ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle
and Graded as Category–I University by MHRD-UGC]

(A State University Established by the Government of Tamil Nadu)

KARAIKUDI – 630 003

Directorate of Distance Education

B.Sc. (Mathematics)

VI - Semester

113 61

DISCRETE MATHEMATICS

Authors:

N Ch S N Iyengar, Professor, Deptt of Computer Applications, Vellore Institute of Technology, Vellore

V M Chandrasekaran, Asstt Professor, Deptt of Mathematics, Vellore Institute of Technology, Vellore

K A Venkatesh, Head - Deptt of Computer Applications, Alliance Business Academy, Bangalore

P S Arunachalam, Senior Lecturer, Department of Mathematics, SRM Engineering College, Chennai

Units (1.0-1.2, 1.3-1.5, 2.0-2.3, 4.0-4.3, 5, 6, 7.0-7.2.1, 7.2.3-7.2.4, 9.0-9.2, 9.4-9.7, 10.3, 11.0-11.2, 12, 13, 14.3-14.4, 14.6-14.8)

Vikas®Publishing House, Units (1.2.1, 1.6-1.10, 2.4-2.10, 3, 4.4-4.9, 7.2.2, 7.3-7.9, 8, 9.3, 9.8-9.12, 10.0-10.2, 10.4-10.12, 11.3-11.9, 14.0-14.2, 14.5, 14.9-14.13)

VIKAS® PUBLISHING HOUSE PVT. LTD.
E-28, Sector-8, Noida - 201301 (UP)
Phone: 0120-4078900 • Fax: 0120-4078999
Regd. Office: A-27, 2nd Floor, Mohan Co-operative Industrial Estate, New Delhi 1100 44
• Website: www.vikaspublishing.com • Email: helpline@vikaspublishing.com

Work Order No.AU/DDE/DE12-27/Preparation and Printing of Course Materials/2020 Dated 12.08.2020 Copies - 500

SYLLABI-BOOK MAPPING TABLE

Discrete Mathematics

Syllabi	Mapping in Book
BLOCK I: LOGIC, TAUTOLOGY AND THEORY OF INFERENCE **UNIT - 1** Logic Introduction – Connectives – Atomic and Compound Statements – Truth Table – Problems. **UNIT - 2** Tautology – Tautological Implications and Equivalence of Formulae – Replacement Process - Law of Duality - Tautological Implications. **UNIT - 3** Normal Forms – Principal Normal Forms - Problems. **UNIT - 4** Theory of Inference - Rules of Inference - Open Statements – Problems.	**Unit 1:** Mathematical Logic **(Pages 3-19)** **Unit 2:** Tautology **(Pages 20-42)** **Unit 3:** Normal Forms **(Pages 43-50)** **Unit 4:** Inference Theory **(Pages 51-57)**
BLOCK II: QUANTIFIERS, LATTICES AND CODING THEORY **UNIT - 5** Quantifiers – Bound and Free Variables - Theory of Inference for Predicate Calculus. **UNIT - 6** Relations – Representation of a Relation – Operations on Relations – Equivalence Relation. **UNIT - 7** Lattices – Some Properties of Lattices, New Lattices – Modular and Distributive Lattices - Boolean Algebra, Boolean Polynomials. **UNIT - 8** Coding Theory – Introduction – Hamming Distance – Encoding a Message – Group Codes – Procedure for Generating Group Codes – Decoding and Error Correction.	**Unit 5:** Quantifiers **(Pages 58-71)** **Unit 6:** Relations **(Pages 72-90)** **Unit 7:** Lattices **(Pages 91-114)** **Unit 8:** Coding Theory **(Pages 115-138)**
BLOCK III: MATRIX OF A GRAPH AND CHROMATIC NUMBERS **UNIT - 9** Definition of a Graph – Finite and Infinite Graphs – Incidence, Degree, Isolated and Pendent Vertices – Isomorphism –Sub Graphs – Walks , Paths and Circuits –Connected and Disconnected Graphs. **UNIT - 10** Matrix Representation of a Graph – Incidence Matrix – Circuit Matrix - Fundamental Circuit Matrix and Rank of the Circuit Matrix – Cut Set Matrix – Adjacency Matrix. **UNIT - 11** Chromatic Number - Chromatic Partitioning – Chromatic Polynomial - Problems.	**Unit 9:** Graph Theory **(Pages 139-156)** **Unit 10:** Circuit Matrix **(Pages 157-178)** **Unit 11:** Chromatic Numbers **(Pages 179-193)**
BLOCK IV: TREES AND CUT SETS **UNIT - 12** Trees – Properties of Trees – Pendent Vertices in a Tree – Distances and Centres in a Tree – Rooted and Binary Trees.	**Unit 12:** Trees **(Pages 194-209)**

UNIT - 13

Spanning Trees – Fundamental Circuits – Finding All Spanning Trees of a Graph – Spanning Trees in a Weighted Graph.

UNIT - 14

Cut Sets – Properties of a Cut Set – All Cut Sets in a Graph – Fundamental Circuits and Cut Sets – Connectivity and Separability - Eulerian and Hamiltonian Graphs –Problems.

CONTENTS

INTRODUCTION

NOTES

Mathematics includes the study of such topics as quantity (number theory), structure (algebra), space (geometry), and change (mathematical analysis). It has no generally accepted definition. Mathematicians search for and use patterns to formulate new conjectures; they resolve the truth or falsity of such by mathematical proof. When mathematical structures are good models of real phenomena, mathematical reasoning can be used to provide insight or predictions about nature. Mathematics has no generally accepted definition. Aristotle defined mathematics as "The Science of Quantity" and this definition prevailed until the 18th century. A great many professional mathematicians take no interest in a definition of mathematics, or consider it undefinable.

Discrete mathematics is the study of mathematical structures that are fundamentally discrete rather than continuous. In contrast to real numbers that have the property of varying smoothly, the objects studied in discrete mathematics, such as integers, graphs, and statements in logic, do not vary smoothly but have distinct and separated values. Discrete mathematics, therefore, excludes topics in continuous mathematics, such as calculus or Euclidean geometry. Discrete objects can often be enumerated by integers. More formally, discrete mathematics has been characterized as the branch of mathematics dealing with countable sets, i.e., finite sets or sets with the same cardinality as the natural numbers. However, there is no exact definition of the term 'Discrete Mathematics'. Indeed, discrete mathematics is described less by what is included than by what is excluded: continuously varying quantities and related notions.

Principally, the discrete mathematics includes the fundamental concepts of sets, relations and functions, mathematical logic, group theory, counting theory, probability, mathematical induction, and recurrence relations, graph theory, trees and Boolean algebra.

This book, *Discrete Mathematics*, is divided into four blocks, which are further subdivided into fourteen units. The topics discussed include logic, tautology and theory of inference, normal forms – principal normal forms, quantifiers, relations, equivalence relation, lattices, new lattices, modular and distributive lattices, Boolean algebra, Boolean polynomials, coding theory, graph, finite and infinite graphs, incidence, degree, isolated and pendent vertices, sub graphs, walks, paths and circuits, connected and disconnected graphs, matrix representation of a graph, incidence matrix, circuit matrix, fundamental circuit matrix, cut set matrix, adjacency matrix, chromatic number, chromatic partitioning, trees, properties of trees, distances and centres in a tree, rooted and binary trees, spanning trees, fundamental circuits, cut sets, cut sets in a graph, Eulerian and Hamiltonian graphs.

The book follows the self-instructional mode wherein each unit begins with an 'Introduction' to the topic. The 'Objectives' are then outlined before going on to the presentation of the detailed content in a simple and structured format. 'Check Your Progress' questions are provided at regular intervals to test the student's understanding of the subject. 'Answers to Check Your Progress Questions', a 'Summary', a list of 'Key Words', and a set of 'Self-Assessment Questions and Exercises' are provided at the end of each unit for effective recapitulation.

BLOCK - I
LOGIC, TAUTOLOGY AND THEORY OF INFERENCE

UNIT 1 MATHEMATICAL LOGIC

Structure

1.0 INTRODUCTION

Every mathematical statement must be precise. Therefore, there has to be proper reasoning in every mathematical proof. Proper reasoning involves logic. Definitions of logic is that, it is the analysis of methods of reasoning. The study of logic helps in increasing one's ability of systematic and logical reasoning. It also helps to develop the skills of understanding various statements and their validity. Logic has a wide scale application in circuit designing, computer programming etc. Hence, the study of logic becomes essential. Mathematical logic is a sub-field of mathematics which deals with the applications of formal logic to mathematics. Mathematical logic has several meanings in common usage. It originally referred to symbolic or formal logic, and then came to be associated with the study of the logical (and even philosophical) foundations of mathematics. In contemporary use by mathematical logicians, the term refers to several branches of pure mathematics whose study involves careful attention to formal axiom systems and formal definability. Its principal aim is to be a precise and adequate understanding of the notion of mathematical proof. Basic Mathematical logics are a negation, conjunction, and disjunction. The symbolic form of mathematical logic is, '$\sim$' for negation '$\wedge$' for conjunction and '$\vee$' for disjunction.

NOTES

In logic, a logical connective (also called a logical operator, sentential connective, or sentential operator) is a symbol or word used to connect two or more sentences (of either a formal or a natural language) in a grammatically valid way, such that the value of the compound sentence produced depends only on that of the original sentences and on the meaning of the connective.

The most common logical connectives are binary connectives (also called dyadic connectives), which join two sentences and which can be thought of as the function's operands. Another common logical connective, negation, is considered to be a unary connective.

A truth table is a mathematical table used in logic-specifically in connection with Boolean algebra, Boolean functions, and propositional calculus-which sets out the functional values of logical expressions on each of their functional arguments, that is, for each combination of values taken by their logical variables. In particular, truth tables can be used to show whether a propositional expression is true for all legitimate input values, i.e., logically valid.

In this unit, you will study about the logic introduction, connectives, atomic statements, compound statement, truth table, and related problems.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the mathematical logic
- Understand the connectives
- Analyse the atomic and compound statements
- Interpret the truth table

1.2 MATHEMATICAL LOGIC: AN INTRODUCTION

One of the main aims of mathematical logic is to provide rules. The rules of logic give precise meaning to mathematical statements and distinguish between valid and invalid mathematical arguments. In addition, logic has numerous applications in computer science. These rules are used in the design of computer circuits, construction of computer programs, verification of the correctness of programs, and in many other ways.

Propositions: A proposition is a statement to which only one of the terms, true or false, can be meaningfully applied.

The value of a proposition if true is denoted by 1and false if denoted by 0. Occasionally they are also denoted by the symbols *T* and *F*.

The following are propositions:

(*i*) 4 + 2 = 6

(*ii*) 4 is an even integer and 5 is not.

(*iii*) 5 is a prime number.

(*iv*) New Delhi is the capital of India.

(*v*) $2 \in \{1, 3, 5, 7\}$

(*vi*) $42 \geq 51$

(*vii*) Paris is in England.

Of the above propositions, (*i*)–(*iv*) are true whereas (*v*)–(*vii*) are false.

The following are *not* propositions:

(*i*) Where are you going?

(*ii*) $x + 2 = 5$

(*iii*) $x + y < z$

(*iv*) Beware of dogs.

The expressions (*i*) and (*iv*) are not propositions since neither this True nor False. The expressions (*ii*) and (*iii*) are not propositions, since the variables in these expressions have not been assigned values and hence they are neither true or false.

Note: Letters are used to denote propositions just as letters are used to denote variables. The conventional letters used for this purpose are $p, q, r, s, \ldots$

1.2.1 Atomic and Compound Statements

In logic and analytic mathematics, an atomic sentence is a type of declarative sentence which is either 'True' or 'False' (may also be referred to as a proposition, statement or truth bearer) and which cannot be broken down into other simpler sentences. For example, the sentence 'The dog ran' is an atomic sentence in natural language, whereas the sentence 'The dog ran and the cat hid' is a molecular sentence in natural language.

For a logical analysis, the truth or falsity of sentences in general is determined by only two things: the logical form of the sentence and the truth or falsity of its simple sentences.

For example, the truth of the sentence 'John is Greek and John is happy' is a function of the meaning of 'and', and the truth values of the atomic sentences 'John is Greek' and 'John is happy'.

In a formal language, a Well-Formed Formula (or WFF) is a string of symbols constituted in accordance with the rules of syntax of the language. A term is a variable, an individual constant or an *n*-place function letter followed by *n* terms. An atomic formula is a WFF consisting of either a sentential letter or an *n*-place predicate letter followed by *n* terms. A sentence is a WFF in which any variables are bound. An atomic sentence is an atomic formula containing no variables.

NOTES

Principally, an atomic sentence contains no logical connectives, variables or quantifiers. A sentence consisting of one or more sentences and a logical connective is a compound (or molecular) sentence.

Assumptions

Consider the following examples:

- Let F, G, H be predicate letters;
- Let a, b, c be individual constants;
- Let x, y, z be variables.

Atomic Sentences: The above mentioned WFFs are atomic sentences because they contain no variables or conjunctions:

Atomic Formulae: The below mentioned WFFs are atomic formulae, but are not sentences (atomic or otherwise) because they include free variables:

- $F(x)$
- $G(a, z)$
- $H(x, y, z)$

Compound Sentences: The below mentioned WFFs are compound sentences. They are sentences, but are not atomic sentences because they are not atomic formulae:

- $\forall x\ (F(x))$
- $\exists z\ (G(a, z))$
- $\exists x\ \forall y\ \exists z\ (H(x, y, x))$
- $\forall x\ \exists z\ (F(x) \wedge G(a, z))$
- $\exists x\ \forall y\ \exists z\ (G(a, z) \vee H(x, y, z))$

Compound Formulae: The below mentioned WFFs are compound formulae. They are not atomic formulae but are built up from atomic formulae using logical connectives. They are also not sentences because they contain free variables:

- $F(x) \wedge G(a, z)$
- $G(a, z) \vee H(x, y, z)$

Therefore, an **atomic statement** is a **declarative statement without logical connectives** that has a truth value. A logical connective is a word or symbol that joins two atomic statements to form a larger logical statement. Here are two declarative statements that are atomic statements:

- P = It is snowing.
- Q = I am cold.

The truth value of a statement is whether the statement is 'True' or 'False'. It can be determined with the help of the truth table that when a logical statement is true and when it is false. The truth table organizes all the possible cases.

Consider the atomic statement P joined with the atomic statement Q. The following sentence can be written using the symbol '$\vee$' for the logical connective 'OR'.

It is snowing or I am cold.

$$P \vee Q$$

This statement implies that one or both of the atomic statements is happening. You have to check the validity of the statement, because it is important to remember that not all statements are always 'True'. To determine that the above statement is true, construct a truth table. A truth table considers all possible combinations of the original atomic statements being 'True' or 'False', and then uses logic to deduce the truth value of the compound statement in each case. Following is the truth table for 'Or ($\vee$)'.

Truth Table for OR

P	Q	P $\vee$ Q
T	T	T
T	F	T
F	T	T
F	F	F

There are four possible truth combinations of P and Q - Both True, First True/Second False, First False/Second True, Both False. Only one of these combinations yields a false statement for P $\vee$ Q.

This means that the statement 'It is snowing or I am cold' is only False if 'It is snowing' is False and 'I am cold' is False. If 'It is snowing' is considered 'True' and 'I am cold' is also considered 'True', then the statement 'It is snowing or I am cold' is also True. In mathematical logic, the word 'OR' does not mean exactly one or the other rather it means 'one or the other or both'.

1.3 CONNECTIVES

There are several ways in which we commonly combine simple statements into compound ones. The words OR, AND, NOT, if… then and if and only if, can be added to one or more propositions to create a new proposition. New propositions are called compound propositions. Logical operators are used to form new propositions or compound propositions. These logical operators are also called connectives.

1.3.1 Conjunction

Conjunction (AND): If p and q are propositions, then the propositions 'p and q', denoted by $p \wedge q$, is true when both p and q are true and is false otherwise.

NOTES

The proposition $p \wedge q$ is called the conjunction of p and q.

The truth table for $p \wedge q$ is shown in Table 1.1. Note that there are four rows in this truth table, one row for each possible combination of truth values of the propositions p and q.

Table 1.1 *Truth Table for Conjunction*

p	q	$p \wedge q$
0	0	0
0	1	0
1	0	0
1	1	1

Example 1.1: Find the conjunction of the propositions p and q where p is the proposition 'Today is Sunday' and q is the proposition 'It is raining today'.

Solution: The conjunction of these two propositions is $p \wedge q$ the proposition, 'Today is Sunday and it is raining today'.

Example 1.2: Let p be 'Ravi is rich' and let q be 'Ravi is happy'. Write each of the following in symbolic form:

(i) Ravi is poor but happy.

(ii) Ravi is neither rich nor happy.

(iii) Ravi is rich and unhappy.

Solution: The symbolic form of the above statements are as follows:

(i) $\sim p \wedge q$

(ii) $\sim p \wedge \sim q$

(iii) $p \wedge \sim q$

1.3.2 Disjunction

Disjunction (OR): If p and q are propositions, then disjunction p or q, denoted as $p \vee q$, is false when p and q are both false and true otherwise. The proposition $p \vee q$ is called the disjunction of p and q.

Note that connectives $\sim$ and $\wedge$ defined earlier have the same meaning as the words 'NOT' and 'AND' in general. However, the connective $\vee$ is not always the same as the word 'OR' because of the fact that the word 'OR' in English is commonly used both as an 'exclusive Or' and as an 'inclusive Or'. For example, consider the following statements:

(i) I shall watch the movie on TV or go to cinema.

(ii) There is something wrong with the fan or with the switch.

(iii) Ten or twenty people were killed in the fire today.

In statement (i), the connective 'OR' is used in the exclusive sense; that is to say, one or the other possibility exists but not both. In statement (ii) the intended

meaning is clearly one or the other or both. The connective 'OR' used in statement (*ii*) is the 'inclusive Or'. In statement (*iii*) the 'OR' is used for indicating an approximate numbr of peoples, and it is not used as a connective. From the definition of disjunction it is clear that $\vee$ is 'inclusive Or'.

NOTES

Table 1.2 *Truth Table for Disjunction*

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Example 1.3: Find the disjunction of the propositions p and q where p is the proposition 'Today is Sunday' and q is the proposition 'It is raining today'.

Solution: The disjunction of p or q, $p \vee q$, is the proposition 'Today is Sunday or it is raining today'.

Example 1.4: Let p be 'Ravi is tall' and let q be 'Ravi is handsome'. Write each of the following statements in symbolic form:

(*i*) Ravi is short or not handsome.
(*ii*) Ravi is tall or handsome.
(*iii*) It is not true that Ravi is short or not handsome.

Solution: The symbolic form of the above statements are as follows:

(*i*) $\sim p \vee \sim q$
(*ii*) $p \vee q$
(*iii*) $\sim (\sim p \vee \sim q)$

Example 1.5: Let p be 'Ravi speaks Tamil' and let q be 'Ravi speaks Hindi'. Give a simple verbal sentence which describes each of the following.

(*i*) $p \vee q$ (*ii*) $p \wedge q$ (*iii*) $p \wedge \sim q$ (*iv*) $\sim p \vee \sim q$ (*v*) $\sim (\sim p)$.

Solution: The verbal statement of the above propositions are as follows:

(*i*) Ravi speaks Tamil or Hindi.
(*ii*) Ravi speaks Tamil and Hindi.
(*iii*) Ravi speaks Tamil but not Hindi.
(*iv*) Ravi does not speak Tamil or he does not speak Hindi.
(*v*) It is not true that Ravi does not speak Tamil.

1.3.3 Negation

***Negation* (*NOT*):** If p is a proposition, its negation not p is another proposition called the negation p. The negation of p is denoted by $\sim p$. The proposition $\sim p$ is read 'not p'.

Alternate symbols used in the literature are $\neg p$, $\overline{p}$ and 'not p'.

Note that a negation is called a connective although it only modifies a statement. In this sense, negation is the only operator that acts on a single proposition.

NOTES

Table 1.3 *Truth Table for Negation*

p	$\sim p$
0	1
1	0

Example 1.6: Find the negation of the propositions:

(*i*) It is cold.

(*ii*) Today is Sunday.

(*iii*) Ravi is poor.

Solution: The negation of the propositions are:

(*i*) It is not cold.

(*ii*) Today is not Sunday.

(*iii*) Ravi is not poor.

1.4 LOGICAL OPERATORS

Conditional Operator (If … Then): Let p and q be propositions. The implication $p \rightarrow q$ is false when p is true and q is false and true otherwise. In the implication, p is called the *premise* or hypothesis and q is called the *consequence* or conclusion.

Table 1.4 *Truth Table for If.... Then*

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

Because implications arise in many places in mathematical argument, a wide variety of terminology is used to express $p \rightarrow q$. Some of the more common ways of expressing this implication are:

(*i*) p implies q.

(*ii*) if p, then q.

(*iii*) q if p.

(*iv*) p only if q.

(*v*) p is sufficient for q.

(*vi*) q whenever p.

(*vii*) q is necessary for p.

We shall avoid the word 'implies' since it might be used in different contexts.

NOTES

Example 1.7: Let p denote 'It is below freezing' and let q denote 'It is snowing'. Write the following statements in a symbolic form:

(*i*) If it is below freezing, it is also snowing.
(*ii*) It is not snowing if it is below freezing.
(*iii*) It is below freezing is a necessary condition for it to be snowing.

Solution: Recall that $p \rightarrow q$ can be read 'if p, then q' or 'p only if q' or 'q is necessary for p'. Then (*i*) $p \rightarrow q$ (*ii*) $p \rightarrow \sim q$ (*iii*) $q \rightarrow p$

Example 1.8: Let p and q be the propositions, where:

p : You drive over 80 kms per hour.
q : You get a speeding ticket.

Write the following propositions in symbolic form:

(*i*) You will get a speeding ticket if you drive over 80 kms per hour.
(*ii*) If you do not drive over 80 kms per hour, then you will not get a speeding ticket.
(*iii*) Driving over 80 kms per hour is sufficient for getting a speeding ticket.
(*iv*) Whenever you get a speeding ticket, you are driving over 80 kms per hour.

Solution: (*i*) $p \rightarrow q$ (*ii*) $\sim p \rightarrow \sim q$ (*iii*) $p \rightarrow q$ (*iv*) $q \rightarrow p$

Example 1.9: Write each of the following statements in the form 'if p then q':

(*i*) To get tenure as a professor, it is sufficient to be world-famous.
(*ii*) It shows whenever the wind blows from the north-east.
(*iii*) The apple trees will bloom if it stays warm for a week.

Solution: The p and q form of the above statements are as follows:

(*i*) If you are world-famous, then you will get tenure as a professor.
(*ii*) If the wind blows from the north-east, then it snows.
(*iii*) If it stays warm for a week, then the apple trees will bloom.

Example 1.10: Determine the truth value of each of the following statements:

(*i*) If Calcutta is in India, then 1 + 1 = 2.
(*ii*) If Calcutta is in Sri Lanka, then 1 + 1 = 2.
(*iii*) If Calcutta is in India, then 1 + 1 = 3.
(*iv*) If Calcutta is in Sri Lanka, then 1 + 1 = 3.

Solution: Since the statement 'if p then q' is false only when p is true and q is false. only statement (*iii*) is false.

Biconditional Operator (If and Only If): Let p and q be propositions. The biconditional $p \leftrightarrow q$ is true when p and q have the same truth values and is false otherwise.

Note that the biconditional $p \leftrightarrow q$ is true when both the implications $p \rightarrow q$ and $q \rightarrow p$ are true. So 'p if and only if q' is used for biconditional. Other common ways of expressing the proposition $p \leftrightarrow q$ or $p = q$ are 'p is necessary and sufficient for q' and 'if p then q, and conversely'.

NOTES

Table 1.5 *Truth Table for If and Only If*

p	q	$p \leftrightarrow q$
0	0	1
0	1	0
1	0	0
1	1	1

Each of the following theorems is well known, and each can be symbolized in the form $p \leftrightarrow q$:

(*i*) Two lines are parallel if and only if they have the same slope.

(*ii*) Two triangles are congruent if and only if all three sets of corresponding sides are congruent.

Example 1.11: Determine the truth value of each of the following statements:

(*i*) Calcutta is in India if and only if $1 + 1 = 2$.

(*ii*) Calcutta is in Sri Lanka if and only if $1 + 1 = 2$.

(*iii*) Calcutta is in India if and only if $1 + 1 = 3$.

(*iv*) Calcutta is in Sri Lanka if and only if $1 + 1 = 3$.

Solution: Statements (*i*) and (*iv*) are true since the substatements are both true in statement (*i*) and both false in statement (*iv*). On the other hand, statements (*ii*) and (*iii*) are false since the substatements have different truth values.

Example 1.12: Let p denote 'He is poor' and let q denote 'He is happy'. Write each of the following statement in symbolic form using p and q:

(*i*) To be poor is to be unhappy.

(*ii*) He is rich if and only if he is unhappy.

(*iii*) Being rich is a necessary and sufficient condition to being happy.

Solution: The symbolic form using p and q of the above statements are as follows:

(*i*) $p \leftrightarrow q$

(*ii*) $\sim p \leftrightarrow q$

(*iii*) $\sim p \leftrightarrow q$

1.5 TRUTH TABLE

The truth table of a logical operator specifies how the truth value of a proposition using that operator is determined by the truth values of the propositions. A truth table lists all possible combination of truth values of the propositions in the left most columns and the truth value of the resulting propositions in the right most column.

Our basic concern is to determine the truth table of a proposition for each possible combination of the truth values of the compound propositions. A table showing all such truth values is called truth table of the formula. In general, if there

are n distinct components in a proposition or formula, we need to consider 2^n possible combinations of truth values in order to obtain the truth table.

Two methods of constructing truth table are shown in the following examples.

NOTES

Example 1.13: Construct the truth table for the statement formula $\sim p \wedge q$.

Solution: It is necessary to consider all possible values of p and q (for the variables, as here, four rows are necessary). These values are entered in two first two columns of table for both methods.

Method 1: In this method, the truth values of $\sim p$ are entered in the third column, and the truth values of $\sim p \wedge q$ are entered in the fourth column.

Truth Table (Method 1)

p	q	$\sim p$	$\sim p \wedge q$
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

Truth Table (Method 2)

p	q	p	$\sim$	$\wedge$	q
0	0	0	1	0	0
0	1	0	1	1	1
1	0	1	0	0	0
1	1	1	0	0	1
Step number		1	2	3	1

Method 2: In this method, a column is drawn for each statement as well as for the connectives that appear. The truth values are entered step by step. The step numbers at the bottom of the table shows the sequence followed in arriving at the final step.

Example 1.14: Construct the truth table for $p \wedge \sim p$.

Solution: The truth table can be constructed as follows:

Truth Table (Method 1)

p	$\sim p$	$p \wedge \sim p$
0	1	0
1	0	0

Truth Table (Method 2)

p	p	$\wedge$	$\sim$	p
0	0	0	1	0
1	1	0	0	1
Step number	1	3	2	1

NOTES

Example 1.15: Construct the truth table for $(p \vee q) \vee \sim p$.

Solution: The following is the truth table:

Truth Table (Method 1)

p	q	$p \vee q$	$\sim p$	$(p \vee q) \vee \sim p$
0	0	0	1	1
0	1	1	1	1
1	0	1	0	1
1	1	1	0	1

Truth Table (Method 2)

p	q	p	$\vee$	q	$\vee$	$\sim$	p
0	0	0	0	0	1	1	0
0	1	0	1	1	1	1	0
1	0	1	1	0	1	0	1
1	1	1	1	1	1	0	1
Step number		1	2	1	3	2	1

Example 1.16: Construct the truth tables for:

(*i*) $\sim (p \wedge q)$ (*ii*) $p \wedge (\sim q)$ (*iii*) $(p \wedge q) \wedge r$

(*iv*) $(p \wedge q) \vee (q \wedge r) \vee (r \wedge p)$ (*v*) $(\sim p) \vee (\sim q)$ (*vi*) $(p \vee q) \vee r \vee s$

Solution: The truth table for the above propositions are constructed below:

(*i*) $\sim (p \wedge q)$ (*ii*) $p \wedge (\sim q)$

Truth Table

p	q	$p \wedge q$	$\sim (p \wedge q)$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table

p	q	$\sim q$	$p \wedge (\sim q)$
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

(*iii*) $(p \wedge q) \wedge r$

Truth Table

p	q	r	$p \wedge q$	$(p \wedge q) \wedge r$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

(*iv*) $(p \wedge q) \vee (q \wedge r) \vee (r \wedge p)$

Truth Table

p	q	r	$p \wedge q$	$q \wedge r$	$r \wedge p$	$(p \wedge q) \vee (q \wedge r)$	$(p \wedge q) \vee (q \wedge r) \vee (r \wedge p)$
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	1	0	0	0	0	0	0
0	1	1	0	1	0	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	1	0	1
1	1	0	1	0	0	1	1
1	1	1	1	1	1	1	1

(*v*) $(\sim p) \vee (\sim q)$

Truth Table

p	q	$\sim p$	$\sim q$	$(\sim p) \vee (\sim q)$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

(*vi*) $(p \vee q) \vee r \vee s$

Truth Table

p	q	r	s	$p \vee q$	$p \vee q \vee r$	$p \vee q \vee r \vee s$
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	0	1	1	0	1	1
0	1	0	0	1	1	1
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

Example 1.17: Construct the truth tables for:

(*i*) $\sim (\sim p \vee \sim q)$ (*ii*) $\sim (\sim p \wedge \sim q)$

(*iii*) $p \wedge (p \vee q)$ (*iv*) $p \wedge (q \wedge p)$

NOTES

NOTES

Solution: The truth tables are constructed as follows:

(*i*) $\sim(\sim p \vee \sim q)$

Truth Table

p	q	$\sim p$	$\sim q$	$(\sim p \vee \sim q)$	$\sim(\sim p \vee \sim q)$
0	0	1	1	1	0
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	0	1

(*ii*) $\sim(\sim p \wedge \sim q)$

Truth Table

p	q	$\sim p$	$\sim q$	$(\sim p \wedge \sim q)$	$\sim(\sim p \wedge \sim q)$
0	0	1	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	0	1

(*iii*) $p \wedge (p \vee q)$

Truth Table

p	q	$p \vee q$	$p \wedge (p \vee q)$
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

(*iv*) $p \wedge (q \wedge p)$

Truth Table

p	q	$q \wedge p$	$p \wedge (q \wedge p)$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Example 1.18: Construct the truth tables for:

(*i*) $(p \to q) \to (p \wedge q)$

(*ii*) $\sim p \to (q \to p)$

(*iii*) $(p \wedge q) \to (p \vee q)$

(*iv*) $p \to (q \wedge r)$

Solution: The truth tables are constructed as follows:

(*i*) $(p \to q) \to (p \wedge q)$

Truth Table

p	q	$p \to q$	$p \wedge q$	$(p \to q) \to (p \wedge q)$
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	1	1	1	1

(*ii*) $\sim p \to (q \to p)$

Truth Table

p	q	$\sim p$	$q \to p$	$\sim p \to (q \to p)$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	1
1	1	0	1	1

(*iii*) $(p \wedge q) \to (p \vee q)$

Truth Table

p	q	$p \wedge q$	$p \wedge q$	$(p \wedge q) \to (p \vee q)$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	1
1	1	1	1	1

(*iv*) $p \to (q \wedge r)$

Truth Table

p	q	r	$q \wedge r$	$p \to (q \wedge r)$
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

NOTES

NOTES

Check Your Progress

1. Why mathematical logic is used?
2. What is a proposition?
3. Explain the atomic and compound statements.
4. Why logical operators are used?
5. Define conjunction.
6. What is disjunction?
7. Give the definition of negation.
8. What is truth table of a logical operator?

1.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. One of the main aims of mathematical logic is to provide rules. The rules of logic give precise meaning to mathematical statements and distinguish between valid and invalid mathematical arguments. In addition, logic has numerous applications in computer science. These rules are used in the design of computer circuits, construction of computer programs, verification of the correctness of programs, and in many other ways.
2. Propositions: A proposition is a statement to which only one of the terms, true or false, can be meaningfully applied.

   The value of a proposition if true is denoted by 1and false if denoted by 0. Occasionally they are also denoted by the symbols *T* and *F*.
3. Principally, an atomic sentence contains no logical connectives, variables or quantifiers. A sentence consisting of one or more sentences and a logical connective is a compound (or molecular) sentence.
4. Logical operators are used to form new propositions or compound propositions. These logical operators are also called connectives.
5. Conjunction (AND): If p and q are propositions, then the propositions 'p and q', denoted by $p \wedge q$, is true when both p and q are true and is false otherwise. The proposition $p \wedge q$ is called the conjunction of p and q.
6. Disjunction *(OR)*: If *p* and *q* are propositions, then disjunction *p* or *q*, denoted as $p \vee q$, is false when *p* and *q* are both false and true otherwise. The proposition $p \vee q$ is called the disjunction of *p* and *q*.
7. Negation (*NOT*): If *p* is a proposition, its negation not *p* is another proposition called the negation *p*. The negation of *p* is denoted by $\sim p$. The proposition $\sim p$ is read 'not *p*'.

NOTES

8. The truth table of a logical operator specifies how the truth value of a proposition using that operator is determined by the truth values of the propositions. A truth table lists all possible combination of truth values of the propositions in the left most columns and the truth value of the resulting propositions in the right most column.

1.7 SUMMARY

- One of the main aims of mathematical logic is to provide rules. The rules of logic give precise meaning to mathematical statements and distinguish between valid and invalid mathematical arguments. In addition, logic has numerous applications in computer science. These rules are used in the design of computer circuits, construction of computer programs, verification of the correctness of programs, and in many other ways.
- Propositions: A proposition is a statement to which only one of the terms, true or false, can be meaningfully applied.

  The value of a proposition if true is denoted by 1and false if denoted by 0. Occasionally they are also denoted by the symbols *T* and *F*.
- Principally, an atomic sentence contains no logical connectives, variables or quantifiers. A sentence consisting of one or more sentences and a logical connective is a compound (or molecular) sentence.
- Logical operators are used to form new propositions or compound propositions. These logical operators are also called connectives.
- Conjunction (AND): If p and q are propositions, then the propositions 'p and q', denoted by $p \wedge q$, is true when both p and q are true and is false otherwise. The proposition $p \wedge q$ is called the conjunction of p and q.
- Disjunction *(OR)*: If *p* and *q* are propositions, then disjunction *p* or *q*, denoted as $p \vee q$, is false when *p* and *q* are both false and true otherwise. The proposition $p \vee q$ is called the disjunction of *p* and *q*.
- Negation (*NOT*): If *p* is a proposition, its negation not *p* is another proposition called the negation *p*. The negation of *p* is denoted by $\sim p$. The proposition $\sim p$ is read 'not *p*'.
- The truth table of a logical operator specifies how the truth value of a proposition using that operator is determined by the truth values of the propositions. A truth table lists all possible combination of truth values of the propositions in the left most columns and the truth value of the resulting propositions in the right most column.

NOTES

1.8 KEY WORDS

- **Proposition:** A proposition is a statement to which only one of the terms, true or false, can be meaningfully applied.
- **Atomic and compound statement:** Principally, an atomic sentence contains no logical connectives, variables or quantifiers. A sentence consisting of one or more sentences and a logical connectives is a compound (or molecular) sentence.
- **Connectives:** The words which combine simple statements to form a compound statement are called connectives.
- **Conjunction (AND):** If p and q are propositions, then the proposition 'p and q', denoted by $p \wedge q$, is true when both p and q are true and false otherwise. The proposition $p \wedge q$ is called the conjunction of p and q.
- **Disjunction (OR):** If p and q are propositions, then disjunction p or q, denoted as $p \vee q$, is false when p and q are both false and true otherwise. The proposition $p \ q$ is called the disjunction of p and q.
- **Negation (NOT):** if p is a proposition, its negation, not p is another proposition called the negation p. The negation of p is denoted by $\sim p$ and read as 'not p'.
- **Truth table:** The truth table of a logical operator specifies how the truth value of a proposition using that operator is determined by the truth values of the proposition.

1.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the significances of mathematical logic.
2. Explain the term propositions.
3. Define the conjunction.
4. Elaborate on the disjunction.
5. What do you understand by the negation?
6. Interpret the atomic and compound statements.
7. Analyse the truth table for a logical operator.

Long-Answer Questions

1. Discuss briefly the significances and applications of mathematical logic giving appropriate examples.

NOTES

2. Explain about the propositions and logical operators with the help of suitable examples.
3. Briefly discuss about the logical connectives giving example of each type.
4. Elaborate on the significance of statements and notations in mathematical logic giving appropriate examples.
5. Explain the significance of truth tables in mathematical logic.
6. Describe briefly the atomic and compound statements.
7. Construct the truth table for the statement formula $\sim p \wedge q$.
8. Construct the truth table for $p \wedge \sim p$.
9. Construct the truth table for $(p \vee q) \vee \sim p$.

1.10 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 2 TAUTOLOGY

NOTES

Structure

2.0 INTRODUCTION

In Mathematical logic, a tautology is a formula or assertion that is true in every possible interpretation. An example is "$x = y$ or $x \neq y$". A less abstract example is "either the ball is green, or the ball is not green". This would be true regardless of the colour of the ball.

Tautologies are a key concept in propositional logic, where a tautology is defined as a propositional formula that is true under any possible Boolean valuation of its propositional variables. A key property of tautologies in propositional logic is that an effective method exists for testing whether a given formula is always satisfied (equiv., whether its negation is unsatisfiable).

The definition of tautology can be extended to sentences in predicate logic, which may contain quantifiers—a feature absent from sentences of propositional logic. Indeed, in propositional logic, there is no distinction between a tautology and a logically valid formula. In the context of predicate logic, many authors define a tautology to be a sentence that can be obtained by taking a tautology of propositional logic, and uniformly replacing each propositional variable by a first-order formula (one formula per propositional variable). The set of such formulas is a proper subset of the set of logically valid sentences of predicate logic (i.e., sentences that are true in every model).

The philosopher Ludwig Wittgenstein first applied the term to redundancies of propositional logic in 1921, borrowing from rhetoric, where a tautology is a repetitive statement. In logic, a formula is satisfiable if it is true under at least one interpretation, and thus a tautology is a formula whose negation is unsatisfiable. Unsatisfiable statements, both through negation and affirmation, are known formally as contradictions.

In logic, a rule of replacement is a transformation rule that may be applied to only a particular segment of an expression. A logical system may be constructed so that it uses either axioms, rules of inference, or both as transformation rules for logical expressions in the system.

In this unit, you will study about the tautology, tautological implications and equivalence of formulae, replacement process, and law of duality.

NOTES

2.1 OBJECTIVES

After going through this unit, you will be able to:

- Describe the concept of tautology
- Explain the tautological implications and equivalence of formulae
- Elaborate on the replacement process
- Analyse the law of duality

2.2 TAUTOLOGY

The final column of a truth table of a given formula contains both 1 and 0. There are some formulae whose truth values are always 1 or always 0 regardless of the truth value assignments to the variables. Consider for example, the statement formula $p \vee \sim p$ and $p \wedge \sim p$ in Truth Table 2.1.

Truth Table 2.1 *Tautology and Contradiction*

p	$\sim p$	$p \vee \sim p$	$p \wedge \sim p$
0	1	1	0
1	0	1	0

The truth values of $p \vee \sim p$ and $p \wedge \sim p$, which are 1 and 0, respectively, are independent of the statement by which the variable p may be replaced.

Tautology: A statement formula which is true regardless of the truth values of the statements which replace the variables in it is called a tautology or a logical truth or a universally valid formula.

Contradiction: A statement formula which is false regardless of the truth values of the statements which replace the variables in it is called a contradiction.

Contingency: A statement formula that is neither a tautology nor a contradiction is called a *contingency*.

Note: A straight forward method to determine whether a given formula is a tautology is to construct its truth table . We may say that a statement formula which is a tautology is identically true (their truth tables consist of a column of ones) and a formula which is a contradiction is identically false (their truth tables consist of a column of zeros). Obviously, the negation of a contradiction is a tautology.

NOTES

Example 2.1: Verify if the following propositions are tautologies:

(*i*) $(p \wedge q) \rightarrow p$

(*ii*) $q \rightarrow (p \vee q)$

(*iii*) $(p \vee q) \leftrightarrow (q \vee p)$

(*iv*) $p \vee \sim (p \wedge q)$

(*v*) $\sim (p \wedge q) \leftrightarrow (\sim p) \vee (\sim q)$

Construct the truth table of the above given propositions.

Solution: The truth table is constructed as follows:

Truth Table

p	q	$\sim p$	$\sim q$	$p \vee q$	$p \wedge q$	$q \vee p$	$\sim (p \wedge q)$	(a)	(b)	(c)	(d)	(e)
0	0	1	1	0	0	0	1	1	1	1	1	1
0	1	1	0	1	0	1	1	1	1	1	1	1
1	0	0	1	1	0	1	1	1	1	1	1	1
1	1	0	0	1	1	1	0	1	1	1	1	1

Since the truth value of all propositions is 1, for all values of p and q, the given propositions are tautologies.

Example 2.2: Verify if the proposition $(p \wedge q) \wedge \sim (p \vee q)$ is a contradiction.

Solution: The proposition is verified as follows:

Truth Table

p	q	$p \wedge q$	$p \vee q$	$\sim (p \vee q)$	$(p \wedge q) \wedge \sim (p \vee q)$
0	0	0	0	1	0
0	1	0	1	0	0
1	0	0	1	0	0
1	1	1	1	0	0

Since the truth value of $(p \wedge q) \wedge \sim (p \vee q)$ is 0 for all values of p and q, the proposition is a contradiction.

Example 2.3: Show that the conjunction of two tautologies is also a tautology.

Solution: Let us denote A and B by two statement formulae which are tautologies. If we assign any truth values to the variables of A and B, then the truth values of both A and B will be 1. Thus, the truth value of $A \wedge B$ will be 1, so that $A \wedge B$ will be a tautology.

Example 2.4: From the formulae given below, indicate if they are tautologies or contradictions.

(*i*) $p \rightarrow (p \vee q)$

(*ii*) $(p \rightarrow \sim p) \rightarrow \sim p$

(*iii*) $(\sim q \wedge p) \wedge q$

(*iv*) $(p \vee q) \wedge (\sim p \wedge \sim q)$

Solution: the solution is obtained as follows:

Truth Table

p	q	$\sim p$	$\sim q$	$p \vee q$	$\sim p \wedge \sim q$	$p \rightarrow \sim p$	$\sim q \wedge p$	(a)	(b)	(c)	(d)
0	0	1	1	0	1	1	0	1	1	0	0
0	1	1	0	1	0	1	0	1	1	0	0
1	0	0	1	1	0	0	1	1	1	0	0
1	1	0	0	1	0	0	0	1	1	0	0

Since the truth values of statements (*i*) and (*ii*) are 1 for all values of p and q, statements (*i*) and (*ii*) are tautologies. Since the truth values of statements (*iii*) and (*iv*) are 0 for all values of p and q, statements (*iii*) and (*iv*) are contradictions.

Substitution instance: A formula A is called a substitution instance of another formula B, if A can be obtained from B by substituting formulae for some variables of B, with the condition that the same formula is substituted for the same variables each time it occurs.

Note: Suppose A (p, q, ….) is a tautology. Then it does not depend upon the particular truth values of its variables p,q,… so we can substitute P for p, Q for q… for any propositions P, Q,… in the tautology A (p,q…) and still have a tautology A(P, Q,…).

Example 2.5: Verify that the proposition $(p \wedge \sim q) \vee \sim (p \wedge \sim q)$ is a tautology.

Solution: This proposition has the form $p \vee \sim p$ where $p = p \wedge \sim q$. Since $p \vee \sim p$ is a tautology, by the above Note, $(p \wedge \sim p) \vee \sim (p \wedge \sim q)$ is also a tautology.

Note: It is possible to substitute more than one variable by other formulae provided all substitutions are considered to occur simultaneously.

Example 2.6: Produce the substitution instance of the following formulae for the given substitutions:

(*i*) $(((p \rightarrow q) \rightarrow p) \rightarrow p)$; substitute $(p \rightarrow q)$ for p and $(p \wedge q \rightarrow r)$ for q.

(*ii*) $((p \rightarrow q) \rightarrow (q \rightarrow p)$; substitute q for p and $(p \wedge \sim p)$ for q.

Solution: The solution is obtained as follows:

(*i*) Substitute $(p \rightarrow q)$ for p and $(p \wedge q \rightarrow r)$ for q simultaneously, we get

$(((p \rightarrow q) \rightarrow (p \wedge q \rightarrow r)\ \ (p \rightarrow q) \rightarrow (p \rightarrow q))$

(*ii*) Substitute q for p and $(p \wedge \sim p)$ for q simultaneously in:

$((p \rightarrow q) \rightarrow\ \ (q \rightarrow p))$, we get $(q \rightarrow (p \wedge \sim p)) \rightarrow ((p \wedge \sim p) \rightarrow q))$

Note: In constructing substitution instances of a formula, substitutions are made for the simple proposition (without connectives) and never for the compound proposition. For example, $p \rightarrow q$ is not a substitution instance of $p \sim r$, because it must be replaced by r and not $\sim r$.

Example 2.7: Verify if the proposition $((p \vee \sim q) \rightarrow r) \leftrightarrow$ s $\vee \sim$ (((p $\vee \sim$ q) $\rightarrow r) \leftrightarrow s)$ is a tautology.

Solution: This proposition has the form $p \vee \sim p$ where $p = (((p \vee \sim q) \rightarrow r) \leftrightarrow s)$

NOTES

2.3 TAUTOLOGICAL IMPLICATIONS AND EQUIVALENCE OF FORMULAE

NOTES

A statement is said to be a *tautology* if it is true for all logical possibilities.

A statement is said to be a *contradiction* if it is false for all logical possibilities.

In the following example, suppose t is a tautology, c is a contradiction and p is any proposition. Then,

According to Tautology Laws:

$p \wedge t \equiv p$

$p \vee t \equiv t$

$\sim t \equiv c$

According to Contradiction Laws:

$p \wedge c \equiv c$

$p \vee c \equiv p$

$\sim c \equiv t$

The following examples and the truth tables are discussed to verify these laws.

Example 2.8: Prove that the statement $p \vee \sim p$ is a tautology while $p \wedge \sim p$ is a contradiction.

Solution: If p is false, then $\sim p$ is true. So, $p \vee \sim p$ is true and $p \wedge \sim p$ is false. If p is true, then $\sim p$ is false. So, $p \vee \sim p$ is true and $p \wedge \sim p$ is false. These are the only logical possibilities and in each case $p \vee \sim p$ is true while $p \wedge \sim p$ is false. This proves our assertion.

Example 2.9: Prove that $(p \wedge q) \Rightarrow (p \vee q)$ is a tautology.

Solution: The following truth table shows that the compound statement, $(p \wedge q) \Rightarrow (p \vee q)$ has truth value T for all logical possibilities and so, it is a tautology.

Since, $p \Rightarrow q = \sim p \vee q$

$(p \wedge q) \Rightarrow (p \vee q)$

$= \sim(p \vee q) \vee (p \vee q)$

Truth Table

p	q	$p \wedge q$	$\sim (p \wedge q)$	$p \vee q$	$(p \wedge q) \Rightarrow (p \vee q)$ $= \sim(p \wedge q) \vee (p \vee q)$
T	T	T	F	T	T
T	F	F	T	T	T
F	T	F	T	T	T
F	F	F	T	F	T

Example 2.10: Prove that $(p \wedge q) \wedge \sim(p \vee q)$ is a contradiction.

Solution: The following truth table shows that the compound statement, $(p \wedge q) \wedge \sim(p \vee q)$ has truth value F for all logical possibilities and so it is a contradiction.

Truth Table

p	q	$p \wedge q$	$p \vee q$	$\sim(p \vee q)$	$(p \wedge q) \wedge \sim(p \vee q)$
T	T	T	T	F	F
T	F	F	T	F	F
F	T	F	T	F	F
F	F	F	F	T	F

NOTES

Example 2.11: Prove that $(p \Rightarrow q) \wedge (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$ is a tautology. This is also called Transitive Law.

Solution: We construct the truth table for the given compound statement to prove the assertion.

Truth Table

p	q	r	$p \Rightarrow q$	$q \Rightarrow r$	$p \Rightarrow r$	$(p \Rightarrow q) \wedge (q \Rightarrow r)$	$(p \Rightarrow q) \wedge (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$
T	T	T	T	T	T	T	T
T	T	F	T	F	F	F	T
T	F	T	F	T	T	F	T
T	F	F	F	T	F	F	T
F	T	T	T	T	T	T	T
F	T	F	T	F	T	F	T
F	F	T	T	T	T	T	T
F	F	F	T	T	T	T	T

Example 2.12: If t denotes tautology and p is any statement, then show that,

(i) $p \wedge t \equiv p$

(ii) $p \vee t \equiv t$

Solution: (i) When p is true, $p \wedge t$ is also true (as t is always true) and when p is false, $p \wedge t$ is false. So, p and $p \wedge t$ are equivalent statements.

(ii) When p is true, $p \vee t$ is also true and when p is false, $p \vee t$ is true. So, $p \vee t$ is always true. Hence, $p \vee t$ and t are equivalent statements.

Example 2.13: If c denotes a contradiction and p be any statement, then show that,

(i) $p \vee c \equiv p$ (ii) $p \wedge c \equiv c$

Solution: (i) If p is true, then $p \vee c$ is clearly true and if p is false, then $p \vee c$ is false (as c is always false). So, $p \vee c$ and p are equivalent statements.

(ii) If p is true, then $p \wedge c$ is false and again p is false implies that $p \wedge c$ is false. So, $p \wedge c$ is a contradiction. Hence, $p \wedge c$ and c are equivalent statements.

Biconditional Statement

Consider the statement,

'If you work hard you will be successful.' Suppose p stands for 'You work hard' and q stands for 'You will be successful.' In symbols, the above statement can be written as $p \Rightarrow q$.

NOTES

Such statements are called *Conditional Statements*. Here q depends upon p, but p need not depend upon q.

The compound statement $p \Rightarrow q$ is true in every case except, when p is true and q is false.

If two conditional statements are combined such as $p \Rightarrow q$ and $q \Rightarrow P$, then it is known as biconditional statement, it is denoted as $p \Rightarrow q$.

Example 2.14: Construct truth table for $p \Rightarrow q$.

Solution: The truth table of $p \Rightarrow q$ will be:

Truth Table

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Example 2.15: Suppose p stands for 'The triangle is isosceles' and q stands for 'Two sides of the triangle are of equal length'. Translate the following compound statement into a symbolic form and give its equivalent statement, both in words and symbols.

'If the triangle is isosceles, then two sides of the triangle are of equal length.'

Solution: The above compound statement in symbolic form is '$p \Rightarrow q$'.

We show that $(p \Rightarrow q) \equiv (\sim q) \Rightarrow (\sim p)$. The following Truth Table proves our assertion:

Truth Table

p	q	$p \Rightarrow q$	$\sim q$	$\sim p$	$\sim q \Rightarrow \sim p$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

So, the equivalent statement in words will be 'If two sides of the triangle are not of equal length, then the triangle is not isosceles.'

Example 2.16: Are the following statements equivalent?

'If the traders do not reduce the prices, then the government will take action against them.'

'It is not true that the traders do not reduce the prices and government does not take action against them.'

NOTES

Solution: Suppose p stands for 'traders do not reduce the prices' and q stands for 'government takes action against them.'

The first statement in symbolic form is $p \Rightarrow q$ and the second statement is $\sim(p \wedge \sim q)$. We prove their equivalence by constructing the Truth Table:

Trtuh Table

p	q	$\sim q$	$p \wedge \sim q$	$\sim(p \wedge \sim q)$	$p \Rightarrow q$
T	T	F	F	T	T
T	F	T	T	F	F
F	T	F	F	T	T
F	F	T	F	T	T

Example 2.17: Construct the truth table for $p \Leftrightarrow q$.

Solution: The following is the truth table for $p \Leftrightarrow q$:

Truth Table

p	q	$p \Rightarrow q$	$q \Rightarrow p$	$p \Leftrightarrow q$
T	T	T	T	T
T	F	F	T	F
F	T	T	F	F
F	F	T	T	T

Example 2.18: A firm of Chartered Accountants makes the following declaration: An articled clerk from the firm passing the final C.A. examination in the first attempt will be awarded a prize of Rs 100. Five clerks *P, Q, R, S, U* appeared in the examination and only *P, Q* could pass. The firm awards prizes not only to them but to *R* and *S* also. Is this action logically justified? *U* claims the prize comparing himself with *R* and *S* but the firm refuses. Is this refusal logically justified? How should the statement be worded so that only *P* and *Q* will be entitled for the prize.

Solution: Suppose p stands for 'passing the examination in first attempt' and q stands for 'getting a prize,' then declaration of the firm is a conditional statement $p \Rightarrow q$ as in given table. It is very clear from truth table of $p \Rightarrow q$ that relationship $p \Rightarrow q$ is false only when p is true and q is false. In other words, if *R* and *S* get a prize, the action is logically justified by looking at 3rd row of the truth table. Again, *U* does not get a prize implies statement q is false for *U*. Since '*U* has not passed the examination' implies p is false for *U*, so, 4th row suggests that action is logically justified. Also *P* and *Q* get a prize implies q is true for *P* and *Q*. Since *P* and *Q* have passed the examination implies p is true for *P* and *Q*. So, first row suggests that action, *P*, *Q* get a prize is logically justified.

