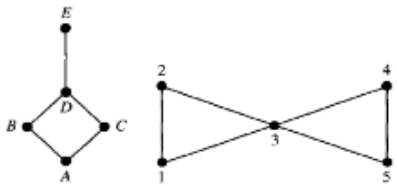
Solution: In the given graph all the vertices have even degree, hence there is an Euler circuit. The Euler circuit is 1,2,3,4,5,10,15,14,13,12,11,6,13,8,9,14,12,7,8,3,10,9,4,2,7,6,1. The graph has no Euler path because it do not have any odd degree vertices.

Example: Consider the graph given below and state whether it has Euler circuit or path. Give reason.



Solution: The given graph has Euler path because there are exactly two vertices of odd degree. The graph does not have Euler circuit because all vertices do not have even degree.

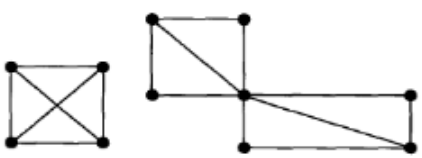
Example: Consider the graphs given below and state whether it has Euler circuit or path



Solution: The first figure has Euler path because there are exactly two vertices of odd degree i.e E and D. One Euler path is E-D-B-A-C-D and the other is D-B-A-C-D-E. One thing can be notice that if a graph has euler path then the path must begin at one of the odd degree vertex and ends at the other vertex having odd degree. This is true for all graph having Euler path.

The Second figure do not contain Euler path but it has Euler Circuit since all the vertices are of even degree. One such Euler circuit is 1-3-5-4-3-2-1.

Example: Consider the graphs given below and state whether it has Euler circuit or path

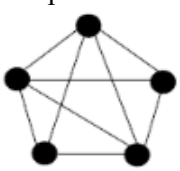


Solution: The first figure neither has Euler path not Euler Circuit. The second figure has Euler path.

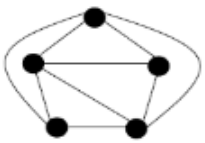
13.2 Planner Graph, Graph Coloring, Chromatic Number:

Planar graphs: A graph is termed **planar** if it can be represented on a plane in such a way that none of its edges intersect or cross each other. This representation is referred to as a **planar drawing** or **planar representation** of the graph.

Example: Draw the graph below as planar representation of the graph.



Solution: The planner representation of the graph is given below.



Euler's Formula: For a connected, planar, simple graph G with e edges and v vertices, let r represent the number of regions in its planar representation. According to Euler's formula for planar graphs: r=e−v+2.

Graph Coloring

It is a way to provide color to each vertex of a simple graph such that non adjacent vertices share the same color.

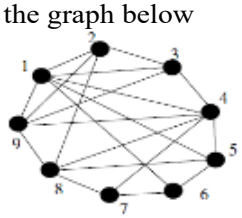
Chromatic Number

It is the minimum number of colors required to properly color the graph.

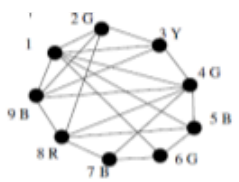
Theorem: The Four Color Theorem

The Four Color Theorem states that the chromatic number of any planar graph is at most four. This means that any planar graph can be properly colored using no more than four distinct colors.

Example: Find the chromatic number of the graph below



Solution: Let's start with vertex 1, it has adjacent vertices as 2, 3, 4, 5, 6, 9 so using only 2 colors would suffice for the graph having the edges from 1 to its adjacent vertices. However, since 9 has its adjacent vertices as 1, 2, 3, 4 we cannot just color the above graph with 2 colors because at least 1, 2 and 9 must have different colors. Trying with 3 colors we found that at least 1, 2, 3, and 9 must have different colors. So trying with four colors we can color the graph. Hence the chromatic number of the above graph is 4. Possible coloring is shown in the figure below.