Now, consider the truth table of $p \Leftrightarrow q$. Since *R*, *S*, *U* fail in the examination implies p is false for *R*, *S*, *U*. If they get a prize, then q is true for them. Third row suggests that, it is not correct. If *R*, *S*, *U* do not get a prize, then q is false for them and 4th row suggests that it is correct. So, truth table of $p \Leftrightarrow q$ shows that if a person fails, he cannot get a prize. Finally, if *P*, *Q* get a prize then p, q are true for

NOTES

both P and Q and by first row, it is correct. If P, Q do not get a prize, then q is false for P and Q and so by 2nd row, it is not true. So, the truth table of $p \Leftrightarrow q$ shows that only P and Q can get a prize.

Hence, the statement should be 'Those and only those persons who pass the examination in first attempt will get a prize of Rs 100.'

Example 2.19: Are the following statements equivalent? Justify 'it is not true that Ashok will get a job if and only if he secures first division.'

'Ashok will not get a job if and only if he secures first division.'

Solution: Suppose p stands for 'Ashok gets a job' and q stands for 'Ashok secures first division.' The two statements in symbolic form will then be:

$$\sim(p \Leftrightarrow q), \sim p \Leftrightarrow q$$

We prove their equivalence by constructing the Truth Table.

Truth Table

p	q	$\sim p$	$\sim p \Leftrightarrow q$	$p \Leftrightarrow q$	$\sim(p \Leftrightarrow q)$
T	T	F	F	T	F
T	F	F	T	F	T
F	T	T	T	F	T
F	F	T	F	T	F

2.3.1 Equivalence and Implication

An important step used in mathematical argument is the replacement of a statement with another statement with the same truth value. Because of this, methods that produce propositions with the same truth value as a given compound proposition are used extensively in the construction of mathematical arguments.

Equivalence

Two propositions are logically equivalent or simply equivalent if they have exactly the same truth values under all circumstances.

We can also define this notion as follows,

The propositions p and q are called logically equivalent, if $p \leftrightarrow q$ is tautology. The equivalence of p and q is denoted by $p \Leftrightarrow q$.

Notes: 1. One way to determine whether two propositions are equivalent is to use a truth table. In particular, the propositions p and q are logically equivalent, if and only if the columns giving their truth values agree.

2. Whenever we find logically equivalent statements, we can substitute one for another as we wish, since this action will not change the truth value of any statement.

Example 2.20: Show that $\sim (p \wedge q)$ and $\sim p \vee \sim q$ are logically equivalent.

Solution: Construct the truth table of these propositions as shown below. Since

the truth values are same for all combinations, it follows that these propositions are logically equivalent.

Truth Table

p	q	$\sim p$	$\sim q$	$p \wedge q$	$\sim (p \wedge q)$	$\sim p \vee \sim q$
0	0	1	1	0	1	1
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	0	0

NOTES

Example 2.21: Show that two propositions $p \rightarrow q$ and $\sim p \vee \sim q$ are logically equivalent.

Solution: Construct the required truth table. Since the truth values of $p \rightarrow q$ and $\sim p \vee q$ agree,

$$p \rightarrow q \Leftrightarrow \sim p \vee q$$

Truth Table

p	q	$\sim p$	$p \rightarrow q$	$\sim p \vee q$
0	0	1	1	1
0	1	1	1	1
1	0	0	0	0
1	1	0	1	1

Example 2.22: Show that $p \rightarrow (q \rightarrow r) \Leftrightarrow p \rightarrow (\sim q \vee r) \Leftrightarrow (p \wedge q) \rightarrow r$.

Solution: Refer Example 2.21 that $q \rightarrow r \Leftrightarrow \sim q \vee r$.

Replacing $q \rightarrow r$ by $\sim q \vee r$, we get

$$\begin{aligned} p \rightarrow (q \rightarrow r) &\Leftrightarrow p \rightarrow (\sim q \vee r) \\ &\Leftrightarrow \sim p \vee (\sim q \vee r) && \text{[by Example 2.21]} \\ &\Leftrightarrow (\sim p \vee \sim q) \vee r && \text{using Associativity of } \vee \\ &\Leftrightarrow \sim (p \wedge q) \vee r && \text{(by Example 2.20)} \\ &\Leftrightarrow (p \wedge q) \rightarrow r && \text{[by Example 2.21]} \end{aligned}$$

Table given below contains some important equivalences. In these equivalences, 1 denotes any proposition that is a tautology, and 0 denotes any proposition that is a contradiction. The symbol *p, q, r* represent arbitrary propositions. Most of the equivalences in this table have straight forward intuitive interpretations and all of them can be verified by constructing truth tables.

NOTES

Table *Logical Equivalences*

	Equivalence	Name
1.	$p \Leftrightarrow (p \vee p)$	Idempotents of $\vee$
2.	$p \Leftrightarrow (p \wedge p)$	Idempotents of $\wedge$
3.	$(p \vee q) \Leftrightarrow (q \vee p)$	Commutativity of $\vee$
4.	$(p \wedge q) \Leftrightarrow (q \wedge p)$	Commutativity of $\wedge$
5.	$(p \vee q) \vee r \Leftrightarrow p \vee (q \vee r)$	Associativity of $\vee$
6.	$(p \wedge q) \wedge r \Leftrightarrow p \wedge (q \wedge r)$	Associativity of $\wedge$
7.	$\sim (p \vee q) \Leftrightarrow \sim p \wedge \sim q$	De Morgan's law 1
8.	$\sim (p \wedge q) \Leftrightarrow \sim p \vee \sim q$	De Morgan's law 2
9.	$p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$	Distributive of $\wedge$ over $\vee$
10.	$p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$	Distributive of $\vee$ over $\wedge$
11.	$p \vee 1 \Leftrightarrow 1$	Null or Domination law 1
12.	$p \wedge 0 \Leftrightarrow 0$	Null or Domination law 2
13.	$p \wedge 1 \Leftrightarrow p$	Identity law 1
14.	$p \vee 0 \Leftrightarrow p$	Identity law 2
15.	$p \vee \sim p \Leftrightarrow 1$	Negation law 1
16.	$p \wedge \sim p \Leftrightarrow 0$	Negation law 2
17.	$\sim (\sim p) \Leftrightarrow p$	Double Negation law (Involution)
18.	$p \rightarrow q \Leftrightarrow \sim p \vee q$	Implication law
19.	$p \leftrightarrow q \Leftrightarrow (p \rightarrow q) \wedge (q \rightarrow p)$	Equivalence law
20.	$(p \wedge q) \rightarrow r \Leftrightarrow p \rightarrow (q \rightarrow r)$	Exportation law
21.	$(p \rightarrow q) \wedge (p \rightarrow \sim q) \Leftrightarrow \sim p$	Absurdity law
22.	$p \rightarrow q \Leftrightarrow \sim q \rightarrow \sim p$	Contrapositive law
23.	$p \vee (p \wedge q) \Leftrightarrow p$	Absorption law 1
24.	$p \wedge (p \vee q) \Leftrightarrow p$	Absorption law 2
25.	$p \leftrightarrow q \Leftrightarrow (p \wedge q) \vee (\sim p \wedge \sim q)$	Biconditional law

Example 2.23: Show that $\sim (p \vee (\sim p \wedge q)$ and $\sim p \wedge \sim q$ are logically equivalent.

Solution: Instead of a truth table we shall establish equivalence by developing a series of logical equivalences using Table of Example 2.22, starting with $\sim (p \vee (\sim p \wedge q)$ and ending with $\sim p \wedge \sim q$.

$\sim (p \vee (\sim p \wedge q) \Leftrightarrow \sim p \wedge \sim (\sim p \wedge q)$ by De Morgan's law 1

$\Leftrightarrow \sim p \wedge (\sim (\sim p) \vee \sim q)$ by De Morgan's law 2

$\Leftrightarrow \sim p \wedge (p \vee \sim q)$ by Involution law

$\Leftrightarrow (\sim p \wedge p) \vee (\sim p \wedge \sim q)$ by Distributive law of $\wedge$ over $\vee$

$\Leftrightarrow 0 \vee (\sim p \wedge \sim q)$ by Negation law 2

$\Leftrightarrow (\sim p \wedge \sim q) \vee 0$ by Commutative law of $\vee$

$\Leftrightarrow \sim p \wedge \sim q$ by Identity law 2

Hence, $\sim (p \vee (\sim p \wedge q))$ and $\sim p \wedge \sim q$ are logically equivalent.

Example 2.24: Show that $(p \wedge q) \rightarrow (p \vee q) \Leftrightarrow 1$.

Solution:

$(p \wedge q) \rightarrow (p \vee q) \Leftrightarrow \sim (p \wedge q) \vee (p \vee q)$ by Implication law

$\Leftrightarrow (\sim p \vee \sim q) \vee (p \vee q)$ by De Morgan's law 2

$\Leftrightarrow \sim p \vee (\sim q \vee p) \vee q$ by Associative law of $\vee$

$\Leftrightarrow \sim p \vee (p \vee \sim q) \vee q$ by Commutative law of $\vee$

$\Leftrightarrow (\sim p \vee p) \vee (\sim q \vee q)$ by Associative law of $\vee$

$\Leftrightarrow 1 \vee 1$ by Negation law 1

$\Leftrightarrow 1$ by Domination law 1

Hence proved.

Note: This could also be done using a truth table.

Example 2.25: Show that $(\sim p \wedge (\sim q \wedge r)) \vee (q \wedge r) \vee (p \wedge r) \Leftrightarrow r$.

Solution:

$(\sim p \wedge (\sim q \wedge r)) \vee (q \wedge r) \vee (p \wedge r)$

$\Leftrightarrow (\sim p \wedge (\sim q \wedge r)) \vee ((q \vee p) \wedge r)$ by Distributive law of $\wedge$ over $\vee$

$\Leftrightarrow ((\sim p \wedge \sim q) \wedge r) \vee ((q \vee p) \wedge r)$ by Associative law of $\wedge$

$\Leftrightarrow ((\sim p \wedge \sim q) \vee (q \vee p)) \wedge r$ by Distributive law of $\wedge$ on $\vee$

$\Leftrightarrow (\sim (p \vee q) \vee (p \vee q)) \wedge r$ by De Morgan's law 1 and Commutative law of $\vee$

$\Leftrightarrow 1 \wedge r$ by Negation law 1

$\Leftrightarrow r$ by Identity law 1

Hence proved.

Example 2.26: Show that

$(p \vee q) \wedge \sim (\sim p \wedge (\sim q \vee \sim r))) \vee (\sim p \wedge \sim q) \vee (\sim p \wedge \sim r)$ is a tautology.

Solution: Consider,

NOTES

$(\sim p \wedge \sim q) \vee (\sim \text{p} \wedge \sim \text{r})$

$\Leftrightarrow \sim (p \vee q) \vee \sim (p \vee r)$ by De Morgan's law 1

$\Leftrightarrow \sim ((p \vee q) \wedge (p \vee r))$ by De Morgan's law 2

$\sim (\sim p \wedge (\sim q \vee \sim r))$

$\Leftrightarrow \sim (\sim p \wedge \sim (q \wedge r))$ by De Morgan's law 2

$\Leftrightarrow \sim (\sim p) \vee \sim (\sim (q \wedge r))$ by De Morgan's law 2

$\Leftrightarrow p \vee (q \wedge r)$ by Involution law

$\Leftrightarrow (p \vee q) \wedge (p \vee r)$ by Distributive law of $\vee$ over $\wedge$

Hence, the given formula is equivalent to,

$((p \vee q) \wedge (p \vee r)) \vee \sim ((p \vee q) \wedge (p \vee r))$

If we substitute $(p \vee q) \wedge (p \vee r)$ for p in $p \vee \sim p$, we get the above formula. But $p \vee \sim p \Leftrightarrow 1$, the given formula is a tautology.

Example 2.27: Prove the following equivalences.

(*i*) $p \rightarrow (q \rightarrow p) \Leftrightarrow \sim p \rightarrow (p \rightarrow q)$

(*ii*) $p \rightarrow (q \vee r) \Leftrightarrow (p \rightarrow q) \vee (p \rightarrow r)$

(*iii*) $(p \rightarrow q) \wedge (r \rightarrow q) \Leftrightarrow (p \vee r) \rightarrow q$

(*iv*) $\sim (p \Leftrightarrow q) \Leftrightarrow (p \vee q) \wedge \sim (p \wedge q)$

Solution:

(*i*) Consider, $p \rightarrow (q \rightarrow p) \Leftrightarrow \sim p \vee (\sim q \vee p)$ by Implication law

$\Leftrightarrow (\sim p \vee p) \vee \sim q$ by Associative and Commutative laws of $\vee$

$\Leftrightarrow 1 \vee \sim q$ by Negation law 1

$\Leftrightarrow 1$ by Null law 1

Again, $\sim p \rightarrow (p \rightarrow q) \Leftrightarrow \sim (\sim p) \vee (\sim p \vee q)$ by Implication law

$\Leftrightarrow p \vee (\sim p \vee q)$ by Involution law

$\Leftrightarrow (p \vee \sim p) \vee q$ by Associative law of $\vee$

$\Leftrightarrow 1 \vee q$ by Negation law 1

$\Leftrightarrow 1$ by Null law 1

Hence, $p \rightarrow (q \rightarrow p) \Leftrightarrow \sim p \rightarrow (p \rightarrow q)$.

(*ii*) $p \rightarrow (q \vee r) \Leftrightarrow \sim p \vee (q \vee r)$ by Implication law

$\Leftrightarrow \sim p \vee \sim p \vee (q \vee r)$, since $\sim p \Leftrightarrow \sim p \vee \sim p$

$\Leftrightarrow (\sim p \vee q) \vee (\sim p \vee r)$

by Associative and Commutative laws of $\vee$

$\Leftrightarrow (p \rightarrow q) \vee (p \rightarrow r)$ by Implication law

(*iii*) $(p \to q) \wedge (r \to q) \Leftrightarrow (\sim p \wedge q) \wedge (\sim r \vee q)$ by Implication law

$\Leftrightarrow (\sim p \wedge \sim r) \vee q$ by Distributive law of $\vee$ over $\wedge$

$\Leftrightarrow \sim (p \vee r) \vee q$ by De Morgan's law 1

$\Leftrightarrow (p \vee q) \to q$ by Implication law

(*iv*) $\sim (p \Leftrightarrow q) \Leftrightarrow \sim ((p \wedge q) \vee \sim (p \vee q))$ by biconditional and De Morgan's law 1

$\Leftrightarrow \sim (p \wedge q) \wedge \sim (\sim (p \vee q))$ by De Morgan's law 1

$\Leftrightarrow \sim (p \wedge q) \wedge (p \vee q)$ by Involution law

$\Leftrightarrow (p \vee q) \wedge \sim (p \wedge q)$ by Commutative law of $\wedge$

Example 2.28: Write an equivalent formula for $p \wedge (q \leftrightarrow r) \vee (r \leftrightarrow p)$ which does not contain the biconditional.

Solution: Since $p \leftrightarrow q \Leftrightarrow (p \to q) \wedge (q \to p)$, $p \wedge (q \leftrightarrow r) \vee (r \leftrightarrow p) \Leftrightarrow p \wedge (q \to r) \wedge (r \to q)) \vee ((r \to p) \wedge (p \to r))$.

Example 2.29: Write an equivalent formula for $p \wedge (q \leftrightarrow r)$ which contains neither the biconditional nor the conditional.

Solution: Since, $p \leftrightarrow q \Leftrightarrow (p \to q) \wedge (q \to p)$ and $p \to q \Leftrightarrow \sim p \vee q$.

$p \wedge (q \leftrightarrow r) \Leftrightarrow p \wedge ((q \to r) \wedge (r \to q))$

$\Leftrightarrow p \wedge ((\sim q \vee r) \wedge (\sim r \vee q))$

Implication

A proposition p is said to logically imply or tautologically imply or simply imply a proposition q if q is true whenever p is true. We can also define this notion as follows:

A proposition p is said to logically imply a proposition q if $p \to q$ is a tautolgoy. The implication of p to q is denoted by $p \Rightarrow q$.

Notes: 1. One can determine whether $p \Rightarrow q$ by constructing the truth table of p and q in the same manner as was done in the determination of $p \Leftrightarrow q$.

2. Since $(p \to q) \Leftrightarrow (p \to q) \wedge (q \to p)$ (equivalence law), it is easy to see from the definition of equivalence and implication that $p \Leftrightarrow q$ and if and only if $p \Rightarrow q$ and $q \Rightarrow p$. This statement is an alternative definition of the equivalence as two propositions.

The implications in Table 2.2 have important applications. All of them can be proved by truth table or by other methods.

NOTES

NOTES

Table 2.2 *Logical Implications*

	Implication	Name
1.	$p \wedge q \Rightarrow p$	Conjunctive simplification
2.	$p \wedge q \Rightarrow q$	Conjunctive simplification
3.	$p \Rightarrow p \vee q$	Disjunctive addition
4.	$p \wedge (p \rightarrow q) \Rightarrow q$	Detachment
5.	$\sim q \wedge (p \rightarrow q) \Rightarrow \sim p$	Contrapositive
6.	$(p \vee q) \wedge \sim q \Rightarrow p$	Disjunctive simplification
7.	$(p \vee q) \wedge \sim p \Rightarrow q$	Disjunctive simplification
8.	$(p \rightarrow q) \wedge (q \rightarrow r) \Rightarrow (p \rightarrow r)$	Chain rule
9.	$\sim p \Rightarrow p \rightarrow q$	
10.	$q \Rightarrow p \rightarrow q$	
11.	$\sim (p \rightarrow q) \Rightarrow p$	
12.	$\sim (p \rightarrow q) \Rightarrow \sim q$	
13.	$(p \vee q) \wedge (p \rightarrow q) \wedge (q \rightarrow r) \Rightarrow r$	Dilemma
14.	$p \wedge q \Rightarrow p \leftrightarrow q$	
15.	$p \rightarrow (q \rightarrow r) \Rightarrow (p \rightarrow q) \rightarrow (p \rightarrow r)$	

Example 2.30: Show that $p \leftrightarrow q$ logically implies $p \rightarrow q$.

Solution: Consider the truth table of $p \leftrightarrow q$ and $p \rightarrow q$. Now $p \rightarrow q$ is true in line 1 and 4, and in these cases $p \leftrightarrow q$ is also true. Hence, $p \leftrightarrow q$ implies $p \rightarrow q$.

Truth Table

p	q	$p \leftrightarrow q$	$p \rightarrow q$
0	0	1	1
0	1	0	1
1	0	0	0
1	1	1	1

Now we will introduce two methods to show $p \Rightarrow q$.

Method 1: To show the implication $p \Rightarrow q$, we assume that p has the truth value 1 and then show that this assumption leads to q having the value 1. Then $p \rightarrow q$ must have the value 1.

Method 2: To show $p \Rightarrow q$, we assume that q has the truth value 0 and then show that this assumption leads to p having the value 0. Then $p \rightarrow q$ must have the truth value 1.

Example 2.31: Prove that the implication given in the Table 2.2 for,

$(p \vee q) \wedge (p \to r) \wedge (q \to r) \Leftrightarrow r$, is true.

Solution: Now by method 1, we assume that $(p \vee q) \wedge (p \to r) \wedge (q \to r)$ is true. This assumption means that $p \vee q$, $p \to r$ and $q \to r$ are true. Since $p \vee q$ is true, at least one p or q is true. If p is true then r must be true since $p \to r$ is true. If q is true then, r must be true since $q \to r$ is true. So, r is true. Hence, the result (13) in Table 2.2 is true..

NOTES

Example 2.32: Prove that the implication given in the Table 2.2 $\sim q \wedge (p \to q) \Rightarrow \sim p$, is true.

Solution: Now by method 2, assume that $\sim p$ is false, so, p is true. If q is true then $\sim q$ is false. If q is false then $p \to q$ is false. So, in both cases, $\sim q \wedge (p \to q)$ is false. Hence, the implication (5) in Table 2.2 is true.

Example 2.33: Show the following implications:

(*i*) $(p \wedge q) \Rightarrow p \to q$

(*ii*) $p \Rightarrow q \to p$

(*iii*) $p \to (q \to r) \Rightarrow (p \to q) \to (p \to r)$

Solution:

(*i*) Assume that $p \wedge q$ is true. This means that p and q are true. So, $p \to q$ must also be true. Hence, by method 1, the result follows.

(*ii*) Assume that p is true. Then, for all possible truth values of q, $q \to p$ is true. Hence, by method 1, the result follows.

(*iii*) Assume that $(p \to q) \to (p \to r)$ is false. Then $p \to r$ is false and $p \to q$ is true. This means that p and q are true. Since r is false, $q \to r$ is false and hence $p \to (q \to r)$ must also be false. Hence, by method 2, the implication follows.

Example 2.34: Show the following implications without constructing the truth tables.

(*i*) $p \to q \Rightarrow p \to (p \wedge q)$

(*ii*) $(p \to q) \to q \Rightarrow p \wedge q$

(*iii*) $((p \vee \sim p) \to q) \to ((p \vee \sim p) \to r) \Rightarrow q \to r$

(*iv*) $(q \to (p \vee \sim p)) \to (r \to (p \vee \sim p)) \Rightarrow r \to q$

Solution:

(*i*) Assume that $p \to (p \wedge q)$ is false. Then, p is true and $p \wedge q$ is false. So, q is false. Hence, $p \to q$ is false. So, by method 2, the implication follows.

(*ii*) Assume that $p \wedge q$ is false. This means that both p and q are false. So, $p \to q$ is true and hence $(p \to q) \to q$ is false. Hence, by method 2, the result follows.

NOTES

(*iii*) Suppose that $q \to r$ is false. Then, q is true and r is false. This implies that $(p \vee \sim p) \to q$ is true and $p \vee \sim q \to r$ is false. Hence $(p \vee \sim p) \to q$
$((p \vee \sim p) \to r)$ is false. By method 2, the result follows.

(*iv*) Assume that $r \to q$ is false. Then, r is true and q is false. So, $q \to ((p \wedge \sim p)$ is true and $r \to (p \wedge \sim p)$ is false. Hence, $(q \to (p \wedge \sim p) \to r\ (p \wedge \sim p))$ is false and the result follows.

Note: Note that $p \vee \sim p$ is always true and $p \wedge \sim p$ is always false.

Example 2.35: If $p_1, p_2, ..., p_n$ and p imply q, then $p_1, p_2, ..., p_n$ imply $p \to q$.

Solution: Let us assume that $(p_1 \wedge p_2 \wedge ... \wedge p_n \wedge p) \Rightarrow q$

This means $(p_1 \wedge p_2 \wedge ... \wedge p_n \wedge p) \to q$ is a tautology:

But we know that (Example 2.22) $(p \wedge q) \to r \Rightarrow p \to (q \to r)$

Using this, we get $(p_1 \wedge p_2 \wedge ... \wedge p_n) \to (p \to q)$ is a tautology.

Hence, $(p_1 \wedge p_2 \wedge ... \wedge p_n) \Rightarrow (p \to q)$.

2.4 REPLACEMENT PROCESS

In discrete mathematics, the replacement process can be defined on the basis of 'Rules of Equivalence or Replacement' which is also termed as 'De Morgan's Rule'.

De Morgan stated that, 'The statements that say the same thing or are equivalent to one another are very important to a system of logical deduction'. De Morgan's Theorems are abbreviated as DeM.

For example, if we have a true conjunction then we can assume that either of its parts is true. If in the statement there is a 'neither…nor' statement, then there is a conjunction, such as 'It will neither rain nor snow'. This statement is similar or equivalent to the statement 'It will not rain and it will not snow either'. Hence, a 'neither…nor' statement can be simplified and replaced if it is equivalent to a conjunction of negatives.

Since two logically equivalent statements have the same truth value on every possible combination of truth values for their component parts, therefore is no change in the truth value of any statement when one of the value is replaced with the other. Consequently, while constructing proofs of validity one has to safely use a statement containing either one of a pair of logical equivalents as the premise for a step whose conclusion is exactly the similar or equivalent, except that it contains the other one.

The tautological statement are represented using the connective '≡'. In order for the '≡' statement to be true on every line, the statement forms on either side of it must always have exactly the similar or equivalent truth value. Statements that

are substitution instances of these two component statement forms are termed as logically equivalent, irrespective of the content, the conditions for their truth or falsity are exactly similar or equivalent. Consider the following example.

Double Negation (DN)	p a≡ ~ ~p
As per De Morgan's Theorems (DeM)	~(p • q) ≡ (~p v ~q) ~(p v q) ≡ (~p • ~q)
Implication	(p⊃q) ≡ (~p v q)
Equivalence	[p≡q] ≡ [(p⊃q) • (q⊃p)] [p≡q] ≡ [(p • q) v (~p • ~q)]
Transposition	(p ⊃ q) ≡ (~q ⊃ ~p)

The tautological biconditional can be used as rules of replacement.

NOTES

2.5 LAW OF DUALITY

Let S be any identity involving sets and the set operations $\cup$, $\cap$ and C. If T is obtained from S by applying the substitutions $\cup \to \cap$; $\cap \to \cup$, $\varnothing \to \mu$, $\mu \to \varnothing$, then T is also true and T is called as dual of S. For example

The dual of $A \cup (B \cap A)$ is $A \cap (B \cup A) = A$.

Two formulae or propositions are said to be *dual* of each other if either one can be obtained from the other by replacing $\wedge$ by $\vee$ and $\vee$ by $\wedge$.

Notes: 1. The connectives $\wedge$ and $\vee$ are called duals. If the proposition contains the special variables 1 or 0, then its dual, is obtained by replacing 1 by 0 and 0 by 1 in addition to the above mentioned interchanges.

2. If any two formulae are equivalent, then their duals are also equivalent to each other.

Example 2.36: Write the duals of the following:

(*i*) $(p \vee q) \wedge r$ (*ii*) $(p \wedge q) \vee 1$

(*iii*) $\sim (p \vee q) \wedge (p \vee \sim (q \wedge \sim s))$

Solution: The duals are:

(*i*) $(p \wedge q) \vee r$ (*ii*) $(p \vee q) \wedge 0$ (*iii*) $\sim (p \wedge q) \vee (p \wedge \sim (q \vee \sim s))$.

Example 2.37: Show that,

(*i*) $\sim (p \wedge q) \to (\sim p \vee (\sim p \vee q)) \Leftrightarrow (\sim p \vee q)$

(*ii*) $(p \vee q) \wedge (\sim p \wedge (\sim p \wedge q)) \Leftrightarrow (\sim p \wedge q)$

NOTES

Solution:

(*i*) $\sim (p \wedge q) \rightarrow (\sim p \vee (\sim p \vee q))$

$\Leftrightarrow$ $(p \wedge q) \vee (\sim p \vee (\sim p \vee q))$ by Implication and Involution laws

$\Leftrightarrow$ $(p \wedge q) \vee (\sim p \vee q)$ by Associative law

$\Leftrightarrow$ $((p \wedge q) \vee \sim p) \vee q$ by Associative law

$\Leftrightarrow$ $((p \vee \sim p) \wedge (q \vee \sim p)) \vee q$ by Distributive law

$\Leftrightarrow$ $(1 \wedge (q \vee \sim p)) \vee q$ by Negation law

$\Leftrightarrow$ $(q \vee \sim p) \vee q$ by Identity law 1

$\Leftrightarrow$ $\sim p \vee q$ by Associative, idempotent and Commutative laws

(*ii*) In (*i*) we have proved that,

$$(p \wedge q) \vee (\sim p \vee (\sim p \vee q)) \Leftrightarrow (\sim p \vee q)$$

By Note (2) of duelity,

$$(p \vee q) \wedge (\sim p \wedge (\sim p \wedge q)) \Leftrightarrow (\sim p \wedge q)$$

Check Your Progress

1. Explain the tautology.
2. Define the contradiction.
3. What is contingency?
4. State the substitution instance.
5. Interpret the conditional statement.
6. What do you understand by the biconditional statement?
7. Explain the equivalence of propositions.
8. Analyse the implications of propositions.
9. Define the replacement process.
10. Elaborate on the law of duality.

2.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Tautology: A statement formula which is true regardless of the truth values of the statements which replace the variables in it is called a tautology or a logical truth or a universally valid formula.

NOTES

2. Contradiction: A statement formula which is false regardless of the truth values of the statements which replace the variables in it is called a contradiction.
3. Contingency: A statement formula that is neither a tautology nor a contradiction is called a contingency.
4. Substitution instance: A formula A is called a substitution instance of another formula B, if A can be obtained from B by substituting formulae for some variables of B, with the condition that the same formula is substituted for the same variables each time it occurs.
5. 'If you work hard you will be successful.' Suppose p stands for 'You work hard' and q stands for 'You will be successful.' In symbols, the above statement can be written as $p \Rightarrow q$.

   Such statements are called Conditional Statements. Here q depends upon p, but p need not depend upon q.
6. If two conditional statements are combined such as $p \Rightarrow q$ and $q \Rightarrow P$, then it is known as biconditional statement, it is denoted as $p \Rightarrow q$.
7. Two propositions are logically equivalent or simply equivalent if they have exactly the same truth values under all circumstances.
8. A proposition p is said to logically imply or tautologically imply or simply imply a proposition q if q is true whenever p is true. We can also define this notion as follows:

   A proposition p is said to logically imply a proposition q if $p \rightarrow q$ is a tautolgoy. The implication of p to q is denoted by $p \Rightarrow q$.
9. In discrete mathematics, the replacement process can be defined on the basis of 'Rules of Equivalence or Replacement' which is also termed as 'De Morgan's Rule'.
10. Let S be any identity involving sets and the set operations $\cup$, $\cap$ and C. If T is obtained from S by applying the substitutions $\cup \rightarrow \cap$; $\cap \rightarrow \cup$, $\varnothing \rightarrow \mu$, $\mu \rightarrow \varnothing$, then T is also true and T is called as dual of S. For example,

    The dual of $A \cup (B \cap A)$ is $A \cap (B \cup A) = A$.

2.7 SUMMARY

- Tautology: A statement formula which is true regardless of the truth values of the statements which replace the variables in it is called a tautology or a logical truth or a universally valid formula.
- Contradiction: A statement formula which is false regardless of the truth values of the statements which replace the variables in it is called a contradiction.

NOTES

- Contingency: A statement formula that is neither a tautology nor a contradiction is called a contingency.
- Substitution instance: A formula A is called a substitution instance of another formula B, if A can be obtained from B by substituting formulae for some variables of B, with the condition that the same formula is substituted for the same variables each time it occurs.
- 'If you work hard you will be successful.' Suppose p stands for 'You work hard' and q stands for 'You will be successful.' In symbols, the above statement can be written as $p \Rightarrow q$.
- Such statements are called Conditional Statements. Here q depends upon p, but p need not depend upon q.
- If two conditional statements are combined such as $p \Rightarrow q$ and $q \Rightarrow P$, then it is known as biconditional statement, it is denoted as $p \Rightarrow q$.
- Two propositions are logically equivalent or simply equivalent if they have exactly the same truth values under all circumstances.
- A proposition p is said to logically imply or tautologically imply or simply imply a proposition q if q is true whenever p is true. We can also define this notion as follows:
- A proposition p is said to logically imply a proposition q if $p \rightarrow q$ is a tautolgoy. The implication of p to q is denoted by $p \Rightarrow q$.
- In discrete mathematics, the replacement process can be defined on the basis of 'Rules of Equivalence or Replacement' which is also termed as 'De Morgan's Rule'.
- Let S be any identity involving sets and the set operations $\cup$, $\cap$ and C. If T is obtained from S by applying the substitutions $\cup \rightarrow \cap$; $\cap \rightarrow \cup$, $\varnothing \rightarrow \mu$, $\mu \rightarrow \varnothing$, then T is also true and T is called as dual of S. For example,

  The dual of $A \cup (B \cap A)$ is $A \cap (B \cup A) = A$.

2.8 KEY WORDS

- **Tautology:** A statement formula which is true regardless of the truth values of the statement which replace the variables in it, is called a tautology or a logical truth or a universally valid formula.
- **Contradiction:** A statement formula which is false regardless of the truth values of the statement which replace the variables in it, is called a contradiction.
- **Contingency:** A statement formula that is neither a tautology nor a contradiction is called a contingency.

NOTES

- **Substitution instance:** A formula A is called a substitution instance of another formula B, if A can be obtained from B by substituting formulae for some variables of B, with the condition that the same formula is substituted for the same variables each time it occurs.
- **Equivalence:** Two propositions are logically equivalent or simply equivalent if they have exactly the same truth values under all circumstances.
- **Implication:** A proposition p is said to logically imply a proposition q if $p \rightarrow q$ is a tautology. The implication of p to q is denoted by $p \Rightarrow q$.
- **Duality:** Two formulae or propositions are said to be dual of each other if either one can be obtained from the other by replacing $\wedge$ by $\vee$ and $\vee$ by $\wedge$.

2.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain the term tautology.
2. What is contradiction?
3. Define contingency.
4. Elaborate on the substitution instance.
5. Analyse the biconditional statement.
6. What do you understand by the equivalence?
7. Interpret the implication.
8. State the replacement process.
9. Explain the law of duality.

Long-Answer Questions

1. Discuss about the equivalence formula and tautology giving suitable examples of each.
2. Show that two propositions $p \rightarrow q$ and $\sim p \vee \sim q$ are logically equivalent.
3. Show that $(p \wedge q) \rightarrow (p \vee q)$ is a tautology.
4. Verify that the proposition $(p \wedge \sim q) \vee \sim (p \wedge \sim q)$ is a tautology.
5. Write the truth tables for $\wedge$ and $\vee$ operators.
6. Write the truth table of the conditional operator.
7. Write the truth table for the biconditional.
8. Write the truth table of c: $q \wedge (p \rightarrow q) \rightarrow p$.

2.10 FURTHER READINGS

NOTES

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 3 NORMAL FORMS

NOTES

Structure

3.0 INTRODUCTION

In mathematics, the normal forms are typically used to solve the mathematical problems specifically in decision making problems for finding whether a given

statement is a tautology or a contradiction or agreeable and acceptable in a finite number of steps. For finding appropriate answers to decision problems, we construct the truth tables, which may not be possible always. Hence, we use an alternate method termed as the reduction to normal forms.

A normal form is a representation such that zero is uniquely represented. This allows testing for equality by putting the difference of two objects in normal form. a formula is in Conjunctive Normal Form (CNF) or clausal normal form if it is a conjunction of one or more clauses, where a clause is a disjunction of literals; otherwise put, it is a product of sums or an AND of ORs. As a canonical normal form, it is useful in automated theorem proving and circuit theory.

All conjunctions of literals and all disjunctions of literals are in CNF, as they can be seen as conjunctions of one-literal clauses and conjunctions of a single clause, respectively. As in the Disjunctive Normal Form (DNF), the only propositional connectives a formula in CNF can contain are and, or, and not. The not operator can only be used as part of a literal, which means that it can only precede a propositional variable or a predicate symbol.

A Disjunctive Normal Form (DNF) is a canonical normal form of a logical formula consisting of a disjunction of conjunctions; it can also be described as an OR of ANDs, a sum of products, or (in philosophical logic) a cluster concept. As a normal form, it is useful in automated theorem proving.

In this unit, you will study about the normal forms, disjunctive normal form, conjunctive normal form, and principal of normal forms.

3.1 OBJECTIVES

NOTES

After going through this unit, you will be able to:

- Understand the normal forms
- Explain the disjunctive normal form
- Define the conjunctive normal form
- Elaborate on the principal of normal forms

3.2 NORMAL AND PRINCIPAL FORMS

In mathematics, the normal forms are typically used to solve the mathematical problems specifically in decision making problems for finding whether a given statement is a tautology or a contradiction or agreeable and acceptable in a finite number of steps. For finding appropriate answers to decision problems, we construct the truth tables, which may not be possible always. Hence, we use an alternate method termed as the reduction to normal forms.

Basically, the normal forms are of following two types:

1. Disjunctive Normal Form (DNF)
2. Conjunctive Normal Form (CNF)

Disjunctive Normal Form (DNF): A formula which is equivalent to a given formula and which consists of a sum of elementary products is called a Disjunctive Normal Form (DNF) of given formula. The notation is given as,

$$(P \wedge \sim Q) \vee (Q \wedge R) \vee (\sim P \wedge Q \wedge \sim R)$$

The Disjunctive Normal Form (DNF) of formula is not exceptional.

If p, q are two statements, then "p or q" is a compound statement, denoted by $p \vee q$ and referred as the disjunction of p and q. The disjunction of p and q is true whenever at least one of the two statements is true, and it is false only when both p and q are false. Consider the following truth table.

Truth Table

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

For example, if p is "4 is a positive integer" and q is "$\sqrt{5}$is a rational number", then $p \vee q$ is true as statement p is true, although statement q is false.

Conjunctive Normal Form (CNF): A formula which is equivalent to a given formula and which consists of a product of elementary products is called a Conjunctive Normal Form (CNF) of given formula. The notation is given as,

$$(P \sim \vee Q) \wedge (Q \vee R) \wedge (\sim P \vee Q \vee \sim R)$$

The Conjunctive Normal Form (CNF) of formula is not exceptional. If every elementary sum in CNF is tautology, then given formula is also tautology.

If p, q are two statements, then "p and q" is a compound statement, denoted by $p \wedge q$ and referred as the conjunction of p and q. The conjunction of p and q is true only when both p and q are true, otherwise, it is false. Consider the following truth table.

Truth Table

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

For example, if statement p is "$6 < 7$" and statement q is "$-3 > -4$" then the conjunction of p and q is true as both p and q are true statements.

PDNF and PCNF

PDNF: PDNF stands for Principal Disjunctive Normal Form. It refers to the Sum of Products, i.e., SOP. For example, if P, Q, R are the variables then **(P . Q' . R) + (P' . Q . R) + (P . Q . R')** is an example of an expression in PDNF. In this expression **'+'** (Sum) is referred as the main operator.

The key difference between DNF (Disjunctive Normal Form) and PDNF (Principal Disjunctive Normal Form) is that in case of DNF, it is not essential that the length of all the variables in the expression is similar. For example,

1. **(P . Q' . R) + (P' . Q . R) + (P . Q)** is an example of an expression in DNF but not in PDNF.
2. **(P . Q' . R) + (P' . Q . R) + (P . Q . R')** is an example of an expression which is both in PDNF and DNF.

PCNF: PCNF stands for Principal Conjunctive Normal Form. It refers to the Product of Sums, i.e., POS. For example, if P, Q, R are the variables then **(P + Q' + R).(P' + Q + R).(P + Q + R')** is an example of an expression in PCNF. In this expression the **'.'** (Product) is referred as the main operator.

NOTES

NOTES

The key difference between PCNF (Principal Conjunctive Normal Form) and CNF (Conjunctive Normal Form) is that in case of CNF, it is not essential that the length of all the variables in the expression is similar. For example,

1. **(P + Q'+ R).(P'+ Q + R).(P + Q)** is an example of an expression in CNF but not in PCNF.
2. **(P + Q'+ R).(P'+ Q + R).(P + Q + R')** is an example of an expression which is both in PCNF and CNF.

Properties of PCNF and PDNF

Following are the properties of PCNF and PDNF:

1. Every PDNF or PCNF corresponds to a unique Boolean Expression and vice versa.
2. If X and Y are two Boolean expressions, then X is equivalent to Y if and only if PDNF(X) = PDNF(Y) or PCNF(X) = PCNF(Y).
3. For a Boolean Expression, if PCNF has m terms and PDNF has *n* terms, then the number of variables in such a Boolean expression = $\log_2 (m + n)$.

Check Your Progress

1. Explain the normal forms.
2. Define disjunctive normal form.
3. Analyse the conjunctive normal form.
4. Interpret the PDNF.
5. Explain the PCNF.
6. State the properties of PCNF and PDNF.

3.3 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. In mathematics, the normal forms are typically used to solve the mathematical problems specifically in decision making problems for finding whether a given statement is a tautology or a contradiction or agreeable and acceptable in a finite number of steps.
2. Disjunctive Normal Form (DNF): A formula which is equivalent to a given formula and which consists of a sum of elementary products is called a Disjunctive Normal Form (DNF) of given formula. The notation is given as,

   $(P \wedge \sim Q) \vee (Q \wedge R) \vee (\sim P \wedge Q \wedge \sim R)$

   The Disjunctive Normal Form (DNF) of formula is not exceptional.

3. Conjunctive Normal Form (CNF): A formula which is equivalent to a given formula and which consists of a product of elementary products is called a Conjunctive Normal Form (CNF) of given formula. The notation is given as,

   $(P \sim \vee Q) \wedge (Q \vee R) \wedge (\sim P \vee Q \vee \sim R)$

4. PDNF: PDNF stands for Principal Disjunctive Normal Form. It refers to the Sum of Products, i.e., SOP. For example, if P, Q, R are the variables then (P . Q' . R) + (P' . Q . R) + (P . Q . R') is an example of an expression in PDNF. In this expression '+' (Sum) is referred as the main operator.
5. PCNF: PCNF stands for Principal Conjunctive Normal Form. It refers to the Product of Sums, i.e., POS. For example, if P, Q, R are the variables then (P + Q'+ R).(P'+ Q + R).(P + Q + R') is an example of an expression in PCNF. In this expression the '.' (Product) is referred as the main operator.
6. Following are the properties of PCNF and PDNF:
   - Every PDNF or PCNF corresponds to a unique Boolean Expression and vice versa.
   - If X and Y are two Boolean expressions, then X is equivalent to Y if and only if PDNF(X) = PDNF(Y) or PCNF(X) = PCNF(Y).
   - For a Boolean Expression, if PCNF has m terms and PDNF has *n* terms, then the number of variables in such a Boolean expression = $\log_2 (m + n)$.

3.4 SUMMARY

- In mathematics, the normal forms are typically used to solve the mathematical problems specifically in decision making problems for finding whether a given statement is a tautology or a contradiction or agreeable and acceptable in a finite number of steps.
- Disjunctive Normal Form (DNF): A formula which is equivalent to a given formula and which consists of a sum of elementary products is called a Disjunctive Normal Form (DNF) of given formula. The notation is given as,

  $(P \wedge \sim Q) \vee (Q \wedge R) \vee (\sim P \wedge Q \wedge \sim R)$

  The Disjunctive Normal Form (DNF) of formula is not exceptional.
- Conjunctive Normal Form (CNF): A formula which is equivalent to a given formula and which consists of a product of elementary products is called a Conjunctive Normal Form (CNF) of given formula. The notation is given as,

  $(P \sim \vee Q) \wedge (Q \vee R) \wedge (\sim P \vee Q \vee \sim R)$

NOTES

- The Conjunctive Normal Form (CNF) of formula is not exceptional. If every elementary sum in CNF is tautology, then given formula is also tautology.
- PDNF: PDNF stands for Principal Disjunctive Normal Form. It refers to the Sum of Products, i.e., SOP. For example, if P, Q, R are the variables then (P . Q' . R) + (P' . Q . R) + (P . Q . R') is an example of an expression in PDNF. In this expression '+' (Sum) is referred as the main operator.
- PCNF: PCNF stands for Principal Conjunctive Normal Form. It refers to the Product of Sums, i.e., POS. For example, if P, Q, R are the variables then (P + Q'+ R).(P'+ Q + R).(P + Q + R') is an example of an expression in PCNF. In this expression the '.' (Product) is referred as the main operator.
- The key difference between DNF (Disjunctive Normal Form) and PDNF (Principal Disjunctive Normal Form) is that in case of DNF, it is not essential that the length of all the variables in the expression is similar.
- Every PDNF or PCNF corresponds to a unique Boolean Expression and vice versa.
- If X and Y are two Boolean expressions, then X is equivalent to Y if and only if PDNF(X) = PDNF(Y) or PCNF(X) = PCNF(Y).
- For a Boolean Expression, if PCNF has m terms and PDNF has *n* terms, then the number of variables in such a Boolean expression = $\log_2 (m + n)$.

3.5 KEY WORDS

- **Normal forms:** In mathematics, the normal forms are typically used to solve the mathematical problems specifically in decision making problems for finding whether a given statement is a tautology or a contradiction or agreeable and acceptable in a finite number of steps. Basically, the normal forms are of two types, Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF).
- **Disjunctive normal form (DNF):** A formula which is equivalent to a given formula and which consists of a sum of elementary products is called a Disjunctive Normal Form (DNF) of given formula. The notation is given as, (P Ù ~ Q) (Q Ù R) Ú (~ P Ù Q Ù ~ R). The Disjunctive Normal Form (DNF) of formula is not exceptional.
- **Conjunctive normal form (CNF):** A formula which is equivalent to a given formula and which consists of a product of elementary products is called a Conjunctive Normal Form (CNF) of given formula. The notation is given as, (P ~ ∨ Q) ∧ (Q ∨ R) ∧ (~ P ∨ Q ∨ ~ R). The Conjunctive Normal Form (CNF) of formula is not exceptional. If every elementary sum in CNF is tautology, then given formula is also tautology.

3.6 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. Define the normal forms.
2. Explain disjunctive normal form.
3. Elaborate on the conjunctive normal form.
4. Analyse the properties of PCNF and PDNF.
5. Interpret the principal of normal forms.

Long-Answer Questions

1. What are normal forms? Discuss the types of normal forms giving examples of each type.
2. Explain briefly about the Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF) giving definition, notations, examples and truth tables.
3. Find the conjunction and disjunction of the propositions p and q where p is the proposition 'Today is Monday' and q is the proposition 'It is raining today'. Write in symbolic form.
4. Obtain the conjunctive normal form of:
   $(i)\ \sim(p \vee q)$ $(ii)\ \sim(p \rightarrow q)$ $(iii)\ \sim(p \leftrightarrow q)$
5. Obtain the conjunctive normal forms of the following formulas:
   $(i)\ (p \wedge q \wedge r) \vee (\sim p \wedge r \wedge q) \vee (\sim p \wedge \sim q \wedge \sim r)$
   $(ii)\ (p \wedge q) \vee (\sim p \wedge q) \vee (p \wedge \sim q)$
   $(iii)\ (p \wedge q) \vee (\sim p \wedge q \wedge r)$
6. Obtain the disjunctive normal forms of:
   $(i)\ (p \rightarrow (q \wedge r)) \wedge (\sim p \rightarrow (\sim q \wedge \sim r))$
   $(ii)\ (\sim p \vee \sim q) \rightarrow (p \leftrightarrow \sim q)$
   $(iii)\ (p \vee (\sim p \rightarrow (q \vee (\sim q \rightarrow r))))$
7. Obtain the disjunctive normal forms of:
   $(i)\ p \wedge (p \vee q)$ $(ii)\ p \wedge (\sim p \vee q)$ $(iii)\ (p \vee q) \wedge (\sim p \vee q) \wedge (q \vee r)$
8. Obtain the disjunctive and conjunctive normal forms of the following formulas and hence state which of the formulas are tautologies?
   $(i)\ q \vee (p \wedge \sim q) \vee (\sim p \wedge \sim q)$ $(ii)\ q \vee (p \vee \sim q)$
   $(iii)\ (q \rightarrow p) \wedge (\sim p \wedge q)$ $(iv)\ p \rightarrow (p \wedge (q \rightarrow p))$

3.7 FURTHER READINGS

NOTES

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 4 INFERENCE THEORY

NOTES

Structure

4.0 INTRODUCTION

Theory of inference is a logical form consisting of a function which takes premises, analyses their syntax, and returns a conclusion (or conclusions). For example, the rule of inference called modus ponens takes two premises, one in the form "If *p* then *q*" and another in the form "*p*", and returns the conclusion "*q*". The rule is valid with respect to the semantics of classical logic (as well as the semantics of many other non-classical logics), in the sense that if the premises are true (under an interpretation), then so is the conclusion.

Typically, a rule of inference preserves truth, a semantic property. In many-valued logic, it preserves a general designation. But a rule of inference's action is purely syntactic, and does not need to preserve any semantic property: any function from sets of formulae to formulae counts as a rule of inference. Usually only rules that are recursive are important; i.e., rules such that there is an effective procedure for determining whether any given formula is the conclusion of a given set of formulae according to the rule. An example of a rule that is not effective in this sense is the infinitary ω-rule.

In this unit, you will study about the theory of inference, rules of inference, and open statements.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- Comprehend the theory of inference
- Explain the rules of inference
- Analyse the open statements

NOTES

4.2 THEORY OF INFERENCE

To draw inference we must have some rule or a set of rules that serves the basis of inference, otherwise inference will not have sound reasoning. This uses the rules of inference given for the statement calculus along with additional rules needed to deal with formulas with quantifiers. We can draw inference on any given statement with symbols and logical connectives either by truth table of by applying rules of inference that are given in subsequent topic.

If case conclusion has the form of a conditional statement, rule of **Conditional Proof** called **CP** is used. For using equivalences and implications, some rules are needed to eliminate quantifiers for such derivation.

Rules of specification, known as rules **US (Universal Specification)** and ES **(Existential Specification)** are used for the purpose of elimination. After eliminating quantifiers the inference is drawn. If the desired conclusion is to be quantified, rules of generalization called rules **UG (Universal Generalization)** and **EG (Existential Generalization)** are used to attach a quantifier.

All these rules are given under the topic 'Rules for inference'.

We can draw inference on any given statement with symbols and logical connectives either by truth table of by applying rules of inference that are given in subsequent topic.

Two statements are equivalent if they have identical truth values. A logical statement is valid when it is a tautology. To check this, truth tables are constructed.

4.3 RULES OF INFERENCE

Table 4.1 illustrates the rules of inference and the implications form of each type.

Table 4.1

Rules of Inference	Implication Form
Addition I_1	
$\therefore \quad \frac{p}{p \vee \varphi}$	$P \Rightarrow p \vee \varphi$
Conjunction I_2	
$\therefore \quad \frac{\varphi}{p \wedge \varphi}$	$P \wedge \varphi \Rightarrow p \wedge \varphi$
Simplification I_3	
$\therefore \quad \frac{p \wedge \varphi}{p}$	$(p \wedge \varphi) \Rightarrow p$

Modus Tollens I_4

$\neg \varphi$

$p \Rightarrow \varphi$ $\quad [\neg \varphi \wedge (p \Rightarrow \varphi)] \Rightarrow \neg p$

$\therefore \quad \neg p$

Disjunctive Syllogism I_5

$\neg p$

$\therefore \quad \dfrac{p \vee \varphi}{\varphi}$ $\quad [\neg p \wedge (p \vee \varphi)] \Rightarrow \varphi$

Modus Ponens

p

$\therefore \quad \dfrac{p \Rightarrow \varphi}{\varphi}$ $\quad [p \wedge (p \Rightarrow \varphi)] \Rightarrow \varphi$

Hypothetical Syllogism I_7

$p \Rightarrow \varphi$

$\therefore \quad \dfrac{\varphi \Rightarrow R}{p \Rightarrow R}$ $\quad [(P \Rightarrow \varphi) \wedge (\varphi \Rightarrow R)] \Rightarrow (P \Rightarrow R)$

Conjunctive Dilemma

$[(P \Rightarrow \varphi) \wedge (R \Rightarrow s)]$

$\therefore \quad \dfrac{p \vee R}{\varphi \vee s}$

$(p \Rightarrow \varphi) \wedge (R \Rightarrow S) \wedge (p \vee R) \Rightarrow (\varphi \vee S)$

Disjunctive Dilemma

$(p \Rightarrow \varphi) \wedge (R \Rightarrow S)$

$\neg \varphi \vee \neg s$

$\therefore \quad p \vee R$ $\quad [(p \Rightarrow \varphi) \wedge (R \Rightarrow S)] \wedge [\neg \varphi \vee \neg S] \Rightarrow (\neg p \vee \neg R)$

NOTES

Besides, we can use the two rules of inference called Rules *P* and *T*.

Rule P: A premise may be introduced at any point in the derivation.

Rule T: A formula *S* may be introduced in a derivation if *S* is tautologically implied by any one or more of the preceeding formulae in the derivation of a truth value.

Example 4.1: For the given set of arguments check the validity of conclusion:

(*i*) If determinism is true then we have no free will.

(*ii*) If Heisenberg's interpretation of quantum physics is correct, then there are events not necessiated by prior events.

(*iii*) If there are events not necessitated by prior events, then we have free will.

Conclusion: If Heisenberg interpretation of physics is correct then we have free will.

NOTES

Solution: D : Determinism is true.

Q : We have no free will.

R : Heisenberg interpretation of quantum physics is correct.

S : There are events not necessitated by prior events.

Arguments: $(D \rightarrow \varphi)\ (R \rightarrow S)(S \rightarrow \urcorner\varphi)$

Conclusion: $R \rightarrow \neg\ \varphi$

(*i*) $R \rightarrow S$ Rule P

(*ii*) $R \rightarrow \ \neg\varphi$ Rule P

From cases (*i*) and (*ii*), we get :

(*iii*) $R \rightarrow \emptyset\ \varphi$ rule T and hypothetical syllogism.

4.4 OPEN STATEMENT

In discrete mathematics, 'Inference Rules' are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal. In inference rules, the implication among all the connectives plays an important role.