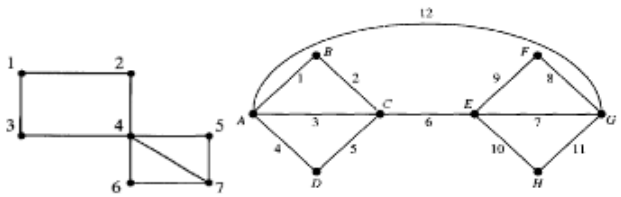
13.3 **Unit Summary:** This unit introduces key concepts in advanced graph theory, focusing on Euler and Hamiltonian graphs, graph coloring, and chromatic numbers. Students will explore Eulerian paths and circuits, learning how to identify graphs with Eulerian properties and the conditions for their existence. The unit also covers Hamiltonian paths and circuits, which are important for solving optimization problems. The unit further explores planner graphs and delves into graph coloring, where the challenge is to assign colors to graph vertices so that no two adjacent vertices

share the same color. The chromatic number, representing the minimum number of colors needed, will also be discussed.

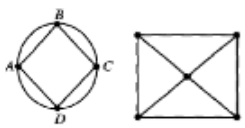
Eulerian graphs are characterized by the ability to traverse every edge exactly once. An Euler path uses every edge exactly once, but does not necessarily return to the starting point, while an Euler circuit also uses every edge once and returns to the starting vertex. Conditions for the existence of these paths and circuits are based on the degree of vertices. In contrast to Eulerian graphs, Hamiltonian graphs deal with paths and circuits that visit every vertex exactly once. A Hamiltonian path visits all vertices without repeating, and a Hamiltonian circuit also returns to the starting vertex. Understanding the distinction between these concepts is key in solving many real-world problems. In this unit, we have explored advanced concepts of graph theory, starting with Eulerian and Hamiltonian paths and circuits. These concepts are foundational in solving problems related to traversal and optimization. We then explored the concept of graph coloring, focusing on the chromatic number, which plays a significant role in resource allocation problems. Additionally, planner graphs were discussed, highlighting the importance of non-crossing edges in real-world applications.

13.4 Check Your Progress:

1. What are the necessary conditions for the existence of an Eulerian path and an Eulerian circuit?
2. How does a Hamiltonian path differ from an Eulerian path, and under what conditions can each exist?
3. What is a planar graph, and how do we determine if a graph is planar?
4. Explain the concept of graph coloring. How is the chromatic number determined for a graph?
5. What are some practical applications of Hamiltonian circuits and Eulerian paths?
6. For the following graph, state whether they have Euler circuit or path or neither?



7. For the following graph, state whether they have Hamiltonian circuit or path or neither?



Unit 14: Introduction to Trees and their applications.

14.0 Introduction and Unit Objectives: This unit provides an introduction to tree data structures, a fundamental concept in computer science. Trees are hierarchical, non-linear structures composed of nodes, with each node storing a value and potentially having multiple child nodes. These structures are highly efficient for representing hierarchical relationships, searching, and sorting operations. The unit will cover the various types of trees and their real-world applications, such as in file systems, databases, and decision-making algorithms. Additionally, we will discuss spanning trees, which are essential in graph theory and network design, allowing for efficient traversal of graph structures with minimal connections. Understanding trees is crucial for solving complex problems in computer science and optimizing various algorithms.

Unit Objectives: On Completion of this unit the learners will be able to:

1. Understand the concept of tree data structures, including their components (nodes, edges, root, leaves).
2. Explore the different types of trees such as binary trees, AVL trees, and heap trees.
3. Learn the practical applications of trees in areas like database indexing, decision trees, and file systems.
4. Understand the concept of a spanning tree and its importance in network design and graph theory.
5. Apply tree structures to solve real-world problems efficiently.

14.1 Tree and its applications:

A tree is a non-linear data structure in which items are arranged in a sequence. It is used to represent hierarchical relationship existing among several data items.

Definition: Consider a set A and suppose that T is a relation on A. Then T can be referred to as a **tree** if there exist a vertex v_0 in A with the property that there is a distinct path in T from v_0 to all vertices in A but no path from v_0 to v_0. The vertex v_0 is unique and it is often called root of the tree T and T is then referred to as rooted tree. We denote (T, v_0) to denote a rooted tree with

root v_0.

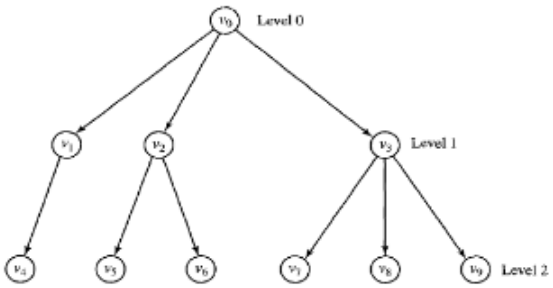
Theorem: Suppose that (T, v_0) is a tree with root v_0. Then

a. T has no cycles.

b. T has only one root i.e. v_0

c. All vertices have in-degree 1 except v_0 which has in-degree 0.

Consider the following Figure



The vertex v_0 is the root. We can see that root has level 0. T has three levels level 0 to level 2. The maximum level number is referred as **tree height**. So the height for this tree is 2. The vertices which do not have any outgoing edges are called **leaves** of the tree.

Theorem: Let (T,v_0) be a rooted tree on set A. Then

a. T is irreflexive.

b. T is asymmetric.

c. T is not transitive.

Example: Let R be defined on set A. Check if R is a tree. If R is tree find its root.

A={a,b,c,d,e}

R={(a,d),(b,c),(c,a),(d,e)}

Solution:

a. R is irreflexive because for any a in A, (a,a) does not belongs to R.

b. R is asymmetric because for any a,b in A, if (a,b) is in R then (b,a) does not belongs to R

c. Also we can see that the relation R is not Transitive.
Therefore, R is a tree.

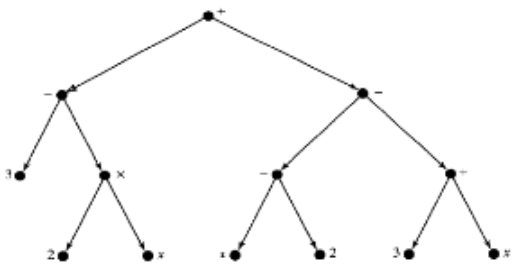
Since the element b do not appear as second element in any of the ordered pair in R so b is the root of the tree.

Labeled Tree: It is sometimes useful to label the vertices or edges of a tree to indicate that the tree is being used for a particular purpose. This is especially true for many use cases of tree in computer science. So a tree whose vertices and edges are labeled with particular name is called labeled tree.

Example: Consider the fully parenthesized algebraic expression given below and draw a labeled tree for it.

$$(3 - (2 \times x)) + ((x - 2) - (3 + x)).$$

Solution: To draw the tree we need to find the central operator. The central operator is the binary operator +. So it is the root of the tree. The tree can be drawn as show below. We omit the parenthesis while drawing the tree.



Tree Searching: There are many occasions when it is useful to consider each vertex of a tree exactly once in some specified order. As each successive vertex is encountered, we may wish to take some action. Performing appropriate task at a vertex is called **visiting** the vertex.

The process of visiting each vertex of a tree in some specified order is called searching a tree. This process is also called **walking or traversing** a tree.

There are three methods of traversing a tree-

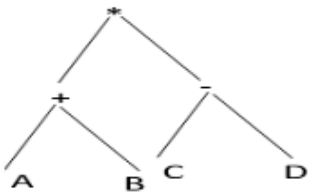
In-order Traversal: In this method, the nodes are visited recursively in the following sequence: left subtree, root node, and then the right subtree. It is particularly useful for retrieving elements of a binary search tree in ascending order.

Pre-order Traversal: This traversal involves visiting nodes in the following order: root node, left subtree, and then the right subtree. It is often used for duplicating a tree or when operations require processing the root node before its child nodes.

Post-order Traversal: In this approach, the nodes are visited in the order of left subtree, right subtree, and finally the root node. It is commonly used for deletion processes, as it ensures child nodes are handled before their parent node.

Let's take the following algebraic expression to demonstrate the tree traversal techniques:

Expression: (A + B) * (C - D). the labeled tree for the expression is as shown below



The output of in-order transversal is A + B * C-D

Preorder traversal yields * + A B – C D

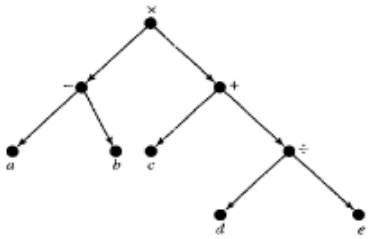
Post order traversal yields A B + C D - *.

The output of in-order, preorder and post-order traversal are known as **Infix** notation, **prefix or polish** notation and **postfix** notation respectively.

Evaluating an Expression:

To evaluate the expression in polish form, we move from right to left until we find a string of the form F_{xy} , where F is the symbol for binary operation and x and y are numbers. Evaluate ${}_xF_y$ and substitute the answer for the string F_{xy}. Continue this process until only one number remains.