Fundamentally, a sentence is a group of words which arrange themselves in a meaning manner. Any sentence which we say or write or think is considered as a statement. In mathematics also we define the statements which are referred as the mathematically valid statements or mathematically invalid statements.

A mathematical statement is the basic unit of any mathematical reasoning. A mathematical reasoning is either inductive (mathematical induction) or deductive. Any assertive sentence which is either 'True' or 'False' but not both is considered as a mathematically acceptable statement. This type of statement is a valid statement. Any ambiguous sentence is not a statement and hence it is invalid.

In mathematics, a **sentence** whose **truth value** can be determined is called a **statement** or **proposition**. A statement is also called a closed sentence because its truth value is not open to question.

Statements that cannot be answered **absolutely** are called **open statements** or **open sentences**. A compound statement is formed by using logical connectives on individual statements.

Definition for Open Statements: A sentence which contains one or more variable such that when certain values are given to the variable it becomes a statement is an open statement.

Consider the following example.

Let P be a statement.

P is an **open statement** if and only if P contains at least **one free occurrence** of a **variables** that appears in it.

NOTES

For example, the statement *x* is a prime number is an open statement, as the variable *x* appears as a free occurrence.

Check Your Progress

1. Define the theory of inference.
2. What are the rules of specifications?
3. State the rules of generalisation.
4. Elaborate on the open statements.

4.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. To draw inference we must have some rule or a set of rules that serves the basis of inference, otherwise inference will not have sound reasoning. This uses the rules of inference given for the statement calculus along with additional rules needed to deal with formulas with quantifiers. We can draw inference on any given statement with symbols and logical connectives either by truth table of by applying rules of inference that are given in subsequent topic.
2. Rules of specification, known as rules US (Universal Specification) and ES (Existential Specification) are used for the purpose of elimination. After eliminating quantifiers the inference is drawn.
3. Rules of generalization called rules UG (Universal Generalization) and EG (Existential Generalization) are used to attach a quantifier.
4. Statements that cannot be answered absolutely are called open statements or open sentences. A compound statement is formed by using logical connectives on individual statements.

4.6 SUMMARY

- To draw inference we must have some rule or a set of rules that serves the basis of inference, otherwise inference will not have sound reasoning. This uses the rules of inference given for the statement calculus along with additional rules needed to deal with formulas with quantifiers. We can draw inference on any given statement with symbols and logical connectives either

NOTES

by truth table of by applying rules of inference that are given in subsequent topic.

- Rules of specification, known as rules US (Universal Specification) and ES (Existential Specification) are used for the purpose of elimination. After eliminating quantifiers the inference is drawn.
- Rules of generalization called rules UG (Universal Generalization) and EG (Existential Generalization) are used to attach a quantifier.
- Statements that cannot be answered absolutely are called open statements or open sentences. A compound statement is formed by using logical connectives on individual statements.

4.7 KEY WORDS

- **Theory of inference:** To draw inference we must have some rule or a set of rules that serves the basis of inference, otherwise inference will not have sound reasoning. Theory of inference is a logical form consisting of a function which takes premises, analyses their syntax, and returns a conclusion (or conclusions).
- **Rules of specification:** These rules known as US (Universal Specification) and ES (Existential Specification) which are used for the purpose of elimination. After eliminating quantifiers the inference is drawn.
- **Rules of generalisation:** Rules of generalisation called rules UG (Universal Generalisation) and EG (Existential generalisation) which are used to attach a quantifier.
- **Open statements:** Statements that cannot be answered absolutely are called open statements or open sentences. A compound statement is formed by using logical connectives on individual statements.

4.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain the inference theory.
2. Define the rules of specification.
3. What are the rules of generalisation?
4. Analyse the rules of inference.
5. Interpret the open statements.

Long-Answer Questions

1. Briefly discuss the theory of inference.
2. Define the significances of rules of specification.
3. Analyse the rules of generalisation.
4. Elaborate on the 'Rules of Inference' in mathematical logic.
5. Explain the open statements with suitable example.

NOTES

4.9 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

BLOCK - II
QUANTIFIERS, LATTICES AND CODING THEORY

NOTES

UNIT 5 QUANTIFIERS

Structure

5.0 INTRODUCTION

In mathematical logic, a quantifier is an operator that specifies how many individuals in the domain of discourse satisfy an open formula. For instance, the universal quantifier $\forall$ in the first order formula $\forall P(x)$ expresses that everything in the domain satisfies the property denoted by p. On the other hand, the existential quantifier $\exists$ in the formula $\exists xP(x)$ expresses that there is something in the domain which satisfies that property. A formula where a quantifier takes widest scope is called a quantified formula. A quantified formula must contain a bound variable and a subformula specifying a property of the referent of that variable.

The mostly commonly used quantifiers are $\forall$ and $\exists$. These quantifiers are standardly defined as duals and are thus interdefinable using negation. They can also be used to define more complex quantifiers, as in the formula $\neg\exists xP(x)$ which expresses that nothing has the property P. Other quantifiers are only definable within second order logic or higher order logics. Quantifiers have been generalised beginning with the work of Mostowski and Lindström. First order quantifiers approximate the meanings of some natural language quantifiers such as "Some" and "All". However, many natural language quantifiers can only be analysed in terms of generalized quantifiers.

A free variable is a notation (symbol) that specifies places in an expression where substitution may take place and is not a parameter of this or any container

expression. Some older books use the terms real variable and apparent variable for free variable and bound variable, respectively. The idea is related to a placeholder (a symbol that will later be replaced by some value), or a wildcard character that stands for an unspecified symbol. A bound variable is a variable that was previously free, but has been bound to a specific value or set of values called domain of discourse or universe.

In this unit, you will study about the quantifiers, bound and free variables, and theory of inference for predicate calculus.

5.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand what quantifiers are
- Explain the bound and free variables
- Analyse the theory of inference for predicate calculus

5.2 QUANTIFIERS

When all the variables in a propositional function are assigned values, the resulting statement has a truth value. However, there is another important way to change propositional functions into propositions, called quantification. It has been broadly classified into two types as follows:

(*i*) Universal Quantification
(*ii*) Existential Quantification

Universe of Discourse: Many mathematical statements assert that a property is true for all values of a variable in a particular domain, called the universe of discourse. Such a statement is expressed using an universal quantification.

Universal Quantification: The universal quantification of $p(x)$ is a proposition only when $p(x)$ is true for all values of x in the universe of discourse.

Notation: $\forall x\, p(x) \rightarrow$ Universal quantification of $p(x)$

It is also expressed as,

'for all $x\, p(x)$' or 'for every $x\, p(x)$'

Example 5.1: 'Every student in this class has studied logic'.

Solution: Let $p(x)$ denote the statement 'x has studied logic',

$$\forall x\, [s(x) \rightarrow p(x)]$$

Where $s(x)$ is the statement 'x is in this class'

Example 5.2: What is the truth value of $\forall x\, p(x)$, where $p(x)$: $x^2 < 10$ and the universe of discourse consists of the positive integers not exceeding 4!

Solution: The statement $\forall x\, p(x)$ is the same as the conjunction. $p(1) \wedge p(2) \wedge p(3) \wedge p(4)$

Since the universe of discourse consists of the integers 1, 2, 3,and 4.

NOTES

$p(4)$, in the statement '$4^2 < 10$' is false, it follows $\forall x\, p(x)$ is also false.

Reason:

$p(1) \wedge p(2) \wedge p(3) \wedge p(4)$

$T \wedge T \wedge T \wedge F$ = false

Existential Quantification: The existential quantification of $p(x)$ is the proposition 'There exists an element x in the universe of discourse such that $p(x)$ is true.'

Notation: $\exists x\, p(x)$

It is also expressed as,

'There is an x such that $p(x)$.'

'There is atleast one x such that $p(x)$', or for some $x\, p(x)$.

Example 5.3: Let $p(x) : x > 3$, what is the truth value of the quantification $\exists x\, p(x)$ where the universe of discourse is the set of real numbers.

Solution: Since $x > 3$ is true, for example, when $x = 4$ the existential quantification of $p(x)$ is $\exists x\, p(x)$ is true.

Example 5.4: Write the predicate 'x is the father of the mother of y'.

Solution: Let $p(x) : x$ is a person.

$p(x, z) : x$ is the father of z.

$m(z, y) : z$ is the mother of y.

We assume that the there exists a person z such that x is the father of z and z is the mother of y.

5.3 BOUND AND FREE VARIABLES

The variable is said to be bound if it is concerned with either universal ($\forall$) or existential ($\exists$) quantifier and the scope of the variable in the formulae immediately following the quantifier. The variable, which is not concerned with any quantifier, is called free variable.

For example, $\forall x\, [p(x, y)]$ in the statement given above x is said to be bound and the scope of x is upto $p(x, y)$, while y is called free variable.

5.4 THEORY OF INFERENCE FOR PREDICATE CALCULUS

NOTES

Consider the two statements 'Rohit is Brilliant' and 'Manaswine is Brilliant'. As propositions, there is no relation between them, but they have something in common. Instead of writing two statements we can write a single statement like 'x is brilliant', because both Rohit and Manaswine share the same nature brilliant. By replacing x by any other name we get many propositions. The common feature expressed by 'Is Brilliant' is called predicate. Predicate calculus deals with sentences involving predicates.

A part of a declaritive sentence describing the properties of an object or relation among objects can be referred as predicate, for example 'Is Brilliant'.

Note: A statement of the form $p(x_1, x_2, ..., x_n)$ is the value of the propositional function P at the nth tuple $(x_1,...,x_n)$ and P is also called a predicate.

For example, Statements involving variables such as $x > 5$, $x = y + 6$, and $x + y = z$.

These statement are neither true nor false. When the values of the variables are not specified.

Let us consider the statement $x > 5$.

Here, the variable x is the subject of the statement. The second part predicate is greater than 5 and > refers to a property that the subject of the statement can have. Therefore, $x > 5$ can be written in the form $p(x)$. The statement $p(x)$ is also said to be the value of the propositional function P at x. Once a value has been assigned to the variable x, the statement can be written as $p(x)$.

5.4.1 Statement Calculus

In logical reasoning, a certain number of propositions is assumed to be true, and based on that assumption, some other propositions are derived.

Hypothesis: The propositions that are assumed to be true. It may also be referred to as premises.

Conclusion: The proposition derived by using the rules of inference.

Valid Argument: The process of deriving conclusions based on the assumptions of a premise.

Example 5.5: Some cats are black but all buffaloes are black.

Solution: $C(x)$: x is a cat.

$B(y)$: y is a buffalo.

$b(x)$: x is black.

Thus, $(\exists x)\ (\forall y)[C(x) \wedge b(x)] \wedge [B(y) \rightarrow b(y)]$

Example 5.6: Sum of two positive integers is greater than either of the integers.

Solution: $I(x)$: x is a positive integer.

$GT(x, y)$: x is greater than y.

$Su(x, y)$: Sum of x and y.

Thus, $(\forall x)\ (\forall y)\ [(I(x) \wedge I(y))] \rightarrow [GT(Su(x, y), x] \vee GT[Su(x, y), y]$

Example 5.7: Every student in this school is either good at studies or good in sports.

Solution: $S(x)$: x is a student of this school.

$ST(x)$: x is good at study.

$SP(x)$: x is good at sports.

Thus, $(\forall x)\ [S(x) \rightarrow (ST(x) \vee SP(x)]$

Quantitiers are distributive over the predicate and negation of universe quantitier is existence quantifier and vice versa. This is being state below.

(*i*) $(\exists x)[A(x) \vee B(x)] \Leftrightarrow (\exists x)\ \{A(x)\} \vee (\exists x)\ \{B(x)\}$

(*ii*) $(\forall x)[A(x) \wedge B(x)] \Leftrightarrow (\forall x) \wedge [A(x)] \wedge (\forall x)\ [B(x)]$

(*iii*) $\neg(\exists x)\ A(x) \Leftrightarrow \forall x \neg A(x)$

(*iv*) $\neg(\forall x)\ A(x) \Leftrightarrow (\exists x)\ \neg A(x)$

Example 5.8: Give an argument which will establish the validity of the following inference.

'All integers are rational numbers. Some integers are power of 2. Therefore, some rational numbers are power of 2.'

Solution: The solution is obtained as follows:

$I(x)$: x is an integer.

$R(x)$: x is a rational number.

$P(x)$: x is a power of 2.

Inference Pattern:

$$\frac{(\forall x)\left[I(x) \rightarrow R(x)\right]}{(\exists x)\left[I(x) \wedge P(x)\right]}$$
$$(\exists x)\left[R(x) \wedge P(x)\right]$$

Argument:

1. $(\exists x)\ I(x) \wedge P(x)]$ *Premise*
2. $I(b) \wedge P(x)$ *ES* (1)
3. $I(b)$
4. $P(b)$
5. $(\forall x)\ [I(x) \rightarrow R(x)]$ *Premise*

6. $I(b) \rightarrow R(b)$ *US*

7. $R(b)$

8. $R(b) \wedge P(b)$

9. $(\exists x)[R(x) \wedge P(x)]$ *EG*

Example 5.9: Verify the validity of the following argument:

All men are mortal. Socrates is a man. Therefore, Socrates is mortal.

Solution: $H(x)$: x is a man.

$M(x)$: x is a mortal.

S : Socrates.

We have to show $(\forall x)[H(x) \rightarrow M(x) \wedge H(s)] \Rightarrow M(s)$

Argument:

1. $(\forall x)[H(x) \rightarrow M(x)$ *Premise*

2. $H(S) \rightarrow M(S)$ *US*

3. $H(S)$ *Premise*

4. $M(S)$

Thus the inference is valid.

Example 5.10: Show that from:

(*i*) $(\exists x)[F(x) \wedge S(x)] \rightarrow \forall y[m(y) \rightarrow w(y)]$

(*ii*) $(\exists y)[m(y) \wedge \neg w(y)]$

The conclusion $(\forall x)[F(x) \rightarrow \neg S(x)]$ follows.

1. $(\exists y)[m(y) \wedge \neg w(y)]$ *Premise*

2. $m(z) \wedge \neg w(z)$ *ES*

3. $\neg[M(z) \rightarrow W(z)]$

4. $(\exists y)\{\neg[m(y) \rightarrow w(y)]\}$ *EG*

5. $\neg(\forall y)[m(y) \rightarrow w(y)]$

6. $(\exists x)[f(x) \wedge s(x)] \rightarrow \forall y[m(y) \rightarrow w(y)]$ *Premise*

7. $\exists, \forall[(f(x) \wedge s(x))]$

8. $\forall x \neg (f(x) \wedge s(x))$

9. $\neg[f(x) \wedge s(x)]$ *US*

10. $F(x) \rightarrow \neg s(x)$

11. $(\forall x)[F(x) \rightarrow \neg s(x)]$ *UG*

Example 5.11: Is the following conclusion validly derivable from the premises given?

If $(\forall x)\ [p(x) \rightarrow \varphi(x)]$, $(\exists y)\ p(y)$, then $(\exists z)\ \varphi(z)$.

NOTES

Solution:

1.	$\neg (\exists z)\ \varphi (z)$	*Premise (Assumed)*
2.	$(\forall z) \neg \varphi (z)$	
3.	$(\exists y)\ p (y)$	*Premise*
4.	$p (a)$	*ES* (3)
5.	$\neg \varphi (a)$	*US* (2)
6.	$p (a) \wedge \neg \varphi (a)$	
7.	$\neg [p (a) \rightarrow \varphi (a)]$	
8.	$(\forall x)\ [p (x) \rightarrow \varphi (x)]$	*Premise*
9.	$p (a) \rightarrow \varphi (a)$	*US* (8)
10.	$[p (a) \rightarrow \varphi (a)] \wedge \neg [p (a) \rightarrow \varphi (a)]$	*Contradiction*

5.4.2 Rule CP (Conditional Proof)

If we can derive S from R and a set of premises then we may derive $R \rightarrow S$ from set of premises alone.

Example 5.12: Derive $P \rightarrow (Q \rightarrow S),$ using the rule CP if necessary from:

$P \rightarrow (Q \rightarrow R), Q \rightarrow (R \rightarrow S)$

Solution: The solution is obtained as follows:

Step 1

In rule CP the conditional part can be taken as an additional premise, i.e., $P \rightarrow (Q \rightarrow S)$

$\therefore$ We include P as an additional premise.

Step 2

1.	$P \rightarrow (Q \rightarrow R)$	*Premise*
2.	P	*Additional Premise*
{1,2}3.	$Q \rightarrow R$	$P \rightarrow Q, P \rightarrow R$
4.	$Q \rightarrow (R \rightarrow S)$	*Premise*
5.	$\neg Q \vee R$	$P \rightarrow Q, \neg P \vee R$

6. $\neg Q \vee (R \rightarrow S)$

7. $\neg Q \vee (R, R \rightarrow S)$

8. $\neg Q \vee S$

9. $Q \rightarrow S$

10. $P \rightarrow (Q \rightarrow S)$ *CP*

Example 5.13: Show that $P \rightarrow S$ can be derived from the premises.
$\neg P \vee Q, \neg Q \vee R, R \rightarrow S$

Solution: The following is the solution,

Step 1

P is additional premise.

Step 2

1.	$\neg P \vee Q$	*Premise*
2.	P	*Additional Premise*
{1,2}3.	Q	
4.	$\neg Q \vee R$	*Premise*
{3,4}5.	R	
6.	$R \rightarrow S$	*Premise*
{5,6}7.	S	
8.	$P \rightarrow S$	*CP*

5.4.3 Consistent and Inconsistent

A set of formulae $S_1, S_2, \ldots S_m$ is said to be inconsistent if their conjunction implies a contradiction, that is, $S_1 \wedge S_2 \wedge S_3 \wedge \ldots \wedge S_m \Rightarrow R \wedge \neg R \Rightarrow F$ for same formula *R*.

A set of formula $S_1, S_2, \ldots S_m$ is said to be consistent if it is not inconsistent.

Indirect Method of Proof

By using the rule of conditional proof and the notion of an inconsistent set of premises, we introduce a method called proof by contradiction or indirect method of proof.

Example 5.14: Prove by indirect method $P \rightarrow (q \wedge r), (q \vee S) \rightarrow t,$ and $p \vee S \Rightarrow t.$

Solution: The following is the proof by indirect method.

NOTES

Step 1

7t is an additional premise.

Step 2

	1.	$P \to (q \wedge \mathrm{r})$	*Premise*
	2.	$(q \vee s) \to t$	*Premise*
	3.	$\neg t$	*Additional Premise*
{1}	4.	$P \to q$	*Premise*
{1}	5.	$P \to r$	
{2}	6.	$S \to t$	*Premise*
{3,6}	7.	$\neg S$	$\neg Q, P \to Q \Rightarrow \neg P$
	8.	$P \vee S$	*Premise*
{8}	9.	S	
	10.	$S \times \neg S = F$	

Example 5.15: Prove by indirect method that,

$$E \to S, \quad S \to H, \quad A \to \neg H \Rightarrow \neg (E \wedge A)$$

Solution: The following is the proof by indirect method.

Step 1

$\neg\neg (E \wedge A)$

$\Rightarrow E \wedge A$ is an additional premise.

Step 2

	1.	$E \to S$	*Premise*
	2.	$S \to H$	*Premise*
{1, 2}	3.	$E \to H$	
	4.	$E \to \neg H$	*Premise*
	5.	$E \wedge A$	*Additional Premise*
{5}	6.	E	
{5}	7.	A	
{3, 6}	8.	$E, E \to H = H$	
{4, 7}	9.	$A, A \to \neg H = \neg H$	

10. $H \wedge \neg H = F$

Example 5.16: Using indirect proof show that $p \to q,\ q \to r,\ \neg (p \wedge r),\ p \vee r \Rightarrow r$

Solution:

Step 1

$\neg r$ is an additional premise.

Step 2

1.	$q \to r$	*Premise*
2.	$\neg r$	*Premise*
{1, 2}3.	$\neg q$	$p \to q, \neg q \Rightarrow \neg q$
4.	$p \to q$	*Premise*
{3, 4}5.	$\neg p$	$p \to q, \neg q \Rightarrow \neg q$
6.	$p \vee r$	*Premise*
{5, 6}7.	r	
8.	$r \wedge \neg r = f$	

Contradiction, hence premise.

Example 5.17: If an integer is divisible by 12 then it is divisible by 6. If an integer is divisible by 6, then it is divisible by 3. Prove that an integer divisible by 12 is divisible by 3.

Solution: The proof is as follows:

$D_{12}(x)$: x is divisible by 12.
$D_6(x)$: x is divisible by 6.
$D_3(x)$: x is divisible by 3.

$$\forall x[D_{12}(x) \to D_6(x)]$$
$$\underline{\forall x[D_6(x) \to D_3(x)]}$$
$$\forall x[D_{12}x \to D_3(x)]$$

1.	$\forall x[D_{12}(x) \to D_6(x)]$	*Premise*
2.	$D_{12}(b) \to D_6(b)$	*US*
3.	$\forall x[D_6(x) \to D_3(x)]$	*Premise*
4.	$D_6(b) \to D_3(b)$	*US*
{2, 4}5.	$D_{12}(b) \to D_3(b)$	$P \to Q, Q \to R \Rightarrow P \to R$
6.	$\forall x[D_{12}(x) \to D_3(x)]$	*Contusion*

NOTES

NOTES

Example 5.18: Prove $(\forall x)[P(x) \to Q(x)], \forall x[R(x) \to \neg Q(x)] \Rightarrow \forall x[R(x) \to \neg P(x)]$.

Solution: The proof is as follows:

Step 1

Taking negated conclusion:

$$\neg(\neg P(x))$$
$$\Rightarrow P(x)$$

Step 2

	1.	$(\forall x)[P(x) \to Q(x)]$	*Premise*
	2.	$(\forall x)[R(x) \to \neg Q(x)]$	*Premise*
{2}	3.	$R(x) \to \neg Q(x)$	*US*
	4.	$R(x)$	*Additional premise*
	5.	$\neg Q(x)$	(3, 4)
	6.	$P(x) \to Q(x)$	*US*
{5, 6}	7.	$\neg P(x)$	Using $\neg Q, P \to Q = \neg P$
	8.	$R(x) \to \neg P(x)$	*CP*
	9.	$(\forall x)[R(x) \to \neg P(x)]$	*UG*

Check Your Progress

1. Explain the quantification.
2. Define the universe of discourse.
3. State the universal quantification.
4. Analyse the existential quantification.
5. Elaborate on the bound and free variables.
6. Interpret the theory of inference for predicate calculus.
7. Explain the consistent and inconsistent.

5.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. When all the variables in a propositional function are assigned values, the resulting statement has a truth value. However, there is another important way to change propositional functions into propositions, called quantification.

NOTES

2. Universe of Discourse: Many mathematical statements assert that a property is true for all values of a variable in a particular domain, called the universe of discourse. Such a statement is expressed using an universal quantification.
3. Universal Quantification: The universal quantification of $p(x)$ is a proposition only when $p(x)$ is true for all values of x in the universe of discourse.
4. Existential Quantification: The existential quantification of $p(x)$ is the proposition 'There exists an element x in the universe of discourse such that $p(x)$ is true.'
5. The variable is said to be bound if it is concerned with either universal ($\forall$) or existential ($\exists$) quantifier and the scope of the variable in the formulae immediately following the quantifier. The variable, which is not concerned with any quantifier, is called free variable.
6. Consider the two statements 'Rohit is Brilliant' and 'Manaswine is Brilliant'. As propositions, there is no relation between them, but they have something in common. Instead of writing two statements we can write a single statement like 'x is brilliant', because both Rohit and Manaswine share the same nature brilliant. By replacing x by any other name we get many propositions. The common feature expressed by 'Is Brilliant' is called predicate. Predicate calculus deals with sentences involving predicates.
7. A set of formulae $S_1, S_2, ...S_m$ is said to be inconsistent if their conjunction implies a contradiction, that is, $S_1 \wedge S_2 \wedge S_3 \wedge ... \wedge S_m \Rightarrow R \wedge \neg R \Rightarrow F$ for same formula R.

   A set of formula $S_1, S_2, ...S_m$ is said to be consistent if it is not inconsistent.

5.6 SUMMARY

- When all the variables in a propositional function are assigned values, the resulting statement has a truth value. However, there is another important way to change propositional functions into propositions, called quantification.
- Universe of Discourse: Many mathematical statements assert that a property is true for all values of a variable in a particular domain, called the universe of discourse. Such a statement is expressed using an universal quantification.
- Universal Quantification: The universal quantification of $p(x)$ is a proposition only when $p(x)$ is true for all values of x in the universe of discourse.
- Existential Quantification: The existential quantification of $p(x)$ is the proposition 'There exists an element x in the universe of discourse such that $p(x)$ is true.'
- The variable is said to be bound if it is concerned with either universal ($\forall$) or existential ($\exists$) quantifier and the scope of the variable in the formulae

NOTES

immediately following the quantifier. The variable, which is not concerned with any quantifier, is called free variable.

- A set of formulae $S_1, S_2, ...S_m$ is said to be inconsistent if their conjunction implies a contradiction, that is, $S_1 \wedge S_2 \wedge S_3 \wedge ... \wedge S_m \Rightarrow R \wedge \neg R \Rightarrow F$ for same formula R.

  A set of formula $S_1, S_2, ...S_m$ is said to be consistent if it is not inconsistent.

5.7 KEY WORDS

- **Quantifiers:** When all the variables in a propositional function are assigned values, the resulting statement has a truth value. However, there is another important way to change propositional function into propositions, called quantification.
- **Universe of discourse:** Many mathematical statements asset that a property is true for all values of a variable in a particular domain, called the universe of discourse.
- **Universal quantification:** The universal quantification of $p(x)$ is a proposition only when $p(x)$ is true for all value of x in the universe of discourse.
- **Bound and free variables:** The variable is said to be bound if it is concerned with either universal ($\forall$) or existential ($\exists$) quantifiers and the scope of the variable in the formulae immediately following the quantifier. The variable, which is not concerned with any quantifiers, is called free variable.

5.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What are quantifiers?
2. Explain the universe of discourse.
3. Define universal quantification.
4. Interpret the bound and free variables.
5. Analyse the theory of inference.
6. State the consistent and inconsistent.

Long-Answer Questions

1. Discuss briefly the quantifiers with their types.
2. Explain the universe of discourse.

3. Analyse the bound and free variables. Give appropriate examples.
4. Elaborate on the theory of inference for predicate calculus.
5. Define consistent and inconsistent.

NOTES

5.9 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 6 RELATIONS

NOTES

Structure

6.0 INTRODUCTION

Relations concept is one of the important topic in mathematics especially in set theory. Whenever sets are being discussed, the relationship between the elements of the sets is the next thing that comes up. Relations may exist between objects of the same set or between objects of two or more sets. A relation is a mathematical tool for describing associations between elements of sets. Relations are widely used in computer science, especially in databases and scheduling applications. A relation can be defined across many items in many sets, but in this text, we will focus on binary relations, which represent an association between two items in one or two sets.

A relation can be represented using a directed graph. The number of vertices in the graph is equal to the number of elements in the set from which the relation has been defined. For each ordered pair (x, y) in the relation R, there will be a directed edge from the vertex 'x' to vertex 'y'. If there is an ordered pair (x, x), there will be self- loop on vertex 'x'. A relation in mathematics defines the relationship between two different sets of information. If two sets are considered, the relation between them will be established if there is a connection between the elements of two or more non-empty sets. In the morning assembly at schools, students are supposed to stand in a queue in ascending order of the heights of all the students. This defines an ordered relation between the students and their heights. Therefore, we can say, 'A set of ordered pairs is defined as a relation.' Sets and relation are interconnected with each other. The relation defines the relation between two given sets. If there are two sets available, then to check if there is any connection between the two sets, we use relations. For example, an empty relation denotes none of the elements in the two sets is same.

NOTES

In Maths, the relation is the relationship between two or more set of values. Suppose, x and y are two sets of ordered pairs. And set x has relation with set y, then the values of set x are called domain whereas the values of set y are called range. There are eight main types of relations which include: Empty Relation, Universal Relation, Identity Relation, Inverse Relation, Reflexive Relation, Symmetric Relation, Transitive Relation, and Equivalence Relation.

An equivalence relation is a binary relation that is reflexive, symmetric and transitive. The relation "Is Equal To" is the canonical example of an equivalence relation.

Each equivalence relation provides a partition of the underlying set into disjoint equivalence classes. Two elements of the given set are equivalent to each other, if and only if they belong to the same equivalence class.

In this unit, you will study about the relations, representation of a relation, operations on relations, and equivalence relation.

6.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the relations
- Analyse the representation of a relation
- Interpret the operations on relations
- Elaborate on the equivalence relation

6.2 RELATIONS AND ORDERING

Let A and B be any two sets. The Cartesian product of A and B is defined as,

$$A \times B = \{(a, b) : a \in A; b \in B\}$$

i.e., the set of all ordered pairs (a_i, b_j) for every $a_i \in A;\ b_j \in B$

For example, $A = \{1, 2\},\ B = \{a, b, c\}$

$$A \times B = \{ (1, a), (2, a), (1, b), (2, b), (1, c), (2, c)\}$$

$B \times A = \{ (a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$ (Clearly $A \times B \neq B \times A$)

Note: We can represent the Cartesian product as a rectangular array having n rows and m columns labelled in order as $a_1, a_2, ..., a_n$ and $b_1, b_2, ..., b_n$, respectively.

6.2.1 Binary Relation

A binary relation R from a set A to a set B is a subset R of the Cartesian product $A \times B$.

NOTES

For example,

1. Let $A = B = N$, the set of natural numbers.

   (*i*) Define the relation R as '='

   Now, $R = \{ (1, 1), (2, 2), (3, 3),....\} \subseteq N \times N$

   $\therefore$ R is a binary relation.

   (*ii*) Define R as '<'

   Then, $R = \{ (1, 2), (2, 3), (3, 4),..... (1, 3), (2, 4), (3, 5), ...\} \subseteq N \times N$

   $\therefore$ R is a binary relation.

2. Let A be the set of all people on earth and $a, b \in A$, and $a\ R\ b$ iff a and b were born on the same day of the same year.

Domain and the Range of a Relation

Let R be a binary relation. The set $D(R)$ of all elements x such that for all y, $(x, y) \in R$ is called the domain of R.

i.e., $D(R) = \{ x : (x, y) \in R$, for all $y\}$

Similarly, $Rg(R)$ of all elements y such that for all x, $(x, y) \in R$ is called the range of R.

i.e., $Rg(R) = \{y : (x, y) \in R$, for all $x\}$

Operations on Relations

Let R and S be relations from a set A to a set B. Now the union and intersection of R and S is defined as,

(*i*) $R \cup S = \{ (a, b) : (a, b) \in R$ or $(a, b) \in S\}$

(*ii*) $R \cap S = \{(a,b) : (a, b) \in R$ and $(a, b) \in S\}$

Example 6.1: Let $X = \{1, 2, 3, 4, 5, 6\}$

Let R and S be relations from X to X as,

$R = \{ (x, y) : (x + y)$ is a multiple of $2\}$

$S = \{ (x, y) : (x + y)$ is a multiple of $3\}$

Find $R \cup S$ and $R \cap S$.

Solution: $R = \{(1, 3), (1, 5)\}$ and $S = \{(2, 4), (1, 5)\}$

$R \cup S = \{(1, 3), (2, 4), (1, 5)\}$

$R \cap S = \{(1, 5)\}$

Inverse of R: Let R be a relation from a set A to set B. The inverse of R is relation from B to A and is given by $R^{-1} = \{(y, x): (x, y) \in R\}$.

6.3 REPRESENTATION OF A RELATION

NOTES

(*i*) A binary relation R from a set A with n elements to a set B with m elements is represented as an $n \times m$ array M_R by marking the positions in M_R. The positions which correspond to the pairs belong to R with 1 and 0 elsewhere.

i.e., $M_R = [a_{ij}]\begin{cases} 1 \text{ if } i\text{th element of } A \text{ is related to } j\text{th element of } B \\ 0, \text{ otherwise.} \end{cases}$

Example 6.2: Let $A = B = X = \{1, 2, 3, 4, 5, 6\}$. Define R as '<' on X.

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

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

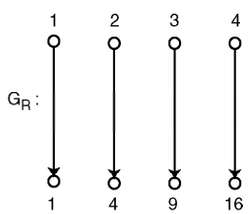
(*ii*) The relation array can be viewed graphically as elements of sets represented by models and an ordered pair is represented by an edge between the vertices that correspond to the paired elements, with an arrow pointing to the second element of the pair.

Example 6.3: Let $A = \{1, 2, 3, 4\}$, $B = \{1, 4, 9, 16\}$ and the relation $R = \{(1, 1), (2, 4), (3, 9), (4, 16)\}$. Draw the relation graph.

Solution: First we shall write the relation matrix M_R.

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

Now we shall draw the relation graph G_R.



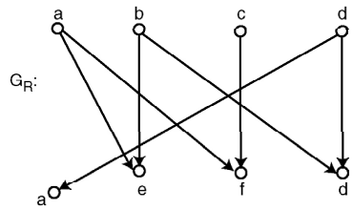
Example 6.4: Let $A = \{a, b, c, d\}$, $B = \{a, e, f, d\}$ and let $R = \{(a, e), (a, f), (b, e), (c, f), (b, d), (d, d), (d, a)\}$. Draw the relation graph.

Solution: First we shall write the relation matrix M_R:

NOTES

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

The relation graph G_R is given as

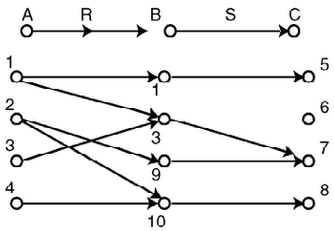


Composition of Two Relations

Let R be a binary relation from the set A to the set B and S be a binary relation from the set B to the set C, then the ordered pair (R, S) is said to be composable. If (R, S) is a composable pair of binary relations, the composite $R \circ S$ and R and S, is a binary relation from the set A to the set C, such that , for $a \in A$ and $c \in C$, $a (R \circ S)c$ if for some $b \in B$, both aRb and bSc are binary relations.

Example 6.5: $A = \{1, 2, 3, 4\}$, $B = \{1, 3, 9, 10\}$ $C = \{5, 6, 7, 8\}$, $R = \{(1, 1), (1, 3), (2, 9), (2, 10), (3, 3), (4, 10)\}$ $S = \{(1, 5), (3, 7), (9, 7), (10, 8)\}$. Find $R \circ S$ and its relation graph.

Solution:

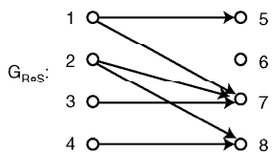


$R \circ S = \{(1, 5), (1, 7), (2, 7), (2, 8), (3, 7), (4, 8)\}$

The corresponding matrix is,

$$M_{R \circ S} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and the corresponding relation graph $G_{R \circ S}$ is,



6.4 EQUIVALENCE RELATIONS AND PARTITION

NOTES

Equivalence Relation

A relation R on a set A is called an equivalence relation if R is reflexive, symmetric and transitive.

For example,

Let N be the set of natural numbers. Define R on N as,

$$R = \{(x,y) : x + y \text{ is even}, x, y \in N\}$$

Proof: Let $x \in N$. Now $x + x = 2x$.

Clearly $2x$ is even. Therefore R is reflexive. Let $x, y \in N$ and $x + y$ is even. Clearly $y + x$ is also even and hence R is symmetric.

Now, if $x + y$ is even and $y + z$ is even then we have to prove that $x + z$ is even.

Since, $x + y$ and $y + z$ are even, both $(x + y)$ and $(y + z)$ are divisible by 2.

$\therefore (x + y) + (y + z)$ is also divisible by 2, i.e., $x + (y + y) + z$ is divisible by 2.

$\therefore (x + z)$ is divisible by 2.

Hence, R is transitive. So, R is an equivalence relation.

Note: From the relation graph or relation matrix, the kind of relation can be identified.

Example 6.6: The relation R on a set is represented by,

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

Is R reflexive, symmetric or antisymmetric?

Solution: In the matrix M_R, the diagonal elements are 1. Therefore, R is reflexive. Since the matrix M_R is symmetric, the relation R is also symmetric.

Example 6.7: The relation R and R_1 on a set is represented by,

$$(i)\ M_R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (ii)\ M_{R_1} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Are the relations R and R_1 reflexive, symmetric, antisymmetric, and/or transitive ?

Solution: The solution is obtained as follows:

(*i*) Since, the matrix M_R is symmetric and its diagonal entries are 1. The relation R is symmetric and reflexive. Since R is not antisymmetric, R is transitive.

NOTES

(*ii*) The relation R_1 is not reflexive.

R_1 is symmetric [$\because$ M_{R_1} is symmetric] and R_1 is transitive.

Example 6.8: Draw the relation graph for the following relations.

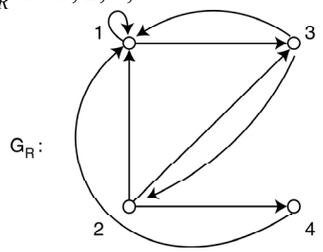
(*i*) $R = \{(1, 1), (1, 3), (2, 1), (2, 3), (2, 4), (3, 1), (3, 2), (4, 1)\}$ on the set $X = \{1, 2, 3, 4\}$.

(*ii*) $R_1 = \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}$ on the set $Y = \{1, 2, 3\}$.

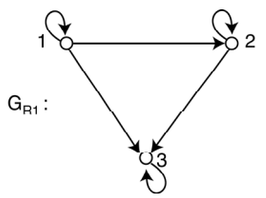
Solution: The relation graph is draws as follows:

(*i*) The relation graph G_R of R is drawn as:

The vertices of G_R are 1, 2, 3, 4.



(*ii*) The relation graph G_{R_1} of R_1 is drawn as:



Example 6.9: Let R be the relation represented by:

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

Find the relation matrices representing (*i*) R^{-1} (*ii*) R^c (*iii*) R^2.

Solution: The solution is obtained as follows:

(*i*) To get the inverse relation matrix $(M_{R^{-1}})$ of a relation matrix (M_R) just write the transpose of M_R.

$$\therefore \quad M_{R^{-1}} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

(*ii*) To find the complement relation matrix, replace 0 by 1 and 1 by 0 in the given relation matrix.

$$\therefore \quad M_{R^c} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

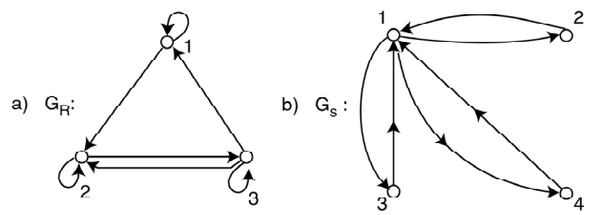
(*iii*) To find the relation matrix of R^2 when $R^2 = R \circ R$.

If the relation matrix M_R is known, then $M_{R^2} = M_R . M_R$ (the matrix multiplication)

NOTES

$$\therefore \quad M_{R^2} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

Example 6.10: Find whether the relations for the directed graphs shown in the following figures are reflexive, symmetric, antisymmetric and/or transitive.



Solution: The solution is obtained as follows:

(*i*) In G_R, there are loops at every vertex of the relation graph and hence it is reflexive.

It is neither symmetric nor antisymmetric since there is an edge between 1 and 2 but not from 2 to 1, but there are edges connecting 2 and 3 in both directions.

Moreover, the relation is not transitive, since there is an edge from 1 to 2 and 2 to 3, but no edge from 1 to 3.

(*ii*) Since loops are not present in G_S, this relation is not reflexive. Further, it is symmetric and not antisymmetric.

Moreover, the relation is not transitive.

Equivalence Class

Let R be an equivalence relation on a set A. Let $x \in A$. The equivalence class a is given by,

$$[a]_R = \{x \in A: (a, x) \in R\}$$

Note: $[a]_R \neq \phi$, because $a \in [a]$.

Example 6.11: Prove that any two equivalence classes are identical or disjoint.

Solution: First we shall prove that $(a,b) \in R$. This implies that $[a]_R = [b]_R$

Suppose $(a, b) \in R$

Case I: $[a] = [b]$

Let $x \in [a] \Rightarrow (x, a) \in R$

$\Rightarrow (x,b) \in R$ [$\because$ $(x,a) \in R$ and $(a,b) \in R$ and R is transitive]

NOTES

$\Rightarrow x \in [b]$

$\Rightarrow [a] = [b]$

$\therefore \quad [a] = [b]$

Now suppose $[a]$, $[b]$ are two equivalence classes.

Case II: $[a] = [b]$ or $[a] \cap [b] = \phi$

If $[a] \cap [b] = \phi$ then nothing to prove.

Suppose $[a] \cap [b] \neq \phi$, then $x \in [a] \cap [b]$

$\Rightarrow x \in [a]$ and $x \in [b]$

$\Rightarrow (x, a) \in R$ and $(x, b) \in R$

$\Rightarrow [x] = [a]$ and $[x] = [b]$

$\Rightarrow [a] = [b]$

$\therefore \quad [a] \cap [b] = \phi$ or $[a] = [b]$

i.e., any two equivalence classes are identical or disjoint.

Example 6.12: Prove that an equivalence relation induces a partition and a partition induces an equivalence relation.

Solution: Let $\{A_i : i \in Z\}$ is a partition of a set A. Define a relation R on A by $(a,b) \in R$ if $a, b \in A_i$ for some i.

Case I: R is an equivalence relation on A.

Let, $a \in A$

$\Rightarrow a \in A_i$ for some i

$\Rightarrow a, a \in A_i$ for some i

$\Rightarrow (a, a) \in R$.

$\therefore R$ is a reflexive relation on A.

Suppose $(a, b) \in R$, then by the definition of R,

$a, b \in A_i$ for some i

$\therefore b, a \in A_i$ for some i

$\Rightarrow (b, a) \in R$.

$\therefore \quad R$ is a symmetric relation on A.

Suppose $(a, b) \in R$ and $(b, c) \in R$, then $a, b \in A_i$ and $b, c \in A_j$ for some i and j.

Here, $b \in A_i$ and $b \in A_j$

$\therefore A_i \cap A_j \neq \phi \Rightarrow A_i = A_j$, otherwise $\{A_i\}_{i \in I}$ is not a partition and hence $a, b, c \in A_i$

$\therefore (a, c) \in R$

$\therefore R$ is a transitive relation on A.

R is also an equivalence relation on A.

Further, we can also show that $A_i = [a]_{a \in A}$

Conversely, we can assume that R is an equivalence relation on set A.

Case II: R induces a partition for A.

Let, $x \in A$, $[x] = \{y \in A / (y, x) \in R\}$ and for any $x, y \in A$, we have

$$[x] \cap [y] = \phi \text{ or } [x] = [y]$$

$\therefore A = \cup_{x \in A} [x]$

i.e., $\{[x] : x \in A\}$ is a partition for A.

Check Your Progress

1. What do you understand by the relations?
2. Explain the binary relation.
3. Define the domain and the range of a relation.
4. Interpret the operations on relations.
5. Elaborate on the representation of a relation.
6. Analyse the composition of two relations.
7. State the equivalence relation.
8. Define the equivalence class.

6.5 GRAPHS OF RELATIONS

We know that a relation is a subset of Cartesian product of two sets. A relation shows relationship of a member of one set to that of another set. Thus, a relationship is shown as an ordered pair and is also called binary relation. If we recall the basic concept of a coordinate plane, also called Cartesian plane, we know that it is constituted by choosing two number lines, intersecting at right angles to each other. One line is horizontal, usually called x-axis and another as y-axis. The intersecting point of these two lines is the origin. When only one number line is used, every point on this number line represents a real number. But when we take two lines at right angles to each other it represents a point in the plain containing an ordered pair. Thus, every point on this plane shows a relation. Thus, relationship can be shown as graphs. A domain with a binary relation can be viewed as *vertices* with *edges* connecting them. Thus, any binary relation can be shown as graph, by

NOTES

taking domain elements as vertices and showing them as dots, with arrows as edges between related elements. Vertices can also represent tasks and edges connecting those showing dependencies.

We can make a graph of any relation. For example, we draw the graph of the relation,
$R = \{(2, 5), (4, 3), (6, 1), (2, 7)\}$. The graph is shown in Figure 6.1.

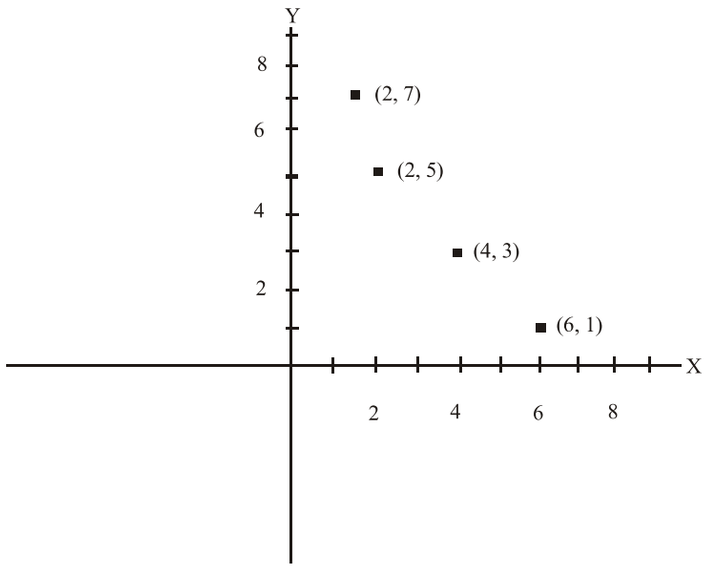


Fig. 6.1 *Graph Showing Relation*

$R = \{(2, 5), (4, 3), (6, 1), (2, 7)\}$

A function is also a relation although, all relations are not functions. So, we can call any function a relation too. We now draw the graph of the relation $2x + 3y = 6$. We must have at least two points. If $x = 0$, $y = 2$ and if $y = 0$, $x = 3$. Thus, graph can be sketched by taking points (0, 2) and (3, 0) and connecting these points to find a straight line. The graph has been plotted in the Figure 6.2.

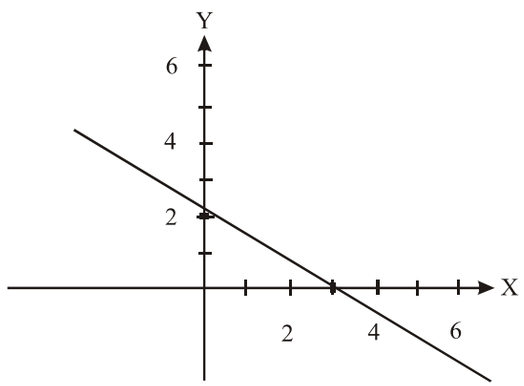
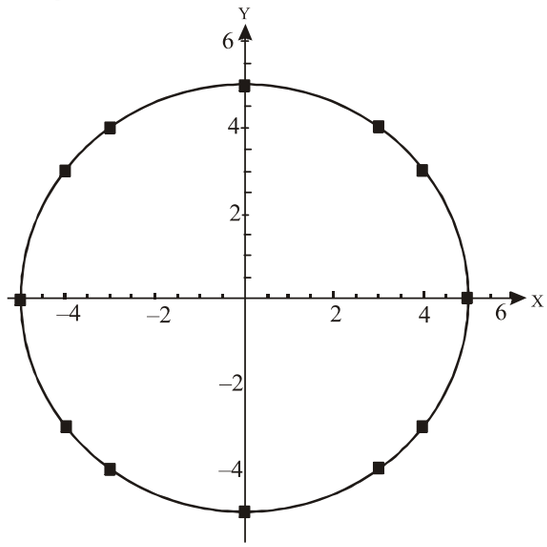


Fig. 6.2 *Graph of Relation 2x + 3y = 6*

NOTES

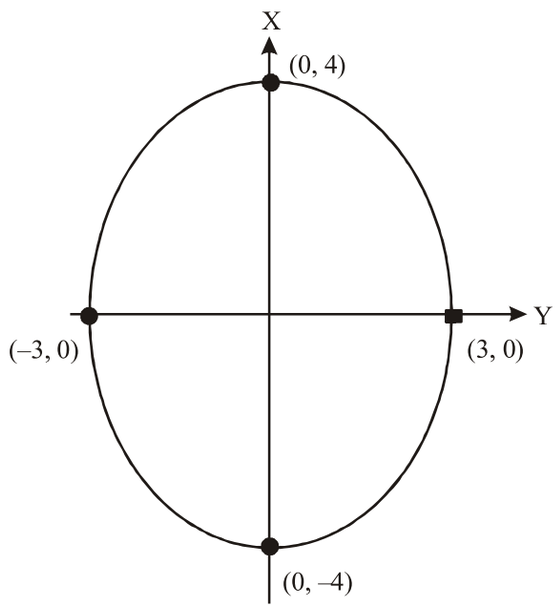
Example 6.13: Draw the graph of the relation $x^2 + y^2 = 25$.

Solution: To sketch the graph we find different points by arbitrarily taking values of x and finding corresponding values of y. Points are: (5, 0), (4, 3), (3, 4), (0, 5), (–3, 4), (–4, 3), (–5, 0), (–4, –3), (–3,–4), (0,–5), (3,–4), (4,–3). We plot this line on the plane and join these.



Example 6.14: Draw the graph of relation $16x^2+9y^2 = 144$ and find the intercepts on axes.

Solution: By equating $y = 0$, we get x-intercepts and by putting $y = 0$, we get y-intercepts. We note different values of y corresponding to some values of x and draw a curve. This shows an ellipse. The x-intercepts are (3, 0) and (–3, 0). The y-intercepts are (4, 0) and (–4, 0). The curve is shown below.



6.6 PROPERTIES OF RELATIONS

NOTES

We have already known what binary relations are. A binary relation has certain properties which apply only to relations on Cartesian product of a single set, i.e., in $A \times A$.

Properties of a Binary Relation

Let R be a relation on a set A (i.e., $R \subseteq A \times A$). R is called:

(*i*) Reflexive: If aRa, $\forall\, a \in A$

(*ii*) Symmetric: If aRb then bRa, $\forall a,b \in A$

(*iii*) Transitive: If aRb and bRc then aRc, $\forall a,b,c \in A$

(*iv*) Irreflexive: If $a \not R a$, and $\forall a \in A$

(*v*) Antisymmetric: If aRb then $b \not R a$, and $a = b$ for $a, b \in A$

(*vi*) Connected: A relation R in A is connected iff for every two *distinct* elements x and y in A, such that $(x,y) \in R$ or $(y, x) \in R$ (or both).

Also, based on these properties, such as reflexivity, symmetricity, transitivity, etc. in combination defines certain classes of relations such as equivalence, tolerance or ordering. Examples of relations that are reflexive are '=' and '' on the set **N** of natural numbers and relations '$\supseteq$' and '$\subseteq$' between sets are reflexive. Relations '>' and '<' on **N** are irreflexive.

Example of the symmetric relation is the relation '*brother of*'. It is symmetric, but not in all cases. It is nonsymmetrical when the set comprises all people. Let us consider a set A = {John, Peter, Bill}. If John is brother of Bill then Bill is also brother of John. So it is symmetric. But if set contains a female named Mary, then it is not symmetric. It is asymmetric. If John is brother of Mary, Mary is not the brother of John. Examples of transitivity relations are =, > and < are transitive in the set of natural numbers.

Partitions: In a non-empty set A, a partition of A is a collection of non-empty subsets of A such that (*i*) for any two distinct subsets X and Y, $X \cap Y = \phi$ and (*ii*) the union of all the subsets in collection equals A. The subsets of A that are members of a partition of A are called *cells* of that partition. There is a close correspondence between partitions and equivalence relations. Given a partition of set A, the relation R = {(x,y): x and y are in the same cell of the partition of A} is an equivalence relation in A. Conversely, given an equivalence relation R in A, there exists a partition of A in which x and y are in the same cell iff $(x,y) \in R$.

Tolerance: A relation R in $A \times A$ is called a *tolerance* or a *tolerance relation* if it is reflexive and symmetric. Thus, tolerance is weaker than equivalence and it need not to be transitive. The notion of tolerance relation is an explication of *similarity* or *closeness*. Relations '*neighbour of*', '*friend of*' can be considered

as examples if we hold that every person is a neighbour and a friend to him(her) self.

Analogous to equivalence classes and partitions, there are tolerance classes and coverings. A set $B \subseteq A$ is called a tolerance *preclass* if it holds that for all x, $y \in B$, x and y are tolerant, i.e., $(x,y) \in R$.

NOTES

Orderings

An order is a binary relation which is essentially transitive and further it can either be (*i*) reflexive and antisymmetric or (*ii*) irreflexive and asymmetric.

For examples, relations $\geq$ and $=$ on the set N of natural numbers are examples of weak order, as are relations $\supseteq$ and $=$ on subsets of any set. The relations $>$ and $\supset$ are examples of strict orders on the corresponding sets. The relations $\geq$ and $>$ are linear orders. Partial ordering sets and well ordered sets also show ordering properties.

Check Your Progress

9. Explain the graphs of a relations.
10. Describe the properties of a binary relation.
11. What do you mean by the tolerance relation?
12. Define the orderings.

6.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. Let A and B be any two sets. The Cartesian product of A and B is defined as,

   $$A \times B = \{(a, b) : a \in A; b \in B\}$$

   i.e., the set of all ordered pairs (a_i, b_j) for every $a_i \in A$; $b_j \in B$

2. A binary relation R from a set A to a set B is a subset R of the Cartesian product $A \times B$.

   For example,

   Let $A = B = N$, the set of natural numbers.

3. Let R be a binary relation. The set $D(R)$ of all elements x such that for all y, $(x, y) \in R$ is called the domain of R.

   i.e., $D(R) = \{ x : (x, y) \in R$, for all $y\}$

   Similarly, $Rg(R)$ of all elements y such that for all x, $(x, y) \in R$ is called the range of R.

   i.e., $Rg(R) = \{y : (x, y) \in R$, for all $x\}$

NOTES

4. Let R and S be relations from a set A to a set B. Now the union and intersection of R and S is defined as,

   (*i*) $R \cup S = \{ (a, b) : (a, b) \in R \text{ or } (a, b) \in S\}$

   (*ii*) $R \cap S = \{(a,b) : (a, b) \in R \text{ and } (a, b) \in S\}$

5. A binary relation R from a set A with n elements to a set B with m elements is represented as an $n \times m$ array M_R by marking the positions in M_R. The positions which correspond to the pairs belong to R with 1 and 0 elsewhere.

   i.e., $M_R = [a_{ij}]\begin{cases} 1 \text{ if } i\text{th element of } A \text{ is related to } j\text{th element of } B \\ 0, \text{ otherwise.} \end{cases}$

6. Let R be a binary relation from the set A to the set B and S be a binary relation from the set B to the set C, then the ordered pair (R, S) is said to be composable. If (R, S) is a composable pair of binary relations, the composite $R\ O\ S$ and R and S, is a binary relation from the set A to the set C, such that, for $a \in A$ and $c \in C$, $a\ (R\ O\ S)c$ if for some $b \in B$, both aRb and bSc are binary relations.

7. A relation R on a set A is called an equivalence relation if R is reflexive, symmetric and transitive.

8. Let R be an equivalence relation on a set A. Let $x \in A$. The equivalence class a is given by,

   $$[a]_R = \{x \in A: (a, x) \in R\}$$

   Note: $[a]_R \neq \phi$, because $a \in [a]$.

9. A relation shows relationship of a member of one set to that of another set. Thus, a relationship is shown as an ordered pair and is also called binary relation. If we recall the basic concept of a coordinate plane, also called Cartesian plane, we know that it is constituted by choosing two number lines, intersecting at right angles to each other.

10. Let R be a relation on a set A (i.e., $R \subseteq A \times A$). R is called:

    (*i*) Reflexive: If aRa, $\forall\ a \in A$

    (*ii*) Symmetric: If aRb then bRa, $\forall a,b \in A$

    (*iii*) Transitive: If aRb and bRc then aRc, $\forall a,b,c \in A$

    (*iv*) Irreflexive: If $a \not R a$, and $\forall a \in A$

    (*v*) Antisymmetric: If aRb then $b \not R a$, and $a = b$ for $a, b \in A$

    (*vi*) Connected: A relation R in A is connected iff for every two distinct elements x and y in A, such that $(x,y) \in R$ or $(y, x) \in R$ (or both).

11. Tolerance: A relation R in $A \times A$ is called a tolerance or a tolerance relation if it is reflexive and symmetric. Thus, tolerance is weaker than equivalence

and it need not to be transitive. The notion of tolerance relation is an explication of similarity or closeness.

12. An order is a binary relation which is essentially transitive and further it can either be (*i*) reflexive and antisymmetric or (*ii*) irreflexive and asymmetric.

NOTES

6.8 SUMMARY

- Let A and B be any two sets. The Cartesian product of A and B is defined as,

  $$A \times B = \{(a, b) : a \in A; b \in B\}$$

  i.e., the set of all ordered pairs (a_i, b_j) for every $a_i \in A$; $b_j \in B$
- A binary relation R from a set A to a set B is a subset R of the Cartesian product $A \times B$.

  For example,

  Let $A = B = N$, the set of natural numbers.
- Let R be a binary relation. The set $D(R)$ of all elements x such that for all y, $(x, y) \in R$ is called the domain of R.

  i.e., $D(R) = \{ x : (x, y) \in R$, for all $y\}$
- Similarly, $Rg(R)$ of all elements y such that for all x, $(x, y) \in R$ is called the range of R.

  i.e., $Rg(R) = \{y : (x, y) \in R$, for all $x\}$
- Let R and S be relations from a set A to a set B. Now the union and intersection of R and S is defined as,

  (*i*) $R \cup S = \{ (a, b) : (a, b) \in R$ or $(a, b) \in S\}$

  (*ii*) $R \cap S = \{(a,b) : (a, b) \in R$ and $(a, b) \in S\}$
- A binary relation R from a set A with n elements to a set B with m elements is represented as an $n \times m$ array M_R by marking the positions in M_R. The positions which correspond to the pairs belong to R with 1 and 0 elsewhere.

  i.e., $M_R = [a_{ij}]$ $\begin{cases} 1 \text{ if } i\text{th element of } A \text{ is related to } j\text{th element of } B \\ 0, \text{ otherwise.} \end{cases}$
- Let R be a binary relation from the set A to the set B and S be a binary relation from the set B to the set C, then the ordered pair (R, S) is said to be composable. If (R, S) is a composable pair of binary relations, the composite $R\,O\,S$ and R and S, is a binary relation from the set A to the set C, such that, for $a \in A$ and $c \in C$, $a\,(R\,O\,S)c$ if for some $b \in B$, both aRb and bSc are binary relations.

NOTES

- A relation R on a set A is called an equivalence relation if R is reflexive, symmetric and transitive.
- Let R be an equivalence relation on a set A. Let $x \in A$. The equivalence class a is given by,

$$[a]_R = \{x \in A: (a, x) \in R\}$$

Note: $[a]_R \neq \phi$, because $a \in [a]$.

- A relation shows relationship of a member of one set to that of another set. Thus, a relationship is shown as an ordered pair and is also called binary relation. If we recall the basic concept of a coordinate plane, also called Cartesian plane, we know that it is constituted by choosing two number lines, intersecting at right angles to each other.
- Tolerance: A relation R in A × A is called a tolerance or a tolerance relation if it is reflexive and symmetric. Thus, tolerance is weaker than equivalence and it need not to be transitive. The notion of tolerance relation is an explication of similarity or closeness.
- An order is a binary relation which is essentially transitive and further it can either be (i) reflexive and antisymmetric or (ii) irreflexive and asymmetric.

6.9 KEY WORDS

- **Binary relation:** A binary relation R from a set A to a set B is a subset R of the Cartesian product $A \times B$.
- **Domain:** Let R be a binary relation. The set $D(R)$ of all elements x such that for all y, $(x,y) \in R$ is called the domain of R.
- **Range:** $Rg(R)$ of all elements y such that for all x, $(x,y) \in R$ is called the range of R.
- **Representation of a relation:** A binary relation R from a set A with n elements is represented as an $n \times m$ array M_R by marking the positions in M_R.
- **Equivalence relation:** A relation R on a set A is called an equivalence relation if R is reflective, symmetric, and transitive.
- **Equivalence class:** Let R be an equivalence relation on a set A. Let $x \in A$. The equivalence class a is given by,

$[a]_R = \{x \in A: (a,x) \in R\}$

- **Tolerance:** A relation R in $A \times A$ is called a tolerance or a tolerance relation if it is reflexive and symmetric.

6.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. Explain the binary relation.
2. Define the domain and range of a relation.
3. Interpret the operations on the relations.
4. Elaborate on the representation of a relation.
5. Analyse the composition of two relations.
6. State the equivalence relation.
7. Explain the equivalence class.
8. Illustrate the properties of relations.
9. Define the tolerance relation.
10. What do you understand by the orderings of relation?

Long-Answer Questions

1. Discuss briefly the binary relation. Give appropriate examples.
2. Define the domain and range of a relation.
3. Analyse the representation of a relation.
4. Illustrate the composition of two relations.
5. Describe the equivalence relation with the help of example.
6. Elaborate on the graphs of relations.
7. Explain the properties of relations.
8. State the relation if any, between sets A and B in the following:
   (*i*) $A = \{1, 3, 5, 7, 9, \ldots\}$
   $B = \{3, 9, 15, 21, \ldots, 3(2n - 1), \ldots\}$
   (*ii*) $A = \{2, 4, 7, 12, 18, 24\}$
   $B = \{1, 3, 7, 11, 16, 22, 29\}$
   (*iii*) $A = \{x : x$ is an even natural number less than 20$\}$
   $B = \{x : x$ is natural number less than 20 which is divisible by 2$\}$
   (*iv*) $A = \{x : x$ is an even integer$\}$
   $B = \{x : x$ is an integer divisible by 3$\}$

6.11 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

NOTES

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 7 LATTICES

NOTES

Structure

7.0 INTRODUCTION

A lattice is an abstract structure studied in the mathematical subdisciplines of order theory and abstract algebra. It consists of a partially ordered set in which every two elements have a unique supremum (also called a least upper bound or join) and a unique infimum (also called a greatest lower bound or meet). An example is given by the natural numbers, partially ordered by divisibility, for which the unique supremum is the least common multiple and the unique infimum is the greatest common divisor.

Lattices can also be characterized as algebraic structures satisfying certain axiomatic identities. Since the two definitions are equivalent, lattice theory draws on both order theory and universal algebra. Semilattices include lattices, which in turn include Heyting and Boolean algebras. These "Lattice-Like" structures all admit order-theoretic as well as algebraic descriptions.

If $(L, \leq)$ is a partially ordered set (poset), and $S \subseteq L$ is an arbitrary subset, then an element $u \in L$ is said to be an upper bound of S if $s \leq u$ for each $s \in S$. A

NOTES

set may have many upper bounds, or none at all. An upper bound u of S is said to be its least upper bound, or join, or supremum, if $u \leq x$ for each upper bound x of S. A set need not have a least upper bound, but it cannot have more than one. Dually, $l \in L$ is said to be a lower bound of S if $l \leq s$ for each $s \in S$. A lower bound l of S is said to be its greatest lower bound, or meet, or infimum, if $x \leq l$ for each lower bound x of S. A set may have many lower bounds, or none at all, but can have at most one greatest lower bound.

A lattice is modular if and only if it doesn't have a sublattice isomorphic to. Besides distributive lattices, examples of modular lattices are the lattice of two-sided ideals of a ring, the lattice of submodules of a module, and the lattice of normal subgroups of a group. The set of first-order terms with the ordering "Is More Specific Than" is a non-modular lattice used in automated reasoning.

Boolean algebra is the branch of algebra in which the values of the variables are the truth values true and false, usually denoted 1 and 0, respectively. Instead of elementary algebra, where the values of the variables are numbers and the prime operations are addition and multiplication, the main operations of Boolean algebra are the conjunction (AND) denoted as $\wedge$, the disjunction (OR) denoted as ($\vee$, and the negation (NOT) denoted as $\neg$. It is thus a formalism for describing logical operations, in the same way that elementary algebra describes numerical operations.

In this unit, you will study about the lattice, some properties of lattices, new lattices, modular and distributive lattices, Boolean algebra, and Boolean polynomials.

7.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the lattices
- Explain some properties of lattices and new lattices
- Illustrate the modular and distributive lattices
- Interpret the Boolean algebra
- Comprehend the Boolean polynomials

7.2 LATTICE

Lattice: A poset in which every pair of elements have both a least upper bound and a greatest lower bound is called a lattice.

For example,

(*i*) The poset $(\{1, 2, 4, 8\}, 1)$ is a lattice.

(*ii*) The poset $(P(S), \subseteq)$ is a lattice.

Here $A \cup B$ = Least upper bound (LUB) of A and B and $A \cap B$ = Greatest lower bound (GLB) of A and B, for any $A \subseteq S, B \subseteq S$.

7.2.1 Properties of Lattice

NOTES

A lattice is a partially ordered set $(L, \leq)$ in which every pair of elements $a, b \in L$ has a Greatest Lower Bound (GLB) and a Least Upper Bound (LUB).

The Greatest Lower Bound (GLB) of a subset $\{a, b\} \subseteq L$ will be denoted by $a \wedge b$ and the Least Upper Bound (LUB) by $a \vee b$. So GLB $\{a, b\} = a \wedge b$, called the meet of a and b and LUB $\{a, b\} = a \vee b$, called the join of a and b.

Note that $\wedge$ and $\vee$ are binary operations and we denote the lattice by $(\wedge, \vee)$. From the definition of $\wedge$ and $\vee$, the following is denoted:

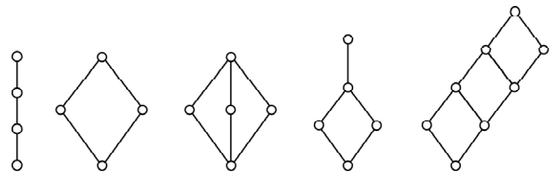
(*i*) $a \leq a \vee b$; $b \leq a \vee b$ (i.e., $a \vee b$ is an UB of a and b).

(*ii*) $a \wedge b \leq a$; $a \wedge b \leq b$ (i.e., $a \wedge b$ is a LB of a and b).

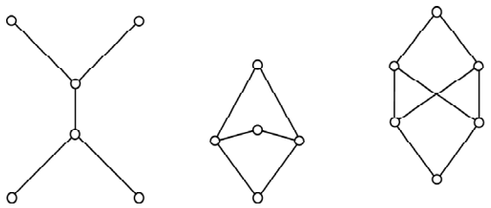
(*iii*) If $a \leq c$ and $b \leq c$ then $a \vee b \leq c$ (i.e., $a \vee b$ is the LUB of a and b).

(*iv*) If $c \leq a$ and $c \leq b$ then $c \leq a \wedge b$. (i.e., $a \wedge b$ is the GLB of a and b).

For example, the following are Lattices:



For example, the following are Posets but not Lattices:



For example, let A be any set and $L = P(A)$ be its power set. The poset $(L, \subseteq)$ is a lattice in which for any $x, y, \in L$, $x \wedge y = x \cap y$ and $x \vee y = x \cup y$.
Similarly let I be the set of positive integers. For any $x, y \in I, x \leq y$ if x/y. Define $x \vee y$ = LCM (x, y) and $x \wedge y$ = GCD $\in (x, y)$. Then $(I, \wedge, \vee)$ is a lattice.

Theorem 7.1: Let $(L, \leq)$ be a lattice in which $\wedge$ and $\vee$ denote the operations of meet and join respectively. For any $a, b, c \in L$, we have

(*i*) $a \wedge a = a$; $a \vee a = a$ (Idempotent)

(*ii*) $a \wedge b = b \wedge a$; $a \vee b = b \vee a$ (Commutative)

(*iii*) $(a \wedge b) \wedge c = a \wedge (b \cap c)$; $(a \vee b) \vee c = a \vee (b \vee c)$ (Associative)

(*iv*) $a \wedge (a \vee b) = a$; $a \vee (a \wedge b) = a$ (Absorption)

NOTES

Proof: Following are the proof of above mentioned statements:

(*i*) Since $a \leq a$, we know that a is a lower bound of $\{a, a\} = \{a\}$. If b is also a lower bound and if $a \leq b$ then we have $b \leq a$ and $a \leq b$. By antisymmetry, $a = b$. So a is the GLB of $\{a\}$. Therefore, $a \wedge a = a$. Dually, $a \vee a = a$ follows.

(*ii*) Let $x = a \wedge b = \text{GLB } \{a, b\}$. Since $\{a, b\} = \{b, a\}$, GLB $\{b, a\} = x$. So $b \wedge a = x$. Hence $x = a \wedge b = b \wedge a$. Dually, $a \vee b = b \vee a$ follows.

(*iii*) Let $x = a \wedge (b \wedge c)$ and $y = (a \wedge b) \wedge c$.

Now $x = a \wedge (b \wedge c) \Rightarrow x \leq a,\ x \leq b \wedge c$

$\Rightarrow x \leq a, x \leq b, x \leq c$

$\Rightarrow x \leq a \wedge b, x \leq c$

$\Rightarrow x \leq (a \wedge b) \wedge c = y.$

Similarly, $y \leq x$ follows. By antisymmetry, $x = y$ and hence, $a \wedge (b \wedge c) = (a \wedge b) \wedge c$.

Dually, $a \vee (b \vee c) = (a \vee b) \vee c$ follows.

(*iv*) By definition, for any $a \in L$, $a \leq a$ and $a \leq a \vee b$

Therefore, $a \leq a \wedge (a \vee b)$. But $a \wedge (a \vee b) \leq a$. Hence $a \wedge (a \vee b) = a$. Dually, $a \vee (a \wedge b) = a$ follows.

Theorem 7.2: Let $(L, \leq)$ be a lattice in which $\wedge$ and $\vee$ denote the operations of meet and join, respectively. For any $a, b \in L$,

$$a \leq b \quad \Leftrightarrow a \wedge b = a$$
$$\Leftrightarrow a \vee b = b$$

Proof: Assume that $a \leq a$. Since $a \leq b$, it follows that $a \leq a \wedge b$. But by definition of $\wedge$, $a \wedge b \leq a$. Therefore $a \wedge b = a$. Conversely, suppose $a \wedge b = a$. Then $a \leq b$. Hence $a \leq b \Leftrightarrow a \wedge b = a$. Similarly $a \leq b \Leftrightarrow a \vee b = b$ follows.

Isotonicity law

Theorem 7.3: Let $(L, \leq)$ be a lattice in which $\wedge$ and $\vee$ denote the operations of meet and join, respectively. For any $a, b, c, \in L$,

$$b \leq c \Rightarrow \begin{cases} a \wedge b \leq a \wedge c \\ a \vee b \leq a \vee c \end{cases}$$

Proof: Assume that $b \leq c$. Since $a \wedge b \leq b$, by transitivity, $a \wedge b \leq c$. Since $a \wedge b \leq a$, it follows that,

$$a \wedge b \leq a \wedge c$$

Now, $b \leq c$ and $c \leq a \vee c$ implies $b \leq a \vee c$. But $a \leq a \vee c$. Hence $a \vee b \leq a \vee c$.

Note: For any $a, b, c \in L$, by Isotonicity law,

$$a \leq b \wedge a \leq c \Rightarrow a \leq b \vee c$$

$a \leq b \wedge a \leq c \Rightarrow a \leq b \vee c$

$c \leq b \wedge a \leq a \Rightarrow b \wedge c \leq a$

$c \leq b \wedge a \leq a \Rightarrow b \vee c \leq a.$

NOTES

Distributive Inequality

Theorem 7.4: Let $(L, \leq)$ be a lattice. For any $a, b, c, \in L$, the following inequalities are hold.

(*i*) $a \vee (b \wedge c) \leq (a \vee b) \wedge (a \vee c)$

(*ii*) $(a \wedge b) \vee (a \wedge c) \leq a \wedge (b \vee c)$

Since $a \leq a \vee b$ and $a \leq a \vee c$, we have,

$$a \leq (a \vee b) \wedge (a \vee c) \quad \ldots (7.1)$$

Since $b \wedge c \leq b \leq a \vee b$ and $b \wedge c \leq c \leq a \vee c$,

$$b \wedge c \leq (a \vee b) \wedge (a \vee c) \quad \ldots (7.2)$$

From Equations (7.1) and (7.2) we have,

$a \vee (b \wedge c) \leq (a \vee b) \wedge (a \vee c)$.

Similarly, Case (ii) follows.

Modular Inequality

Theorem 7.5: Let $(L, \leq)$ be a lattice. For any $a, b, c \in L$,

$a \leq c \Leftrightarrow a \vee (b \wedge c) \leq (a \vee b) \wedge c$

Proof: Suppose $a \leq c$. Then $a \vee c = c$.

By distributive inequality, $a \vee (b \wedge c) \leq (a \vee b) \wedge (a \vee c)$

Since $a \vee c = c$,

$$a \vee (b \wedge c) \leq (a \vee b) \wedge c.$$

Conversely, let us assume that $a \vee (b \wedge c) \leq (a \vee b) \wedge c$.

Since,

$$\begin{aligned} a &\leq a \vee (b \wedge c) \\ &\leq (a \vee b) \wedge c \\ &\leq c \end{aligned}$$

We get $a \leq c$. Hence proved.

Example 7.1: Prove that in a lattice $(L, \leq)$, for any $a, b, c \in L$, if $a \leq b \leq c \Rightarrow a \vee b = bc$, and $(a \wedge b) \vee (b \wedge c) = b = (a \vee b) \wedge (a \vee c)$.

Solution: Since $a \leq b$ and $a \leq c$, $a \leq b \wedge c$. Again $b \leq b$ and $b \leq c$ implies $b \leq b \wedge c$.

Now $a \leq b \wedge c$ and $b \leq b \wedge c \Rightarrow a \vee b \leq b \wedge c$ ... (1)

Again, $b \wedge c \leq b \leq a \vee b$ ... (2)

From Equations (1) and (2), $a \vee b = b \wedge c$.

NOTES

Hence, $a \wedge b \leq b$ and $b \wedge c \leq b$, we get $(a \wedge b) \vee (b \wedge c) \leq b$. Since $b \leq c$ and $b \leq b$ implies $b \leq b \wedge c$. Again this implies $b \leq (b \wedge c) \vee (a \wedge b)$. Hence, $(a \wedge b) \vee (b \wedge c) = b$. Similarly, $(a \vee b) \wedge (a \vee c) = b$ follows.

Example 7.2: Prove that in a lattice $(L, \leq)$, for any $a, b, c, d, \in L$, if $a \leq b$ and $c \leq d$ then $a \wedge c \leq b \wedge d$.

Solution: Since $a \wedge c \leq a \leq b$ and $a \wedge c \leq c \leq d$, $a \wedge c \leq b \wedge d$.

7.2.2 New Lattice

A **lattice** is defined as an abstract structure which is typically studied in the mathematical subdisciplines of order theory and abstract algebra. It consists of a **partially ordered set (poset)** in which every two elements have a unique **supremum**, also termed as **Least Upper Bound (LUB)** or join, and a unique infimum, also termed as **Greatest Lower Bound (GUB)** or meet.

A new lattice means creation of a lattice as a special kind of an ordered set with two binary operations.

Definition: A lattice is a partially ordered set (L, ≤) in which every subset $\{a, b\}$ consisting of two elements has a least upper bound and a greatest lower bound.

The LUB ({a, b}) is denoted by $a \vee b$ and is termed as join or sum of a and b. In the same way, GLB ($\{a, b\}$) is denoted by $a \wedge b$ and is termed as meet or product of a and b.

Consequently, 'Lattice' is a mathematical structure with two binary operations, join and meet. A totally ordered set is evidently a lattice but not all partially ordered sets are lattices.

We generate new propositions from existing ones by means of well-formed formulations.

7.2.3 Distributive Lattice

A Lattice $(L, \leq)$ is said to be distributive lattice if for any $a, b, c, \in L$,

$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

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

Theorem 7.6: Let $a, b, c \in L$, where $(L, \leq)$ is a distributive lattice. Then $a \vee b = a \vee c$ and $a \wedge b = a \wedge c \Rightarrow b = c$.

Proof: We know that,

$b = b \vee (b \wedge a)$ (Absorption)

$= b \vee (a \wedge b)$ (Commutative)

$= b \vee (a \wedge c)$ ($\because a \wedge b = a \wedge c$)

$= (b \vee a) \wedge (b \vee c)$ (Distributive)

$= (a \vee b) \wedge (c \vee b)$ (Commutative)

NOTES

$= (a \vee c) \wedge (c \vee b)$ $(\because\ a \vee b = a \vee c)$

$= (c \vee a) \wedge (c \vee b)$ (Commutative)

$= c \vee (a \wedge b)$ (Distributive)

$= c \vee (a \wedge c)$ $(\because\ a \wedge b = a \wedge c)$

$= c \vee (c \wedge a)$ (Commutative)

$= c$ (Absorption)

Hence proved.

Modular Lattice: A Lattice $(L, \leq)$ is said to be modular lattice if

$a \leq c \Rightarrow a \vee (b \wedge c) = (a \vee b) \wedge c$.

Bounded Lattice: A Lattice $(L, \leq)$ which has both, a least element denoted by 0, and the greatest element denoted by 1 is called a bounded lattice.

Note: If $L = \{a_1, a_2, ..., a_n\}$ with $\overset{n}{\underset{i=1}{\wedge}} a_i = 0$ and $\overset{n}{\underset{i=1}{\vee}} a_i = 1$. It satisfies $a \vee 0 = a$, $a \vee 1 = 1$, $a \wedge 1 = a$ and $a \wedge 0 = 0$.

Complement of an Element: In a bounded Lattices $(L, \leq)$, an element $b \in L$ is called a complement of an element $a \in L$ if $a \wedge b = 0$ and $a \vee b = 1$, we denote b by a.

Complement Lattice: A Lattice $(L, \leq)$ is said to be complemented lattice if every element of L has at least one complement.

Example 7.3: Show that De Morgan's laws hold in a complemented distributive lattice.

Solution: To process that $(a \wedge b)' = a' \vee b'$ and $(a \vee b)' = (a' \wedge b')$, consider

$$
\begin{aligned}
(a \wedge b)' \wedge (a' \vee b') &= \big((a \wedge b) \wedge a'\big) \vee \big((a \wedge b) \wedge b'\big) && \text{(Distributive)} \\
&= \big((b \wedge a) \wedge a'\big) \vee \big((a \wedge b \wedge \wedge b'\big) && \text{(Commutative)} \\
&= \big(b \wedge (a \wedge a')\big) \vee \big(a \wedge (b \wedge b')\big) && \text{(Associative)} \\
&= (b \wedge 0) \vee (a \wedge 0) \\
&= 0 \vee 0 = 0
\end{aligned}
$$

Again,

$$
\begin{aligned}
(a \wedge b)' \wedge (a' \vee b') &= \big(a \vee (a' \vee b')\big) \wedge \big(b \vee (a' \vee b')\big) && \text{(Distributive)} \\
&= \big(a \vee (a' \vee b')\big) \wedge \big(b \vee (b' \vee a')\big) && \text{(Commutative)} \\
&= \big((a \vee a') \vee b'\big) \wedge \big((b \vee b') \vee a'\big) && \text{(Associative)} \\
&= (1 \vee b') \wedge (1 \vee a') \\
&= 1 \wedge 1 = 1
\end{aligned}
$$

Hence, $a' \vee b'$ is the complement of $(a \wedge b)$. So $a' \vee b' = (a \wedge b)'$. Similarly, $(a \vee b)' = a' \wedge b'$ follows.

Example 7.4: Show that in a complemented lattice $(L, \leq)$,

NOTES

$$a \leq b \Leftrightarrow a' \vee b = 1 \Leftrightarrow a \wedge b' = 0 \Leftrightarrow b' \leq a'$$

Solution: Consider $a \leq b \Leftrightarrow a \wedge b = a$

$$\Leftrightarrow a' \vee a = a' \vee (a \wedge b) = 1$$
$$\Leftrightarrow (a' \vee a) \wedge (a' \vee b) = 1$$
$$\Leftrightarrow 1 \wedge (a' \vee b) = 1$$
$$\Leftrightarrow a' \vee b = 1$$

Again, $a \leq b \Leftrightarrow a \vee b = b$

$$\Leftrightarrow b \wedge b' = (a \vee b) \wedge b' = 0$$
$$\Leftrightarrow (a \wedge b') \vee (b \wedge b') = 1$$
$$\Leftrightarrow (a \wedge b') \vee 0 = 0$$
$$\Leftrightarrow a \wedge b' = 0.$$

To prove the last one,

$$a \leq b \Leftrightarrow a \vee b = b$$
$$\Leftrightarrow (a \vee b') = b'$$
$$\Leftrightarrow a' \wedge b' = b'$$
$$\Leftrightarrow a' \wedge a' = b' \quad \text{(Commutative)}$$
$$\Leftrightarrow b' \leq a'.$$

Example 7.5: Consider the lattice $L = \{1, 2, 3, 4, 6, 12\}$, the divisions of 12 ordered by divisibility. Find the following:

(*i*) The lower bound and upper bound of L

(ii) The complement of 4.

(*iii*) Is L a complemented lattice?

Solution: The solution is obtained as follows:

(*i*) The lower bound of L is 1and the upper bound is 12.

(*ii*) Since $4 \wedge 3 = \gcd(4,3) = 1$ and
$4 \vee 3 = \text{lcm}(4,3) = 12$, then the complement of 4 is 3.

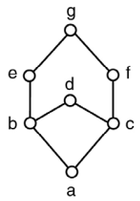
(*iii*) Since $6 \wedge x = \gcd(6, x)$
$\neq 1$ for $x \neq 1$ and $6 \vee 1 = \text{lcm}(6,1)$
$\neq 12$, 6 has no complement and hence L is not a complemented lattice.

7.2.4 Sublattice

Let M be a non-empty subset of a Lattice $(L, \leq)$. We say that M is a sublattice of L if M itself is a lattice with respect to the operations of L.

Note: So M is a sublattice of L if and only if M is closed under the operations $\wedge$ and $\vee$ of L.

Example 7.6: Consider the following lattice L.



Determine whether each of the following is a sublattice of L.

$$M = \{a, b, c, g\}$$

$$N = \{a, b, f, g\}$$

$$O = \{b, d, e, g\}$$

$$P = \{a, d, e, g\}$$

Solution: Since $b \vee c = d$, and $d \notin M$, M is not a sublattice. Since $d \wedge e = b$ and $b \notin p$, p is not a sublattice. But N and O are sublattices.

Example 7.7: Suppose M is a sublattice of a distributive lattice L. Show that M is a distributive lattice.

Solution: For a distributive lattice L, $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ and $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$ for all $a, b, c \in L$. Since M is closed, each element of M is also in L, the distributive laws hold for all elements in M. Hence M is a distributive lattice.

Example 7.8: Prove that in a distributive lattice $(L, \leq)$, if an element has a complement then this complement is unique.

Solution: Suppose for any $a \in L$ has two complements say b and c in L. Then $a \vee b = 1$; $a \wedge b = 0$ and $v\ a \vee c = 1$; $a \wedge c = 0$.

Consider
$$\begin{aligned} b &= b \wedge 1 = b \wedge (a \vee c) \\ &= (b \wedge a) \vee (b \wedge c) && \text{(Distributive)} \\ &= 0 \vee (b \wedge c) = (a \wedge c) \vee (b \wedge c) \\ &= (a \vee b) \wedge c && \text{(Distributive)} \\ &= 1 \wedge c = c \end{aligned}$$

7.3 BOOLEAN ALGEBRA

Boolean algebra is named after George Boole, who used it to study human logical reasoning. For example, any event can be true or false. Similarly, connectives can be of any of the following three basic forms:

NOTES

1. a OR b
2. a AND b
3. NOT a

NOTES

Boolean algebra consists of a set of elements B, with two binary operations {+} and {.} and a unary operation {′}, such that the following axioms hold:

- The set B contains at least two distinct elements x and y.
- **Closure**: For every x, y in B,
  - x + y
  - x . y
- **Commutative laws**: For every x, y in B,
  - x + y = y + x
  - x . y = y . x
- **Associative laws**: For every x, y, z in B,
  - (x + y) + z = x + (y + z) = x + y + z
  - (x . y) . z = x .(y . z) = x . y . z
- **Identities** (0 and 1):
  - 0 + x = x + 0 = x for every x in *B*
  - 1 . x = x . 1 = x for every x in B
- **Distributive laws**: For every x, y, z in B,
  - x . (y + z) = (x . y) + (x . z)
  - x + (y . z) = (x + y) . (x + z)
- **Complement**: For every x in B, there exists an element x′ in B such that,
  - x + x′ = 1
  - x . x′ = 0

Duality Principle: Every valid Boolean expression (equality) remains valid if the operators and identity elements are interchanged.

+ ↔ .

1 ↔ 0

For example, given the expression,

$$a + (b .c) = (a + b). (a + c)$$

Its dual expression is:

$$a. (b + c) = (a. b) + (a. c)$$

The advantage of this theorem is that if you prove one theorem, the other follows automatically.

For example, if $(x + y + z)' = x'. y.' z'$ is valid, then its dual is also valid:

$$(x. y. z)' = x' + y' + z'$$

Apart from the axioms/postulates, there are other useful theorems. These entire theorems are useful for reducing the expression.

NOTES

1. **Idempotency**
   (a) $x + x = x$
   (b) $x \cdot x = x$
   Proof of (a):

$$\begin{aligned} x + x &= (x + x).1 \text{ (Identity)} \\ &= (x + x).(x + x') \text{ (Complement)} \\ &= x + x.x' \text{ (Distributive)} \\ &= x + 0 \text{ (Complement)} \\ &= x \text{ (Identity)} \end{aligned}$$

2. **Null elements** for '+' and '.' operators
   (a) $x + 1 = 1$
   (b) $x \cdot 0 = 0$
3. **Involution**

$$(x')' = x$$

4. **Absorption**
   (a) $x + x \cdot y = x$
   (b) $x \cdot (x + y) = x$
5. **Absorption (variant)**
   (a) $x + x' \cdot y = x + y$
   (b) $x \cdot (x' + y) = x.y$
6. **De Morgan**
   (a) $(x + y)' = x' \cdot y'$
   (b) $(x \cdot y)' = x' + y'$
7. **Consensus**
   (a) $x.y + x' \cdot z + y.z = x \cdot y + x' \cdot z$
   (b) $(x + y) \cdot (x' + z) \cdot (y + z) = (x + y) \cdot (x' + z)$

The set B = {0, 1} and the logical operations OR, AND and NOT satisfy all the axioms of Boolean algebra.

A **Boolean expression** is an algebraic statement containing Boolean variables and operators. Theorems can be proved using the truth table method. They can also be proved by an algebraic manipulation using axioms/postulates or other basic theorems.

7.4 BOOLEAN POLYNOMIALS

A **Boolean function** is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, and the equal and parenthesis

signs. Its result is also a binary value. The general usage is '.' for AND, '+' for OR and '′' for NOT.

7.4.1 Precedence of Operators

NOTES

To lessen the brackets used in writing Boolean expressions, operator precedence can be used. Precedence (highest to lowest): $' \rightarrow . \rightarrow +$

For example,

$$a .b + c = (a. b) + c$$

$$b' + c = (b') + c$$

$$a + b' . c = a + ((b'). c)$$

In order to avoid confusion, use brackets to overwrite precedence.

7.4.2 Truth Table

A truth table is a table, which consists of every possible combination of inputs and its corresponding outputs.

INPUTS	OUTPUTS
...	...
...	...

For basic logic gates, the truth table is already being discussed. Now, for the complex digital systems, it is very important to derive the truth table.

A truth table describes the behaviour of a system that is to be designed. This is the starting point for any digital system design. A designer must formulate the truth table first. It is the responsibility of the designer to decide the number of output bits to represent the behaviour of the system.

For example, if you have to design a 2-bit multiplier, which multiplies two inputs A and B, each of the two bits, then it should be noted that the output must be at least of 4 bits since the maximum result that you can have from this multiplication is 1001(9) corresponding to the maximum value of both the inputs, i.e., 11(3). The block diagram and the truth table are shown as follows:

Fig. 7.1 *2-Bit Multiplier Block Diagram*

In the truth table formation, inputs are taken as A_1A_0 for A input and B_1B_0 for B input. Output resulting from multiplication is to be represented as $P_3P_2P_1P_0$, where P_3 is the MSB and P_0 is the LSB bit. If A = 10, i.e., 2 and B = 11, i.e., 3, then the result of multiplication will be 0110, i.e., 6. So, the bits at the output will be $P_3 = 0$, $P_2 = 1$, $P_1 = 1$, $P_0 = 0$. The complete truth table for the multiplier will be as shown in Table 7.1.

NOTES

Table 7.1 *Truth Table for 2-Bit Multiplier*

A_1	B_0	B_1	B_0	P_3	P_2	P_1	P_0
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
0	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**
0	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0	**1**	**0**	**1**	**0**	**0**	**0**	**1**
0	**1**	**1**	**0**	**0**	**0**	**1**	**0**
0	**1**	**1**	**1**	**0**	**0**	**1**	**1**
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**0**	**0**	**1**	**0**	**0**	**1**	**0**
1	**0**	**1**	**0**	**0**	**1**	**0**	**0**
1	**0**	**1**	**1**	**0**	**1**	**1**	**0**
1	**1**	**0**	**0**	**0**	**0**	**0**	**0**
1	**1**	**0**	**1**	**0**	**0**	**1**	**1**
1	**1**	**1**	**0**	**0**	**1**	**1**	**0**
1	**1**	**1**	**1**	**1**	**0**	**0**	**1**

After the truth table, you have to write the Boolean expression for the output bit and then realize the reduced expression using logic gates.

Whenever a Boolean expression for any output signal is to be written from the truth table, only those input combinations for which the output is high is to be written. As an example, let us write the Boolean expression for Table 7.2.

Table 7.2 *Truth Table*

x	**y**	**z**	**F_1**	**F_2**	**F_3**	**F_4**
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

The Boolean expression for the output F_1 will be $F_1 = x.\ y.\ z'$. This is in the Sum-of-Products form, which will be discussed later.

As can be seen from Table 7.2, output F_1 is 1 only when input xyz is 110. This is represented as x. y. z′. Similarly, you can write the output expression for the rest of the output signals.

$$F_2 = x'.\ y'.\ z + x\ .y'.\ z' + x.\ y'.\ z + x.\ y.\ z' + x.\ y.\ z$$

F_2 can be reduced using Boolean algebra and can be written as follows:

$$F_2 = x'.\ y'.\ z + x\ .y'.\ (z' + z) + x.\ y.\ (z' + z)$$
$$= x'.\ y'.\ z + x\ .y' + x.\ y$$
$$= x'.\ y'.\ z + x\ .(y' + y)$$

NOTES

$= x'.\ y'.\ z + x$

$= (x' + x).\ (y'.\ z + x)$ (Using **Absorption rule)**

$= \mathbf{1.}\ (y'.\ z + x)$

$= (y'.\ z + x)$

Similarly, it can be shown that $F_3 = F_4 = x.\ y' + x'.\ z$

7.4.3 Complement of Functions

For a function F, the **complement** of this function F′ is obtained by interchanging 1 with 0 and vice versa in the function's output values. As an example, take the following function F_1 and its complement, F′:

Table 7.3 *Truth Table of Function and its Complement*

x	y	z	F_1	F_1'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

The same can also be verified using the Boolean algebra technique. In Table 7.3, if $F_1 = xyz'$, then its complement will be:

$F_1' = (x.\ y.\ z')'.$

$= x' + y' + (z')'$ (Using De Morgan's theorem)

$= x' + y' + z$

This is the same as that obtained from the truth table by algebraic manipulation, which is given as follows:

$F_1' = x'.y'.z' + x'.y'.z + x'.y.z' + x'.y.z + x.y'.z' + x.y'.z + x.y.z$

$= x'.y'.(z' + z) + x'.y.(z' + z) + x.y'.(z' + z) + x.y.z$

$= x'.y' + x'.y + x.y' + x.y.z$

$= x'.(y' + y) + x.(y' + y.z)$

$= x' + x.(y' + y.z)$

$= x' + x(y' + y).\ (y' + z)$

$= x' + x.\ (y' + z)$

$= (x' + x).(x' + y' + z)$

$= (x' + y' + z)$

The following are some more general forms of **De Morgan's theorems** used for obtaining complement functions:

$$(A + B + C + ... + Z)' = A'.\ B'.C' \ldots .\ Z'$$

$$(A.\ B.\ C... .Z)' = A' + B' + C' + \ldots + Z'$$

NOTES

7.4.4 Standard Forms

Certain types of Boolean expressions lead to gating networks, which are desirable from the implementation point of view. The following are two standard forms for writing a Boolean expression:

- Sum-Of-Product (SOP)
- Product-Of-Sum (POS)

Before using SOP and POS forms, you must know the following terms:

- **Literal:** A variable on its own or in its complemented form is known as a literal.

  Examples: x, x′, y, y′

- **Product Term**: It is a single literal or a logical product (AND) of several literals.

  Examples: x, x.y.z′, A′.B, A.B

- **Sum Term**: It is a single literal or a logical sum (OR) of several literals.

  Examples: x, x +y + z′, A′+B, A+B

- **Sum-Of-Products (SOP) Expression**: It is a product term or a logical sum (OR) of several product terms.

  Examples: x, x + y. z′, x .y′ + x′. y. z , A.B+A′.B′

- **Product-Of-Sum (POS) Expression**: It is a sum term or a logical product (AND) of several sum terms.

  Examples: x, x.(y + z′), (x +y′).(x′ + y+ z), (A+B).(A′+B′)

Every Boolean expression can either be expressed as a Sum-Of-Product or Product-Of-Sum expression. For example,

SOP:	x′.y + x.y′ + x.y.z
POS:	(x + y′).(x′ + y).(x′ + z′)
Both:	x′ + y + z or x.y.z′
Neither:	x.(w′ + y.z) or z′ + w.x′.y + v.(x.z + w′)

7.4.5 Minterm and Maxterm

Consider two binary variables x, y. Each variable may appear as itself or in the complemented form as literals (i.e., x, x′ and y, y′). For two variables, there are four possible combinations with the AND operator, namely:

$$x'.y',\ x'.y,\ x.y' \text{ and } x.y$$

NOTES

These product terms are called **Minterms**. In other words, A **Minterm** of n variables is the product of n literals from the different variables. In general, n variables can give 2^n Minterms.

Similarly, a **Maxterm** of n variables is the sum of n literals from the different variables.

Examples: x′+y′, x′+y, x+y′, x+y

In general, n variables can give 2^n Maxterms.

The Minterms and Maxterms of 2 variables are denoted by m_0 to m_3 and M_0 to M_3, respectively. In Table 7.4, all the Minterms and Maxterms are written.

Table 7.4 *Minterms and Maxterms*

Minterms				Maxterms	
x	**y**	**Term**	**Notation**	**Term**	**Notation**
0	0	x' .y'	m_0	x + y	M_0
0	1	x' .y	m_1	x + y'	M_1
1	0	x .y'	m_2	x' + y	M_2
1	1	x .y	m_3	x' + y'	M_3

If you examine carefully, each Minterm is the complement of the corresponding Maxterm. For example, $m_2 = x.y'$ and $m_2' = (x.y')' = x' + (y')' = x'+y = M_2$. In other words, **Maxterm is the sum of terms of the corresponding Minterm with its literal complemented.**

7.4.6 Canonical Form: Sum of Minterms

Canonical form is a unique way of representing Boolean expressions. Any Boolean expression can be written in the form of the sum of Minterm. A Σ symbol is used for showing the sum of Minterms. For example,

Table 7.5 *Sum of Minterms*

x	y	x	F_1	F_2	F_3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	1	0

Sum-of-Minterms by gathering/summing the Minterms of the function (where result is a 1) can be obtained as follows:

$$F_1 = x.y.z' = \Sigma m\,(6)$$

$$F_2 = x'.y'.z + x.y'.z' + x.y'.z + x.y.z' + x.y.z = \Sigma m(1,4,5,6,7)$$

$$F_3 = x'.y'.z + x'.y.z + x.y'.z' + x.y'.z = \Sigma m(1,3,4,5)$$

NOTES

7.4.7 Canonical Form: Product of Maxterms

Maxterms are sum terms. For Boolean functions, the Maxterms of a function are the terms for which the result is 0. Boolean functions can be expressed as Products-of-Maxterms. For Table 7.5, each output F_1, F_2 and F_3 can be represented in Product-of-Maxterm. A Π symbol is used to represent Product-of-Maxterms.

$$F_1 = (x + y + z).(x + y + z').(x + y' + z).(x + y' + z').(x' + y + z) .(x' + y + z').(x' + y' + z')$$

$$= \Pi\, M(0,1,2,3,4,5,7)$$

$$F_2 = (x + y + z).(x + y' + z).(x + y' + z')$$

$$= \Pi\, M(0,2,3)$$

$$F_3 = (x + y + z).(x + y' + z).(x' + y' + z).(x' + y' + z')$$

$$= \Pi\, M(0,2,6,7)$$

7.4.8 Conversion of Canonical Forms

Sum-of-Minterms $\Rightarrow$ Product-of-Maxterms

- Rewrite Minterm shorthand using Maxterm shorthand.
- Replace Minterm indices with indices not already used.

For example, $F_1(x,y,z) = \Sigma m(6) = \prod M(0,1,2,3,4,5,7)$.

Product-of-Maxterms $\Rightarrow$ Sum-of-Minterms

- Rewrite Maxterm shorthand using Minterm shorthand.
- Replace Maxterm indices with indices not already used.

For example, $F_2(x,y,z) = \prod M(0,2,3) = \Sigma m(1,4,5,6,7)$.

Sometimes, you are given the reduced expression for any Boolean expression. In this case, you need to find Minterms or Maxterms present in the expression. To convert from a general expression to a Minterm or Maxterm expression, you can use either the truth table or the algebraic manipulation.

For example, suppose you wish to find all the Minterm expansions of $F = AB' + A'C$.

The truth table for the expression is represented as shown in Table 7.6:

NOTES

Table 7.6

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

From the Table 7.6, $F = A'.B'.C + A'.B.C + A.B'.C' + A.B'.C$

$= \Sigma m\ (1, 3, 4, 5)$

Using Algebraic Manipulation

Use $X + X' = 1$ to introduce the missing variables in each term; this introduction will not change the overall expression value. Therefore, for the Boolean expression $F = AB' + A'C$, the missing variable in the first term is C and in the second term is B. So, the missing variable can be introduced as follows:

$= A.B'.(C + C') + A'.C.(B + B')$

$= A.B'.C + A.B'.C' + A'.B.C + A'.B'.C$

$= m_5 + m_4 + m_3 + m_1$

$= \sum m\ (1, 3, 4, 5)$

Similarly, you can find all the Maxterms for reduced expressions**.** Find the Maxterms expansion of $F = (A + B')\ (A' + C)$

Using Algebraic Expression: In this case, $XX' = 0$ is used to introduce missing variables in each term.

Therefore, **$F = (A + B' + CC').\ (A' + C + BB')$**

Assuming that $(A + B') = X$ and $C.C' = YZ$, you can use the expression rule

$= X+YZ = (X+Y)(X+Z)$

$F = (A + B'+C)(A+B'+C')(A'+B+C)(A'+B'+C)$

$= \prod(2, 3, 4, 6)$

Using the Truth Table:

A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$F\ (A,B,C) = \prod(2, 3, 4, 6)$

7.4.9 Boolen Algebra as Lattices

NOTES

Let B be a non-empty set with two binary operations + (or $\vee$) and, (or $\wedge$), a unary operation, and two distinct elements 0 and 1. Then B is called a Boolean algebra if the following axioms hold wher a,b, c are any elements in B.

(i) $a + b = b + a$; $a \cdot b = b \cdot a$ (commutative laws)

(ii) $a + (b \cdot c) = (a + b) \cdot (a + c)$; $a.(b + c) = (a \cdot b)\ (a \cdot c)$ (Distributie laws)

(iii) $a + 0 = a$; $a \cdot 1 = a$ (Identity laws)

(iv) $a + a' = 1$; $a \cdot a' = 0$ (Complement laws)

Boolean algebra is a lattice which contains a least element and a greatest element and which is both complemented and distributive.

We denote the Boolean algebra B by (B, +, ., 1, 0, 1). Here we call 0 as the zero element, 1 as the unit element, and a′ is complement of a, + and . are called sum and product.

Let B = {0,1}, the set of binary digits with the binary operations of + and . and the unary operation defined by

+	1	0
1	1	1
0	1	0

·	1	0
1	1	0
0	1	0

′	1	0
	0	1

Then B is a Boolean algebra.

7.4.10 Atom

A non zero element 'a' in a Boolean algebra (B, +, .,′) is called an atom if for every $x \in B$, $x \wedge a = a$ or $x \wedge a = 0$.

Note: Here the condition $x \wedge a = a$ means that x is a successor of a and $x \wedge a = 0$ is true only when x and a are 'not connected'. So in any Boolean algebra, the immediate successors of the 0-element are called atoms.

Let A be any non-empty set and P(A) the power set of A. In Boolean algebra $(p(A), \cup, \cap, ')$ over $\subseteq$, the singleton sets are the atoms since each element p(A) can be described completely and uniquely as the union of singleton sets.

Let B = {1,2, 3, 5, 6, 10, 15, 30} and let the relation $\leq$ be divides. The operation $\wedge$ is GCD and $\vee$ is LCM. The 0-element is 1. Then the set of atoms of the Boolean algebra is {2,3,5}.

Notes:

1. Let (B, +, . ,′) be any finite Boolean algebra and let A be the set of all atoms. Then (B, +,., ′) is isomorphic to $(p(A), \cup, \cap, ')$.
2. Every finite Boolean algebra (B,+,.,′) has 2^n elements for some position integer n.
3. All Boolean algebra of order 2^n are isomorphic to each other. Finite Boolean algebras are n-tuples of 0′s and 1′s.

NOTES

The simplest nontrivial Boolean algebra is the Boolean algebra B = {0, 1}, the set of binary digits with the binary operations of + and . and the unary operation ´ given by,

+	1	0
1	1	1
0	1	0

·	1	0
1	1	0
0	1	0

´	1
1	0
0	1

If we form $B^2 = B \times B$, we obtain the set $B^2 = \{(0,0), (0,1), (1, 0), (1, 1)\}$. Define +, . and ´ by

$(0, 1) + (1, 1) = (0 + 1, 1+1) = (1, 1)$,

$(0, 1).(1,1) = (0.1, 1.1) = (0, 1)$ and

$(0,1)´ = (0´, 1´) = (1, 0)$.

The B^2 is a Boolean algebra.

Note: Here B^2 is a Boolean algebra of order 4 under component wise operations. Since all Boolean algebra of order 4 are isomorphic to each other, this is a simple way of describing all Boolean algebras of order 4. In general, any Boolean algebra of order 2^n are isomorphic to B^n.

Example 7.9: Find the atoms of the Boolean algebra (i) B^2 (ii) B^4 (iii) B^n for $n \geq 1$.

Solution:

(i) (0, 1) and (1, 0)

(ii) (1,0,0,0), (0,1,0,0), (0,0,1,0) and (0,0,0,1)

(iii) The n-tuples with exactly one 1.

Check Your Progress

1. What do you understand by the lattice?
2. Explain the distributive lattice.
3. Define the modular lattice.
4. Elaborate on the bounded lattice.
5. State the complement of an element.
6. Interpret the complement lattice.
7. Explain the sublattice.
8. Define the Boolean algebra.
9. Illustrate the Boolean function.
10. Analyse the term 'atom' in Boolean algebra.

7.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A lattice is a partially ordered set $(L, \leq)$ in which every pair of elements a, $b \in L$ has a Greatest Lower Bound (GLB) and a Least Upper Bound (LUB).

2. A Lattice $(L, \leq)$ is said to be distributive lattice if for any $a, b, c, \in L$,

   $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

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

3. Modular Lattice: A Lattice $(L, \leq)$ is said to be modular lattice if

   $a \leq c \Rightarrow a \vee (b \wedge c) = (a \vee b) \wedge c$.

4. Bounded Lattice: A Lattice $(L, \leq)$ which has both, a least element denoted by 0, and the greatest element denoted by 1 is called a bounded lattice.

5. Complement of an Element: In a bounded Lattices $(L, \leq)$, an element $b \in L$ is called a complement of an element $a \in L$ if $a \wedge b = 0$ and $a \vee b = 1$, we denote b by a.

6. Complement Lattice: A Lattice $(L, \leq)$ is said to be complemented lattice if every element of L has at least one complement.

7. Let M be a non-empty subset of a Lattice $(L, \leq)$. We say that M is a sublattice of L if M itself is a lattice with respect to the operations of L.

8. Boolean algebra is named after George Boole, who used it to study human logical reasoning. For example, any event can be true or false. Similarly, connectives can be of any of the following three basic forms:

   1. a OR b

   2. a AND b

   3. NOT a

9. A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, and the equal and parenthesis signs. Its result is also a binary value. The general usage is '**.**' for AND, '**+**' for OR and '**′**' for NOT.

10. A non zero element 'a' in a Boolean algebra (B, +, .,′) is called an atom if for every x ∈ B, x $\wedge$ a = a or x $\wedge$ a = 0.

7.6 SUMMARY

- A lattice is a partially ordered set $(L, \leq)$ in which every pair of elements a, $b \in L$ has a Greatest Lower Bound (GLB) and a Least Upper Bound (LUB).
- A Lattice $(L, \leq)$ is said to be distributive lattice if for any $a, b, c, \in L$,

  $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

  $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.
- Modular Lattice: A Lattice $(L, \leq)$ is said to be modular lattice if

  $a \leq c \Rightarrow a \vee (b \wedge c) = (a \vee b) \wedge c$.