Consider the following labeled tree



If we perform the pre-order traversal, then we will find × - a b + c ÷ d e. This notation is called prefix or polish form. For example in this expression, let a= 4, b=6, c=5,d=2 and e=2. Then we are to evaluate × - 6 4 + 5 ÷ 2 2. This is done in the following sequence of steps.

1. × − 6 4 + 5 ÷ 2 2.
2. × 2 + 5 ÷ 2 2 since the first string of the correct type is −6 4 and 6 − 4 = 2.
3. × 2 + 5 1 replacing ÷ 2 2 by 2 ÷ 2 or 1.
4. × 2 6 replacing + 5 1 by 5 + 1 or 6.
5. 12 replacing × 2 6 by 2 × 6.

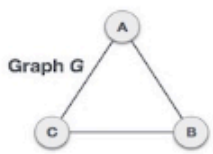
If we perform the post-order traversal, then we will find a b – c d e + ÷ ×. This notation is called post-fix notation. To evaluate the expression in postfix, we move from right to left until

we find a string of the form $_{xy}F$, where F is the symbol for binary operation and x and y are numbers. Evaluate $_xF_y$ and substitute the answer for the string $_{xy}F$. Continue this process until only one number remains. Let a=2, b=1, c=3, d=4 and e=2. To evaluate a b – c d e + ÷ × following sequence of steps.

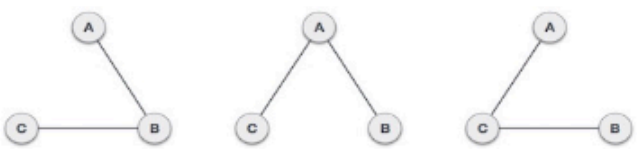
1. 2 1 – 3 4 2 ÷ + ×.	
2. 1 3 4 2 ÷ + ×	replacing 2 1 – by 2 – 1 or 1.
3. 1 3 2 + ×	replacing 4 2 ÷ by 4 ÷ 2 or 2.
4. 1 5 ×	replacing 3 2 + by 3 + 2 or 5.
5. 5	replacing 1 5 × by 1 × 5 or 5.

14.2 Spanning Tree: Suppose G is a graph. A spanning tree T, is subset of G having all vertices of G and (n-1) edges where n is the number of vertices in G which ensures that there are no cycles in T.

Consider the following graph G.



Few spanning tree of G are given blow



Real World Applications of a Spanning Tree:

a. Several pathfinding algorithms, such as Dijkstra's algorithm and A* search algorithm, internally build a spanning tree as an intermediate step.

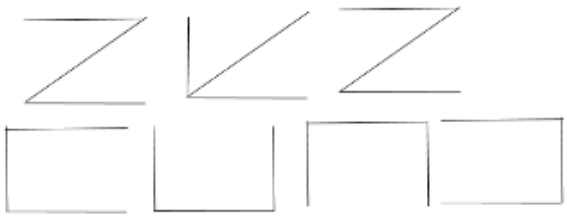
b. Building telecommunication networks.

c. Image segmentation to break an image into distinguishable components.

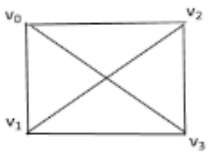
d. Computer network routing protocols.

Example: Find few spanning trees for the graph given below

Solution: Since the given graph is connected, so it has some spanning trees which are shown below.



Example: How many different spanning trees can be formed from k_4 shown in the figure



Solution: Since k_4 consist of n=4 vertices, there are $(4)^{4-2} = 16$ different ways of joining them to form a spanning tree. This means that 16 different trees can be formed.

Note: According to English mathematician Arthur Cayley, for a complete graph k_n with n vertices there are n^{n-2} different ways of joining them to form a tree.

14.3 Unit Summary: This unit provides an introduction to tree data structures, focusing on their properties, types, and applications. The unit begins with an overview of **trees**, a hierarchical structure used to organize data efficiently, and explores their foundational concepts such as nodes, edges, and the tree's acyclic nature. A tree is a non-linear data structure used extensively in various algorithms and applications due to its efficient search and traversal capabilities. The unit covers different types of trees, such as binary trees, binary search trees, AVL trees, and heap trees, each suited for specific tasks like searching, sorting, or optimizing operations. The unit explains the significance of spanning trees in algorithms like Kruskal's and Prim's, which find minimum spanning trees for network optimization.