NOTES

- Bounded Lattice: A Lattice $(L, \leq)$ which has both, a least element denoted by 0, and the greatest element denoted by 1 is called a bounded lattice.
- Complement of an Element: In a bounded Lattices $(L, \leq)$, an element $b \in L$ is called a complement of an element $a \in L$ if $a \wedge b = 0$ and $a \vee b = 1$, we denote b by a.
- Complement Lattice: A Lattice $(L, \leq)$ is said to be complemented lattice if every element of L has at least one complement.
- Let M be a non-empty subset of a Lattice $(L, \leq)$. We say that M is a sublattice of L if M itself is a lattice with respect to the operations of L.
- Boolean algebra is named after George Boole, who used it to study human logical reasoning. For example, any event can be true or false. Similarly, connectives can be of any of the following three basic forms:

  1. a OR b

  2. a AND b

  3. NOT a
- A Boolean function is an expression formed with binary variables, the two binary operators OR and AND, the unary operator NOT, and the equal and parenthesis signs. Its result is also a binary value. The general usage is '**.**' for AND, **'+'** for OR and **''** for NOT.
- A non zero element 'a' in a Boolean algebra (B, +, .,′) is called an atom if for every x $\in$ B, x $\wedge$ a = a or x $\wedge$ a = 0.

7.7 KEY WORDS

- **Lattice:** A poset in which every pair of elements have both a least upper bound and a greatest lower bound is called a lattice.
- **Distributive lattice:** A lattice $(L,\leq)$ is said to be distributive lattice if for any $a, b, c, \in L$

  $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$

  $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
- **Modular lattice:** A lattice $(L,\leq)$ is said to be modular lattice if $a \leq c \Rightarrow a \vee (b \wedge c) = (a \vee b)\ \ c$.
- **Bounded lattice:** A lattice $(L,\leq)$ which has both, a least element denoted by 0, and the greatest element denoted by 1 is called a bounded lattice.
- **Complement lattice:** A lattice $(L,\leq)$ is said to be complemented lattice if every element of L has at least one complement.
- **Subset:** Let M be a non-empty subset of a lattice $(L,\leq)$. We say that *M is a* sublattice of L if M itself is a lattice with respect to the operation of L.

- **Boolean algebra:** Boolean algebra is named after George Boole, who used it to study human logical reasoning. For example, any event can be true or false.

NOTES

7.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain the lattice.
2. Define distributive lattice.
3. Elaborate on the modular lattice.
4. State bounded lattice.
5. Illustrate the complement of an element.
6. Interpret the complement lattice.
7. Explain the sublattice.
8. Define Boolean algebra.
9. Analyse the Boolean function.
10. Elaborate on the minterm and maxterm.

Long-Answer Questions

1. Explain the term lattice with their basic properties.
2. Differentiate between the modular lattice and bounded lattice giving appropriate examples.
3. Prove that every finite subset of a lattice has an LUB and a GLB.
4. Show that in a direct product of any two distributive lattices is a distributive lattice.
5. Show that in a lattice with two or more elements, no element is its own complement.
6. A lattice is said to the modular if $a \leq c \Rightarrow a \vee (b \wedge c) = (a \vee b) \wedge c$. Show that every distributive lattice is modular but not conversely.
7. Show that a lattice is modular if and only if $a \vee (b \wedge (a \vee c)) = (a \vee b) \wedge (a \vee c)$.
8. Is the cartesian product of two lattices always a lattice? Prove your claim.

7.9 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

NOTES

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 8 CODING THEORY

NOTES

Structure

8.0 INTRODUCTION

Coding theory is the study of the properties of codes and their respective fitness for specific applications. Codes are used for data compression, cryptography, error detection and correction, data transmission and data storage. Codes are studied by various scientific disciplines—such as information theory, electrical engineering, mathematics, linguistics, and computer science for the purpose of designing efficient and reliable data transmission methods. This typically involves the removal of redundancy and the correction or detection of errors in the transmitted data.

In 1948, Claude Shannon published "A Mathematical Theory of Communication", an article in two parts in the July and October issues of the Bell System Technical Journal. This work focuses on the problem of how best to encode the information a sender wants to transmit. In this fundamental work he used tools in probability theory, developed by Norbert Wiener, which were in their nascent stages of being applied to communication theory at that time. Shannon developed information entropy as a measure for the uncertainty in a message while essentially inventing the field of information theory.

The binary Golay code was developed in 1949. It is an error-correcting code capable of correcting up to three errors in each 24-bit word, and detecting a fourth.

NOTES

Error correction adds extra data bits to make the transmission of data more robust to disturbances present on the transmission channel. The ordinary user may not be aware of many applications using error correction. A typical music compact disc (CD) uses the Reed-Solomon code to correct for scratches and dust. In this application the transmission channel is the CD itself. Cell phones also use coding techniques to correct for the fading and noise of high frequency radio transmission. Data modems, telephone transmissions, and the NASA Deep Space Network all employ channel coding techniques to get the bits through, for example the turbo code and LDPC codes.

In this unit, you will study about the coding theory, computer codes, Hamming distance, encoding a message, group codes, procedure for generating group codes, and decoding and error correction.

8.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the coding theory
- Explain the computer codes
- Elaborate on the Hamming distance
- Interpret the encoding a message
- Define the groups codes
- Analyse the procedure for generating group codes
- Describe the decoding and error correction

8.2 COMPUTER CODES

A code is a symbol or group of symbols that stands for something. It is a representation of discrete elements of information, which may be in the form of numbers, letters or any other varying physical quantities. Binary bits 1 and 0 are often used in groups. The codes are used to communicate the information to the digital computer and to retrieve from it. The purpose of the code is that the operator can feed data into computers directly in decimal numbers, alphabets and special characters. The computer in turn converts these data in binary code which it can process and after computation, it again converts the binary data into decimal numbers, alphabets and special characters which we could understand easily.

Certain binary codes are used for arithmetic operations. Other codes facilitate the creation of digital transducers for entering information into a system.

NOTES

Bits and Bytes

All data to be stored and processed in computers are transformed or coded as strings of two symbols, one symbol to represent each state. The two symbols normally used are 0 and 1. These are known as BITS, an abbreviation for BInary digiTS.

Let us now understand some commonly used terms.

BITS A bit is the smallest element used by a computer. It holds one of the two possible values.

Value	Meaning
0	Off
1	On

A bit which is OFF is also considered to be FALSE or NOT SET; a bit which is ON is also considered to be TRUE or SET.

Since a single bit can only store two values, there could possibly be only four unique combinations as follows,

00 01 10 11

Bits are therefore, combined together into larger units in order to hold greater range of values.

NIBBLE A nibble is a group of FOUR bits (4 bits). This gives a maximum number of sixteen possible different values.

$2^4 = 16$ (2 to the power of the number of bits)

BYTES Bytes are a grouping of 8 bits (two nibbles) and are often used to store characters. They can also be used to store numeric values.

$2^8 = 256$ (2 to the power of the number of bits)

Representation of Characters

Binary data is not the only data handled by the computer. We also need to process alphanumeric data like alphabets (upper and lower case), digits (0 to 9) and special characters like + – * / ( ) space or blank etc. These also must be internally represented as bits.

8.2.1 Binary Coded Decimal (BCD)

Binary Coded Decimal (BCD) is one of the early memory codes. It is based on the concept of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. It further uses four digits to represent each of the digits. Table 8.1 shows the BCD equivalent of the decimal digits.

NOTES

Table 8.1 BCD Equivalent of Decimals

Decimal Number	Binary Equivalent
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Converting $(42)_{10}$ into its BCD equivalent, would result in:

$$(42)_{10} = \underset{4}{0100}\ \underset{2}{0100} \quad \text{or} \quad 01000010 \text{ in BCD}$$

As seen, 4 bit BCD code can be used to represent decimal numbers only. Since 4 bits are insufficient to represent the various other characters used by the computer, instead of using only 4-bits (giving 16 possible combinations), computer designers commonly use 6 bits to represent characters in BCD code. In this, the 4 BCD numeric place positions are retained, but two additional zone positions are added. With 6 bits it is possible to represent 2^6 or 64 different characters. This is therefore sufficient to represent the decimal digits (10), alphabetic characters (26), and special characters (28).

8.2.2 Extended Binary Coded Decimal Interchange (EBCDIC)

The abbreviation EBCDIC stands for the *Extended Binary Coded Decimal Interchange Code*. It is an 8-bit code in which the decimal digits are represented by the 8421 BCD code preceded by 1111.

The major drawback with the BCD code is that it allows only 64 different characters to be represented. This is not sufficient to provide for decimal numbers (10), lowercase letters (26), uppercase letters (26), and a fairly large number of special characters (28 plus).

The BCD code was therefore extended from a 6 bit to an 8 bit code. The added 2 bits are used as additional zone bits, expanding the zone bits to 4. This resulting code is called the Extended Binary Coded Decimal Interchange Code (EBCDIC). Using the EBCDIC it is possible to represent 2^8 or 256 characters. This takes care of the character requirement along with a large quantity of printable and several non-printable control characters (movement of the cursor on the screen, vertical spacing on printer etc.).

NOTES

Since EBCDIC is an 8 bit code, it can easily be divided into two 4 bit groups. Each of these groups can be represented by one hexadecimal digit (explained earlier in this unit). Thus, hexadecimal number system is used as a notation for memory dump by computers that use EBCDIC for internal representation of characters.

Developed by IBM, EBCDIC code is used in most IBM models and many other computers.

8.2.3 American Standard Code for Information Interchange (ASCII)

The abbreviation ASCII stands for the *American Standard Code for Information Interchange*. The ASCII code is a 7-bit code used in transferring coded information from keyboards and to computer displays and printers. It is used to represent numbers, letters, punctuation marks as well as control characters. For example, the letter *A* is represented by 100 0001. Several computer manufacturers have adopted it as their computers' internal code. This code uses 7 digits to represent 128 characters. Now an advanced ASCII is used having 8 bit character representation code allowing for 256 different characters. This representation is being used in micro computers.

Let us look at the encoding method. Table 8.2 below shows the bit combinations required for each character.

Table 8.2 Bit Combinations for Each Character

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20		!	"	#	$	%	&	‘	(	)	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Thus, to code a text string 'Hello' in ASCII using hexadecimal digits:

H	e	l	l	o	.
48	65	6C	6C	6F	2E

The string is represented by the byte sequence 48 65 6C 6C 6F 2E.

8.2.4 Excess-3 Code

The Excess-3 is a digital code that is derived by adding to each decimal digit and then converting the result to four-bit binary. The Excess-3 code is used in some

arithmetic circuits because it is self-complementing. Table 8.3 shows Excess-3 codes to represent single decimal digit and its BCD code.

Table 8.3 *BCD and Excess-3 Codes*

NOTES

Decimal Digit	*BCD*				*Excess-3 Code*			
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

8.2.5 Gray Code

The Gray code belongs to a class of codes called *minimum change codes*, in which only one bit in the code group changes when going from one step to the next. Gray code is not an arithmetic code.

Table 8.4 *Gray Code*

Decimal Digit	*Binary Code*				*Gray Code*			
0	0	0	0	0	0	0	0	0 ←—————————
1	0	0	0	1	0	0	0	1 ————————
2	0	0	1	0	0	0	1	1 ←————————
3	0	0	1	1	0	0	1	0 ————————
4	0	1	0	0	0	1	1	0 ←————————
5	0	1	0	1	0	1	1	1 ←————
6	0	1	1	0	0	1	0	1 ←——
7	0	1	1	1	0	1	0	0 ←
8	1	0	0	0	1	1	0	0 ←
9	1	0	0	1	1	1	0	1 ←——
10	1	0	1	0	1	1	1	1 ←————
11	1	0	1	1	1	1	1	0 ←————————
12	1	1	0	0	1	0	1	0 ←————————
13	1	1	0	1	1	0	1	1 ————————
14	1	1	1	0	1	0	0	1 ←—————————
15	1	1	1	1	1	0	0	1 ←—————————

Table 8.4 shows the Gray code representation for the decimal numbers 0 through 9 together with the straight binary code.

8.2.6 Alphanumeric Codes

Alphanumeric codes are the codes that represent alphabetic characters (letters), punctuation marks and other special characters. Alphanumeric code represents all of the various characters and functions that are found on a computer keyboard.

NOTES

Check Your Progress

1. Define the computer codes.
2. Explain the binary coded decimal.
3. What do you understand by the 'EBCDIC'?
4. Interpret the American standard code for information interchange.
5. Elaborate on the excess-3 code.
6. State the Gray code.
7. Explain the alphanumeric code.

8.3 HAMMING DISTANCE

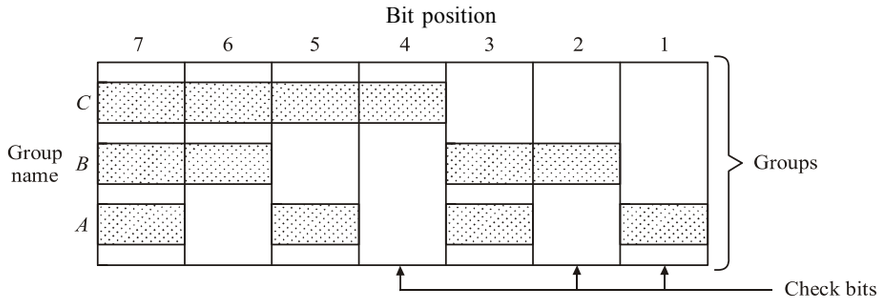
In 1950, R.W. Hamming developed a system that provides an orderly way to add one or more parity bits to a data character to allow detection or both error detection and correction. The **Hamming distance** between two code words is defined as the number of bits that must be changed for one code word to another. It is actually a method for constructing codes with a minimum distance of 3.

For any value of i, Hamming code method yields a $(2^i - 1)$—bit code with i check bits and $2^i - 1 - i$ information bits. Distance-3 codes with a smaller number of information bits are obtained by deleting information bits from a Hamming code with a larger number of bits.

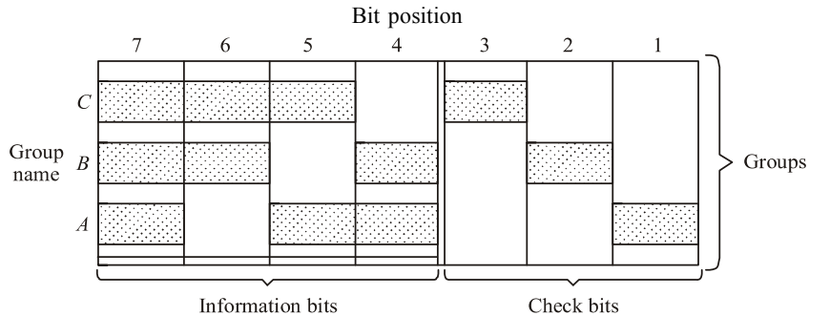
The bit positions in a Hamming code word can be numbered from 1 through $2^i - 1$. In this case, any position whose number is a power of 2 is a check bit, and the remaining positions are information bits. Each check bit is grouped with a subset of the information bits, as specified by a *parity-check matrix* as shown in Figure 8.1. Each check bit is grouped with the information positions whose numbers have a 1 in the same bit when expressed in binary. For example, check bit 2 (010) is grouped with information bits 3(011), 6(110) and 7(111). For a given combination of information bit values, each check bit is chosen to produce even parity, that is, so the total number of 1s in its group is even.

The bit positions of a parity-check matrix and the resulting code words are rearranged so that all of the check bits are on the right, as in Figure 8.1(*b*). The first two columns of Table 8.5 list the resulting code words.

NOTES



(a) Hamming codes with bit positions in numerical order



(b) Hamming codes with check bits and information bits separated

Fig. 8.1 *Parity-Check Matrices for 7-bit Hamming Codes*

We can prove that the minimum distance of a Hamming code is 3 by proving that at least a 3-bit change must be made to code a word to obtain another code word.We can also prove that a 1-bit or 2-bit change in a code word yields a non-code word.

If we change 1-bit of a code word in position *j*, then we change the parity of every group that contains position *j*. Since every information bit is contained in at least one group, at least one group has incorrect parity, and the result is a non-code word.

If we change two bits in positions *j* and *k*, parity groups that contain both positions *j* and *k* will still have correct parity. This is because parity is not affected when an even number of bits are changed. However, since *j* and *k* are different, their binary representations differ in at least one bit, corresponding to one of the parity groups. This group has only one bit changed, resulting in incorrect parity and a non-code word.

For the proof of 1-bit errors, the position numbers must be non-zero. For the proof of 2-bit errors, no two positions have the same number. Thus, with an *i*-bit position number, we can construct a Hamming code with up to $2^i - 1$ bit positions.

The proof also suggests how can an error-correcting decoder be designed for a received Hamming code word. First, check all of the parity groups. If all the

NOTES

groups have even parity, then the received word is assumed to be correct. If one or more groups have odd parity, then a single error is assumed to have occurred. The pattern of groups that have odd parity called the *syndrome* must match one of the columns in the parity-check matrix; the corresponding bit position is assumed to contain the wrong value and is complemented. For example, using the code defined by Figure 8.1(*b*), suppose we receive the word 0101011. Groups *B* and *C* have odd parity, corresponding to position 6 of the parity-check matrix (the syndrome is 110_2, or 6_{10}). By complementing the bit in position 6 of the received word, we determine that the correct word is 0001011.

A distance-3 Hamming code can easily be modified to increase its minimum distance to 4. Add one more check bit, chosen so that the parity of all the bits, including the new one, is even. As in the 1-bit even-parity code, this bit ensures that all errors affecting an odd number of bits are detectable.

Distance-3 and distance-4 Hamming codes are commonly used to detect and correct errors in computer memory systems, especially in large mainframe computers where memory circuits account for the bulk of the system's failures.

Table 8.5 *Code Words in Distance-3 and Distance-4 Hamming codes with Four Information Bits*

Minimum distance-3 Code		*Minimum distance-4 Code*	
Information bits	*Parity bits*	*Information bits*	*Parity bits*
0000	000	0000	0000
0001	011	0001	0111
0010	101	0010	1011
0011	110	0011	1100
0100	110	0100	1101
0101	101	0101	1010
0110	011	0110	0110
0111	000	0111	0001
1000	111	1000	1110
1001	100	1001	1001
1010	010	1010	0101
1011	001	1011	0010
1100	001	1100	0011
1101	010	1101	0100
1110	100	1110	1000
1111	111	1111	1111

Example 8.1: Encode data bits 0101 into a seven bit even-parity Hamming code.

Solution:

D_7	D_6	D_5	P_4	D_3	P_2	P_1
0	1	0	1	1	0	1

NOTES

Example 8.2: A seven-bit Hamming code is received as 1111101. What is the correct code?

Solution:

D_7	D_6	D_5	P_4	D_3	P_2	P_1	
1	1	1	1	1	0	1	
					0	1	0

Bits 4, 5, 6 and 7, no error

Bits 2, 3, 6 and 7, error

Bits 1, 3, 5 and 7, no error

Bit 2 is in error, and the correct code is 1111111

Example 8.3: For a received data 1100010, determine whether a single error occurred and, if so, correct the error.

Solution: Checking the three parity bit, groups for even-parity, we have:

P_1	P_2	8	P_4	4	2	1
1	1	0	0	0	1	0

$P_1 + 8 + 4 + 1 = 1 + 0 + 0 + 0 = F$

$P_2 + 8 + 2 + 1 = 1 + 0 + 1 + 0 = E$ (even-parity)

$P_4 + 4 + 2 + 1 = 0 + 0 + 1 + 0 = F$ (failed even parity check)

Any even parity failure indicates an error has occurred, the bit 5 was in error. Thus, the correct digit is 6.

1	2	3	4	5	6	7	
1	1	0	0	0	1	0	
				↓			
1	1	0	0	1	1	0	(= digit 6)

Example 8.4: Determine the single error-correcting code for the BCD number 1001 (information bits) using even parity.

Solution: First, find the number of parity bits required. Let $P = 3$.

Then $2^P = 2^3 = 8$

Since $2^P \geq m + p + 1$, we have $m + p + 1 = 4 + 3 + 1 = 8$

Three parity bits are sufficient.

Total code bits = 4 + 3 = 7

Construct a bit position table.

NOTES

Bit designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4
Bit position	1	2	3	4	5	6	7
Binary position number	001	010	011	100	101	110	111
Information bits			1		0	0	1
Parity bits	0	0		1			

Parity bits are determined in the following steps:

P_1 checks bit positions 1, 3, 5 and 7 and must be a 0 in order to have an even number of 1s (2) in this group.

P_2 checks bit positions 2, 3, 6 and 7 and must be a 0 in order to have an even number of 1s (2) in this group.

P_3 checks bit positions 4, 5, 6 and 7 and must be a 1 in order to have an even number of 1s (2) in this group.

These parity bits are entered into the table, and the resulting combined code is 0011001.

Example 8.5: Determine the single error-correcting code for the information code 10110 for odd-parity.

Solution: Determine the number of parity bits required. In this case the number of information bits, m, is five.

Let $p = 4$, $2^p = 2^4 = 16$

We know that $m + p + 1 = 5 + 4 + 1 = 10$

Four parity bits are sufficient

Total code bits $= 5 + 4 = 9$

Construct a bit position table

Bit designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4	P_4	M_5
Bit position	1	2	3	4	5	6	7	8	9
Binary position number	0001	0010	0011	0100	0101	0110	0111	1000	1001
Information bits			1		0	1	1		0
Parity bits	1	0		1				1	

Parity bits are determined as follows:

P_1 checks bit positions 1, 3, 5, 7 and 9 and must be a 1 to have an odd number of 1s (3) in this group.

P_2 checks bit positions 2, 3, 6 and 7 and must be a 0 to have an odd number of 1s (3) in this group.

P_3 checks bit positions 4, 5, 6 and 7 and must be a 1 to have an odd number of 1s (3) in this group.

NOTES

P_4 checks bit positions 8 and 9 and must be a 1 to have an odd number of 1s (1) in this group.

These parity bits are entered into the table, and the resulting combined code is 101101110.

Example 8.6: The code word 0011001 is transmitted and that 0010001 is received. The receiver does not know what was transmitted and must look for proper parities to determine if the code is correct. Designate any error that has occurred in transmission if even-parity is used.

Solution: First, prepare a bit position table:

Bit designation	P_1	P_2	M_1	P_3	M_2	M_3	M_4
Bit position	1	2	3	4	5	6	7
Binary position number	001	010	011	100	101	110	111
Received code	0	0	1	0	0	0	1

First parity check: P_1 checks positions 1, 3, 5 and 7

There are two 1s in this group.

Parity check is good, ——————→ 0 (LSB)

Second parity check: P_2 checks positions 2, 3, 6 and 7.

There are two 1s in this group.

Parity check is good, ——————→ 0

Thirty parity check: P_3 checks positions 4, 5, 6 and 7.

There is one in this group.

Parity check is bad, ——————→ 1 (MSB)

Result: The error position code is 100 (binary 4). This says that the bit in the number 4 position is in error. It is a 0 and should be a 1. The correctd code is 0011001, which agrees with the transmitted code.

8.4 ENCODING A MESSAGE

The encoding of a message is the production of the message. It is a system of coded meanings, and in order to create that, the sender needs to understand how the world is comprehensible to the members of the audience.

In the process of encoding, the sender (i.e. encoder) uses verbal (e.g. words, signs, images, video) and non-verbal (e.g. body language, hand gestures, face expressions) symbols for which he or she believes the receiver (that is, the decoder) will understand. The symbols can be words and numbers, images, face expressions, signals and/or actions. It is very important how a message will be encoded; it partially depends on the purpose of the message.

Production – This is where the encoding, the construction of a message begins. Production process has its own "Discursive" aspect, as it is also framed by meanings and ideas; by drawing upon society's dominant ideologies, the creator of the message is feeding off of society's beliefs, and values. Numerous factors are involved in the production process. On one hand "Knowledge-in-use concerning the routines of production, technical skills, professional ideologies, institutional knowledge, definitions and assumptions, assumptions about the audience" form the "production structures of the television." On the other hand, "topics, treatments, agendas, events, personnel, images of the audience, 'definitions of the situation' from other sources and other discursive formations" form the other part of wider socio-cultural and political structure.

NOTES

The Encoding/Decoding model of communication was first developed by cultural studies scholar Stuart Hall in 1973. Titled "Encoding and Decoding in the Television Discourse", Hall's essay offers a theoretical approach of how media messages are produced, disseminated, and interpreted. Hall proposed that audience members can play an active role in decoding messages as they rely on their own social contexts, and might be capable of changing messages themselves through collective action.

8.5 GROUPS CODES

In coding theory, group codes are a type of code. Group codes consist of *n* linear block codes which are subgroups of G^n, where G is a finite Abelian group. A systematic group code C is a code over G^n of order $|G|^k$ defined by $n-k$ homomorphisms which determine the parity check bits. The remaining k bits are the information bits themselves.

8.5.1 Procedure for Generating Group Codes

Group codes can be constructed by special generator matrices which resemble generator matrices of linear block codes except that the elements of those matrices are endomorphisms of the group instead of symbols from the code's alphabet. For example, considering the generator matrix

$$G = \begin{pmatrix} \begin{pmatrix} 00 \\ 11 \end{pmatrix} & \begin{pmatrix} 01 \\ 01 \end{pmatrix} & \begin{pmatrix} 11 \\ 01 \end{pmatrix} \\ \begin{pmatrix} 00 \\ 11 \end{pmatrix} & \begin{pmatrix} 11 \\ 11 \end{pmatrix} & \begin{pmatrix} 00 \\ 00 \end{pmatrix} \end{pmatrix}$$

The elements of this matrix are 2×2 matrices which are endomorphisms. In this scenario, each code word can be represented as $g_1^{m_1} g_2^{m_2} \ldots g_r^{m_r}$ where $g_1, \ldots g_r$ are the generators of G.

8.6 DECODING ERROR CORRECTION

NOTES

A code that uses n-bit strings need not contain 2^n valid code words. An error-detecting code has the property that corrupting or garbling a code word will likely produce a bit string that is not a code word (a non-code word).

A system that uses an error-detecting code generates, transmits, and stores only code words. Thus, errors in a bit string can be detected by a simple rule—if the bit string is a code word, it is assumed to be correct; if it is a non-code word, it contains an error.

Errors may occur while recording data on magnetic surfaces due to bad spots on the surface. They may also be introduced during data transmission between units, due to electrical disturbances. Most of these errors result in change of a bit from 0 to 1 or 1 to 0. One of the simplest and most commonly used error detection code is called the parity bit.

A parity bit is an extra bit added to the binary data so that it makes the total number of 1s in the data either odd or even. In case of Even parity**,** a bit is added to each character in such a way that the total number of 1s in each character code is even. Similarly, in case of odd parity, the bit added to each character makes the number of 1s in the character code odd.

In a 7-bit data 0110101, for example, let us add the eighth bit, which is a parity bit. In case of even parity, the added bit, should be 0 since there are already four 1s in the given 7-bit number. While in case of odd parity, the added bit would be a 1 making the total number of 1s odd (5 in number).

All the characters would then have 8 bits including the parity check bit. Whenever a character is read from storage or received from a remote location, the number of bits in its code is counted. It has to be even (in case of even parity). If it is odd, then at least one bit is wrong. A single error in any of the 8 bits of the code will definitely be detected. However, two errors cannot be detected by this scheme, as the total number of 1s will remain even after 2 bits change. The probability of more than one error occurring at a time is usually very small and that is why this scheme is commonly accepted as adequate.

Codes have also been devised which use more than one check bit in order to not only detect but also correct errors. These are called error-correcting codes.

Parity

The most simple and commonly used error detecting method is the *parity check method*, in which an extra bit called *parity bit* is included with the binary message, to make the total number of 1s either odd or even, resulting in two methods; (*i*) Even-parity method and (*ii*) Odd-parity method.

NOTES

The ability of a code to detect single errors can be stated in terms of the *concept of distance*. A code detects all single errors if the minimum distance between all possible pairs of code words is 2.

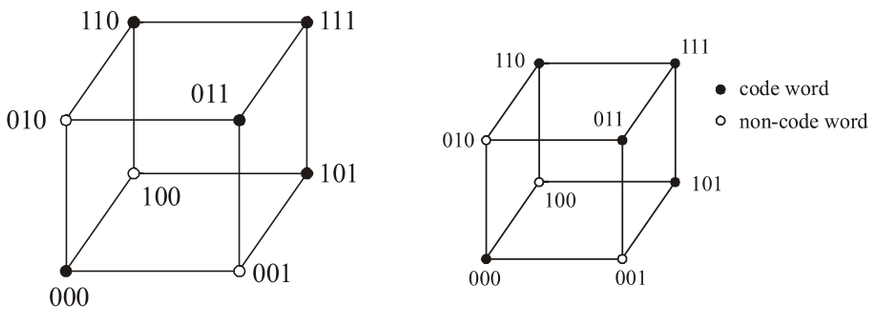
In general, $(n + 1)$ bits are needed to construct a single-error detecting code with 2^n code words. The first n bits of a code word, called *information bits*, may be any of the 2^n n-bit strings minimum error bit.

A code in which the total number of 1s in a valid $(n + 1)$ bit code word is even; this is called an *even-parity code*.

A code in which the total number of 1s in a valid $(n + 1)$ bit code word is odd and this code is called an *odd-parity code*. These codes are also sometimes called *1-bit parity codes*, since they each use a single parity bit. The parity bit can be placed at either end of the code word, such that the receiver must be able to understand the parity bit and the actual data.

An n-bit code and its error-detecting properties under the independent error model are easily explained in terms of an n-cube. A code is simply a subset of the vertices of the n-cube. In order for the code to detect all single errors, no code-word vertex can be immediately adjacent to another code-word vertex.

Figure 8.2(a) shows a 3-bit code with five code words. Code word 111 is immediately adjacent to code words 110, 011 and 101. Since a single failure could change 111 to 110, 011 or 101 bits code does not detect all single errors. If we make 111 a non-code word, we obtain a code that does have the single-error-detecting property, as shown in Figure 8.2. No single error can change one code word into another.



(a) Minimum distance = 1 does not detect all single errors

(b) Minimum distance = 2 detects all single errors

Fig. 8.2 *Code Words in Two-Different 3-Bit Codes*

NOTES

Table 8.6 *Distance-2 Codes with Three Information Bits*

Information Bits XYZ	*Even-parity Code XYZ P*	*Odd-parity Code XYZ P*
000	000 1	000 1
001	001 1	001 0
010	010 1	010 0
011	011 0	011 1
100	100 1	100 0
101	101 0	101 1
110	110 0	110 1
111	111 1	111 0

The 1-bit parity codes do not detect 2-bit errors, since changing two bits does not affect the parity. However, the codes can detect errors in any *odd* number of bits. Actually, 1-bit parity codes error-detection capability stops after 1-bit errors. Other codes, with minimum distance greater than 2, can be used to detect multiple errors.

Checksum

Since the double error will not change the parity of the bits, the parity checker will not indicate any error. The check sum method is used to detect double errors and pinpoint erroneous bits. The working of checksum method is explained as follows:

Initially word A 10110111 is transmitted. Next word B 00100010 is transmitted. The binary digits in the two words are added and the sum obtained is retained in the transmitter. Then, a word *C* is transmitted and added to the previous sum and the new sum is retained. Thus, each word is added to the previous sum and after the transmission of all the words, the final sum, called *checksum* is also transmitted. The same operation is also done independently at the receiver and the final sum obtained at the receiver is checked against the transmitted checksum. If the two sums are equal, then there is no error.

8.6.1 Error-Correcting Codes

A code that is used to correct errors is called an *error-correcting code*. In general, if a code has minimum distance 2c+1, it can be used to correct errors, that affect up to *c* bits. If a code's minimum distance is 2c + d + 1, it can be used to correct errors in up to *c* bits and to detect in up to *d* additional bits.

Consider a fragment—cube for a code with minimum of 3. There are at least two non-code words between each pair of code words. Now, let us transmit code words and assume that failures affect at most one bit of each received code word. Then a received non-code word with a 1-bit error will be closer to the

originally transmitted code word than to any other code word. Therefore, when we receive a non-code word, we can correct the error by changing the received non-code word to the nearest code word, as indicated by the arrows in the Figure. Deciding which code word was originally transmitted to produce a received word is called *decoding*, and the hardware that does this is an *error-correcting decoder.*

NOTES

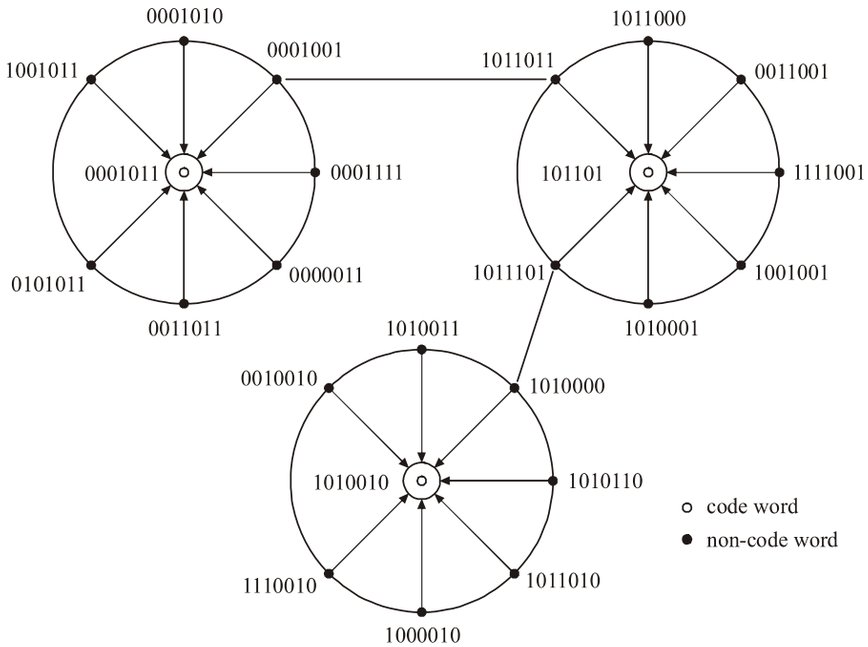
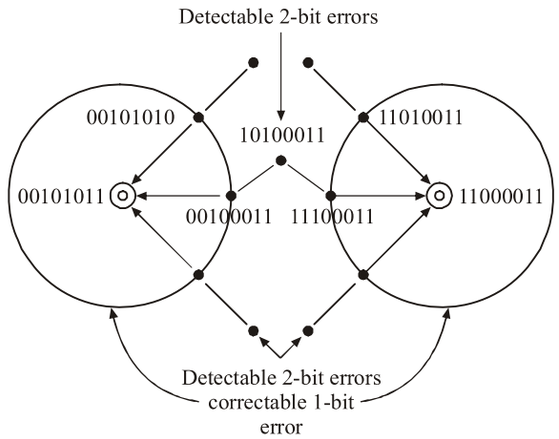


Fig. 8.3 *Some Code Words and Non-code Words in a 7-bit Distance 3 Code*

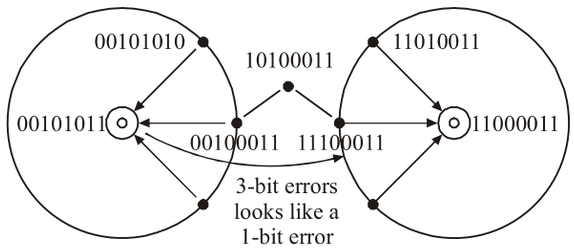
For example, consider a fragment of the *n*-cube for a code with minimum distance 4 ($c = 1, d = 1$) [Figure 8.4(*a*)]. Single-bit errors that produce non-code words 00101010 and 11010011 can be corrected. However, an error that produces 10100011 cannot be corrected, because no single-bit error can produce this non-code word, and either of two 2-bit errors could have produced it. So the code can detect a 2-bit error, but it cannot correct it.

When a non-code word is received, we do not know which code word was originally transmitted; we only know which code word is closest to what we have received. Thus, a 3-bit error may be corrected to the wrong value as shown in Figure 8.4(*b*). The possibility of making this kind of mistake may be acceptable if 3-bit errors are very unlikely to occur. On the other hand, if we are concerned about 3-bit errors, we can change, the decoding policy for the code. Instead of connecting the errors, we can just flag all non-code words as uncorrectable errors. Thus, we can use the same distance 4-code to detect up to 3-bit errors but correct no errors ($c = 0, d = 3$) as shown in Figure 8.4(*c*).

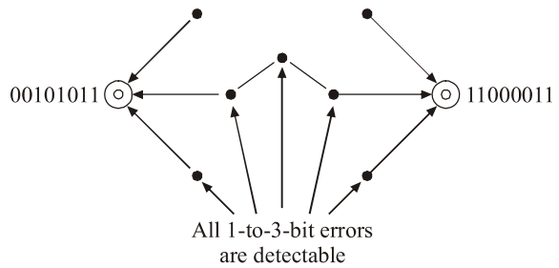
NOTES



(a) Correcting 1-bit and detecting 2-bit errors



(b) Incorrectly Correcting a 3-Bit Errors



(c) Correcting no errors but detecting upto 3-bit errors

Fig. 7.4 *Some Code and Non-code Words in an 3-bit Distance 4-code*

Check Your Progress

8. Analyse the Hamming distance.
9. Illustrate the encoding a message.
10. Define the groups codes.
11. Describe the procedure for generating group codes.
12. Elaborate on the error-detecting codes.
13. Interpret the error-correcting codes.

8.7 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. A code is a symbol or group of symbols that stands for something. It is a representation of discrete elements of information, which may be in the form of numbers, letters or any other varying physical quantities. Binary bits 1 and 0 are often used in groups.
2. Binary Coded Decimal (BCD) is one of the early memory codes. It is based on the concept of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. It further uses four digits to represent each of the digits.
3. The abbreviation EBCDIC stands for the Extended Binary Coded Decimal Interchange Code. It is an 8-bit code in which the decimal digits are represented by the 8421 BCD code preceded by 1111.
4. The abbreviation ASCII stands for the American Standard Code for Information Interchange. The ASCII code is a 7-bit code used in transferring coded information from keyboards and to computer displays and printers. It is used to represent numbers, letters, punctuation marks as well as control characters.
5. The Excess-3 is a digital code that is derived by adding to each decimal digit and then converting the result to four-bit binary. The Excess-3 code is used in some arithmetic circuits because it is self-complementing.
6. The Gray code belongs to a class of codes called *minimum change codes*, in which only one bit in the code group changes when going from one step to the next. Gray code is not an arithmetic code.
7. Alphanumeric codes are the codes that represent alphabetic characters (letters), punctuation marks and other special characters. Alphanumeric code represents all of the various characters and functions that are found on a computer keyboard.
8. The Hamming distance between two code words is defined as the number of bits that must be changed for one code word to another. It is actually a method for constructing codes with a minimum distance of 3.

   For any value of i, Hamming code method yields a $(2^i - 1)$—bit code with i check bits and $2^i - 1 - i$ information bits. Distance-3 codes with a smaller number of information bits are obtained by deleting information bits from a Hamming code with a larger number of bits.
9. The encoding of a message is the production of the message. It is a system of coded meanings, and in order to create that, the sender needs to understand how the world is comprehensible to the members of the audience.

NOTES

10. In coding theory, group codes are a type of code. Group codes consist of n linear block codes which are subgroups of G^n, where G is a finite Abelian group. A systematic group code C is a code over G^n of order $|G|^k$ defined by $n - k$ homomorphisms which determine the parity check bits. The remaining k bits are the information bits themselves.
11. Group codes can be constructed by special generator matrices which resemble generator matrices of linear block codes except that the elements of those matrices are endomorphisms of the group instead of symbols from the code's alphabet. For example, considering the generator matrix

$$G = \begin{pmatrix} \begin{pmatrix} 00 \\ 11 \end{pmatrix} \begin{pmatrix} 01 \\ 01 \end{pmatrix} \begin{pmatrix} 11 \\ 01 \end{pmatrix} \\ \begin{pmatrix} 00 \\ 11 \end{pmatrix} \begin{pmatrix} 11 \\ 11 \end{pmatrix} \begin{pmatrix} 00 \\ 00 \end{pmatrix} \end{pmatrix}$$

12. Errors may occur while recording data on magnetic surfaces due to bad spots on the surface. They may also be introduced during data transmission between units, due to electrical disturbances. Most of these errors result in change of a bit from 0 to 1 or 1 to 0. One of the simplest and most commonly used error detection code is called the parity bit.
13. A code that is used to correct errors is called an error-correcting code. In general, if a code has minimum distance 2c+1, it can be used to correct errors, that affect up to *c* bits. If a code's minimum distance is 2c + d + 1, it can be used to correct errors in up to *c* bits and to detect in up to *d* additional bits.

8.8 SUMMARY

- A code is a symbol or group of symbols that stands for something. It is a representation of discrete elements of information, which may be in the form of numbers, letters or any other varying physical quantities. Binary bits 1 and 0 are often used in groups.
- Binary Coded Decimal (BCD) is one of the early memory codes. It is based on the concept of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. It further uses four digits to represent each of the digits.
- The abbreviation EBCDIC stands for the Extended Binary Coded Decimal Interchange Code. It is an 8-bit code in which the decimal digits are represented by the 8421 BCD code preceded by 1111.

NOTES

- The abbreviation ASCII stands for the American Standard Code for Information Interchange. The ASCII code is a 7-bit code used in transferring coded information from keyboards and to computer displays and printers. It is used to represent numbers, letters, punctuation marks as well as control characters.
- The Excess-3 is a digital code that is derived by adding to each decimal digit and then converting the result to four-bit binary. The Excess-3 code is used in some arithmetic circuits because it is self-complementing.
- The Gray code belongs to a class of codes called minimum change codes, in which only one bit in the code group changes when going from one step to the next. Gray code is not an arithmetic code.
- Alphanumeric codes are the codes that represent alphabetic characters (letters), punctuation marks and other special characters. Alphanumeric code represents all of the various characters and functions that are found on a computer keyboard.
- The Hamming distance between two code words is defined as the number of bits that must be changed for one code word to another. It is actually a method for constructing codes with a minimum distance of 3.
- For any value of i, Hamming code method yields a $(2^i - 1)$—bit code with i check bits and $2^i - 1 - i$ information bits. Distance-3 codes with a smaller number of information bits are obtained by deleting information bits from a Hamming code with a larger number of bits.
- The encoding of a message is the production of the message. It is a system of coded meanings, and in order to create that, the sender needs to understand how the world is comprehensible to the members of the audience.
- In coding theory, group codes are a type of code. Group codes consist of n linear block codes which are subgroups of G^n, where G is a finite Abelian group. A systematic group code C is a code over G^n of order $|G|^k$ defined by $n - k$ homomorphisms which determine the parity check bits. The remaining k bits are the information bits themselves.
- Errors may occur while recording data on magnetic surfaces due to bad spots on the surface. They may also be introduced during data transmission between units, due to electrical disturbances. Most of these errors result in change of a bit from 0 to 1 or 1 to 0. One of the simplest and most commonly used error detection code is called the parity bit.
- A code that is used to correct errors is called an error-correcting code. In general, if a code has minimum distance 2c+1, it can be used to correct errors, that affect up to c bits. If a code's minimum distance is 2c + d + 1,

it can be used to correct errors in up to c bits and to detect in up to d additional bits.

NOTES

8.9 KEY WORDS

- **Computer code:** A code is a symbol or group of symbols that stands for something. It is a representation of discrete elements of information.
- **Bits:** A bit is the smallest element used by a computer. It holds one of the two possible values.
- **Nibble:** A nibble is a group of FOUR bits (4 bits). This gives a maximum number of sixteen possible different values.
- **Bytes:** Bytes are a grouping of 8 bits (two nibbles) and are often used to store characters.
- **Binary coded decimal:** Binary coded decimal is one of the early memory codes. It is based on the concept of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form.
- **Excess-3 code:** The excess-3 code is a digital code that is derived by adding to each decimal digit and then converting the result to four-bit binary.
- **Gray code:** The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code group changes when going from one step to the next. Gray code is not an arithmetic code.
- **Alphanumeric codes:** Alphanumeric codes are the codes that represent alphabetic characters (letters).
- **Hamming distance:** The hamming distance between two code words is defined as the number of bits that must be changed for one code to another.
- **Error-detecting codes:** Errors may occur while recording data on magnetic surfaces due to bad spots on the surface. One of the simplest and most commonly used error detection code is called the parity bit.
- **Error-correcting codes:** A code that is used to correct errors is called an error-correcting code.

8.10 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. What do you understand by the computer code?
2. Define the bits and bytes.

3. Elaborate on the binary coded decimal.
4. Illustrate the excess-3 code.
5. State the Gray code.
6. Explain the alphanumeric codes.
7. Analyse the Hamming distance.
8. Interpret the error-detecting codes.
9. Define the error-correcting codes.

Long-Answer Questions

1. Discuss briefly the computer codes with the help of examples.
2. Explain bits, bytes, and nibble. Give appropriate example.
3. Define binary coded decimal. How does it works?
4. Analyse the excess-3 code.
5. Interpret the significances of Hamming distance.
6. Elaborate on the encoding a message.
7. What are the groups codes? Describe the procedure for generating group codes.
8. Analyse the error-detecting codes.
9. Briefly define the error- correcting codes.

8.11 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

NOTES

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

BLOCK - III
MATRIX OF A GRAPH AND
CHROMATIC NUMBERS

NOTES

UNIT 9 GRAPH THEORY

Structure

9.0 INTRODUCTION

Graph theory, which is mainly topological, touch the quantitative as well as qualitative approaches of mathematics. Essentially, graph theory has great possibilities for characterising systems and analysing the information from a set of objects. Graph theory deals with connection amongst points such as vertices, lines, and edges. Indeed, graph theory concerned with the network of points connected by lines. A graph contains a set of vertices, a set of edges and a function. That is, *graphs*, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of *vertices* (also called *nodes* or *points*) which are connected by *edges* (also called *links* or *lines*).

The history of graph theory may be specifically traced to 1735, when the Swiss mathematician Leonhard Euler solved the Königsberg bridge problem. Graph theory had its beginnings basically in recreational math problems, but, nowadays, it is used into a significant area of mathematical research, with applications in chemistry, operations research, social sciences, and computer science. Graphs are one of the prime objects of study in discrete mathematics.

Every graph has its associated graph which is too very important for illustrate the mathematical problems. A graph can be represented by the set of vertices, denoted by V and edges denoted by E. it can be expressed as $G = (V, E)$. A

NOTES

simple graph, also called a strict graph is an unweighted, undirected graph which contains no graph loops or multiple edges. A simple graph may be either connected or disconnected.

A finite graph is a graph in which the vertex set and the edge set are finite sets. Otherwise, it is called an infinite graph. Most commonly in graph theory it is implied that the graphs discussed are finite. If the graphs are infinite, that is usually specifically stated.

In this unit, you will study about the definition of a graph, finite and infinite graphs, incidence, degree, isolated and pendent vertices, isomorphism, sub graphs, walks, paths and circuits, connected and disconnected graphs.

9.1 OBJECTIVES

After going through this unit, you will be able to:

- Define a graph
- Explain the finite and infinite graphs
- Analyse incidence, degree, isolated and pendent vertices
- Elaborate on the isomorphism
- Understand the subgraphs
- Comprehend the walks, paths and circuits
- Interpret the connected and disconnected graphs

9.2 DEFINITION OF A GRAPH

The following are the basic terminologies and symbolic representations of various graphs.

Graph

A graph *G*, a triplet $(V(G), E(G), \theta_G)$ consisting of a non-empty set $V(G)$ of vertices, a set $E(G)$ of edges, and a function θ_G assigns to each edge, a subset $\{u, v\}$ of $V(G)$ (u, v need not be distinct). If e is an edge and u, v are vertices such that $\theta_G(e) = uv$, then e is a line (edge) between u and v; the vertices u and v are the end points of the edge e.

For example, (i) $G = (V(G), E(G), \theta_G)$

Where, $V(G) = \{v_1, v_2, v_3, v_4\}$

$E(G) = \{e_1, e_2, e_3, e_4, e_5, e_6\}$

$\theta_G(e_1) = \{v_1v_2\}$, $\theta_G(e_2) = \{v_2v_2\}$, $\theta_G(e_3) = \{v_2v_3\}$

$\theta_G(e_4) = \{v_1v_3\}$; $\theta_G(e_5) = \{v_4v_5\}$; $\theta_G(e_6) = \{v_1v_4\}$

(*ii*) $G = (V(G), E(G), \theta_G)$

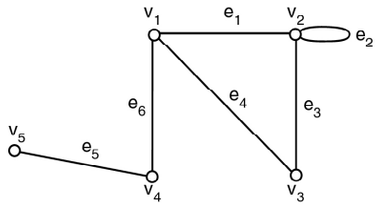
Where, $V(G) = \{v_1, v_2, v_3\}$; $E(G) = \{e_1, e_2, e_3\}$

$\theta_G(e_1) = \{v_1v_2\}$; $\theta_G(e_2) = \{v_2v_3\}$; $\theta_G(e_3) = \{v_3v_1\}$

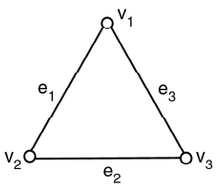
NOTES

Every graph has a diagram associated with it. These diagrams are useful for understanding problems involving such a graph. In the pictorial representation, we represent the vertices by small circles and the edges by lines whenever the corresponding pair of vertices forms an edge.

The following are the pictorial representation of Examples (*i*) and (*ii*):



(i)



(ii)

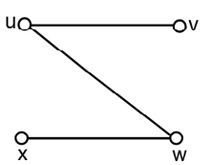
Notes:

1. In Example (*i*), e_2 joins the vertex v_2 to itself. Such an edge is called self loop (loop).
2. Suppose there is more than one edge between a pair of vertices in a graph, these edges are called parallel edges.
3. Hereafter, we denote the graph $G = (V, E)$ for simplicity.
4. A graph, which consists of parallel edges, is called a multigraph.

Simple Graph: A graph with no self-loops and parallel edges is called a simple graph.

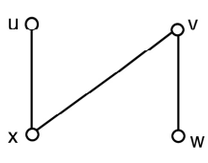
Complement of a Graph: The complement $\overline{G}$ of a graph G is a graph with $V(G) = V(\overline{G})$ and such that uv is an edge of $\overline{G}$ if and only if uv is not an edge of G.

For example,

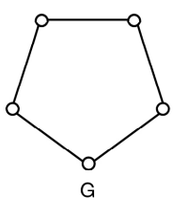


G

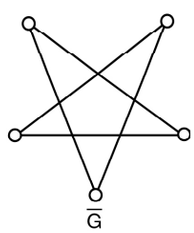
(i)



$\overline{G}$



G



$\overline{G}$

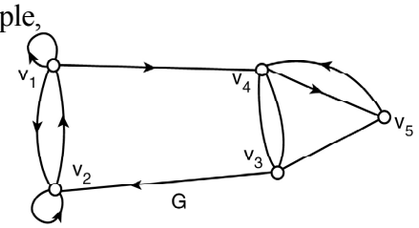
(ii)

NOTES

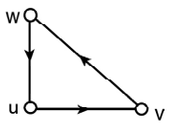
There are also some useful terminology for graphs with directed edges.

Graphs with Directed Edges: When (u, v) is an edge of the graph G with directed edges, u is said to be adjacent to v and v is said to be adjacent from u. The vertex u is called the initial vertex of (u, v) and v is called the terminal or end vertex of the edge (u, v).

For example,



(i)

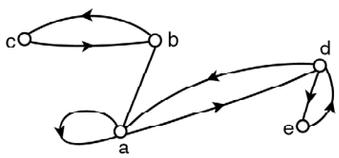


(ii)

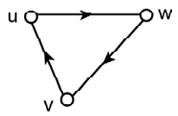
In-Degree and Out-Degree: In a graph with directed edges, the in-degree of a vertex v denoted by $d^-(v)$ is the number of edges with v as their terminal vertex. The out-degree of v denoted by $d^+(v)$ is the number of edges with v as their initial vertex.

Note: Self loop at a vertex contributes 1 to both in-degree and out-degree of this vertex.

Example 9.1: Find the in-degree and out-degree of the following graphs:



(i)



(ii)

Solution: The in-degree and out-degree of Cases (i) and (ii) are as follows:

(*i*) $d^-(a) = 3;\ d^-(b) = 1;\ d^-(c) = 1;\ d^-(d) = 2;$ and $d^-(e) = 1$

$d^+(a) = 2;\ d^+(b) = 2;\ d^+(c) = 1;\ d^+(d) = 2;\ d^+(e) = 1$

(*ii*) $d^-(u) = 1;\ d^-(v) = 1;\ d^-(w) = 1$

$d^+(u) = 1;\ d^+(v) = 1;\ d^+(w) = 1$

Notes:

1. Let $G = (V, E)$ be a graph with directed edges. Then, $\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) = e$.
2. By ignoring directions of edges in a graph with directed edges, we will get an undirected graph. Such graphs are called underlying undirected graphs.

9.2.1 Directed and Undirected Graphs

In a directed graph every edge has a direction (Refer Figure 9.1). If there is movement from a vertex to its adjacent vertex the direction is notified. If movement is from vertex v_1 to vertex v_2, then v_1v_2 and v_2v_1 are different. Here movement is

in one direction only. But in undirected graph, if there is movement in between v_1 and v_2, then movement in both the direction is possible. Such graphs are known as undirected graphs (Refer Figure 9.2).