14.4 Check Your Progress:

1. Explain the difference between a tree and a graph.
2. What is a spanning tree, and why is it important in graph theory?
3. Consider the relation R defined on the set A. In case determine if R is a tree and if it is, find its root.

1. $A = \{a, b, c, d, e\}$
   $R = \{(a, d), (b, c), (c, a), (d, e)\}$

2. $A = \{a, b, c, d, e\}$
   $R = \{(a, b), (b, e), (c, d), (d, b), (c, a)\}$

3. $A = \{a, b, c, d, e, f\}$
   $R = \{(a, b), (c, e), (f, a), (f, c), (f, d)\}$

4. $A = \{1, 2, 3, 4, 5, 6\}$
   $R = \{(2, 1), (3, 4), (5, 2), (6, 5), (6, 3)\}$

1. Construct label trees for following expressions
   1. $(7 + (6 - 2)) - (x - (y - 4))$
   2. $(x + (y - (x + y))) \times ((3 \div (2 \times 7)) \times 4)$
   3. $3 - (x + (6 \times (4 \div (2 - 3))))$
2. Show the result of performing pre-order, in-order, post-order traversal on the graph given below.

Unit15: Minimal Spanning Tree and shortest path algorithms.

15.0 Introduction and Unit Objectives: In graph theory, a Minimal Spanning Tree (MST) is a subgraph that connects all the vertices of a graph with the minimum total edge weight, ensuring no cycles exist. It plays a crucial role in network design, such as telecommunications, where the goal is to connect various nodes with minimal cost. Two popular algorithms used to find the MST of a graph are **Prim's Algorithm** and **Kruskal's Algorithm**, each using different approaches but achieving the same result. These algorithms are foundational in solving problems where efficient network connectivity is essential, such as in designing roads, pipelines, or circuit layouts.

In addition to MSTs, shortest path algorithms are another important aspect of graph theory, particularly when it comes to finding the shortest route between nodes. These algorithms, such as Dijkstra's and

Bellman-Ford, are used in various applications, including GPS systems, network routing, and logistics. Although shortest path algorithms can be used in the process of finding MSTs, their primary function is to find the minimum distance between two vertices. This unit will explore the concepts of both MSTs and shortest path algorithms, their respective algorithms, and applications.

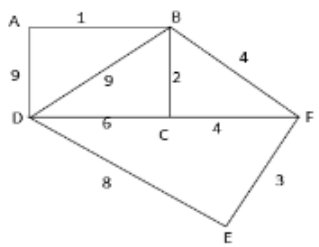
Unit Objectives: On completion of this unit, the students will be able to:

1. Understand the concept of Minimal Spanning Trees (MST) and their significance in graph theory.
2. Learn how to find the MST of a graph using **Prim's Algorithm** and **Kruskal's Algorithm**.
3. Compare and contrast Prim's and Kruskal's algorithms in terms of their approach to solving MST problems.
4. Apply MST algorithms in real-world applications like network design and optimization.
5. Gain familiarity with common shortest path algorithms, such as Dijkstra's Algorithm, and understand their applications in graph theory.

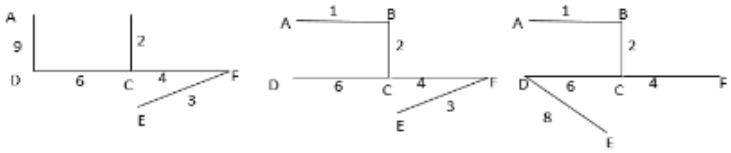
15.1 Minimal Spanning Tree(MST), Finding MST from graph: A Minimum Spanning Tree (MST) is a subset of edges in a connected, undirected graph that links all vertices without forming any cycles and with the smallest total edge weight. MSTs are particularly significant in areas like network design, where minimizing the cost of connecting all nodes while ensuring complete connectivity is essential.

The significance of an MST lies in its ability to optimize solutions in many practical problems. For instance, in telecommunications, it helps design networks that minimize the total wiring or cost of communication channels. MSTs are also fundamental in algorithms like Prim's and Kruskal's algorithms, which help find the MST in an efficient manner.