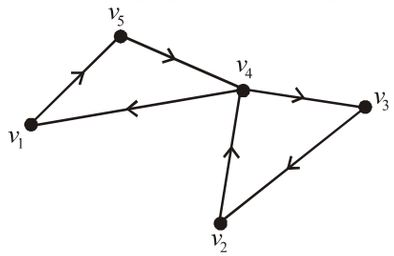


Fig. 9.1 *Directed Graph*

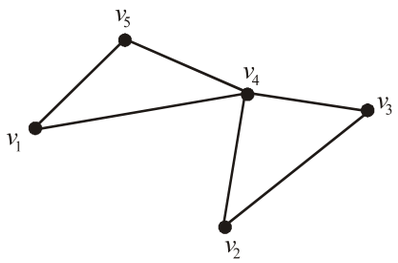


Fig. 9.2 *Undirected Graph*

9.3 FINITE AND INFINITE GRAPHS

In discrete mathematics and more specifically in graph theory, a graph is defined as a structure amounting to a set of objects in which some pairs of the objects are in some capacity 'Related'. The objects correspond to mathematical abstractions called vertices (also sometimes termed as nodes or points) and each of the related pairs of vertices is called an edge (also sometimes termed as link or line). Typically, a graph is depicted in diagrammatic form as a set of dots or circles for the vertices, joined by lines or curves for the edges. Fundamentally, a graph is a collection of vertices connected to each other through a set of edges.

There are finite and infinite graph types. A finite graph is a graph in which the vertex set and the edge set are finite sets. Otherwise, it is called an infinite graph. Most commonly in graph theory it is implied that the graphs discussed are of finite type. If the graphs are infinite, that is usually specifically stated.

Finite Graph: A graph consisting of finite number of vertices and edges is called as a **finite graph**. If the graph has n nodes and has no multiple edges or graph loops (i.e., it is simple), it is a subgraph of the complete graph K_n.

A graph which is not finite is called infinite. If every node has finite degree, the graph is called **locally finite**. The Cayley graph of a group with respect to a finite generating set is always locally finite, even if the group itself is infinite.

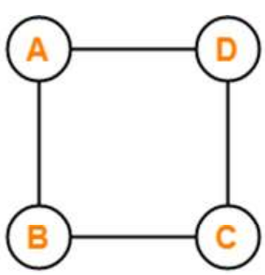


Fig. 9.3 *Illustrates an Example of Finite Graph.*

NOTES

NOTES

The graph shown in Figure 9.3 consists of finite number of vertices and edges. Therefore, it is a finite graph.

Infinite Graph: A graph consisting of infinite number of vertices and edges is called as an **infinite graph**.

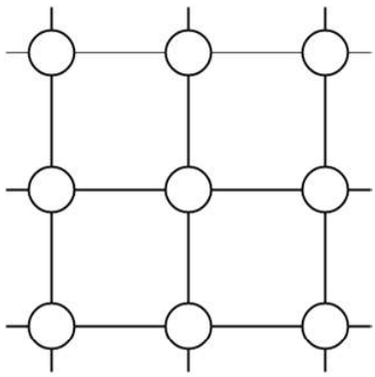


Fig. 9.4 *Illustrates an Example of Infinite Graph.*

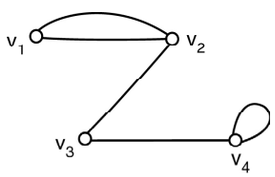
The graph shown in Figure 9.4 consists of consists of infinite number of vertices and edges. Therefore, it is an infinite graph

9.4 INCIDENCE, DEGREE AND PENDENT VERTICES ISOMORPHISM

The concept of incidence associates an edge to the nodes that are connected by that edge. For example, the edge $\{u, v\}$ is incident to the nodes u and v. If there is an edge that connects two nodes, we say that those nodes are adjacent. The degree of a node v, denoted as deg(v), is the number of edges incident to it. In a directed graph, the in-degree and out-degree count the number of directed edges coming into and out of a vertex, respectively.

Degree of a Vertex: The degree of a vertex v is the number of edges incident with that vertex. In other words, the degree of a vertex is the number of edges, having that vertex as an end point, and is denoted by $d(v)$. Figure 9.5 illustrates the degree of vertex.

For example,



Here, $d(v_1) = 2$

$d(v_2) = 3$

$d(v_3) = 2$

$d(v_4) = 3$

Fig. 9.5 *Degree of Vertex*

A loop contributes 2 to the degree of vertex.

NOTES

Isolated Vertex: A vertex with degree zero is called an isolated vertex.

Pendant Vertex: A vertex with degree one is called a pendant vertex.

Adjacent Vertices: A pair of vertices that determine an edge are called adjacent vertices.

Note: A vertex is even or odd if as its degree is even or odd.

Example 9.2: Let G be a simple graph with n vertices. Prove that the number of edges $E(G)$ is atmost ${}^nC_2$.

Solution: Let $G = (V(G), E(G), \theta_G)$ be a simple graph with $|V(G)| = n$.

Since θ_G assigns to each edge, a 2 element subset $\{u, v\}$ of $V(G)$, there are atmost ${}^nC_2$ number of 2 element subsets.

Hence, $E(G) \leq \dfrac{n(n-1)}{2}$

Theorem 9.1: Let G be a graph with n vertices and e edges. Then $\sum_{i=1}^{n} d(v_i) = 2e$

Proof: Let G be a graph with n vertices and e edges.

Since every edge contributes degree 2 to this sum, $\sum_{i=1}^{n} d(v_i) = 2e$

Theorem 9.2: In a graph G, the number of odd vertices is an even number.

Proof: Let G be a graph with n vertices and e edges.

By Theorem 3.1, we have,

$$\sum_{i=1}^{n} d(v_i) = 2e = \text{Even number} \qquad (9.1)$$

Among n vertices, some are even vertices and some are odd vertices. Let V_e and V_o be the even and odd vertices, respectively.

Now Equation (9.1) can be written as,

$$\sum_{v \in V_e}^{n} d(v) + \sum_{v \in V_0} d(v) = \text{Even number}$$

$$\therefore \quad \sum_{v \in V_0} d(v) = \text{Even number} - \sum_{v \in V_e}^{n} d(v) \qquad (9.2)$$

Since every term in the right side of Equation (9.2) is even, the sum on the left side must contain an even number of terms, i.e., the number of odd vertices in G is even.

Minimum and Maximum Degrees: Let G be a graph. The minimum and maximum degrees of G are, respectively, $\delta(G)$ and $\Delta(G)$ and given as:

$$\delta(G) = \min \{d(v); v \in V(G)\}$$
$$\Delta(G) = \max \{d(v); v \in V(G)\}$$

NOTES

k-Regular: A graph G is k-regular or regular of degree k, if every vertex of G has degree k.

Isomorphism

Two graphs G and H are said to be isomorphic if there exists bijections $\psi : V(G) \to V(H)$ and $\phi : E(G) \to E(H)$ such that iff $\theta_G(e) = uv$ iff $\theta_H(\phi(e)) = \psi(u)\ \psi(v)$.

Such a pair of mappings (ψ, ϕ) is called an isomorphism between G and H and is written as $G \cong H$.

In other words, two simple graphs G and H are isomorphic iff there is a bijection $\psi : V(G) \to V(H)$ such that $uv \in E(G)$ iff $\psi(u)\ \psi(v) \in E(H)$. Figure 9.6 illustrates the isomorphic graphs as G and H.
For example,

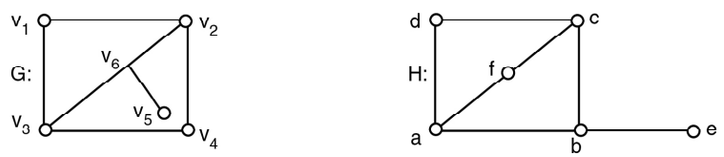


Fig. 9.6 *Isomerphic Graphs*

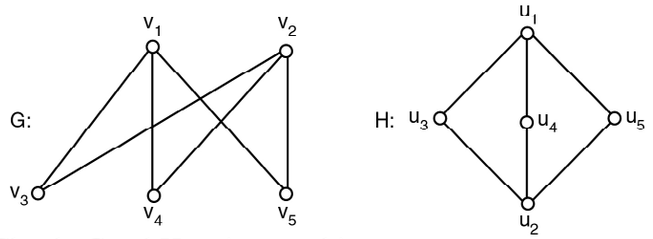
Here, G and H are isomorphic.
The correspondence which gives isomorphism between G and H is as follows:

$$v_1\ v_2 \in E(G) \Leftrightarrow dc = \psi(v_1)\ \psi\ (v_2) \in E(H)$$
$$v_1\ v_3 \in E(G) \Leftrightarrow da = \psi(v_1)\ \psi\ (v_3) \in E(H)$$
$$v_3\ v_6 \in E(G) \Leftrightarrow ab = \psi(v_3)\ \psi\ (v_6) \in E(H)$$
$$v_6\ v_5 \in E(G) \Leftrightarrow be = \psi(v_6)\ \psi\ (v_5) \in E(H)$$
$$v_3\ v_4 \in E(G) \Leftrightarrow af = \psi(v_3)\ \psi\ (v_4) \in E(H)$$
$$v_6\ v_2 \in E(G) \Leftrightarrow bc = \psi(v_6)\ \psi\ (v_2) \in E(H)$$
$$v_4\ v_2 \in E(G) \Leftrightarrow fc = \psi(v_4)\ \psi\ (v_2) \in E(H)$$
$$\therefore \quad G \cong H$$

Notes:

1. Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are said to be isomorphic if a one-to-one correspondence ϕ exists from V_1 to V_2 such that u and v are adjacent in G_1, iff $\phi(u)$ and $\phi(v)$ are adjacent in G_2.
2. If $G \cong H$, then, degrees of corresponding vertices are equal.

Example 9.3: Prove that the graphs G and H are non-isomorphic.



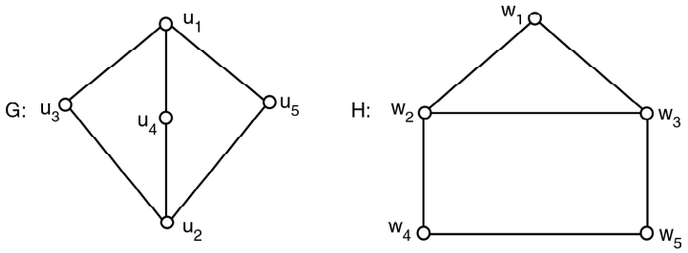
Solution: Clearly *G* and *H* are isomorphic.

In *G*, V_1 is adjacent to the vertices V_3, V_4, V_5 ; V_2 is adjacent to the vertices V_3, V_4, V_5.

In *H*, u_1 is adjacent to u_3, u_4, u_5 and u_2 is adjacent to u_3, u_4, u_5.

Here, the function defined by $\phi(v_i) = u_i$, $1 \le i \le 5$ gives the isomorphism.

Example 9.4: Prove that the graphs *G* and *H* are non-isomorphic.



Solution: Clearly *G* and *H* are non-isomorphic graphs.

In *G*, these two vertices (u_1 and u_2) are adjacent with three other vertices (u_3, u_4, u_5) whereas in *H*, the vertex w_2 is adjacent to w_1, w_2, w_4 and the vertex w_3 is adjacent to w_1, w_2, w_5. w_2 and w_3 are adjacent to each other.

In *G*, u_1 and u_2 are non-adjacent. Hence *G* is not isomorphic to *H*.

Note: From the above example, it is clear that two graphs are isomorphic if they have same number of vertices and same number of edges and the degrees of the corresponding vertices are equal, but the converse is not true.

Check Your Progress

1. What do you understand by graph?
2. Define the complement of a graph.
3. Explain the graphs with directed edges.
4. Elaborate on the in-degree and out-degree.
5. Analyse the finite graphs.
6. Describe the infinite graph
7. Illustrate the degree of vertex.
8. Explain the isolated and pendent vertex.
9. Define the minimum and maximum degree.
10. Elaborate on the isomorphism of graphs.

NOTES

NOTES

9.5 SUB GRAPHS

Let there be a graph given by *G(V, E)*. If another graph (denoted as *H(V', E')*) is obtained by deleting few vertices and edges then it is the subgraph of G, if *V'* in graph *H* contains all the terminal points of edges in *E'*. If we remove an edge, its terminal points remain in place, but if a vertex is removed, then edges that are meeting on this vertex are also removed. Examples of graph and its sub-graph are shown below in Figures 9.7 and 9.8.

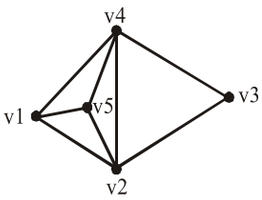


Fig. 9.7 *Graph G*

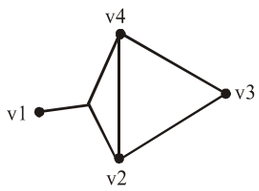


Fig. 9.8 (i) *Sub-graph H of G with Edges, v_1v_4 and v_1v_2 Removed*

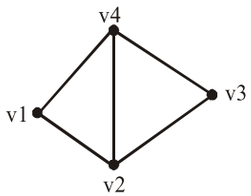


Fig. 9.8 (ii) *Sub-graph H of G with Vertex v_5 Removed.*

9.6 WALK, PATHS AND CIRCUITS

A walk is a sequence of vertices and edges starting from any vertex and travelling through edges to a destination vertex, such that no edge appears more than once. But in a walk a vertex may be visited more than once. Examples of walk are illustrated in Figures 9.9 and 9.10.

Fig. 9.9 *This Shows a Walk with Single Visit on Every Vertex*

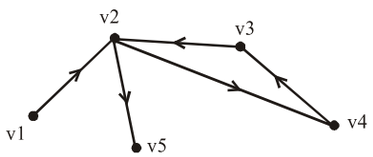


Fig. 9.10 *This Shows a Walk with Two Visits on Vertex v_2*

A path is a sequence of consecutive edges in a graph and the length of the path is the number of edges traversed. Path is thus a sequence of adjacent edges, where the edges are traversed only once. A circuit is a path which ends at the vertex it begins, so a loop is an circuit of length one.

9.7 CONNECTED AND DISCONNECTED GRAPHS

NOTES

In this section, we study the structure of graphs (Refer Figure 9.11). A walk in a graph G is an alternating sequence.

$W : v_0, e_1, v_1, e_2, ..., v_{n-1}, e_n, v_n \ (n \geq 0)$ of vertices and edges, beginning and ending with vertices, such that $e_i = v_{i-1}\ v_i,\ i = 1, 2, ..., n.$ It is denoted by $(v_0 - v_n)$ walk. The number of edges (not necessarily distinct) is called the length of walk. In graph G, $u, e_1, v, e_2, w, e_6, x, e_4, u$ is a walk of length 4.

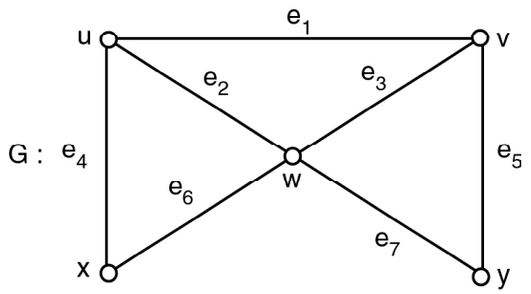


Fig. 9.11 *Structure of Graph*

A trail is a walk in which no edge is repeated and a path is a trail in which no vertex is repeated. Thus, a *path* is a trail, but not every trail is a path. In the above graph G, $x, e_6, w, e_3, v, e_1, u, e_2, w, e_7, y$ is a trail that is not a path, and $u, e_4, x, e_6, w, e_3, v,$ is a path.

Result: Every $(u - v)$ walk in a graph contains a $(u - v)$ path.

Proof: Let W be a $(u - v)$ walk in a graph G. If $u = v$, then w is the trail path, i.e., walk of length zero.

Suppose $u \neq v$ and $W : u = u_0, u_1, u_2, ..., u_n = v.$ If no vertex of G appears in W more than once, then w itself is a $(u - v)$ path. Otherwise, there are vertices of G that occur in w twice or more. Let i and j be distinct positive integers such that $i < j$ with $ui = uj$. Then say $u_i, u_{i+1}, ..., u_{j-2}, u_{j-1}$ are removed from w, and the resulting sequence is $(u - v)$ walk w_1 whose length is less than that of w. (By induction hypothesis, this w_1 contains a $(u - v)$ path and hence w has a $(u - v)$ path). If no vertex of G appears more than once in w_1, then w_1 is a $(u - v)$ path. If not, apply the above procedure, until we get a $(u - v)$ path.

Cycle: A cycle is a walk. $v_0, v_1, ..., v_n$ is a walk in which $n \geq 3,\ v_0 = v_n$ and the 'n'-vertices $v_1, v_2, \ldots, v_n$ are distinct. We say that a $(u - v)$ walk is closed if $u = v$ and open if $u \neq v.$

Connection: Let u and v be vertices in a graph G. We say that u is connected to v if G contains a $(u - v)$ path. The graph G is connected, if u is connected to v for every pair u, v of vertices of G.

Disconnection: A graph G is disconnected, if there exists two vertices u and v for which there is no $(u - v)$ path.

Component: A subgraph H of a graph G is called a component of G, if H is a maximal connected subgraph of G and component is denoted by $\omega(G)$.

Note: If $\omega(G) > 1$, then G is disconnected.

NOTES

For example,

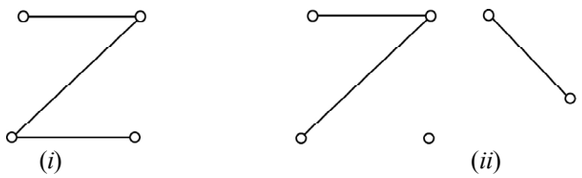


Fig. 9.12 *Components of Graph*

Graph (*i*) is connected and (*ii*) is disconnected.

Note that graph (*ii*) has 3 components.

Connectedness in Directed Graph

Strongly Connected: A directed graph is strongly connected if there is a path from u to v and v to u, whenever u and v are vertices in the graph.

Weakly Connected: A directed graph is weakly connected, if there is a path between any two vertices in the underlying undirected graph.

Unilaterally Connected: A directed graph is said to be unilaterally connected, if in the two vertices u and v, there exists a directed path either from u to v or from v to u.

For example,

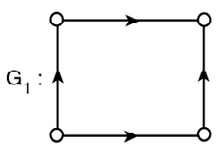


Fig. 9.13 *G_1 is Weakly Connected*

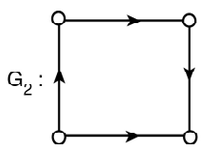


Fig. 9.14 *G_2 Unilaterally Connected*

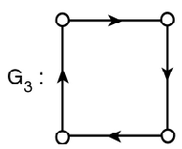


Fig. 9.15 *G_3 is Strongly Connected*

Check Your Progress

11. What do you mean by the subgraph?
12. State the walk of a graph.
13. Interpret the paths of a graph.
14. Analyse the connected graph.
15. Define the disconnected graph.
16. Illustrate the strongly and weakly connected graph.
17. Describe the unilaterally connected graph.

9.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

NOTES

1. A graph G, a triplet $(V(G), E(G), \theta_G)$ consisting of a non-empty set $V(G)$ of vertices, a set $E(G)$ of edges, and a function θ_G assigns to each edge, a subset $\{u, v\}$ of $V(G)$ (u, v need not be distinct). If e is an edge and u, v are vertices such that $\theta_G(e) = uv$, then e is a line (edge) between u and v; the vertices u and v are the end points of the edge e.
2. Complement of a Graph: The complement $\overline{G}$ of a graph G is a graph with $V(G) = V(\overline{G})$ and such that uv is an edge of $\overline{G}$ if and only if uv is not an edge of G.
3. When (u, v) is an edge of the graph G with directed edges, u is said to be adjacent to v and v is said to be adjacent from u. The vertex u is called the initial vertex of (u, v) and v is called the terminal or end vertex of the edge (u, v).
4. In a graph with directed edges, the in-degree of a vertex v denoted by $d^-(v)$ is the number of edges with v as their terminal vertex. The out-degree of v denoted by $d^+(v)$ is the number of edges with v as their initial vertex.
5. Finite Graph: A graph consisting of finite number of vertices and edges is called as a finite graph. If the graph has n nodes and has no multiple edges or graph loops (i.e., it is simple), it is a subgraph of the complete graph K_n.
6. A graph consisting of infinite number of vertices and edges is called as an infinite graph.
7. The degree of a vertex v is the number of edges incident with that vertex. In other words, the degree of a vertex is the number of edges, having that vertex as an end point, and is denoted by $d(v)$.
8. Solated Vertex: A vertex with degree zero is called an isolated vertex.

   Pendant Vertex: A vertex with degree one is called a pendant vertex.
9. Minimum and Maximum Degrees: Let G be a graph. The minimum and maximum degrees of G are, respectively, $\delta(G)$ and $\Delta(G)$ and given as:

   $$\delta(G) = \min \{d(v); v \in V(G)\}$$
   $$\Delta(G) = \max \{d(v); v \in V(G)\}$$
10. Two graphs G and H are said to be isomorphic if there exists bijections $\psi : V(G) \to V(H)$ and $\phi : E(G) \to E(H)$ such that iff
    $\theta_G(e) = uv$ iff $\theta_H(\phi(e)) = \psi(u)\, \psi(v)$.

NOTES

11. Let there be a graph given by *G(V, E)*. If another graph (denoted as *H(V', E')*) is obtained by deleting few vertices and edges then it is the subgraph of G, if *V'* in graph *H* contains all the terminal points of edges in *E'*.
12. A walk is a sequence of vertices and edges starting from any vertex and travelling through edges to a destination vertex, such that no edge appears more than once.
13. A path is a sequence of consecutive edges in a graph and the length of the path is the number of edges traversed. Path is thus a sequence of adjacent edges, where the edges are traversed only once.
14. Let *u* and *v* be vertices in a graph *G*. We say that *u* is connected to *v* if *G* contains a $(u - v)$ path. The graph *G* is connected, if *u* is connected to *v* for every pair *u*, *v* of vertices of *G*.
15. A graph *G* is disconnected, if there exists two vertices *u* and *v* for which there is no $(u - v)$ path.
16. Strongly Connected: A directed graph is strongly connected if there is a path from u to v and v to u, whenever u and v are vertices in the graph.

    Weakly Connected: A directed graph is weakly connected, if there is a path between any two vertices in the underlying undirected graph.
17. Unilaterally Connected: A directed graph is said to be unilaterally connected, if in the two vertices u and v, there exists a directed path either from u to v or from v to u.

9.9 SUMMARY

- A graph *G*, a triplet $(V(G), E(G), \theta_G)$ consisting of a non-empty set $V(G)$ of vertices, a set $E(G)$ of edges, and a function θ_G assigns to each edge, a subset $\{u, v\}$ of $V(G)$ (u, v need not be distinct). If e is an edge and u, v are vertices such that $\theta_G(e) = uv$, then e is a line (edge) between u and v; the vertices u and v are the end points of the edge e.
- Complement of a Graph: The complement $\overline{G}$ of a graph G is a graph with $V(G) = V(\overline{G})$ and such that uv is an edge of $\overline{G}$ if and only if uv is not an edge of G.
- When (u, v) is an edge of the graph G with directed edges, u is said to be adjacent to v and v is said to be adjacent from u. The vertex u is called the initial vertex of (u, v) and v is called the terminal or end vertex of the edge (u, v).
- In a graph with directed edges, the in-degree of a vertex v denoted by $d^-(v)$ is the number of edges with v as their terminal vertex. The out-degree of v denoted by $d^+(v)$ is the number of edges with v as their initial vertex.

NOTES

- Finite Graph: A graph consisting of finite number of vertices and edges is called as a finite graph. If the graph has n nodes and has no multiple edges or graph loops (i.e., it is simple), it is a subgraph of the complete graph K_n.
- A graph consisting of infinite number of vertices and edges is called as an infinite graph.
- The degree of a vertex *v* is the number of edges incident with that vertex. In other words, the degree of a vertex is the number of edges, having that vertex as an end point, and is denoted by $d(v)$.
- Solated Vertex: A vertex with degree zero is called an isolated vertex.

  Pendant Vertex: A vertex with degree one is called a pendant vertex.
- Minimum and Maximum Degrees: Let G be a graph. The minimum and maximum degrees of G are, respectively, $\delta(G)$ and $\Delta(G)$ and given as:

$$\delta(G) = \min \{d(v); v \in V(G)\}$$
$$\Delta(G) = \max \{d(v); v \in V(G)\}$$

- Two graphs *G* and *H* are said to be isomorphic if there exists bijections $\psi : V(G) \to V(H)$ and $\phi : E(G) \to E(H)$ such that iff

  $\theta_G(e) = uv$ iff $\theta_H(\phi(e)) = \psi(u)\ \psi(v)$.
- Let there be a graph given by *G(V, E)*. If another graph (denoted as *H(V', E')*) is obtained by deleting few vertices and edges then it is the subgraph of G, if *V'* in graph *H* contains all the terminal points of edges in *E'*.
- A walk is a sequence of vertices and edges starting from any vertex and travelling through edges to a destination vertex, such that no edge appears more than once.
- A path is a sequence of consecutive edges in a graph and the length of the path is the number of edges traversed. Path is thus a sequence of adjacent edges, where the edges are traversed only once.
- Let *u* and *v* be vertices in a graph *G*. We say that *u* is connected to *v* if *G* contains a $(u - v)$ path. The graph *G* is connected, if *u* is connected to *v* for every pair *u*, *v* of vertices of *G*.
- A graph *G* is disconnected, if there exists two vertices *u* and *v* for which there is no $(u - v)$ path.
- Strongly Connected: A directed graph is strongly connected if there is a path from u to v and v to u, whenever u and v are vertices in the graph.
- Weakly Connected: A directed graph is weakly connected, if there is a path between any two vertices in the underlying undirected graph.

- Unilaterally Connected: A directed graph is said to be unilaterally connected, if in the two vertices u and v, there exists a directed path either from u to v or from v to u.

NOTES

9.10 KEY WORDS

- **Graph:** A graph G, a triplet ($V(G)$, $E(G)$, θ_G) consisting of a non-empty set $V(G)$ of vertices, a set $E(G)$ of degree, and a function θ_G assigns to each edge, a subset $\{u,v\}$ of $V(G)$ (u,v need not be distinct).
- **Simple graph:** A graph with no self-loop and parallel edges is known as simple graph.
- **Graph with directed edge:** When (u,v) is an edge of the graph G with directed edges, u said to be adjacent to v and v is said to be adjacent from u.
- **In-degree and out-degree:** In a graph with directed edges, the in-degree of a vertex v denoted by $d^-(v)$ is the number of edges with v as their terminal vertex. The out degree of v denoted by $d^+(v)$ is the number of edges with v as their initial vertex.
- **Degree of vertex:** The degree of a vertex v is the number of edges incident with that vertex.
- **Isolated vertex:** A vertex with degree zero is called an isolated vertex.
- **Pendant vertex:** A vertex with degree one is called a pendant vertex.
- **Adjacent vertices:** A pair of vertices that determine an edge are called adjacent vertices.
- ***k*-regular graph:** A graph G is k-regular or regular of degree k, if every vertex of G has degree k.
- **Subgraph:** Let there be a graph given by $G(V,E)$. If another graph (denoted as $H(V',E')$) is obtained by deleting few vertices and edges then it is the subgraph of G, if V' in graph H contains all the terminal points of edges in E'.
- **Walk:** A walk is a sequence of vertices and edges starting from any vertex and travelling through edges to a destination vertex, such that no edge appears more than once.
- **Path:** A path is a sequence of consecutive edges in a graph and the length of the path is the number of edges traversed.
- **Component:** a sub-group H of a group G is called a component of G, if H is a maximal connected subgroup of G and component is denoted by $\omega(G)$.

9.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

NOTES

Short-Answer Questions

1. Give the definition of graph.
2. Elaborate on the complement of a graph.
3. Explain the graphs with directed edges.
4. Illustrate the in-degree and out-degree.
5. Define the finite and infinite graphs.
6. Analyse the term incidence in graph theory.
7. State the degree of a vertex.
8. Describe the isolated and pendent vertex.
9. Explain the isomorphism of graph.
10. Give the definition of subgraph.
11. Interpret the walk and path.
12. Define connected and disconnected graph.

Long-Answer Questions

1. Discuss briefly the graph with suitable examples.
2. Explain the graph with directed edges.
3. Analyse the finite and infinite graphs. Give appropriate examples.
4. Elaborate on the degree of vertex.
5. Describe briefly the isomorphism of a graph.
6. Illustrate the subgraphs with the help of examples.
7. Define the terms walks, paths, and circuits.
8. Interpret the connected and disconnected graphs.

9.12 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

NOTES

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 10 CIRCUIT MATRIX

NOTES

Structure

10.0 INTRODUCTION

Matrix structures include the incidence matrix, a matrix of 0's and 1's whose rows represent vertices and whose columns represent edges, and the adjacency matrix, in which both the rows and columns are indexed by vertices. In both cases a 1 indicates two adjacent objects and a 0 indicates two non-adjacent objects. The degree matrix indicates the degree of vertices. The Laplacian matrix is a modified form of the adjacency matrix that incorporates information about the degrees of the vertices, and is useful in some calculations such as Kirchhoff's theorem on the number of spanning trees of a graph. The distance matrix, like the adjacency matrix, has both its rows and columns indexed by vertices, but rather than containing a 0 or a 1 in each cell it contains the length of a shortest path between two vertices. List structures include the edge list, an array of pairs of vertices, and the adjacency list, which separately lists the neighbors of each vertex: Much like the edge list, each vertex has a list of which vertices it is adjacent to.

Matrix structures on the other hand provide faster access for some applications but can consume huge amounts of memory. Implementations of sparse matrix structures that are efficient on modern parallel computer architectures are an object of current investigation. An incidence matrix is a logical matrix that shows the relationship between two classes of objects, usually called an incidence relation. If the first class is X and the second is Y, the matrix has one row for each element of X and one column for each element of Y. The entry in row x and column y is 1 if x and y are related (called incident in this context) and 0 if they are not.

NOTES

A cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut, and in some cases cuts are identified with their cut-sets rather than with their vertex partitions.

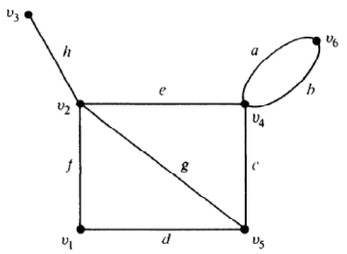
In this unit, you will study about the matrix representation of a graph, incidence matrix, circuit matrix, fundamental circuit matrix and the rank of the circuit matrix, cut set matrix, and adjacency matrix.

10.1 OBJECTIVES

After going through this unit, you will be able to:

- Explain the matrix representation of a graph
- Elaborate on the incidence matrix
- Illustrate the circuit matrix
- Understand the fundamental circuit matrix and rank of the circuit matrix
- Comprehend the cut set matrix
- Analyse the adjacency matrix

10.2 MATRIX REPRESENTATION OF A GRAPH



(a)

	a	b	c	d	e	f	g	h
v_1	0	0	0	1	0	1	0	0
v_2	0	0	0	0	1	1	1	1
v_3	0	0	0	0	0	0	0	1
v_4	1	1	1	0	1	0	0	0
v_5	0	0	1	1	0	0	1	0
v_6	1	1	0	0	0	0	0	0

(b)

Fig. 10.1 *Graph and its incidence matrix*

NOTES

Such a matrix A is called the *vertex-edge incidence matrix,* or simply *incidence matrix.* Matrix A for a graph G is sometimes also written as A(G). A graph and its incidence matrix are shown in Refer Figure 10.1.

The incidence matrix contains only two elements, 0 and 1. Such a matrix is called a *binary matrix* or a *(0, Ti-matrix,* Let us stipulate that these two elements are from Galois field modulo 2. Given any geometric representation of a graph without self-loops, we can readily write its incidence matrix.

Although matrices are customarily defined over a commutative ring with identity, which need not be a field (such as the ring of integers), we have defined matrix A over a field, GF(2), in keeping with our definition of the vector space *WG.*

On the other hand, if we are given an incidence matrix A(G), we can construct its geometric graph G without ambiguity. The incidence matrix and the geometric graph contain the same informationt–they are simply two alternative ways of representing the same (abstract) graph.

The following observations about the incidence matrix A can readily be made:

1. Since every edge is incident on exactly two vertices, each column of A has exactly two 1'so
2. The number of 1's in each row equals the degree of the corresponding vertex.
3. A row with all O's, therefore, represents an isolated vertex.
4. Parallel edges in a graph produce identical columns in its incidence matrix, for example, columns 1 and 2 in Refer Figure 10.1.
5. If a graph G is disconnected and consists of two components *gland* g_1 and g_2 the incidence matrix A(G) of graph G can be written in a block- diagonal form as:

$$A(G) = \begin{bmatrix} A(g_1) & 0 \\ 0 & A(g_2) \end{bmatrix}, \qquad (10.1)$$

   Where $A(g_1)$ and $A(g_2)$ are the incidence matrices of components g_1 and g_2. This observation results from the fact that no edge in g_1 is incident on vertices of g_2, and vice versa. Obviously, this remark is also true for a disconnected graph with any number of components.
6. Permutation of any two rows or columns in an incidence matrix simply corresponds to relabeling the vertices and edges of the same graph. This observation leads us to Theorem 10.1.

Theorem 10.1: Two graphs G_1 and G_2 are isomorphic if and only if their incidence matrices A(G_1) and A(G_2) differ only by permutations of rows and columns.

Rank of the Incidence Matrix: Each row in an incidence matrix A(G) may be regarded as a vector over GF(2) in the vector space of graph G. Let the vector in the first row be called A_1, in the second row A_2, and so on. Thus,

NOTES

$$A(G) = \begin{bmatrix} A_1 \\ A_2 \\ \cdot \\ \cdot \\ \cdot \\ A_n \end{bmatrix}, \qquad (10.2)$$

Since there are exactly two 1's in every column of A, the sum of all these vectors is 0 (this being a modulo 2 sum of the corresponding entries). Thus vectors $A_1, A_2, \ldots, A_n$ are not linearly independent. Therefore, the rank of A is less than *n;* that is, rank A $\leq n - 1$.

Now consider the sum of any *m* of these *n* vectors *(m* $\leq n - 1$). If the graph is connected, A(G) cannot be partitioned, as in Equation (10.1), such that $A(g_1)$ is with *m* rows and $A(g_2)$ with *n–m* rows. In other words, no *m* by *m* submatrix of A(*G*) can be found, for $m \leq n - 1$, such that the modulo 2 sum of those *m* rows is equal to zero.

Since there are only two constants 0 and I in this field, the additions of all vectors taken *m* at a time for $m = 1, 2, \ldots, n - 1$ exhausts all possible linear combinations of $n - 1$ row vectors. Thus we have just shown that no linear combination of *m* row vectors of A (for $m \leq n - 1$) can be equal to zero. Therefore, the rank of A(*G*) must be at least $n - 1$.

Since the rank of A(*G*) is no more than $n - 1$ and is no less than $n - 1$, it must be exactly equal to $n - 1$. Hence Theorem 10.2.

Theorem 10.2: If A(*G*) is an incidence matrix of a connected graph G with *n* vertices, the rank of A(*G*) is $n - 1$.

The argument leading to Theorem 10.2 can be extended to prove that the rank of A(*G*) is $n - k$, if *G* is a disconnected graph with *n* vertices and *k* components. This is the reason why the number $n - k$ has been called the rank of a graph with *k* components.

If we remove anyone row from the incidence matrix of a connected graph, the remaining *(n* – 1) by *e* submatrix is of rank $n - 1$ (Theorem 10.2). In other words, the remaining $n - 1$ row vectors are linearly independent. Thus we need only $n - 1$ rows of an incidence matrix to specify the corresponding graph completely, for $n - 1$ rows contain the same amount of information as the entire matrix. (This is obvious, since given $n - 1$ rows we can easily reconstitute the missing row, because each column in the matrix has exactly two I's.)

Such an *(n* – 1) by *e* submatrix A_f of A is called a *reduced incidence matrix.* The vertex corresponding to the deleted row in A_f is called the *reference vertex.* Clearly, any vertex of a connected graph can be made the reference vertex.

Since a tree is a connected graph with *n* vertices and $n - 1$ edges, its reduced incidence matrix is a square matrix of order and rank $n - 1$.

10.3 INCIDENCE MATRIX

NOTES

To any graph G, there corresponds a $V \times E$ matrix called the incidence matrix of G and is denoted by $I(G) = [a_{ij}]_{V \times E}$, where,

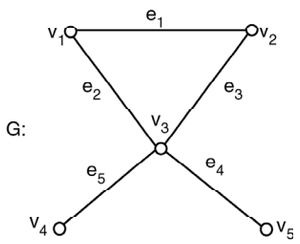
$$a_{ij} = \begin{cases} 1, \text{ if } j\text{th edge is incident with } i\text{th vertex} \\ 0, \text{ otherwise} \end{cases}$$

One more matrix associated with graph G is the adjacency matrix, e is denoted by $A(G) = [b_{ij}]_{V \times V}$,

$$a_{ij} = \begin{cases} 1, \text{ if } j\text{th edge is incident with } i\text{th vertex} \\ 0, \text{ otherwise} \end{cases}$$

Some authors used to define a_{ij} as the number of times (0, 1, and 2) v_i and e_j are incident ; b_{ij} is the number of edges v_i and v_j.

For example,



	e_1	e_2	e_3	e_4	e_5
v_1	1	1	0	0	0
v_2	1	0	1	0	0
v_3	0	1	1	1	1
v_4	0	0	0	0	1
v_5	0	0	0	1	0

$I(G)$, Incidence Matrix of G

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	1	0	1	0	0
v_3	1	1	0	1	1
v_4	0	0	0	1	0
v_5	0	0	0	1	0

$A(G)$, Adjacency Matrix of G

Fig. 10.2 *Matrix Representation of Graphs*

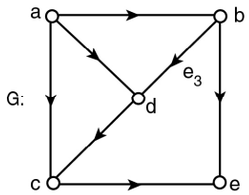
NOTES

The adjacency matrix $A(G) = [b_{ij}]$ of a directed graph is also a $V \times V$ matrix,

$$\text{Where,} \quad b_{ij} = \begin{cases} 1, \text{ if there is a directed edge from } v_i \text{ to } v_j \\ 0, \text{ otherwise} \end{cases}$$

Similarly one can define the incidence matrix of a directed graph as shown in Figure 10.3.

For example,

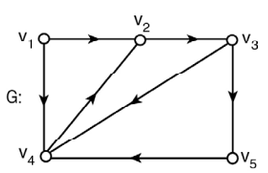


$$A(G) = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

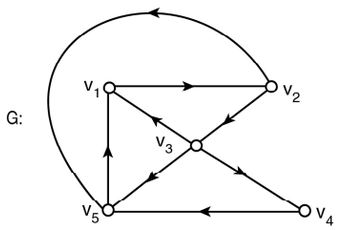
Fig. 10.3 *Incidence Matrix of a Directed Graph*

Example 10.1: Write the adjacency matrix of the following graphs:

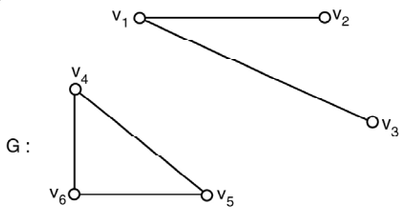
(*i*)



(*ii*)

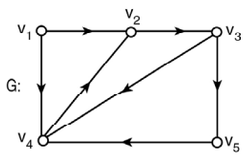


(*iii*)



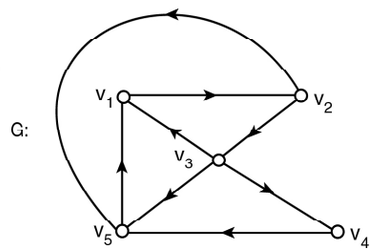
Solution: The adjency matrix of the graphs are as follows:

(*i*)



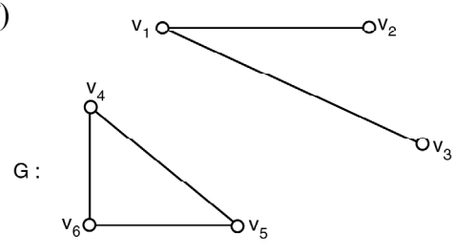
$$A(G) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(*ii*)



$$A(G) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(*iii*)



$$A(G) = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

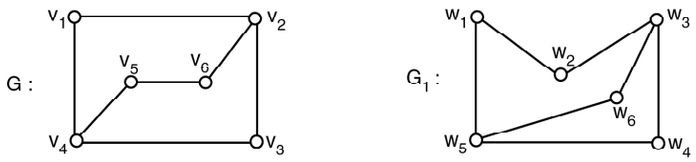
Notes: From Example 10.1 one can conclude that:

1. The diagonal entries of an adjacency matrix are all zero, iff the graph is a graph with no self-loops.
2. If G is disconnected and it has two components, then its adjacency matrix $A(G)$ can be written as,

   $A(G) = \begin{bmatrix} A(G_1) & 0 \\ 0 & A(G_2) \end{bmatrix}$, G_1 and G_2 are components.

   With the help of these matrices, one can verify whether the given graphs are isomorphic or not.

Example 10.2: Verify if G and G_1 are isomorphic.



Solution: First we shall write the adjacency matrices of G and G_1.

$$A(G) = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \quad A(G_1) = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

By keeping one matrix fixed, and by applying permutation of rows and corresponding columns permutations on the unfixed matrix, yields the fixed one. Then the given graphs are isomorphic.

Here, keep $A(G)$ fixed.

NOTES

Also, G and G_1 have 4 vertices of degree 2 and two vertices of degree 3. Since $d(v_1) = 2$ and v_1 is not adjacent to any other vertex of degree 2, corresponding vertex in G_1 is either w_4 or w_6, the only vertices of degree 2 in G_1 not adjacent to a vertex of degree 2.

Without loss of generality, let us take $v_1 \to w_6$. Suppose this $v_1 \to w_6$ is not ending with isomorphism, we have to take $v_1 \to w_4$.

Similarly, for other vertices of G, we can set

$v_2 \to w_3; v_3 \to w_4; v_4 \to w_5; v_5 \to v_1;\ \ v_6 \to v_2$.

Thus we can modify $A(G_1)$ as

$$A(G_1) = \begin{array}{c} \\ w_6 \\ w_3 \\ w_4 \\ w_5 \\ w_1 \\ w_2 \end{array} \begin{array}{c} \begin{array}{cccccc} w_6 & w_3 & w_4 & w_5 & w_2 & w_1 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$\therefore\ A(G) = A(G_1)$ and hence $G \cong G_1$.

10.4 CIRCUIT MATRIX

Let the number of different circuits in a graph G be q and the number of edges in G be e. Then a *circuit matrix* $B = [b_{ij}]$ of G is a q *bye,* (0, 1)-matrix defined as follows:

$b_{ij} = 1$, if ith circuit includes jth edge, and

$=0$, otherwise.

To emphasize the fact that B is a circuit matrix of graph G, the circuit matrix may also be written as B(G).

The graph in Figure 10.1(a) has four different circuits, $\{a, b\}$, $\{c, e, g\}$, $\{d, f, g\}$, and $\{c, d, f, e\}$. Therefore, its circuit matrix is a 4 by 8, (0, 1)-matrix as shown:

$$B(G) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \end{array} \qquad (10.3)$$

The following observations can be made about a circuit matrix B(G) of a graph G:

1. A column of all zeros corresponds to a noncircuit edge (i.e., an edge that does not belong to any circuit).

NOTES

2. Each row of B(G) is a circuit vector.
3. Unlike the incidence matrix, a circuit matrix is capable of representing a self-loop– the corresponding row will have a single I.
4. The number of I's in a row is equal to the number of edges in the corresponding circuit.
5. If graph G is separable (or disconnected) and consists of two blocks (or components) g_1 and g_2, the circuit matrix B(G) can be written in a block-diagonal form as:

$$B(G) = \begin{bmatrix} B(g_1) & 0 \\ 0 & B(g_2) \end{bmatrix},$$

   Where $B(g_1)$ and $B(g_2)$ are the circuit matrices of gland g_1 and g_2. This observation results from the fact that circuits in $g1$ have no edges belonging to g_2, and vice versa.
6. Permutation of any two rows or columns in a circuit matrix simply corresponds to relabeling the circuits and edges.
7. Two graphs G_1 and G_2 will have the same circuit matrix if and only if G_1 and G_2 are 2-isomorphic. In other words, (unlike an incidence matrix) the circuit matrix does not specify a graph completely. It only specifies the graph within 2-isomorphism. For instance, it can be easily verified that the two graphs have the same circuit matrix, yet the graphs are not isomorphic.

An important theorem relating the incidence matrix and the circuit matrix of a self-loop-free graph G is

Theorem 10.3: Let B and A be, respectively, the circuit matrix and the incidence matrix (of a self-loop-free graph) whose columns are arranged using the same order of edges. Then every row of B is orthogonal to every row A; that is,

$$A \cdot B^T = B \cdot A^T = 0 \qquad (\text{mod } 2), \qquad (10.4)$$

Where superscript T denotes the transposed matrix.

Proof: Consider a vertex υ and a circuit Γ in the graph G. Either υ is in Γ or it is not. If υ is not in Γ, there is no edge in the circuit Γ that is incident on υ. On the other hand, if υ is in Γ, the number of those edges in the circuit Γ that are incident on υ is exactly two.

With this remark in mind, consider the ith row in A and the jth row in B. Since the edges are arranged in the same order, the nonzero entries in the corresponding positions occur only if the particular edge is incident on the ith vertex and is also in the jth circuit.

If the ith vertex is not in the jth circuit, there is no such non-zero entry, and the dot product of the two rows is zero. If the ith vertex is in the jth circuit, there will be exactly two 1's in the sum of the products of individual entries. Since $1 + 1 = 0$ (mod 2), the dot product of the two arbitrary rows– one from A and the other from B– is zero. Hence the theorem.

NOTES

As an example, let us multiply the incidence matrix and transposed circuit of the graph in Figure 10.1(a), after making sure that the edges are in the same order in both.

$$A \cdot B^T = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{(mod 2).}$$

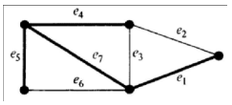
10.5 FUNDAMENTAL CIRCUIT MATRIX AND RANK OF THE CIRCUIT MATRIX

A set of fundamental circuits (or basic circuits) with respect to any spanning tree in a connected graph, are the only independent circuits in a graph. The rest of the circuits can be obtained as ring sums (i.e., linear combinations) of these circuits. Thus, in a circuit matrix, if we retain only those rows that correspond to a set of fundamental circuits and remove all other rows, we would not lose any information. The remaining rows can be reconstituted from the rows corresponding to the set of fundamental circuits. For example, in the circuit matrix in Equation (10.3), the fourth row is simply the mod 2 sum of the second and third rows.

A submatrix (of a circuit matrix) in which all rows correspond to a set of fundamental circuits is called a *fundamental circuit matrix* B_f. A graph and its fundamental circuit matrix with respect to a spanning tree (indicated by heavy lines) are shown in Figure 10.4.

As in matrices A and B, permutations of rows (and/or of columns) do not affect B_f. If n is the number of vertices and e the number of edges in a connected graph, then B_f is an $(e - n + 1)$ by e matrix, because the number of fundamental circuits is $e - n + 1$, each fundamental circuit being produced by one chord.

Let us arrange the columns in B_f such that all the $e - n + 1$ chords correspond to the first $e - n + 1$ columns. Furthermore, let us rearrange the rows such that the first row corresponds to the fundamental circuit made by the chord in the first column, the second row to the fundamental circuit made by the second, and so on. This indeed is how the fundamental circuit matrix is arranged in Figure 10-4(b).



(a)

e_2	e_3	e_6	e_1	e_4	e_5	e_7
1	0	0	1	1	0	1
0	1	0	0	1	0	1
0	0	1	0	0	1	1

(b)

Fig 10.4 *Graph and its fundamental circuit matrix (with respect to the spanning tree shown in heavy lines)*

A matrix B_f thus arranged can be written as

$$B_f = [I_\mu | B_t] \qquad (10.5)$$

Where I_μ is an identity matrix of order $\mu = e - n + 1$, and B, is the remaining μ by $(n - 1)$ submatrix, corresponding to the branches of the spanning tree.

From Equation (10.5) it is clear that the

Rank of $B_f = \mu = e - n + 1$.

Since B_f is a submatrix of the circuit matrix B, the

Rank of B $\geq e - n + 1$.

In fact, we can prove Theorem 10.4.

Theorem 10.4: If B is a circuit matrix of a connected graph *G* with *e* edges and *n* vertices, rank of B = $e - n + 1$.

Proof: If A is an incidence matrix of *G*, from Equation (10.4) we have

$$A \cdot B^T = 0 \pmod 2.$$

Therefore, according to Sylvester's theorem (Appendix B),

rank of A + rank of B $\leq e$;

That is,

rank of B $\leq e$ – rank of A.

Since rank of A = $n - 1$

We have rank of B $\leq e - n + 1$.

But rank of B $\geq$ e – n + 1.

Therefore, we must have

Rank of B = e - n + 1.

NOTES

NOTES

An Alternative Proof: Theorem 10.4 can also be proved by considering the circuit subspace W_Γ in the vector space W_G of a graph.

Every row in circuit matrix B is a vector in W_Γ, and since the rank of any matrix is equal to the number of linearly independent rows (or columns) in the matrix, we have.

Rank of matrix B = number of linearly independent rows in B;

but the number of linearly independent rows in B ≤ number of linearly independent vectors in W_Γ, and the number of linearly independent vectors in W_Γ dimension of $W_\Gamma = \mu$. Therefore, rank of B ≤ $e - n + 1$. Since we already showed that rank of B ≥ $e - n + 1$, Theorem 10.4 follows.

Note that in talking of spanning trees of a graph G it is necessary to assume that G is connected. In the case of a disconnected graph, we would have to consider a spanning forest and fundamental circuits with respect to this forest. It is not difficult to show (considering component by component) that if G is a disconnected graph with k components, e edges, and n vertices,

$$\text{Rank of B} = \mu = e - n + k.$$

10.6 CUT SET MATRIX

Analogous to a circuit matrix, we can define a *cut-set matrix* C = $[c_{ij}]$ in which the rows correspond to the cut-sets and the columns to the edges of the graph, as follows:

$c$ij = 1, if i_{th} cut-set contains j_{th} edge, and

= 0, otherwise

For example, a graph and its cut-set matrix are shown in Figure 10.5.

The following remarks may be made about a cut-set matrix C(G) of a graph G.

1. As in the case of the incidence matrix, a permutation of rows or columns in a cut-set matrix corresponds simply to a renaming of the cut-sets and edges, respectively.
2. Each row in C(G) is a cut-set vector.
3. A column with all 0's corresponds to an edge forming a self-loop.
4. Parallel edges produce identical columns in the cut-set matrix (e.g., first two columns in Figure 10.5).
5. In a nonseparable graph, every set of edges incident on a vertex is a cut-set. Therefore, every row of incidence matrix A(G) is included as a row in the cut-set matrix C(G). In other words, for a nonseparable graph G, C(G) contains A(G). For a separable graph, the incidence matrix of each block is contained in the cut-set matrix. For example, the incidence matrix of the

block {*c, d, e, f, g*} in Figure 10.5 is the 4 by 5 submatrix of C left after deleting rows *a, b,* and *h* and columns 1, 2, 5, and 8.

6. In view of observation 5,

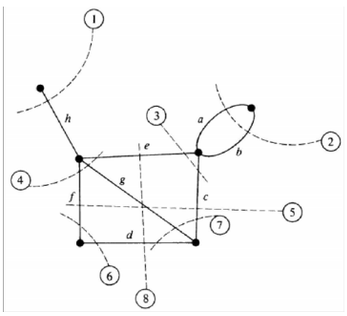
$$\text{rank of C}(G) \geq \text{rank of A}(G).$$

Hence, for a connected graph of n vertices,

$$\text{rank of C}(G) \geq n - 1. \tag{10.6}$$

7. Since the number of edges common to a cut-set and a circuit is always even, every row in C is orthogonal to every row in B, provided the edges in both Band C are arranged in the same order. In other words,

$$\text{B} \cdot \text{C}^T = \text{C}.\ B^T = 0 \pmod 2. \tag{10-7}$$



$$\text{C} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} \begin{array}{c} \begin{array}{cccccccc} a & b & c & d & e & f & g & h \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Fig. 10.5 *Graph and its cut-set matrix.*

On applying Sylvester's theorem to Equation (10.7),

$$\text{Rank of B} + \text{rank of C} \leq e,$$

And since for a connected graph

$$\text{Rank of B} = e - n + 1,$$

$$\text{Rank of C} \leq n - 1. \tag{10.8}$$

Combining Equations (10.6) and (10.8),

$$\text{Rank of C} = n - 1.$$

Thus we have the following important theorem for a connected graph G.

NOTES

NOTES

Theorem 10.5: The rank of cut-set matrix C(*G*) is equal to the rank of the incidence matrix A(G), which equals the rank of graph *G*.

As in the case of the circuit matrix, the cut-set matrix generally has many redundant (or linearly dependent) rows. Therefore, it is convenient to define a fundamental cut-set matrix, C_f as follows:

A fundamental cut-set matrix C_f (of a connected graph *G* with *e* edges and *n* vertices) is an $(n-1)$ by *e* submatrix of C such that the rows correspond to the set of fundamental cut-sets with respect to some spanning tree.