Minimal Spanning Tree(MST): A minimal spanning tree of G is a spanning tree whose weight is as small as possible among all possible spanning tree of that graph. Consider the figure given below.



Among all the spanning tree there will one or more spanning trees whose weight is smallest. Now to find that minimum spanning tree, we need to construct all the spanning trees and then determine the Minimum spanning tree.



There are several algorithms to find the minimal spanning tree of the given weighted graphs with drawing all. Those algorithms will be discussed in next section.

15.2 Shortest Path algorithms (to find MST): Two well-known MST algorithms are:

Prim's Algorithm: This algorithm starts from any node and grows the MST by adding the smallest edge connecting a node in the MST to a node outside the MST.

Kruskal's Algorithm: This algorithm selects the smallest edges available and adds them to the MST, ensuring no cycles are formed.

Let us discuss them in detail one by one.

Kruskal's Algorithms: The input to this algorithm is a weighted connected graph G. This algorithm involves following steps

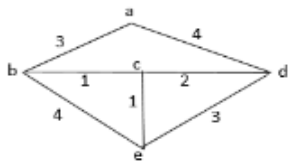
Step1: Arrange the edges of the graph with increasing order of their weights.

Step2: Choose a minimum weight edge Select an edge of minimum weight (If there are more than one such edge, choose any one of them).

Step3: Select an edge having minimum weight from the remaining edges, if it does not results a cycle with the previously selected edges in T. Then add the edge to T.

Ste4: Repeat step3 until (n-1) edges have been selected.

Example: Show step by step, how Kruskal's algorithm is used to find the Minimal Spanning Tree for the graph given below.



Solution:

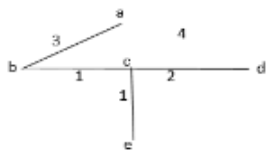
Step1: List the edges by non-decreasing order of their weights.

Edge	bc	ce	cd	ab	de	ad	be
Weight	1	1	2	3	3	4	4

Step2: The edge bc has the smallest weight, so include it in T.

Step3: An edge with next smallest weight is ce. So include it in T.

Step4: Similarly, next smallest weight including is cd, since it does nor form any cycle with the existing edges in T, include it in T. Similarly, we follow this process until T has (n-1=5-1) 4 edges are added. Then T contain all the vertices of the given graph and (n-1) edges. The result is the Minimum spanning tree as shown in figure.

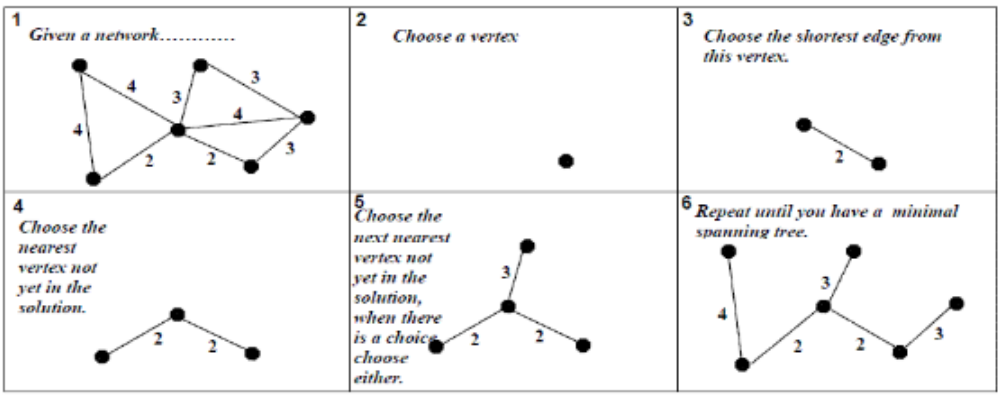


The weight of the MST is bc + cd + ab + ce= 1+2+3+1=7

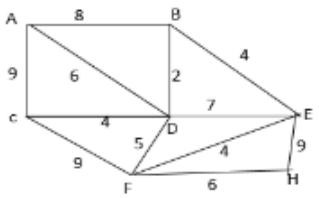
Prim's Algorithm: The input to this algorithm is a weighted connected graph G. The algorithm involves the following steps.