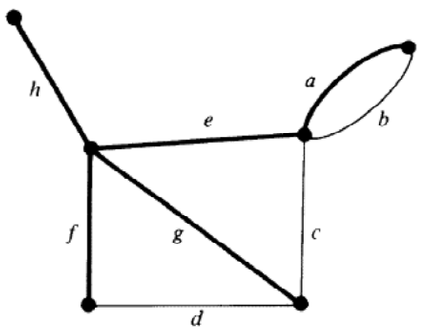
As in the case of a fundamental circuit matrix, a fundamental cut-set matrix C_f can also be partitioned into two submatrices, one of which is an identity matrix I_{n-1} of order $n-1$. That is,

$$C_f = [C_c | I_n - 1] \qquad 10.9$$

Where the last $n-1$ columns forming the identity matrix correspond to the $n-1$ branches of the spanning tree, and the first $e-n+1$ columns forming C, correspond to the chords.

A connected graph and a fundamental cut-set matrix with respect to a spanning tree (shown in heavy lines) are given in Figure 10.6.

Again note that in talking of cut-set matrices we have confined ourselves to connected graphs only. This treatment can be generalized to include disconnected graphs by considering one component at a time.



$$C_f = \begin{array}{c} \begin{matrix} b & c & d & a & e & f & g & h \end{matrix} \\ \left[\begin{array}{ccc:ccccc} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} = [C_c \vdots I_5]$$

Fig. 10.6 *Spanning tree in a graph and the corresponding fundamental cut-set matrix*

10.7 ADJACENCY MATRIX

NOTES

The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i,v_j) according to whether v_i and v_j are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric.

In computers, a graph can be represented in two ways, viz., adjacency matrix and adjacency list. Adjacency matrix uses arrays while adjacency list uses linked lists in the representation of graphs.

Adjacency can be explained with the help of an undirected graph as shown in Figure 10.7

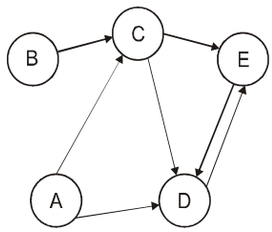


Fig. 10.7 *An Undirected Graph*

In undirected graphs, the edges of the graph do not indicate the direction from one vertex to another.

10.7.1 Adjacency Matrix

Adjacency matrix is an array of vertices of the graph. The matrix can be made for both undirected as well as directed graphs. The adjacency matrix A for an undirected graph G is formed according to the procedure described herein.

If there is an edge from vertex v_i to v_j in G, then the element a_{ij} in A is marked as one, else it is marked as zero. Also, note that since it is an undirected graph, a_{ij} is equivalent to a_{ji}. Therefore, if a_{ij} is one, then a_{ji} will also be one and vice versa.

Table 10.1 shows the adjacency matrix representation for undirected graph.

Table 10.1 *Adjacency Matrix Representation for Undirected Graph*

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

NOTES

The adjacency matrix for directed graph is defined in the same way as it is defined in the undirected graph except for the fact that a_{ij} is not equivalent to a_{ji}. This means that if a_{ij} is one, then a_{ji} will be zero. Table 10.2 shows the adjacency matrix for the directed graph.

Table 10.2 *Adjacency Matrix Representation for Directed Graph*

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

Adjacency matrix has several disadvantages associated with it. Some of them are as follows:

- For a graph with 'n' vertices, an adjacency matrix requires n^2 elements to represent it.
- For a directed graph with n vertices, n^2–e edges are zero. Thus, for a graph with few edges, the matrix becomes sparse. This means that matrix for a graph with few edges carries a lot of zeros.
- An adjacency matrix cannot represent parallel edges.

Adjacency List

To avoid the disadvantages of adjacency matrix, adjacency lists are used. They are especially efficient in case of sparse matrix. They make use of the linked lists of adjacent vertices for all vertices V in graph G. For example, for the undirected graph shown in Figure 10.8, the adjacency list for vertex A is shown in Figure 10.9.

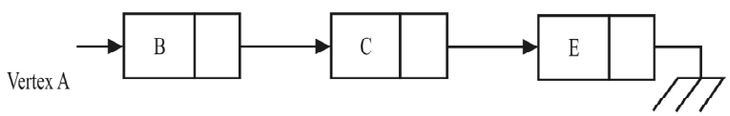


Fig. 10.8 *Adjacency List for Vertex A*

This is because, the vertices adjacent to A are B, C and E. Figure 10.9 shows the complete adjacency list for the graph used in adjacency matrix.

NOTES

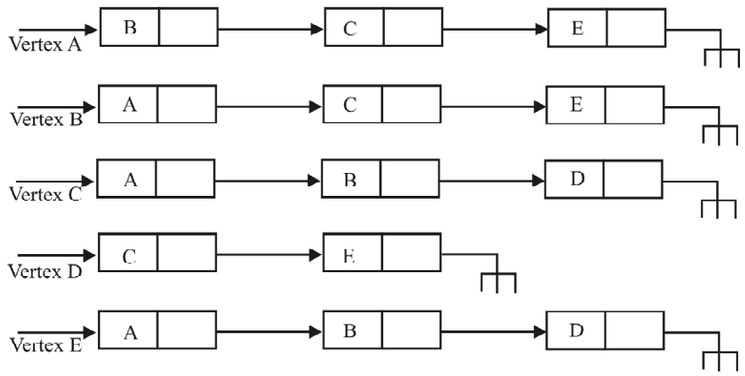


Fig. 10.9 *The Complete Adjacency List*

This adjacency list representation uses less memory. However, if the number of edges and vertices increases, the adjacency list becomes inefficient, i.e., it uses more memory as the overhead of maintaining pointers increases. In C language, adjacency list is represented by an array of pointers, where each pointer points to a linked list of vertices that are adjacent to a particular vertex.

10.7.2 Path Matrix

Path matrix represents the path of certain length. To show adjacent nodes in a simple directed graph we use adjacency matrix. For example, consider the following simple directed graph *G* as shown in Figure 10.10.

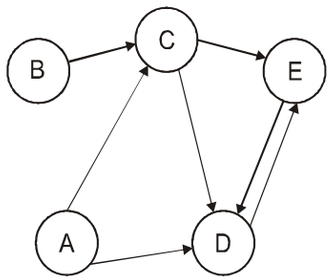


Fig. 10.10 *Simple Directed Graph G*

Following Table 10.3 represents the adjacency matrix for this simple directed graph *G*:

Table 10.3 *Adjacency Matrix for Simple Directed Graph G*

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

The adjacency matrix describes the relationships between the adjacent nodes. Besides these, other relationships also exist between nodes. For example, if we

NOTES

consider the three nodes B, C and E then we see that <B, C> = 1 and also <C, E> = 1. Hence there is a link directed from B to E through C. Thus in order to show such relationship we use the concept of path matrix. To establish a relationship between the nodes B, C and E only two nodes were used and hence we say that it is a path of length 2. Considering all such paths of length 2 we can represent them using a path matrix of length 2 as shown below:

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

Moving on further, consider all paths of length 3 in the above graph to obtain the following path matrix of length 3:

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

In this way, we can define path matrices of certain lengths to show paths among nodes in a graph if they exist.

Check Your Progress

1. Explain the matrix representation of graph.
2. Define incidence matrix.
3. Elaborate on the circuit matrix.
4. Analyse the fundamental circuit matrix.
5. Illustrate the cut-set matrix.
6. Interpret the adjacency matrix.
7. Why adjacency lists are used?
8. What do you understand by the path matrix?

10.8 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. The incidence matrix contains only two elements, 0 and 1. Such a matrix is called a binary matrix or a (0, Ti-matrix, Let us stipulate that these two

elements are from Galois field modulo 2. Given any geometric representation of a graph without self-loops, we can readily write its incidence matrix.

2. To any graph G, there corresponds a $V \times E$ matrix called the incidence matrix of G and is denoted by $I(G) = [a_{ij}]_{V \times E}$, where,

$$a_{ij} = \begin{cases} 1, \text{ if } j\text{th edge is incident with } i\text{th vertex} \\ 0, \text{ otherwise} \end{cases}$$

3. Let the number of different circuits in a graph G be q and the number of edges in G be e. Then a circuit matrix $B = [b_{ij}]$ of G is a q *bye,* (0, 1)-matrix defined as follows:

$b.,$ = 1, if ith circuit includes jth edge, and

=0, otherwise.

4. A set of fundamental circuits (or basic circuits) with respect to any spanning tree in a connected graph, are the only independent circuits in a graph. The rest of the circuits can be obtained as ring sums (i.e., linear combinations) of these circuits. Thus, in a circuit matrix, if we retain only those rows that correspond to a set of fundamental circuits and remove all other rows, we would not lose any information.

5. Analogous to a circuit matrix, we can define a cut-set matrix $C = [c_{ij}]$ in which the rows correspond to the cut-sets and the columns to the edges of the graph, as follows:

$c$ij = 1, if i_{th} cut-set contains j_{th} edge, and

= 0, otherwise

6. The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i, v_j) according to whether v_i and v_j are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric.

7. To avoid the disadvantages of adjacency matrix, adjacency lists are used. They are especially efficient in case of sparse matrix. They make use of the linked lists of adjacent vertices for all vertices V in graph G.

8. Path matrix represents the path of certain length. To show adjacent nodes in a simple directed graph we use adjacency matrix.

10.9 SUMMARY

- The incidence matrix contains only two elements, 0 and 1. Such a matrix is called a binary matrix or a (0, Ti-matrix, Let us stipulate that these two

NOTES

elements are from Galois field modulo 2. Given any geometric representation of a graph without self-loops, we can readily write its incidence matrix.

- To any graph G, there corresponds a $V \times E$ matrix called the incidence matrix of G and is denoted by $I(G) = [a_{ij}]_{V \times E}$, where,

$$a_{ij} = \begin{cases} 1, \text{ if } j\text{th edge is incident with } i\text{th vertex} \\ 0, \text{ otherwise} \end{cases}$$

- Let the number of different circuits in a graph G be q and the number of edges in G be e. Then a *circuit matrix* $B = [b_{ij}]$ of G is a q *bye,* (0, 1)-matrix defined as follows:

$b.,$ = 1, if ith circuit includes jth edge, and

=0, otherwise.

- A set of fundamental circuits (or basic circuits) with respect to any spanning tree in a connected graph, are the only independent circuits in a graph. The rest of the circuits can be obtained as ring sums (i.e., linear combinations) of these circuits. Thus, in a circuit matrix, if we retain only those rows that correspond to a set of fundamental circuits and remove all other rows, we would not lose any information.
- Analogous to a circuit matrix, we can define a *cut-set matrix* $C = [c_{ij}]$ in which the rows correspond to the cut-sets and the columns to the edges of the graph, as follows:

$c$ij = 1, if i_{th} cut-set contains j_{th} edge, and

= 0, otherwise

- The adjacency matrix, sometimes also called the connection matrix, of a simple labeled graph is a matrix with rows and columns labeled by graph vertices, with a 1 or 0 in position (v_i, v_j) according to whether v_i and v_j are adjacent or not. For a simple graph with no self-loops, the adjacency matrix must have 0s on the diagonal. For an undirected graph, the adjacency matrix is symmetric.
- To avoid the disadvantages of adjacency matrix, adjacency lists are used. They are especially efficient in case of sparse matrix. They make use of the linked lists of adjacent vertices for all vertices V in graph G.
- Path matrix represents the path of certain length. To show adjacent nodes in a simple directed graph we use adjacency matrix.

10.10 KEY WORDS

- **Incidence matrix:** To any graph G, there corresponds a $V \times E$ matrix called the incidence matrix of G and is denoted by $I(G)=[a_{ij}]_{V \times E}$.

- **Circuit matrix:** Let the number of different circuits in a graph G be q and the number of edges in G be e. Then a circuit matrix $B = [b_{ij}]$ of G is a q by e, (0,1).
- **Fundamental circuit matrix:** A submatrix (of a circuit matrix) in which all rows correspond to asset of fundamental circuits is called a fundamental circuit matrix B_f.
- **Cut-set matrix:** Analogous to a circuit matrix, we can define a cut-set matrix C=$[c_{ij}]$ in which the rows correspond to the cut-sets and the column to the edges of the graph.
- **Adjacency matrix:** Adjacency matrix is an array of vertices of the graph. The matrix can be made for both undirected as well as directed graphs.
- **Adjacency lists:** To avoid the disadvantages of adjacency matrix, adjacency lists are used. They are especially efficient in case of sparse matrix.
- **Path matrix:** Path matrix represents the path of certain length. To show adjacency nodes in a simple directed graph we use adjacency matrix.

10.11 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define matrix representation of a graph.
2. Explain the incidence matrix.
3. Analyse the circuit matrix.
4. Elaborate on the fundamental circuit matrix.
5. Interpret the cut-set matrix.
6. Illustrate the adjacency matrix.
7. State the path matrix.

Long-Answer Questions

1. Describe the matrix representation of a graph.
2. Briefly discuss the incidence matrix with the help of example.
3. Analyse the circuit matrix.
4. Interpret the fundamental circuit matrix. Give appropriate example.
5. Elaborate on the cut-set matrix.
6. Define the adjacency matrix.
7. Explain in brief the path matrix.

10.12 FURTHER READINGS

NOTES

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 11 CHROMATIC NUMBERS

NOTES

Structure

11.0 INTRODUCTION

The smallest number of colours needed to colour a graph G is called its chromatic number, and is often denoted $\chi(G)$. Sometimes $\gamma(G)$ is used, since $\chi(G)$ is also used to denote the Euler characteristic of a graph. A graph that can be assigned a (proper) k-colouring is k-colourable, and it is k-chromatic if its chromatic number is exactly k. A subset of vertices assigned to the same colour is called a colour class, every such class forms an independent set. Thus, a k-colouring is the same as a partition of the vertex set into k independent sets, and the terms k-partite and k-colourable have the same meaning.

The first results about graph colouring deal almost exclusively with planar graphs in the form of the colouring of maps. While trying to colour a map of the counties of England, Francis Guthrie postulated the four colour conjecture, noting that four colours were sufficient to colour the map so that no regions sharing a common border received the same colour. Guthrie's brother passed on the question to his mathematics teacher Augustus de Morgan at University College, who mentioned it in a letter to William Hamilton in 1852. Arthur Cayley raised the problem at a meeting of the London Mathematical Society in 1879. The same year, Alfred Kempe published a paper that claimed to establish the result, and for a decade the four colour problem was considered solved. For his accomplishment Kempe was elected a Fellow of the Royal Society and later President of the London Mathematical Society.

The chromatic polynomial counts the number of ways a graph can be coloured using no more than a given number of colours. For example, using three colours, the graph in the adjacent image can be coloured in 12 ways. With only two colours, it cannot be coloured at all. With four colours, it can be coloured in

NOTES

24 + 4.12 = 72 ways: using all four colours, there are 4! = 24 valid colourings (every assignment of four colours to any 4-vertex graph is a proper colouring); and for every choice of three of the four colours, there are 12 valid 3-colorings.

In this unit, you will study about the chromatic numbers, chromatic partitioning, and chromatic polynomial.

11.1 OBJECTIVES

After going through this unit, you will be able to:

- Understand the chromatic number
- Explain the chromatic partitioning
- Analyse the chromatic polynomial

11.2 CHROMATIC NUMBERS

A graph G of n vertices can be properly coloured in many different ways using a sufficiently large number of colours. This property of a graph is expressed by a polynomial. This polynomial can be referred to as chromatic polynomial.

The value of chromatic polynomial is given by,

$$P_n(\lambda) = \sum_{i=1}^{n} c_i \begin{pmatrix} \lambda \\ . \\ l \end{pmatrix}$$

Where, C_i is the different ways of properly colouring G using exactly l colours out of λ colours in $\begin{pmatrix} \lambda \\ . \\ i \end{pmatrix}$ different ways.

Theorem 11.1: *A graph of n vertices is a complete graph if and only if its chromatic polynomial is,*

$$P_n(\lambda) = \lambda(\lambda-1)(\lambda-2)...(\lambda-n+1)$$

Proof: Let us use λ colours, then there are λ different ways of colouring any selected vertex of a graph. A second vertex can be coloured properly in exactly $(\lambda-1)$ ways, the third in $(\lambda-2)$ ways, the fourth in $(\lambda-3)$ ways, etc., and nth in $(\lambda-n+1)$ ways iff every vertex is adjacent to every other. That is, if and only if the graph is complete.

Note: An n - vertex graph is a tree if its chromatic polynomial

$$p_n(\lambda) = \lambda(\lambda-1)^{n-1}.$$

11.3 CHROMATIC PARTITIONING

NOTES

Let $\chi(G)$ be the chromatic number of a graph $G = (V, E)$, and $k \geq 1$ be an integer. The general chromatic number $\chi_k(G)$ of G is the minimum order of a partition P of V such that each set in P induces a subgraph H with $\chi(H) \leq k$.

Let $G = (V, E)$ be a graph. For a property P, let $n(P)$ be the minimum number of sets into which V can be partitioned so that each set induces a subgraph H with property P. The number $n(P)$ specifies various properties P. For example, suppose P_1, P' and P'' be the properties defined as follows:

P_1 : H is totally disconnected or irivial.

P' : H is a forest

P'' : H is k-degenerate.

Then $n(P_1)$ is the chromatic number of G, $n(P')$ is the point arboricity of G and $n(P'')$ is the point partition number.

Let $k \geq 1$ be an integer, and $\chi(G)$ denote the chromatic number of G. We will define number $n(P_k)$ where P_k is the property: $\chi(H) \leq$ k.

A set $S \subset V$ is a P_k-set if $\chi(\langle S \rangle) \leq k$, where $\langle$S$\rangle$ is the subgraph of G induced by S. A partition $\{V_1, V_2, ..., V_n\}$ of V is a P_k-partition if each V_i is a P_k-set. A P_k-coloring of G is defined as the coloring of the vertices of G such that the set of all vertices receiving the same color is a P_k-set. A P_k-coloring which uses r colors is defined as (k, r)-coloring. If there exists a (k, r)-coloring of G for some $r \leq n$, then G is said to be (k, n)-colorable.

The chromatic partition number $\chi_k(G)$ of G is the minumum number of colors required in a P_k-coloring of G. If $\chi_k(g) = n$, then G is said to be (k, n)-chromatic.

Clearly, $\chi_1(G) = \chi(G)$ and $\chi_k(g) = 1$ for all $k \geq \chi(G)$. Thus, if G is any bipartite graph, $\chi_n(G) = 1$ for all $n \geq 2$, and for an odd cycle C_γ, $\chi_2(C_\gamma) = 2$ and $\chi_n(C_\chi) = 1$, for all $n \geq 3$. For any graph G, $\chi_k(G) \leq \chi_i(G)$ when $j \leq$ K.

For a real number r, let $[r]$ and $\{r\}$, respectively, denote the greatest integer not exceeding r, and the least integer not less than r.

Following are some significant proposition that define chromatic partitioning of a graph.

Proposition 1: For the complete graph K_p

$$\chi_k(K_p) = \left\{\frac{p}{k}\right\}. \quad (1)$$

Corollary: For any graph G of order p

$$\chi_k(G) \leq \left\{\frac{p}{k}\right\}. \quad (2)$$

Proposition 2: For any graph G

$$\frac{\chi(G)}{k} \leq \chi_k(G) \leq \chi(G). \tag{3}$$

NOTES

Proof: We establish only the first half in Equation (3), the second half obvious. Let $\{V_1, V_2, ..., V_r\}$ be a $\chi_k(G)$-partition of V and $\chi(\langle V_i \rangle) = T_i$. Since $t_i \leq K$, we have

$$\chi(G) \leq \sum t_i \leq k\chi_k(g).$$

Let $\beta_0 = \beta_0(G)$ be the independence number of G, and M_k be the maxmimum number of points in a P_k-set of G.

Proposition 3: For any graph G of order P

$$\frac{p}{M_k} \leq \chi_k(G) \leq \left\{\frac{p - M_k}{k}\right\} + 1 \tag{4}$$

$$\frac{p}{\beta_0 k} \leq \chi_k(G) \leq \left\{\frac{p - \beta_0}{k}\right\} + 1 \tag{5}$$

Proof: Let $\{V_1, V_2, ..., V_n\}$ be a minimum P_k-partition of $V(G)$. Then $n = k(G)$ and $|V_i| \leq M_k$, for $1 \leq i \leq n$. Therefore,

$$P = G \sum |V_{i|} \leq nM_k = \chi_k(G)M_k.$$

The lower bound in Equation (4) holds. To establish the upper bound in Equation (4), let $S \subset V$ be a P_k-set with $|S| = M_k$. Clearly, $\chi_K(G - S) \geq \chi_k(G) - 1$. Since $G - S$ has $p - M_k$ points, we have

$$\chi_k(G - S) \leq \left\{\frac{p - M_k}{k}\right\} \quad \text{by (2).}$$

Therefore, $\chi_k(G) \leq \left\{\frac{p - M_k}{k}\right\} + 1$, and Equation (4) follows. To establish Equation (5), let $\langle V_i \rangle = G_i$, and $|V_i| = p_i$. It is well known that,

$$\frac{p_i}{\beta_0(G_i)} \leq \chi(G_i)$$

Since $(Gi) \leq k$, we have

$$P_i \leq \beta_0(G_i)k \leq \beta_0(G)k, \text{ and}$$

$$p = \sum p_i \leq k\beta_0(g) \leq_k (g),$$

This establishes the lower bound in Equation (5). Similarly, we can establish the upper bound in Equation (5).

11.4 CHROMATIC POLYNOMIAL

The **'Chromatic Polynomial'** is a graph polynomial studied in algebraic graph theory, a branch of mathematics. It **counts** the **number of graph colourings** as a function of the number of colours and was originally defined by George David

NOTES

Birkhoff to study the four colour problem. It was generalised to the Tutte polynomial by Hassler Whitney and W. T. Tutte, linking it to the Potts model of statistical physics.

George David Birkhoff introduced the concept of chromatic polynomial in 1912, defining it only for planar graphs, in an attempt to prove the four colour theorem. If $P(G, k)$ denotes the number of proper colourings of G with k colours then one could establish the four colour theorem by showing $P(G, 4) > 0$ for all planar graphs G. In this manner, he hoped to apply the powerful tools of analysis and algebra for studying the roots of polynomials to the combinatorial colouring problem.

Hassler Whitney generalised Birkhoff's polynomial from the planar case to general graphs in 1932. In 1968, Read asked which polynomials are the chromatic polynomials of some graph, a question that remains open, and introduced the concept of chromatically equivalent graphs. Today, chromatic polynomials are one of the central objects of algebraic graph theory.

Following Figure 11.1 illustrates the all non-isomorphic graphs on 3 vertices and their chromatic polynomials, clockwise from the top. The independent 3-set: k^3, an edge and a single vertex: $k^2 (k - 1)$, the 3-path: $k (k - 1)^2$ and the 3-clique: $k (k\ 1) (k - 2)$.

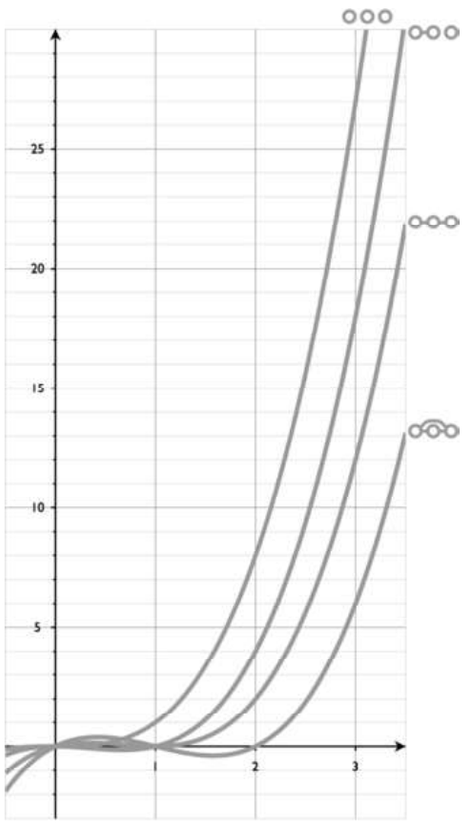


Fig. 11.1 *All Non-Isomorphic Graphs on 3 Vertices and Their Chromatic Polynomials*

NOTES

Definition

For a graph G, $P(G, k)$ counts the number of its (proper) vertex k-colourings. Other commonly used notations include $P_G(k)$, $\chi_G(k)$, or $\pi_G(k)$. There is a unique polynomial $P(G, x)$ which evaluated at any integer $k \geq 0$ coincides with $P(G, k)$; it is called the **'Chromatic Polynomial** of ***G*'**.

For example, to colour the path graph P_3 on 3 vertices with k colours, one may choose any of the k colours for the first vertex, any of the $(k-1)$ remaining colours for the second vertex, and lastly for the third vertex, any of the colours that are different from the second vertex's choice.

Therefore, $P(P_3, k) = k \,.\, (k-1) \,.\, (k-1)$ is the number of k-colourings of P_3. For a variable x (not necessarily integer), we thus have $P(P_3, x) = x(x-1)^2 = x^3 - 2x^2 + x$. Colourings which differ only by permuting colours or by automorphisms of G are still counted as different.

Following Figure 11.2 illustrates all the proper vertex colourings of vertex graphs with 3 vertices using k colours for $k = 0, 1, 2, 3$. The chromatic polynomial of each graph interpolates through the number of proper colourings.

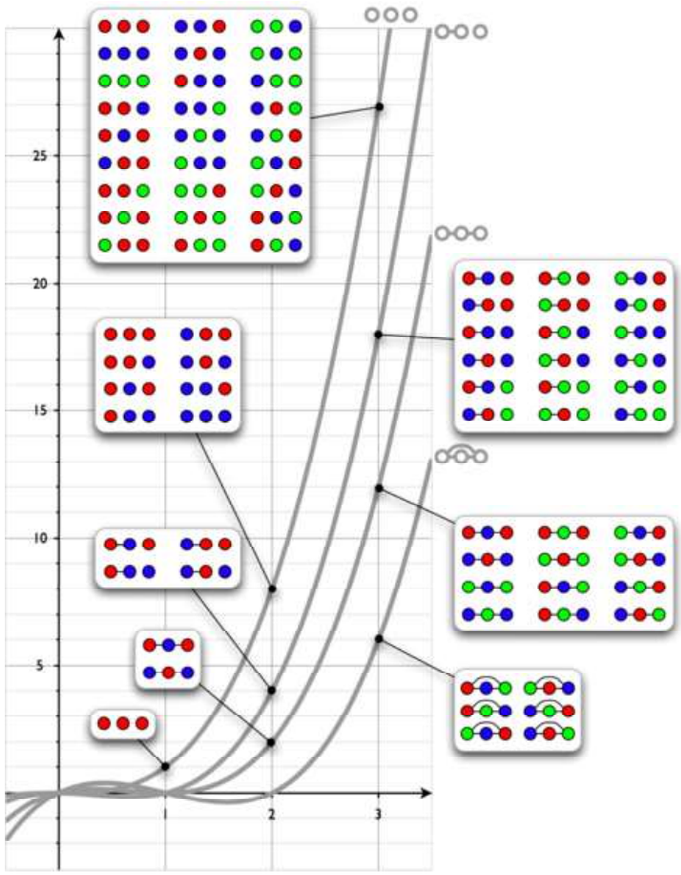


Fig. 11.2 *All Proper Vertex Colourings of Vertex Graphs with 3 Vertices using* k *Colours for* k = *0, 1, 2, 3*

NOTES

Deletion–Contraction Formula

The fact that the number of k-colourings is a polynomial in k follows from a recurrence relation called the **'Deletion–Contraction Recurrence'** or **'Fundamental Reduction Theorem'**. It is based on edge contraction, for a pair of vertices u and v the graph G/uv is obtained by merging the two vertices and removing any edges between them. If u and v are adjacent in G, then let $G - uv$ denote the graph obtained by removing the edge uv. At that point the numbers of k-colourings of these graphs satisfy:

$$P(G, k) = P(G - uv, k) - P(G/uv, k)$$

Equivalently, if u and v are not adjacent in G and $G + uv$ is the graph with the edge uv added, then,

$$P(G, k) = P(G + uv, k) + P(G/uv, k)$$

This follows from the observation that every k-colouring of G either gives different colours to u and v, or the similar colours. In the first case this gives a (proper) k-colouring of $G + uv$, while in the second case it gives a colouring of G/uv. Conversely, every k-colouring of G can be uniquely obtained from a k-colouring of $G + uv$ or G/uv, if u and v are not adjacent in G.

The chromatic polynomial can hence be recursively defined as,

- $P(G, x) = x^n$ for the edgeless graph on n vertices.
- $P(G, x) = P(G - uv, x) - P(G/uv, x)$ for a graph G with an edge uv (arbitrarily chosen).

Since the number of k-colourings of the edgeless graph is certainly k^n, it follows by induction on the number of edges that for all G, the polynomial $P(G, x)$ coincides with the number of k-colourings at every integer point $x = k$. In particular, the chromatic polynomial is the unique interpolating polynomial of degree at most n through the points,

$$\{(0, P(G, 0)), (1, P(G, 1)), \ldots, (n, P(G, n))\}$$

Tutte's curiosity about which other graph invariants satisfied such recurrences led him to discover a bivariate generalization of the chromatic polynomial, the Tutte polynomial $T_G(x, y)$.

Chromatic Polynomials for Certain Graphs

Following Table 11.1 shows some examples of chromatic polynomials for certain graphs.

NOTES

Table 11.1 *Chromatic Polynomials for Certain Graphs*

Triangle K_3	$t(t-1)(t-2)$
Complete Graph K_n	$t(t-1)(t-2)\cdots(t-(n-1))$
Tree with n Vertices	$t(t-1)^{n-1}$
Cycle C_n	$(t-1)^n+(-1)^n(t-1)$
Petersen Graph	$t(t-1)(t-2)(t^7-12t^6+67t^5-230t^4+529t^3-814t^2+775t-352)$

11.4.1 Properties of the Chromatic Polynomial

For fixed G on n vertices, the chromatic polynomial $P(G,x)$ is a monic polynomial of degree exactly n, with integer coefficients.

The chromatic polynomial includes at least as much information about the colourability of G as does the chromatic number. Definitely, the chromatic number is the smallest positive integer that is not a zero of the chromatic polynomial,

$$\chi(G) = \min \{k \in \mathbb{N} : P(G, k) > 0\}$$

The polynomial evaluated at -1, that is $P(G, -1)$, yields $(-1)^{|V(G)|}$ times the number of acyclic orientations of G.

The derivative evaluated at 1, $P'(G, 1)$ equals the chromatic invariant $\theta(G)$ up to sign.

If G has n vertices and c components $G_1, \ldots, G_c$, then,

- The coefficients of $x^0, \ldots, x^{c-1}$ are zeroes.
- The coefficients of $x^c, \ldots, x^n$ are all non-zero and alternate in signs.
- The coefficient of x^n is 1 (the polynomial is monic).
- The coefficient of x^{n-1} is $-|E(G)|$.
- The coefficient of x^1 is $(-1)^{n-1}$ times the number of acyclic orientations that have a unique sink, at a specified, arbitrarily chosen vertex.
- The absolute values of coefficients of every chromatic polynomial form a log-concave sequence.
- $P(G, x) = P(G_1, x)\, P(G_2, x), \ldots, P(G_c, x)$.

The last property is generalized by the fact that if G is a k-clique-sum of G_1 and G_2, i.e., a graph obtained by gluing the two at a clique on k vertices, then

$$P(G,x) = \frac{P(G_1,x)P(G_2,x)}{x(x-1)\cdots(x-k+1)}$$

A graph G with n vertices is a tree if and only if,

$$P(G, x) = x(x-1)^{n-1}$$

Chromatic Equivalence

Two graphs are said to be chromatically equivalent if they have the same chromatic polynomial. Isomorphic graphs have the same chromatic polynomial, but non-isomorphic graphs can be chromatically equivalent. For example, all trees on n vertices have the same chromatic polynomial. In particular, $(x-1)^3 x$ is the chromatic polynomial of both the claw graph and the path graph on 4 vertices. Following Figure 11.3 illustrates the three graphs with a chromatic polynomial equal to $(x-2)(x-1)^3 x$.

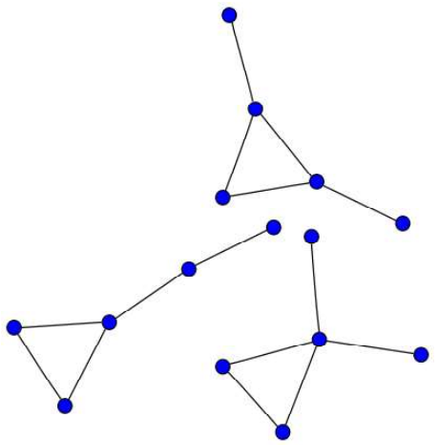


Fig. 11.3 *Three Graphs with a Chromatic Polynomial Equal To (x-2) (x-1)³ x*

A graph is chromatically unique if it is determined by its chromatic polynomial, up to isomorphism. In other words, if G is chromatically unique, then $P(G,x) = P(H,x)$ would imply that G and H are isomorphic. All cycle graphs are chromatically unique.

Chromatic Roots

A root or zero of a chromatic polynomial, called a **'Chromatic Root'**, is a value x where $P(G, x) = 0$. Chromatic roots have been very well studied, in fact, Birkhoff's original motivation for defining the chromatic polynomial was to show that for planar graphs, $P(G,x) > 0$ for $x \geq 4$. This would have established the four colour theorem.

No graph can be 0-coloured, so 0 is always a chromatic root. Only edgeless graphs can be 1-coloured, so 1 is a chromatic root of every graph with at least one edge. On the other hand, except for these two points, no graph can have a chromatic root at a real number smaller than or equal to 32/27. A result of Tutte connects the golden ratio ϕ (phi) with the study of chromatic roots, showing that chromatic roots exist very close to ϕ^2: If G_n is a planar triangulation of a sphere then,

$$P(G_n, \phi^2) \leq \phi^{5-n}$$

While the real line thus has large parts that contain no chromatic roots for any graph, every point in the complex plane is arbitrarily close to a chromatic root

in the sense that there exists an infinite family of graphs whose chromatic roots are dense in the complex plane.

Categorification

NOTES

The chromatic polynomial is categorified by a homology theory closely related to Khovanov homology.

11.4.2 Algorithms for the Chromatic Polynomial

Computational problems associated with the chromatic polynomial include the following:

Problem 1: Finding the chromatic polynomial $P(G, x)$ of a given graph G.

Problem 2: Evaluating $P(G, x)$ at a fixed point x for given graph G.

The first problem is considered as more general and universal because if we know and identify the coefficients of $P(G, x)$ then we can evaluate or compute it at any point in polynomial time because the degree is n. The difficulty of the second type of problem depends strongly on the value of x and has been intensively studied in computational complexity. When x is a natural number, then this problem is normally viewed as computing the number of x-colourings of a given graph. For example, this includes the problem **#3-Colouring** of counting the number of 3-Colourings, a canonical problem in the study of complexity of counting, complete for the counting class **#P**.

Efficient Algorithms

For some basic graph classes, closed formulas for the chromatic polynomial are known. For instance this is true for trees and cliques, as listed in the Table 8.1.

Polynomial time algorithms are known for computing the chromatic polynomial for wider classes of graphs, including chordal graphs and graphs of bounded clique-width. The latter class includes cographs and graphs of bounded tree-width, such as outerplanar graphs. In graph theory, an outerplanar graph is a graph that has a planar drawing for which all vertices belong to the outer face of the drawing.

Deletion–Contraction Algorithm

The deletion-contraction recurrence gives a method of computing the chromatic polynomial, called the deletion–contraction algorithm. In the first form (with a **minus '–'**), the recurrence terminates in a collection of empty graphs. In the second form (with a **plus '+'**), it terminates in a collection of complete graphs. This forms the basis of many algorithms for graph colouring. The worst case running time of either formula satisfies the same recurrence relation as the Fibonacci numbers, so in the worst case, the algorithm runs in time within a polynomial factor of,

$$\phi^{n+m} = \left(\frac{1+\sqrt{5}}{2}\right)^{n+m} \in O\left(1.62^{n+m}\right)$$

This is on a graph with n vertices and m edges. The analysis can be improved to within a polynomial factor of the number $t(G)$ of spanning trees of the input graph. In practice, branch and bound strategies and graph isomorphism rejection are employed to avoid some recursive calls, the running time depends on the heuristic used to pick the vertex pair.

Cube Method

There is a natural geometric perspective on graph colourings by observing that, as an assignment of natural numbers to each vertex, a graph colouring is a vector in the integer lattice. Since two vertices ***i*** and ***j*** being given the same colour is equivalent to the ***i***th and ***j***th coordinate in the colouring vector being equal, each edge can be associated with a hyperplane of the form,

$$\{x \in R^d : x_i = x_j\}$$

The collection of such hyperplanes for a given graph is called its graphic arrangement. The proper colourings of a graph are those lattice points which avoid forbidden hyperplanes. Restricting to a set of k colours, the lattice points are contained in the cube $[0, k]^n$. In this context the chromatic polynomial counts the number of lattice points in the $[0, k]$-cube that avoid the graphic arrangement.

Computational Complexity

The problem of computing the number of 3-Colorings of a given graph is a canonical example of a **#P-Complete** problem, so the problem of computing the coefficients of the chromatic polynomial is **#P-Hard**. In the same way, by evaluating $P(G, 3)$ for given G is **#P-Complete**. Alternatively, for $k = 0, 1, 2$ it is easy to compute $P(G, k)$, so the corresponding problems are polynomial-time computable. For integers $k > 3$ the problem is #P-Hard, which is established similar to the case $k = 3$. Essentially, it is known that $P(G, x)$ is #P-hard for all $\boldsymbol{x}$ including negative integers and even all complex numbers except for the three easy points. Thus, from the perspective of #P-Hardness, the complexity of computing the chromatic polynomial is completely understood.

In the expansion,

$$P(G, x) = a_1 x + a_2 x^2 + \ldots\ldots + a_n x^n$$

The coefficient a_n is always equal to 1, and several other properties of the coefficients are known. However, the computational problem of computing a_r for a fixed $r \geq 1$ and a given graph G is #P-Hard, even for bipartite planar graphs.

No approximation algorithms for computing $P(G, x)$ are known for any x except for the three easy points. At the integer points $k = 3, 4, \ldots$, the corresponding decision problem of deciding if a given graph can be k-coloured is NP-Hard. Such problems cannot be approximated to any multiplicative factor by a bounded-error probabilistic algorithm unless NP = RP, because any multiplicative approximation would distinguish the values 0 and 1, effectively solving the decision version in bounded-error probabilistic polynomial time. In particular, under the same assumption, this rules out the possibility of a Fully Polynomial time

NOTES

Randomised Approximation Scheme (FPRAS). For other points, more complicated arguments are required, and the question is the focus of active research. As of today, it is known that there is no FPRAS for computing $P(G, x)$ for any $x > 2$, unless NP = RP holds.

Check Your Progress

1. Explain the chromatic numbers.
2. Elaborate on the chromatic partitioning.
3. Interpret the chromatic polynomial.
4. Define the deletion-contraction formula.
5. Analyse the properties of the chromatic polynomial.
6. What do you understand by the chromatic equivalence?
7. State the chromatic roots.

11.5 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A graph G of n vertices can be properly coloured in many different ways using a sufficiently large number of colours. This property of a graph is expressed by a polynomial. This polynomial can be referred to as chromatic polynomial.
2. Let $G = (V, E)$ be a graph. For a property P, let $n(P)$ be the minimum number of sets into which V can be partitioned so that each set induces a subgraph H with property P. The number $n(P)$ specifies various properties P.
3. The 'Chromatic Polynomial' is a graph polynomial studied in algebraic graph theory, a branch of mathematics. It counts the number of graph colourings as a function of the number of colours and was originally defined by George David Birkhoff to study the four colour problem.
4. The fact that the number of k-colourings is a polynomial in k follows from a recurrence relation called the 'Deletion–Contraction Recurrence' or 'Fundamental Reduction Theorem'. It is based on edge contraction, for a pair of vertices u and v the graph G/uv is obtained by merging the two vertices and removing any edges between them.
5. The chromatic polynomial includes at least as much information about the colourability of G as does the chromatic number. Definitely, the chromatic number is the smallest positive integer that is not a zero of the chromatic polynomial,

$$\chi(G) = \min \{k \in \mathbb{N} : P(G, k) > 0\}$$

6. Two graphs are said to be chromatically equivalent if they have the same chromatic polynomial. Isomorphic graphs have the same chromatic polynomial, but non-isomorphic graphs can be chromatically equivalent.
7. A root or zero of a chromatic polynomial, called a 'Chromatic Root', is a value x where $P(G, x) = 0$. Chromatic roots have been very well studied, in fact, Birkhoff's original motivation for defining the chromatic polynomial was to show that for planar graphs, $P(G, x) > 0$ for $x \geq 4$. This would have established the four colour theorem.

11.6 SUMMARY

- A graph G of n vertices can be properly coloured in many different ways using a sufficiently large number of colours. This property of a graph is expressed by a polynomial. This polynomial can be referred to as chromatic polynomial.
- Let $G = (V, E)$ be a graph. For a property P, let $n(P)$ be the minimum number of sets into which V can be partitioned so that each set induces a subgraph H with property P. The number $n(P)$ specifies various properties P.
- The 'Chromatic Polynomial' is a graph polynomial studied in algebraic graph theory, a branch of mathematics. It counts the number of graph colourings as a function of the number of colours and was originally defined by George David Birkhoff to study the four colour problem.
- The fact that the number of k-colourings is a polynomial in k follows from a recurrence relation called the 'Deletion–Contraction Recurrence' or 'Fundamental Reduction Theorem'. It is based on edge contraction, for a pair of vertices u and v the graph G/uv is obtained by merging the two vertices and removing any edges between them.
- For fixed G on n vertices, the chromatic polynomial $P(G, x)$ is a monic polynomial of degree exactly n, with integer coefficients.
- The chromatic polynomial includes at least as much information about the colourability of G as does the chromatic number. Definitely, the chromatic number is the smallest positive integer that is not a zero of the chromatic polynomial,

$$\chi(G) = \min \{k \in \mathbb{N} : P(G, k) > 0\}$$

- Two graphs are said to be chromatically equivalent if they have the same chromatic polynomial. Isomorphic graphs have the same chromatic polynomial, but non-isomorphic graphs can be chromatically equivalent.
- A root or zero of a chromatic polynomial, called a 'Chromatic Root', is a value x where $P(G, x) = 0$. Chromatic roots have been very well studied, in fact, Birkhoff's original motivation for defining the chromatic polynomial

was to show that for planar graphs, $P(G, x) > 0$ for $x \geq 4$. This would have established the four colour theorem.

NOTES

11.7 KEY WORDS

- **Chromatic numbers:** A graph G of n vertices can be properly coloured in many different ways using a sufficiently large number of colours. This property of a graph is expressed by a polynomial.
- **Chromatic partitioning:** Let $G = (V,E)$ be a graph. For a property P, let $n(P)$ be the minimum number of sets into which V can be partitioned so that each set induces a subgraph H with property P.
- **Chromatic polynomial:** The chromatic polynomial is a graph polynomial studied in algebraic graph theory, a branch of mathematics. It counts the number of graph colourings as a function of the number of colours and was originally defined by George David Birkhoff to study the four colour problem.
- **Chromatic equivalence:** Two graphs are said to be chromatically equivalent if they have the same chromatic polynomial.
- **Chromatic roots:** A root or zero of a chromatic polynomial, called a chromatic root, is a value x where $P(G,x) = 0$.
- **Categorification:** The chromatic polynomial is categorified by a homology theory closely related to Khovanov homology.

11.8 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the chromatic numbers.
2. Explain the chromatic partitioning.
3. State the chromatic polynomial.
4. Elaborate on the deletion-contraction formula.
5. Interpret the properties of the chromatic polynomial.
6. Define the chromatic equivalence.
7. Illustrate the chromatic roots.

Long-Answer Questions

1. Discuss briefly the chromatic numbers with the help of example.
2. Analyse the chromatic partitioning. Give appropriate example.
3. Explain the chromatic polynomial.

4. Interpret the deletion-contraction formula.
5. Briefly define the properties of the chromatic polynomial.
6. Elaborate on the chromatic equivalence.
7. Define chromatic roots with suitable example.

NOTES

11.9 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

BLOCK - IV
TREES AND CUT SETS

NOTES

UNIT 12 TREES

Structure

12.0 INTRODUCTION

In graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path, or equivalently a connected acyclic undirected graph. A forest is an undirected graph in which any two vertices are connected by at most one path, or equivalently an acyclic undirected graph, or equivalently a disjoint union of trees. A polytree (or directed tree or oriented tree or singly connected network) is a directed acyclic graph (DAG) whose underlying undirected graph is a tree. A polyforest (or directed forest or oriented forest) is a directed acyclic graph whose underlying undirected graph is a forest.

The various kinds of data structures referred to as trees in computer science have underlying graphs that are trees in graph theory, although such data structures are generally rooted trees. A rooted tree may be directed, called a directed rooted tree, either making all its edges point away from the root—in which case it is called an arborescence or out-tree—or making all its edges point towards the root—in which case it is called an anti-arborescence or in-tree. A rooted tree itself has been defined by some authors as a directed graph. A rooted forest is a disjoint union of rooted trees. A rooted forest may be directed, called a directed rooted forest, either making all its edges point away from the root in each rooted tree—in which case it is called a branching or out-forest—or making all its edges point towards the root in each rooted tree—in which case it is called an anti-branching or in-forest.

A rooted tree is a tree in which one vertex has been designated the root. The edges of a rooted tree can be assigned a natural orientation, either away from or towards the root, in which case the structure becomes a directed rooted tree.

The term "Tree" was coined in 1857 by the British mathematician Arthur Cayley.

In this unit, you will study about the trees, properties of trees, pendent vertices in a tree, distance and centres in a tree, rooted and binary trees.

NOTES

12.1 OBJECTIVES

After going through this unit, you will be able to:

- Define the trees
- Explain the properties of trees
- Interpret the pendent vertices in a tree
- Elaborate on the distance and centres in a tree
- Comprehend the rooted and binary trees

12.2 TREES

In mathematics, and more specifically in graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph without simple cycles is a tree. A forest is considered as a disjoint union of trees.

12.2.1 Properties of Trees

In this section we shall study the characteristics of a tree.

Acyclic Graph: A graph G which has no cycles is called an acyclic graph.

Tree: A connected acyclic graph G is called a tree.

For example,

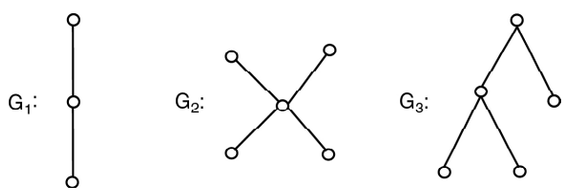


Fig. 12.1 *Trees*

Notes:

1. Trees are often known as open graphs.
2. Any organizational hierarchy is also an example of tree.

NOTES

Theorem 12.1: Every two vertices in a tree, are joined by a unique path.

Proof: By contradiction: Let G be a tree and assume that there are two distinct (v, w) paths P_1 and P_2 in G. Since $P_1 \neq P_2$, there is an edge $e = V_1V_2$ of P_1 that is not in P_2. Clearly $(P_1 \cup P_2) - e$ is connected. Therefore it contains a $(V_1 - V_2)$ path P. Now $P + e$ is a cycle in the acyclic graph G, which is a contradiction to the fact that G is a tree.

Theorem 12.2: If G is a tree on n vertices, then G has $(n-1)$ edges.

Proof: By induction on the number of vertices.

When $n = 1$, $E(G) = 0 = n - 1$ $(G \cong K_1)$

When $n = 2$, $E(G) = 1 = n - 1$ $(G \cong K_2)$

Let us assume that this theorem is true for all trees of G with fewer than n vertices.

Now, let G be a tree on n vertices. Let $e = uv$ be an edge in G. Then $G - e$ is disconnected and G has two components say G_1 and G_2 of $G - e$. Since G is acyclic, G_1 and G_2 are also acyclic and hence G_1 and G_2 are also trees. Moreover G_1 and G_2 has fewer than n vertices say n_1 and n_2, respectively. Therefore, by induction hypothesis,

G_1 has $(n_1 - 1)$ edges and G_2 has $(n_2 - 1)$ edges.

$\therefore E(G) = E(G_1) + (G_2) + 1$ (Here 1 in the sum corresponds to the edge e)

$$= (n_1 - 1) + (n_2 - 1) + 1$$
$$= n_1 + n_2 - 1$$
$$= n - 1$$

Therefore, an n vertex tree has $(n - 1)$ edges.

Theorem 12.3: Every tree has atleast two vertices of degree one in a tree, i.e., there are atleast two pendant vertices.

Proof: Let G be a tree on n vertices. Then,

$$d(v) \geq 1, \ \forall v \in \nu(G) \tag{12.1}$$

Already we have, $$\text{“} \sum_{v \in V} d(v) = 2.E(G) = 2.e \text{”} \tag{12.2}$$

Since G is an n-vertex tree, it has $(n-1)$ edges.

$$\therefore \sum_{v \in V(G)} d(v) = (2n - 2) \tag{12.3}$$

From Equations (12.1) and (12.3), it follows that $d(v) = 1$ for atleast two vertices.

Note: In a tree, every edge is a cut-edge.

12.3 PENDENT VERTICES IN A TREES

In this section, we shall discuss problems using trees.

NOTES

Binary Search Trees

Binary search tree is a binary tree in which each child is either a left or right child; no vertex has more than one left child and one right child, and the data are associated with vertices.

Example 12.1: Build a binary search tree for the words banana, peach, apple, pear, coconut, mango and papaya using the alphabetical order.

Solution: The binary tree is build as follows:

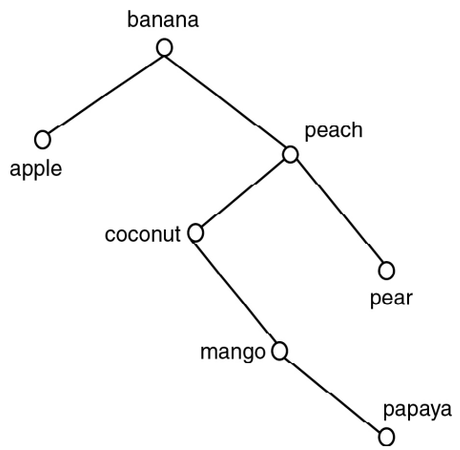


Fig. 12.2 *Binary Search Tree*

Further mango is the right child of coconut and papaya is the right child of mango.

Decision Trees

A rooted tree in which each internal vertex is assigned to a decision with a subtree at the vertices, then each possible outcome of the decision is called a decision tree.

Traversal of a Tree

A systematic method for visiting every vertex of an ordered rooted tree is called as a 'Traversal Algorithm'.

Pre-Order: Let T be an ordered rooted tree with root r. Suppose T has one and only vertex say r, then r is the pre-order traversal of T. Suppose that $T_1, T_2, ..., T_k$ are the subtrees at r from left to right in T, then pre-order traversal begins by visiting r. It continues by traversing T_1 in pre-order, then T_2 in pre-order and so on, until T_k is reached.

NOTES

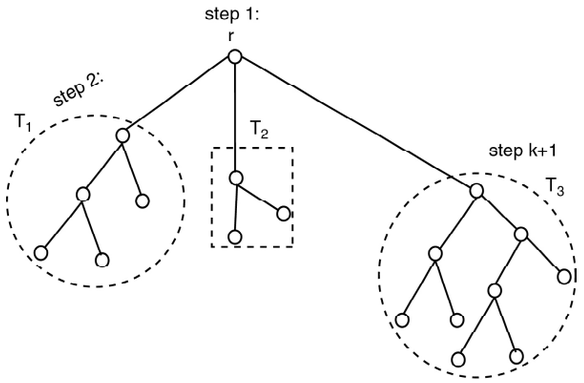


Fig. 12.3 *Pre-Order Traversal*

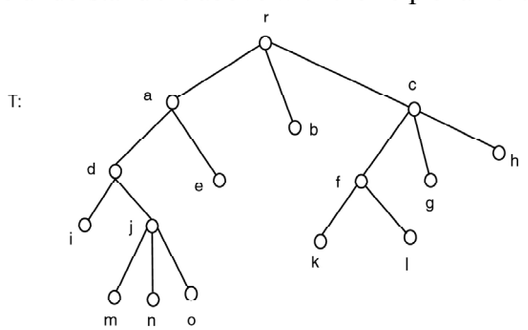
Step 1: Visit the root *r*.

Step 2: Visit T_1 in pre-order.

Step 3: Visit T_2 in pre-order.

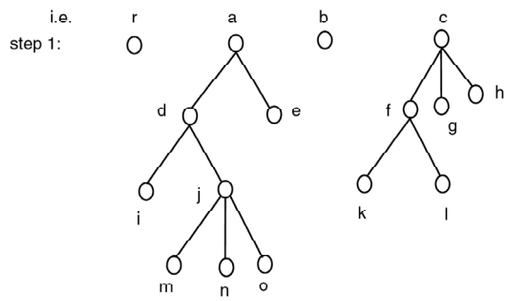
Step k+1: Visit T_k in pre-order.