1. Choose a Starting Vertex: Pick any vertex from the graph to begin the MST.
2. Mark the Vertex: Mark the starting vertex as "visited."
3. Add Edges: Look at all the edges connected to the visited vertex. Add them to a list of possible edges to consider, but only include those that lead to vertices not yet visited.
4. Pick the Smallest Edge: From the list of possible edges, choose the one with the smallest weight (the smallest cost).
5. Visit the New Vertex: Mark the vertex connected by this smallest edge as "visited."
6. Repeat the Process: Keep adding the smallest edges that connect a new, unvisited vertex to the MST. Repeat steps 4 and 5 until all vertices are included in the MST.
7. Finish the MST: Once all vertices are connected, the MST is complete.

Prim's Algorithm

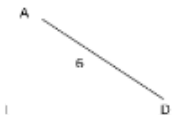


Example: Find a minimal spanning tree of the graph given below using Prim's algorithm

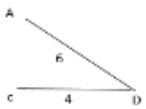


Solution:

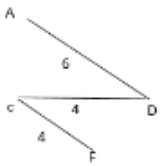
Step1: Choose a vertex A as the initial vertex. Now the edge with the smallest weight incident on A is D. So we include it in T (Spanning Tree)



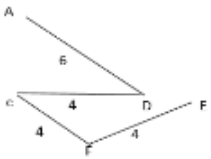
Step2: Now DB=8, DE=7,DC=4, DF=5 AB=8,AC=9. From these edges we chose DC since it is of minimum weight. Then include it in T.



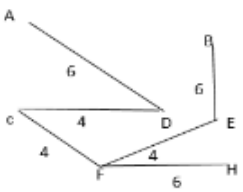
Step3: Again CF=4, DB=8, DE=7,DF=5,AB=8. So we choose CF since it is minimum. Include it in T.



Step4: Now FH=6, FE=4, DB=8, AB=8,DE=7. So we choose FE and include it in T.



Step 5: EH=9, FH=6, EB=6,BD=8,AB=8. So we choose FH or EB as they both have same weight. If we choose FH in this step then we must choose EB in the next step, so we include both in T.



Hence we have a minimum sanning tree of having (7-1) edges, which is shown in the figure above. The weight of the MST is 4+4+4+6+6+6=30.

Dijkstra's algorithm

Dijkstra's algorithm is an efficient method for determining the shortest path from a source node to all other nodes in a weighted graph. It is especially useful for single-source shortest path problems in graphs with non-negative edge weights. This algorithm is commonly used in applications such as network routing, GPS navigation, and graph analysis**.**

Example:

Consider a map of cities, where each city is a node and roads between them are edges with certain distances (weights). If you want to find the shortest path from City A to City D, Dijkstra's algorithm will calculate the minimum distance from City A to all other cities, ultimately selecting the shortest route to City D. Dijkstra's algorithm guarantees the most efficient route, such as finding the quickest road between two cities, even when many routes exist.

15.3 Unit Summary: This unit delves into two fundamental topics in graph theory: the Minimum Spanning Tree (MST) and Shortest Path Algorithms, both crucial for optimization in network design and pathfinding. The Minimum Spanning Tree (MST) is designed to connect all vertices in a graph while minimizing the total edge weight, ensuring no cycles are formed. Two prominent techniques for constructing an MST are Prim's Algorithm and Kruskal's Algorithm. Prim's Algorithm operates by initiating from a chosen vertex and iteratively incorporating the closest vertex with the minimum edge weight, maintaining a connected, cycle-free structure. In contrast, Kruskal's Algorithm follows a global approach by sorting all edges based on their weights and adding edges sequentially while avoiding cycles.

Beyond MSTs, Shortest Path Algorithms play a pivotal role in identifying the optimal path between two vertices in a graph. Algorithms such as Dijkstra's Algorithm are instrumental not only in determining the shortest paths between nodes but also in supporting MST construction by focusing on edges with minimal weights. This unit emphasizes the practical utility of these algorithms in domains

such as network optimization, traffic flow management, and cost reduction, equipping learners with the ability to apply graph-theoretic techniques to complex real-world challenges

15.4 Check Your Progress.

1.What is the primary goal of finding MST in a graph?

2.What is the difference between Kruskal's and Prim's algorithms for finding an MST?

3.In Kruskal's algorithm, how are the edges selected to form the MST?

4.What type of graph is suitable for applying Kruskal's or Prim's algorithm?

5.For the following graph, find the MST using Kruskals as well as Prim's algorithm.