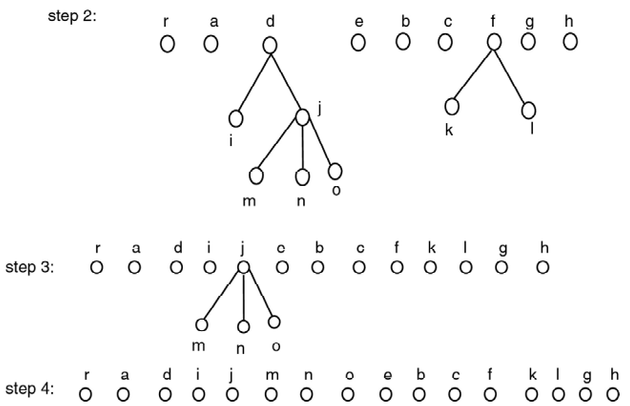
Let us try to understand the above with the help of an example.



Let *T* be an ordered root tree. The steps of the pre-order traversal of *T* are as follows:

We traverse *T* in pre-order by listing the root *r*, followed by the pre-order list of subtree with root *a*, the pre-order list of subtree with root *b*, and the pre-order list of subtree with root *c*.





Algorithm: Pre-Order Traversal

Step 1: Visit root *r* and then list *r.*

Step 2: For each child of *r* from left to right, list the root of first subtree then next subtree and so on until we complete listing the roots of subtrees at level 1.

Step 3: Repeat Step 2, until we arrive at the leaves of the given tree.

Step 4: Stop.

In-Order Traversal: Let T be an ordered rooted tree with its root at vertex r. Suppose T consists only root r, then r is the in-order traversal of T. If not, i.e., suppose T has subtrees $T_1, T_2, ..., T_k$ at r from left to right. The in-order traversal begins by traversing T_1 in-order, then visiting r. It continues by traversing T_2 in-order, then T_s inorder and so on and finally T_k in-order.

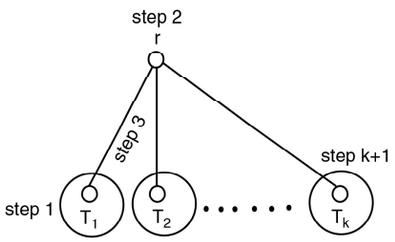


Fig. 12.4 *In-Order Traversal*

Step 1: Visit T_1 in-order.

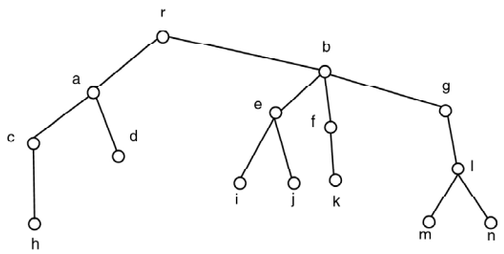
Step 2: Visit root.

Step 3: Visit T_2 in-order.

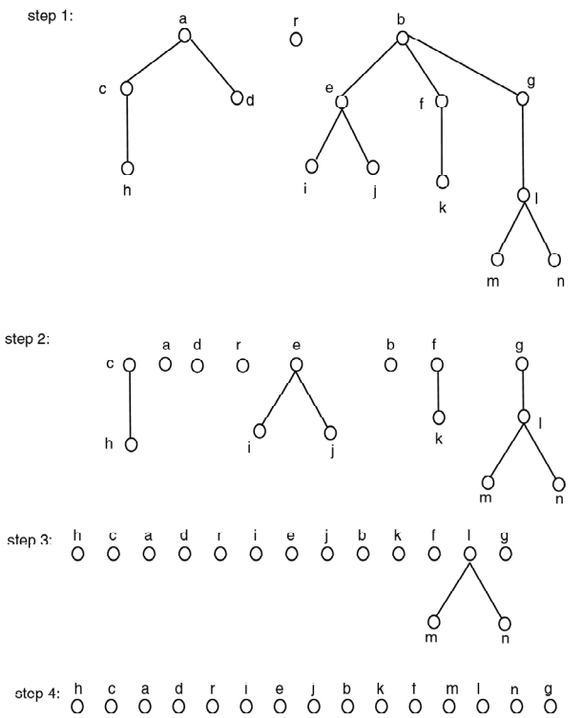
Step k+1: Visit T_k in-order.

Example 12.2: Determine the order in which the vertices of the following rooted tree is visited using an in-order traversal.

NOTES



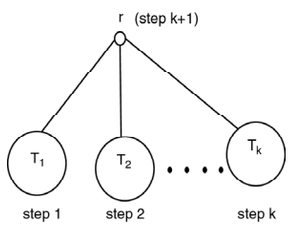
Solution: The in-order traversal begins with an in-order traversal of the subtree with root at *a*, followed by the root *r*, and the in-order listing of the subtree with root *b*.



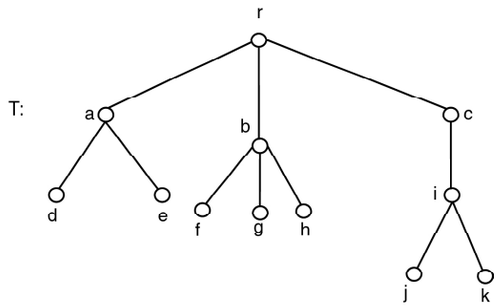
Definition of Post-Order Traversal

Let *T* be an ordered rooted tree with root *r*. If *T* has only one vertex *r,* then *r* is the post-order traversal of *T*. But if *T* has subtrees $T_1, T_2, ..., T_k$ at *r* from left to right, the post-order traversal begins by traversing T_1 in post-order, then T_2 in post-order, and so on, until T_k and ends by visiting *r*.

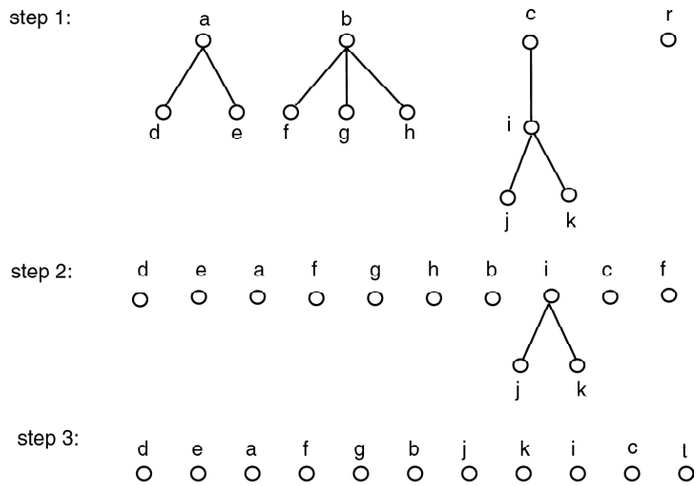
For example,



For example,



The post-order traversal begins with the post-order traversal of the subtree with root *a*, the post-order traversal of the subtree with root *b*, and the post-order traversal of the subtree with root *c*, followed by the root *r*.



Infix, Prefix, and Postfix Notation

We can represent any expression (like arithmetic, compound proposition) using ordered rooted trees. An ordered rooted tree can be used to represent expressions, where the internal vertices represent operations, the leaves represent the variables or numerals.

Example 12.3: What is the ordered rooted tree that represents the expression $((a + b)\uparrow 3) + ((a - 6)/3)$?

Solution: First construct a subtree for $a + b$. Then this tree is included as a part of the next subtree of $((a + b) \uparrow 3)$. Similarly a subtree is constructed for $(a - 6)$ then this tree is included as a part of the next subtree of $(a - 6)/3$. Finally the subtrees $((a + b)\uparrow 3)$ and $(a - 6)/3$ are combined to form the required tree corresponding to the given expression.

Ordered rooted tree corresponding to the expression $((a + b)\uparrow 3) + ((a - 6)/3)$.

NOTES

NOTES

An in-order traversal of the binary tree representing an expression, produces the original expression with the elements and operations in the same order as they originally appeared (except unary operator).

If we use parenthesis, whenever we encounter an operation there will be no ambiguity. Such fully parenthesised expression is said to be infix form.

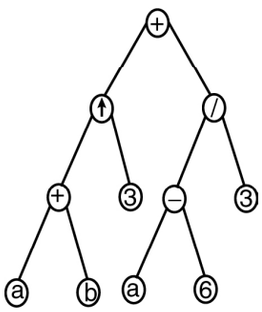
To get the *prefix form* of an expression, we traverse its rooted tree in pre-order.

Expressions written in prefix form are called Polish notation.

Example 12.4: What is the prefix form of $((a + b)\uparrow 3) + ((a - 6)/3)$?

Solution: The ordered rooted tree corresponding to the expression

$((a + b)\uparrow 3) + ((a - 6)/3)$ is,



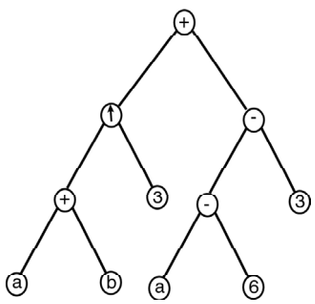
To obtain the prefix form of the given expression, we have to traverse the binary tree in pre-order. Prefix form of the expression,

$((a + b)\uparrow 3) + ((a - 6)/3)$ is $+\uparrow ab\ 3/-a\ 63$

We obtain the postfix form of an expression by traversing its binary tree in post-order.

Example 12.5: What is the postfix form of $((a + b)\uparrow 3) + ((a - 6)/3)$?

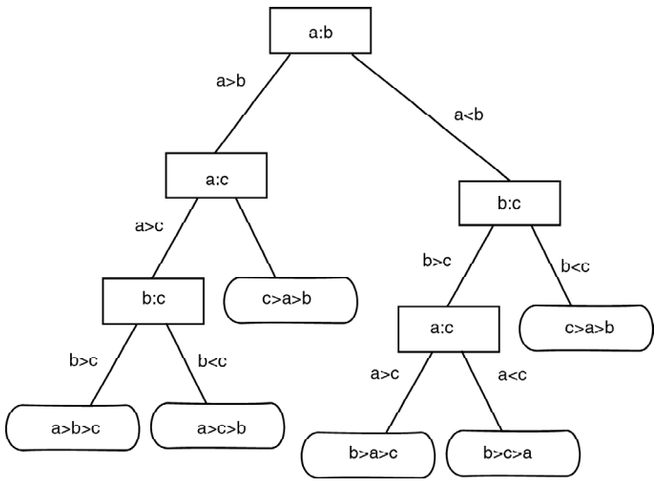
Solution: The binary tree corresponding to the expression is given as,



To obtain the postfix form of the given expression, we have to traverse its binary tree in post-order. The required post fix form is $ab + 3 \uparrow a6 - 3/+$.

Example 12.6: Draw the decision tree that orders the elements of the list *a*,*b*,*c*.

Solution: The following is decision tree that contains the list *a*,*b*,*c*.



12.4 DISTANCE AND CENTERS IN A TREES

In a rooted tree, every vertex has a path length which is given by the number of edges it has to traverse from the root to that vertex. Every vertex has a unique path length. A tree has been shown here. To find the path length of any vertex, one has to start from the root and travel up to that vertex. For example, path length of vertex D, E, F and G is 2 whereas for H, I and J it is 3 and for K and L it is 4.

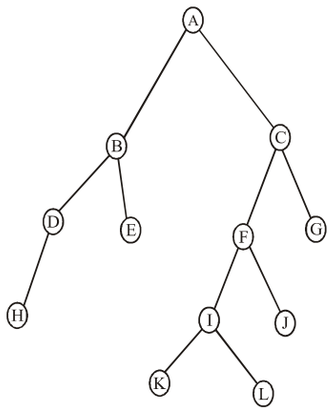


Fig. 12.5 *Path Length*

12.5 ROOTED AND BINARY TREES

Rooted Tree: In a directed tree (every edge assigned with a direction), a particular vertex is called a root if that vertex is of degree zero. A tree together with its root produces a graph called a rooted tree.

(Note that in the rooted tree, every edge is directed away from the root) For example,

NOTES

Suppose T is a rooted tree. If a vertex u is a vertex in T other than the root, the parent of u is the unique vertex u_1 such that there is a directed edge from u_1 to u. Here u is called as a child of u_1. Vertices of the same parent are called as siblings. A vertex of a rooted tree is called as a leaf if it has no children and those vertices which have children, are called as internal vertices.

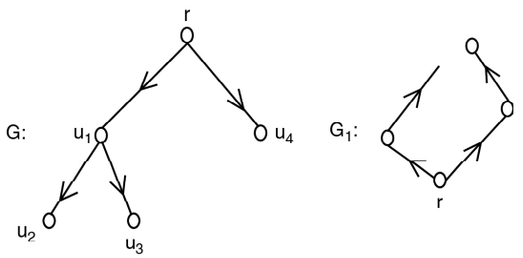


Fig. 12.6 *Rooted Trees*

If v is a vertex in a tree, the subtree with v as its root is the subgraph of the tree consisting of v and its children and all edges incident to these children.

For example,

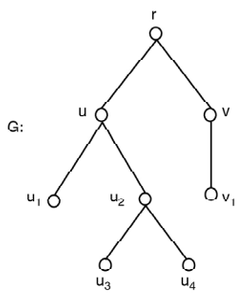


Fig. 12.7 *Rooted Tree T*

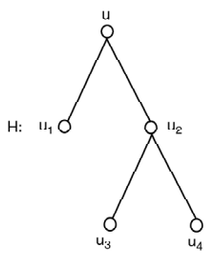


Fig. 12.8 *Subtree of T with its Root u*

k-Ary Tree: A rooted tree is called a k-ary tree if every internal vertex has, not more than k-children. The tree is called a full k-ary tree if every internal vertex has, exactly k-children. A k-ary tree with $k = 2$ is called a binary tree.

For example,

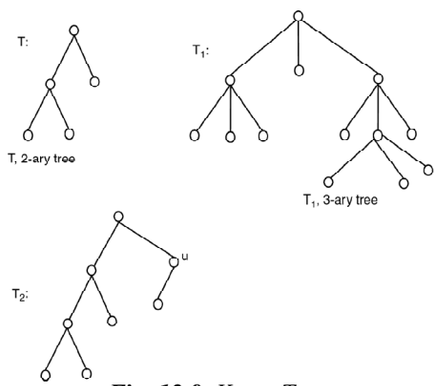


Fig. 12.9 *K-ary Trees*

T_2, not a 2-ary tree. (vertex u has only one child, whereas all the other vertices have two children).

A tree T is called as a binary tree if there is only one vertex with degree 2 and the remaining vertices are of degree 1 or 2.

NOTES

Example 12.7: Prove that a full k-ary tree with i-internal vertices contains k_{i+1} vertices.

Solution: In a full k-ary tree, every internal vertex has k-children and hence a full k-ary tree with i-internal vertices can have ki vertices. If we include the root, the tree has $k_i + 1$ vertices. By looking at the full k-ary tree, we can observe the following:

(*i*) n vertices has $i = (n-1)/k$ internal vertices and $p = [(k-1)n + 1]/k$ leaves.

(*ii*) i internal vertices has $n = ki + 1$ vertices and $p = (m-1)i + 1$ leaves.

(*iii*) p leaves has $n = (kp-1)/(k-1)$ vertices and $i = (p-1)/(k-1)$ internal vertices.

Level and Height in a Rooted Tree: The level of a vertex v in a rooted tree is the length of the path from the root to this vertex (Refer Figure 12.10). The height of a rooted tree is the length of the longest path from the root to any vertex.

For example,

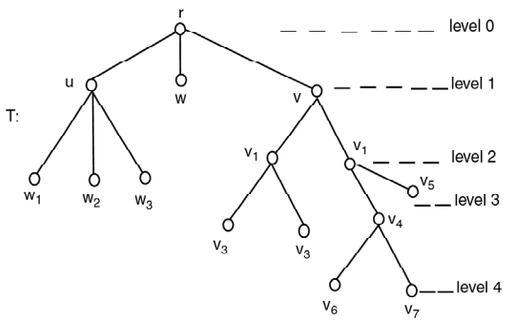


Fig. 12.10 *Levels of Trees*

A rooted tree T with its levels. Height of T is 4.

Balanced Tree: A rooted k-ary tree of height h is balanced if all the leaves are at level h or $(h-1)$.

Check Your Progress

1. What do you understand by a tree in graph theory?
2. Explain the properties of trees.
3. Define the binary search trees.
4. Elaborate on the decision trees.
5. State the traversal algorithm.
6. Explain the pre-order.

NOTES

7. Illustrate the in-order traversal.
8. Interpret the distance and centres in a tree.
9. Analyse the rooted tree.
10. Define the k-ary tree.

12.6 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. In mathematics, and more specifically in graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph without simple cycles is a tree. A forest is considered as a disjoint union of trees.
2. Acyclic Graph: A graph G which has no cycles is called an acyclic graph. Tree: A connected acyclic graph *G* is called a tree.
3. Binary search tree is a binary tree in which each child is either a left or right child; no vertex has more than one left child and one right child, and the data are associated with vertices.
4. A rooted tree in which each internal vertex is assigned to a decision with a subtree at the vertices, then each possible outcome of the decision is called a decision tree.
5. A systematic method for visiting every vertex of an ordered rooted tree is called as a 'Traversal Algorithm'.
6. Let T be an ordered rooted tree with root r. Suppose T has one and only vertex say r, then r is the pre-order traversal of T. Suppose that T_1, T_2, ..., T_k are the subtrees at r from left to right in T, then pre-order traversal begins by visiting r. It continues by traversing T_1 in pre-order, then T_2 in pre-order and so on, until T_k is reached.
7. Let T be an ordered rooted tree with its root at vertex r. Suppose T consists only root r, then r is the in-order traversal of T. If not, i.e., suppose T has subtrees $T_1, T_2, ..., T_k$ at r from left to right. The in-order traversal begins by traversing T_1 in-order, then visiting r. It continues by traversing T_2 in-order, then T_s inorder and so on and finally T_k in-order.
8. In a rooted tree, every vertex has a path length which is given by the number of edges it has to traverse from the root to that vertex. Every vertex has a unique path length. A tree has been shown here. To find the path length of any vertex, one has to start from the root and travel up to that vertex.
9. In a directed tree (every edge assigned with a direction), a particular vertex is called a root if that vertex is of degree zero. A tree together with its root produces a graph called a rooted tree.

10. A rooted tree is called a *k*-ary tree if every internal vertex has, not more than *k*-children. The tree is called a full *k*-ary tree if every internal vertex has, exactly *k*-children. A *k*-ary tree with $k = 2$ is called a binary tree.

NOTES

12.7 SUMMARY

- In mathematics, and more specifically in graph theory, a tree is an undirected graph in which any two vertices are connected by exactly one path. Alternatively, any connected graph without simple cycles is a tree. A forest is considered as a disjoint union of trees.
- Acyclic Graph: A graph G which has no cycles is called an acyclic graph. Tree: A connected acyclic graph *G* is called a tree.
- Binary search tree is a binary tree in which each child is either a left or right child; no vertex has more than one left child and one right child, and the data are associated with vertices.
- A rooted tree in which each internal vertex is assigned to a decision with a subtree at the vertices, then each possible outcome of the decision is called a decision tree.
- A systematic method for visiting every vertex of an ordered rooted tree is called as a 'Traversal Algorithm'.
- Let *T* be an ordered rooted tree with root *r*. Suppose *T* has one and only vertex say *r*, then *r* is the pre-order traversal of *T*. Suppose that T_1, T_2, ..., T_k are the subtrees at *r* from left to right in *T*, then pre-order traversal begins by visiting *r*. It continues by traversing T_1 in pre-order, then T_2 in pre-order and so on, until T_k is reached.
- Let *T* be an ordered rooted tree with its root at vertex *r*. Suppose *T* consists only root *r,* then *r* is the in-order traversal of *T*. If not, i.e., suppose *T* has subtrees $T_1, T_2, ..., T_k$ at *r* from left to right. The in-order traversal begins by traversing T_1 in-order, then visiting *r*. It continues by traversing T_2 in-order, then T_s inorder and so on and finally T_k in-order.
- In a rooted tree, every vertex has a path length which is given by the number of edges it has to traverse from the root to that vertex. Every vertex has a unique path length. A tree has been shown here. To find the path length of any vertex, one has to start from the root and travel up to that vertex.
- In a directed tree (every edge assigned with a direction), a particular vertex is called a root if that vertex is of degree zero. A tree together with its root produces a graph called a rooted tree.
- A rooted tree is called a *k*-ary tree if every internal vertex has, not more than *k*-children. The tree is called a full *k*-ary tree if every internal vertex has, exactly *k*-children. A *k*-ary tree with $k = 2$ is called a binary tree.

NOTES

12.8 KEY WORDS

- **Trees:** A tree is an undirected graph in which any two vertices are connected by exactly one path.
- **Acyclic graph:** A graph *G* which has no cycles is called an cyclic graph.
- **Binary search tree:** Binary search tree is a binary tree in which each child is either a left or right child; no vertex has more than one left child and one right child, and the data are associated with vertices.
- **Decision trees:** A rooted tree in which each internal vertex is assigned to a decision with a subtree at the vertices, then each possible outcome of the decision is called a decision tree.
- **Traversal algorithm:** A systematic method for visiting every vertex of an ordered rooted tree is called as a traversal algorithm.
- **Rooted tree:** In a directed tree (every edge assigned with a direction), a particular vertex is called a root if that vertex is of degree zero.
- ***K*-ary tree:** A rooted tree is called a *k*-ary tree if every internal vertex has, not more than *k*-children.
- **Balanced tree:** A rooted *k*-ary tree of height *h* is balanced if all the leaves are at level *h* or (*h*–1).

12.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain the tree.
2. Elaborate on the properties of trees.
3. Analyse the binary search trees.
4. Illustrate the decision trees.
5. Define the traversal algorithm.
6. State the pre-order.
7. Explain the in-order traversal.
8. Interpret the distance and centres in a tree.
9. Describe the rooted tree.
10. Analyse the *k*-ary tree.

Long-Answer Questions

1. Discuss briefly the trees with the help of example.

2. Explain the properties of trees.
3. Elaborate on the binary search trees. Give appropriate example.
4. Define the post-order traversal.
5. Analyse the distance and centres in a tree.
6. Describe briefly rooted tree.
7. Interpret the *k*-ary tree. Give example.

NOTES

11.10 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 13 SPANNING TREES

NOTES

Structure

13.0 INTRODUCTION

In graph theory, a spanning tree T of an undirected graph G is a subgraph, i.e., a tree which includes all of the vertices of G. In general, a graph may have several spanning trees, but a graph that is not connected will not contain a spanning tree (see spanning forests below). If all of the edges of G are also edges of a spanning tree T of G, then G is a tree and is identical to T (that is, a tree has a unique spanning tree and it is itself).

A special kind of spanning tree, the Xuong tree, is used in topological graph theory to find graph embedding with maximum genus. A Xuong tree is a spanning tree such that, in the remaining graph, the number of connected components with an odd number of edges is as small as possible. A Xuong tree and an associated maximum-genus embedding can be found in polynomial time.

A tree is a connected undirected graph with no cycles. It is a spanning tree of a graph G if it spans G (that is, it includes every vertex of G) and is a subgraph of G (every edge in the tree belongs to G). A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices. Adding just one edge to a spanning tree will create a cycle; such a cycle is called a fundamental cycle. There is a distinct fundamental cycle for each edge not in the spanning tree; thus, there is a one-to-one correspondence between fundamental cycles and edges not in the spanning tree. For a connected graph with V vertices, any spanning tree will have $V - 1$ edges, and thus, a graph of E edges and one of its spanning trees will have $E - V + 1$ fundamental cycles (The number of edges subtracted by number of edges included in a spanning tree; giving the number of edges not included in the spanning tree). For any given spanning tree the set of all $E - V + 1$ fundamental cycles forms a cycle basis, a basis for the cycle space.

In this unit, you will study about the spanning trees, fundamental circuits, finding all spanning trees of a graph, and spanning trees in a weighted graph.

13.1 OBJECTIVES

After going through this unit, you will be able to:

- Analyse the spanning trees
- Define the fundamental circuits
- Finding out all spanning trees of a graph
- Elaborate on the spanning trees in a weighted graph

NOTES

13.2 SPANNING TREES

A connected graph might contain more than one spanning tree. Consider the following graphs shown in Figure 13.1 illustrating the concept of spanning tree.

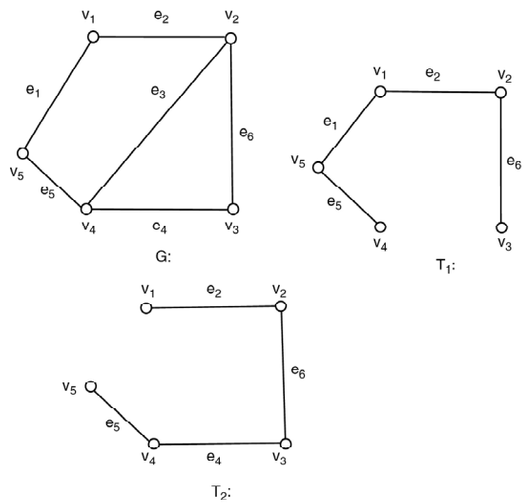


Fig. 13.1 *Spanning Trees*

In T_1, the edges e_1, e_2, e_5, e_6 are present, whereas in T_2, edges e_2, e_4, e_5, e_6 are present.

Edges of G, which are present in a spanning tree T, are called as the *branches* of G with respect to T. The edges of G, which are not present in its spanning tree T, are called the *chords* of G with respect to T.

In the above example, the branches of G are e_1, e_2, e_5, e_6, with respect to T_1 and the branches of G are e_2, e_4, e_5, e_6, with respect to T_2.

Note: Let G be a connected graph on n vertices; e-edges and T be one of its spanning tree. Since T is a tree on n vertices, it has $(n-1)$ edges, i.e., the number of branches of G with respect to T is $(n-1)$; the number of chords of G with respect to T is $e-(n-1)$. Often the number of branches of G is called as rank of G and is denoted by $r(G)$; the number of chords of G is called as the nullity of G, denoted by $\mu(G)$. In general, for a connected graph on n-vertices and e-edges, $r(G)$, the rank of G is $(n-1)$; $\mu(G)$, the nullity of G is $e-n+1$.

NOTES

13.3 FUNDAMENTAL CIRCUITS

Let *T* be the spanning tree of a connected graph *G*, and *e* be a chord of *G* with respect to *T*. Since the spanning tree *T* is minimally acyclic, the graph *T*+*e* contains a unique cycle. This cycle is called a fundamental cycle in *G* with respect to *T*.

Every chord of *G* gives rise to a fundamental cycle. Therefore, the number of fundamental cycles possible for a connected graph is atmost $\mu(G)$. (Refer Figure 13.2).

For example,

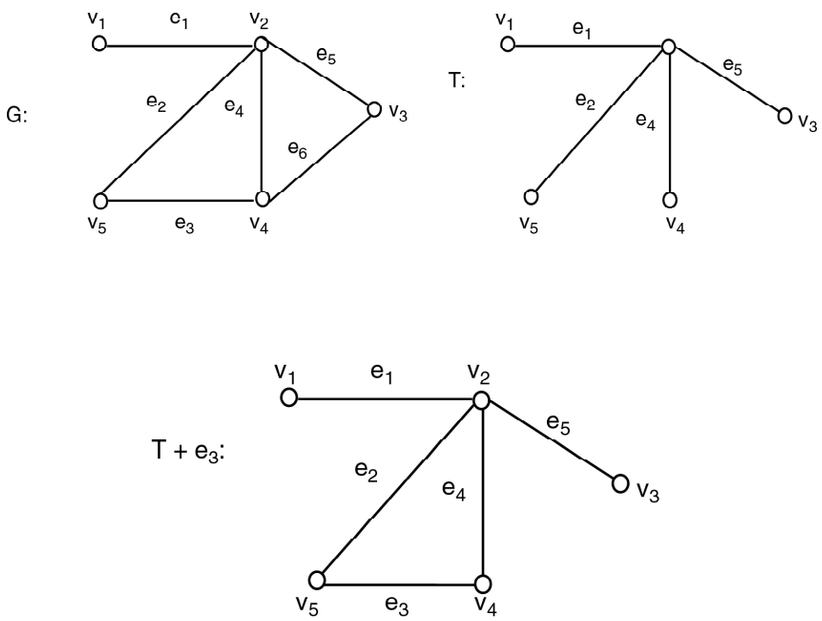


Fig. 13.2 *Fundamental Circuit in a Graph*

13.4 FINDING ALL SPANNING TREES OF A GRAPH

We can build the spanning tree of a connected graph using DFs and BFs. First we shall see how DFs are useful in construction of a spanning tree from a given connected graph.

Depth-First Search

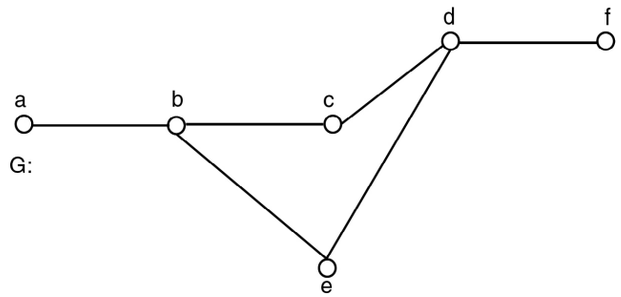
Let *G* be the given connected graph. Arbitrarily choose a vertex as the root. Find a path starting from this choosen vertex by successively adding edges, where each edge is incident with the last vertex in the path and a vertex not already in the path.

NOTES

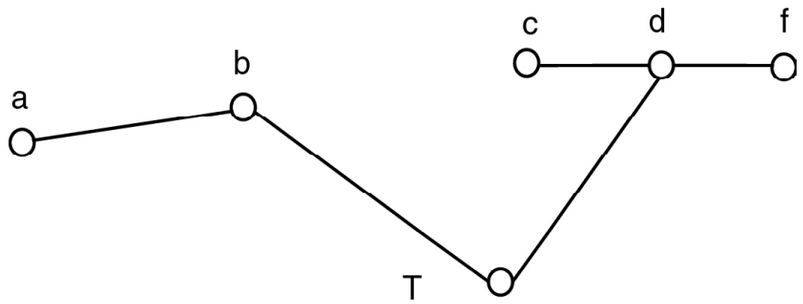
Continue adding edges to this path as long as possible. If this path consists of all the vertices of *G*, this path is the required spanning tree. If not, more edges should be added. Move back to the next to last vertex in this path, and if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this is not possible, move back to another vertex in this path (i.e., 2 vertices back from the last) and try again. Repeat this procedure, beginning at the last vertex visited, moving back up the path one vertex at a time, forming new long paths until no more edges can be added. This process ends with a spanning tree.

When this procedure returns to vertices previously visited, it is also called as backtracking.

Example 13.1: Construct a spanning tree for the following graph *G*.



Solution: First we choose arbitrarily a vertex say *e* as the root. Form a path at *e*, i.e., *c d f* is the path. Backtrack to *d.* Form a path beginning at *d* in such a way that it has to visit the vertices which where not visited in the previous path, *d e b a*. Since all the vertices of *G* are visited, this procedure gives the spanning tree *T*.

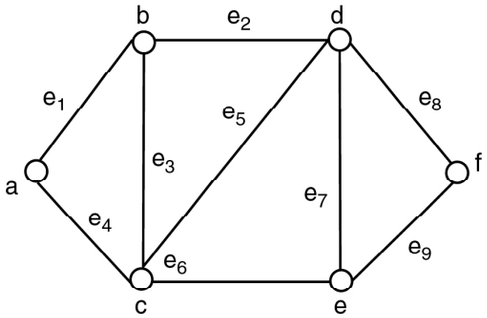


Breadth-First Search

First choose a vertex arbitrarily as the root. Add the edges of *G* which are incident with this vertex. The new vertices added at this stage becomes level 1 in the spanning tree. Order these vertices arbitrarily. Next, for each vertex at level 1 visited in order, add each edge incident to this vertex to the tree as long as it does not create a simple circuit. Order the children of each vertex at level 1 arbitrarily. This produces the vertices at level 2 in the tree. Continue in this manner until all the vertices of *G* have been added. Ultimately we end with a spanning tree.

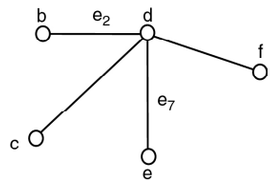
NOTES

Example 13.2: Construct a spanning tree of the following graph *G*.



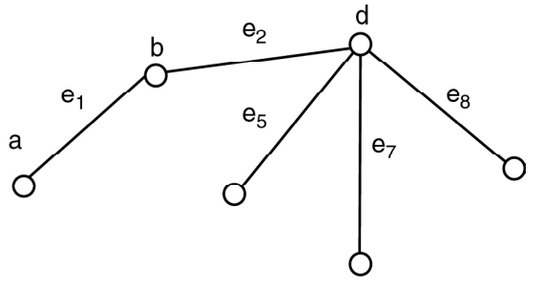
Solution: First choose a vertex say *d* (arbitrarily) as the root. Add the edges incident to this vertex *d*. Hence the edges e_2,e_5,e_7,e_8 are incident with the vertex *d*. These vertices create level 1.

i.e.,



Now add the edges which are incident to the vertices *b,c,e,f* in such a way that the resulting graph does not contain any circuit.

T:



Thus at this level itself we have got the spanning tree T.

Note: If the given graph is directed graph then we construct the underlying undirected graph and apply DFs or BFs to obtain a spanning graph.

13.5 SPANNING TREES IN A WEIGHTED GRAPH

Let *T* be a spanning tree of *G* and *e* be a chord of *G* with respect to *T*. The graph *T*+*e* is a fundamental circuit. In this circuit other than edge *e*, all the other edges are branches of *G* with respect to *T*. On removal of any of the branches from the fundamental circuit, we get a spanning tree T_1, i.e., *b* is a branch in the fundamental circuit with respect to a chord *e*, then spanning tree T_1 is obtained by removing *b* from $T + e$, i.e, $T_1 = T + e - b$. This process is called cyclic interchange.

For example,

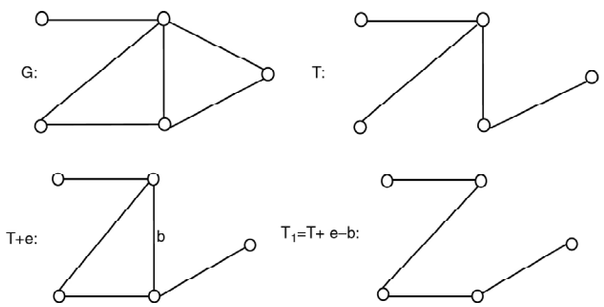


Fig. 13.3 *Cycle Interchange*

G – Connected Graph, T – Spanning Tree

T + e – Fundamental Circuit, T_1 – Spanning Tree Obtained by Cyclic Interchange

Check Your Progress

1. Define the spanning tress.
2. Analyse the fundamental circuits.
3. Explain the depth-first search.
4. Elaborate on the breath-first search.
5. Interpret the spanning trees in a weighted graph.

13.9 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. A connected graph might contain more than one spanning tree. Consider the following graphs shown in Figure illustrating the concept of spanning tree.
2. Let *T* be the spanning tree of a connected graph *G*, and *e* be a chord of *G* with respect to *T*. Since the spanning tree *T* is minimally acyclic, the graph *T*+*e* contains a unique cycle. This cycle is called a fundamental cycle in *G* with respect to *T*.
3. Let *G* be the given connected graph. Arbitrarily choose a vertex as the root. Find a path starting from this choosen vertex by successively adding edges, where each edge is incident with the last vertex in the path and a vertex not already in the path. Continue adding edges to this path as long as possible. If this path consists of all the vertices of *G*, this path is the required spanning tree.
4. First choose a vertex arbitrarily as the root. Add the edges of *G* which are incident with this vertex. The new vertices added at this stage becomes level 1 in the spanning tree. Order these vertices arbitrarily. Next, for each vertex at level 1 visited in order, add each edge incident to this vertex to the tree as long as it does not create a simple circuit.

NOTES

NOTES

5. Let T be a spanning tree of G and e be a chord of G with respect to T. The graph $T+e$ is a fundamental circuit. In this circuit other than edge e, all the other edges are branches of G with respect to T. On removal of any of the branches from the fundamental circuit, we get a spanning tree T_1, i.e., b is a branch in the fundamental circuit with respect to a chord e, then spanning tree T_1 is obtained by removing b from $T + e$, i.e, $T_1 = T + e - b$. This process is called cyclic interchange.

13.7 SUMMARY

- A connected graph might contain more than one spanning tree.
- Let T be the spanning tree of a connected graph G, and e be a chord of G with respect to T. Since the spanning tree T is minimally acyclic, the graph $T+e$ contains a unique cycle. This cycle is called a fundamental cycle in G with respect to T.
- Let G be the given connected graph. Arbitrarily choose a vertex as the root. Find a path starting from this choosen vertex by successively adding edges, where each edge is incident with the last vertex in the path and a vertex not already in the path. Continue adding edges to this path as long as possible. If this path consists of all the vertices of G, this path is the required spanning tree.
- First choose a vertex arbitrarily as the root. Add the edges of G which are incident with this vertex. The new vertices added at this stage becomes level 1 in the spanning tree. Order these vertices arbitrarily. Next, for each vertex at level 1 visited in order, add each edge incident to this vertex to the tree as long as it does not create a simple circuit.
- Let T be a spanning tree of G and e be a chord of G with respect to T. The graph $T+e$ is a fundamental circuit. In this circuit other than edge e, all the other edges are branches of G with respect to T. On removal of any of the branches from the fundamental circuit, we get a spanning tree T_1, i.e., b is a branch in the fundamental circuit with respect to a chord e, then spanning tree T_1 is obtained by removing b from $T + e$, i.e, $T_1 = T + e - b$. This process is called cyclic interchange.

13.8 KEY WORDS

- **Spanning trees:** A connected graph might contain more than one spanning tree. Edges of G, which are present in a spanning tree T, are called as the branches of G with respect to T.
- **Fundamental circuits:** Let T be the spanning tree of a connected graph G, and e be the chord of G with respect to T. Since the spanning tree T is minimally acyclic, the graph $T+e$ contains a unique cycle. This cycle is called a fundamental cycle in G with respect to T.

- **Depth-first search:** Let *G* be the given connected graph. Arbitrarily choose a vertex as the root. Find a path starting from this choose vertex by successively adding edges, where each edge is incident with the last vertex in the path and a vertex not already in the path.
- **Breadth-first search:** First choose a vertex arbitrarily as the root. Add the edges of *G* which are incident with this vertex. The new vertices added at this stage becomes level 1 in the spanning tree.

13.9 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Explain the spanning trees.
2. Elaborate on the fundamental circuits.
3. Analyse the depth-first search.
4. State the breath-first search.
5. Define the spanning trees in a weighted graph.

Long-Answer Questions

1. Discuss briefly the spanning trees with the help of example.
2. Describe the fundamental circuits. Give appropriate example.
3. Analyse the depth-first search in brief.
4. Elaborate on the breadth-first search.
5. Explain briefly the spanning trees in a weighted graph.

13.10 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

NOTES

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.

UNIT 14 CUT SETS AND CONNECTIVITY OF GRAPHS

Structure

14.0 INTRODUCTION

In graph theory, a cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut, and in some cases cuts are identified with their cut-sets rather than with their vertex partitions.

In a flow network, an s–t cut is a cut that requires the source and the sink to be in different subsets, and its cut-set only consists of edges going from the source's side to the sink's side. The capacity of an s–t cut is defined as the sum of the capacity of each edge in the cut-set.

A cut $C = (S, T)$ is a partition of V of a graph $G = (V, E)$ into two subsets S and T. The cut-set of a cut $C = (S, T)$ is the set $\{(u, \upsilon) \in E, \upsilon \in T\}$ of edges that have one endpoint in S and the other endpoint in T. If s and t are specified vertices of the graph G, then an s–t cut is a cut in which s belongs to the set S and t belongs to the set T.

In an unweighted undirected graph, the size or weight of a cut is the number of edges crossing the cut. In a weighted graph, the value or weight is defined by

NOTES

the sum of the weights of the edges crossing the cut. A bond is a cut-set that does not have any other cut-set as a proper subset.

In this unit, you will study about the cut sets, properties of a cut set, all cut sets in a graph, fundamental circuits and cut sets, connectivity and separability, Eulerian and Hamiltonian graphs.

14.1 OBJECTIVES

After going through this unit, you will be able to:

- Comprehend the cut sets
- Understand the properties of a cut sets
- Explain all cut sets in a graph
- Elaborate on the fundamental circuits and cut sets
- Define the connectivity and separability
- Analyse the Eulerian and Hamiltonian graphs

14.2 CUT SET

In graph theory, a cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut, and in some cases cuts are identified with their cut-sets rather than with their vertex partitions.

In a flow network, an s–t cut is a cut that requires the source and the sink to be in different subsets, and its cut-set only consists of edges going from the source's side to the sink's side. The capacity of an s–t cut is defined as the sum of the capacity of each edge in the cut-set.

A cut $C = (S, T)$ is a partition of V of a graph $G = (V, E)$ into two subsets S and T. The cut-set of a cut $C = (S, T)$ is the set $\{(u, \upsilon) \in E, \upsilon \in T\}$ of edges that have one endpoint in S and the other endpoint in T. If s and t are specified vertices of the graph G, then an s–t cut is a cut in which s belongs to the set S and t belongs to the set T.

In an unweighted undirected graph, the size or weight of a cut is the number of edges crossing the cut. In a weighted graph, the value or weight is defined by the sum of the weights of the edges crossing the cut. A bond is a cut-set that does not have any other cut-set as a proper subset.

14.3 PROPERTIES OF A CUT SETS

NOTES

Cut-Vertex: A vertex v in a graph G is said to be a cut-vertex if $\omega(G - v) > \omega(G)$, where $\omega(G)$ is the component of G and component is a maximal connected subgraph of G, i.e., a vertex v of a connected graph is a cut-vertex, iff $(G - v)$ is disconnected.

For example, in Figure 14.1 the Cut-vertices and Cut-edges of graph G is shown.

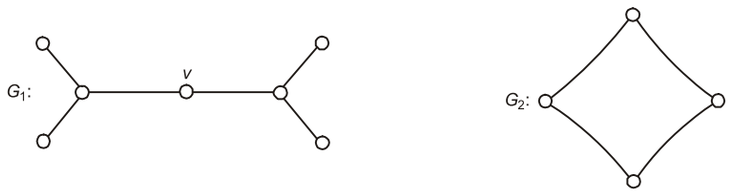


Fig. 14.1 *Cut-Vertices and Cut-Edges*

G_1 contains one cut-vertex v and G_2 contains no cut-vertices.

Theorem 14.1: *A vertex v in a connected graph G is a cut-vertex iff there exists vertices u and w (both are different from v) such that every path connecting u and w contain v.*

Proof: Let G be a connected graph and v be a cut-vertex.

***Corollary* 1:** There exists vertices u and w such that every path between u and w contains v.

Since v is a cut-vertex, $(G - v)$ is disconnected and $(G - v)$ contains two components say G_1 and G_2. Let u and w be the vertices of G_1 an G_2 respectively. Clearly, there is no $(u - w)$ path in $(G - v)$. Hence, every path connecting u and w must contain v.

Conversely, let us assume that there exist vertices u and w such that every $(u - w)$ path contains v.

***Corollary* 2:** v is a cut-vertex.

Suppose v is not a cut-vertex. Then $(G - v)$ is connected. Since u, w are vertices in $G - v$, there is a path between u and w in $G - v$, which does not contain the vertex v. This is a contradiction. Hence, v is a cut-vertex.

Cut-Edge: An edge e in a graph G is said to be a Cut-edge, if $(G - e)$ is disconnected.

For example, in Figure 14.2 the graph G_1 contains one Cut-edge and graph G_2 contains no Cut-edge.

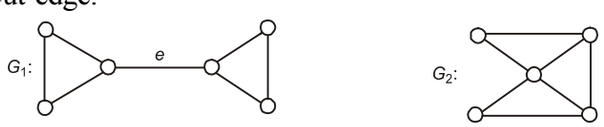


Fig. 14.2 *G_1 contains One Cut-edge and G_2 has No Cut-edge*

NOTES

As in Cut-vertex, a similar result can be furnished.

Theorem 14.2: *An edge e in a connected graph G is a cut-edge iff there exists vertices u and w such that every path connecting u and w must contain the edge 'e'.*

Proof: Let G be a connected graph and e be a cut-edge.

***Corollary* 1:** There exist vertices u and w such that every $(u - w)$ path must contain the edge e, since e is a cut-edge in G, $(G - e)$ is disconnected and $(G - e)$ contains atleast two components say G_1 and G_2.

Let u and w be the vertices respectively in G_1 and G_2. Thus, there is no path between u and w in $(G - e)$. Hence, every path connecting u and w must contain the edge e.

Conversely, suppose that there exist vertices u and w such that every path connecting u and w must contain the edge e.

14.4 ALL CUT SET IN A GRAPH

Let G be a connected graph. Let us recollect the definition of cut-edge (bridge) and cut-vertex. If G contains an edge e such that $G–e$ is disconnected, then e is a bridge of G. Further if G contains a vertex v such that $G–v$ is disconnected, then v is a cut-vertex of G.

Edge Cut-Set: A subset S of the edge set of a connected graph G is called an edge cut-set or cut-set of G if

(i) $G - S$ is disconnected.

(ii) $G - S_1$ is connected for every proper subset S_1 of S.

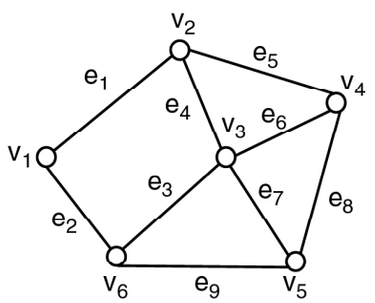
Vertex Cut-Set: A subset u of the vertex set of G is called a vertex cut-set if

(i) $G - u$ is disconnected

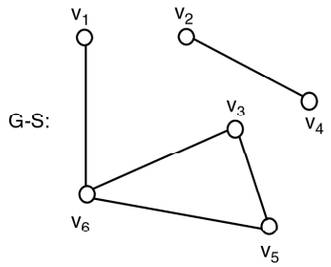
(ii) $G - u_1$ is connected for every proper subset u_1 of u.

For Example:

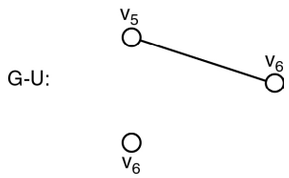
G:



(a) $S = \{e_1, e_4, e_6, e_8\}$ is a cut-set



(b) $u = \{v_1, v_3, v_5\}$ is a vertex-cut-set



Note: For a connected graph, there may be more than one cut-set.

For example, consider the above graph *G*. Some cut-sets of *G* are

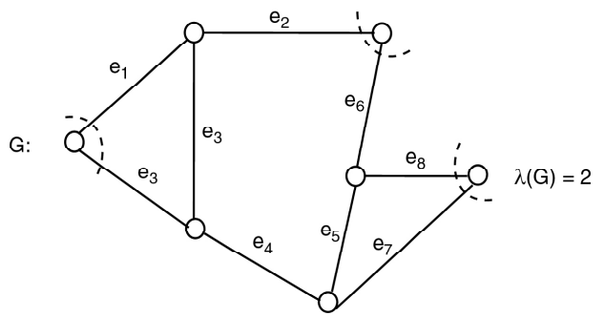
$S_1 = \{e_1, e_4, e_6, e_8\}$

$S_2 = \{e_1, e_2\}$, $S_3 = \{e_1, e_3, e_9\}$

From the above note, we are forced to introduce two more parameters for graphs viz edge-connectivity $\lambda(G)$ and vertex connectivity $k(G)$.

Edge Connectivity: The edge connectivity $\lambda(G)$ of a graph is the minimum cardinality of a set *S* of edges of *G* such that *G–S* is disconnected, i.e., the edge (line) connectivity of a connected graph is the number of edges in a minimum cut-set in the graph.

For Example:



Note:

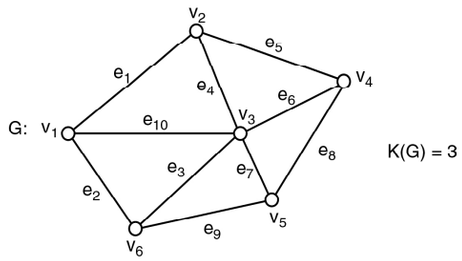
1 If *G* is a tree then $\lambda(G) = 1$.

2 *G* has $\lambda(G) = 0$ iff *G* is disconnected or trivial.

Vertex Connectivity: The *vertex connectivity K*(G) of a graph G is the minimum number of vertices whose deletion makes *G* a disconnected or trivial graph, i.e., the number of vertices in a minimum *vertex* cut is called the connectivity of the graph.

NOTES

For Example:



Result For every graph *G*, $K(G) \leq \lambda(G) \leq \delta(G)$

Proof Let *v* be a vertex of *G* with minimum degree. i.e., $d(v) = \delta(G)$. Removing $\delta(G)$ edges of *G* incident with *v* produces a graph G_1, in which *v* is isolated. Clearly *G* is disconnected or trivial.

$$\therefore \lambda(G) \leq \delta(G) \qquad ...(14.1)$$

Claim $K(G) \leq \delta(G)$

If $\delta(G) = 0$ then *G* is disconnected.

$\therefore K(G) = 0$.

If $\delta(G) = 1$ then *G* is a connected graph containing a cut-edge (bridge). Therefore either *G* is isomorphic to K_2 or *G* is a connected graph having atleast one cut-vertex.

$\therefore$ In both cases, $K(G) = 1$.

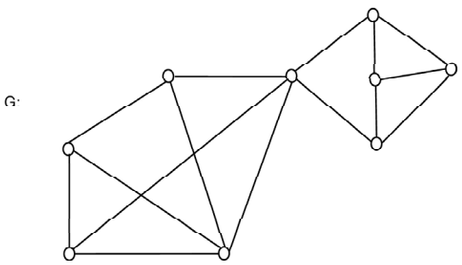
Now let us assume that $\lambda(G) \geq 2$. Let *S* be a cut-set of *G* with $\lambda(G)$ edges and $e = xy$ be an edge in *S*. If the edges of $S-\{e\}$ are deleted from *G*, the resulting subgraph H_1 is connected and contains *e* as a cut-edge. Now select an incident vertex different from *x* and *y* for each and every edge in $S-\{e\}$ .Remove these vertices from H_1, the resulting subgraph H_2 is disconnected, then

$K(G) \leq \lambda(G) - 1 < \lambda(G)$.

Suppose the subgraph H_2 is connected, then H_2 is isomorphic to K_2 or the subgraph H_2 has a cut-vertex [since H_2 is an induced subgraph of H_1]. In any case, there exists a vertex of H_2 whose removal results in a disconnected graph. Therefore

$$K(G) \leq \lambda(G) \qquad ...(14.2)$$

From Equations (14.1) and (14.2), $K(G) \leq \lambda(G) \leq \delta(G)$



In the figure $K(G) = 1$; $\lambda(G) = 3$ and $\delta(G) = 3$.

n-Edge Connected: A graph G is n-edge connected ($n \geq 1$) if $\lambda(G) \geq n$ and G is n-connected if $K(G) \geq n$.

NOTES

Check Your Progress

1. What do you understand by the cut sets?
2. Explain the cut vertex.
3. Define cut edge.
4. What is edge cut-set?
5. Elaborate on the vertex cut-set.
6. Explain the edge connectivity.
7. Analyse the vertex connectivity.

14.5 FUNDAMENTAL CIRCUITS AND CUT SET

Let T be a spanning tree in a connected graph G. When a chord is added to a spanning tree T then it forms exactly one circuit. Such a circuit is termed as a fundamental circuit.

Theorem 14.3: If a connected graph G it ehre is no circuit, then G itselt is considered as a tree through all its vertices. Therefore, G is its own spanning tree.

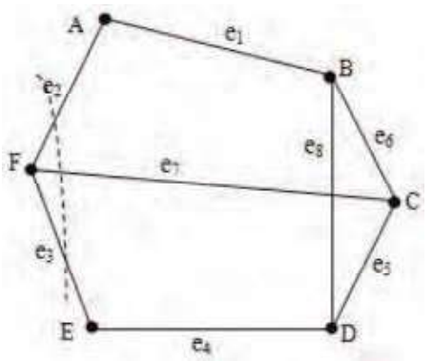


Fig. 14.3 *Fundamental Circuit*

Now, If G is has one or more circuits, then select any circuit and delete an edge from the circuit. In the selected circuit there atleast one more path that connects the end vertices of the deleted edge. Consequently the delection of the edge will leave the graph connected.

If there remains more circuits after the deletion then again select any other circuit from the graph and delete any one of its edge. Similarly the resultant graph will also remain connected.

NOTES

Keep repeating the delection process so that all the circuits are 'broken' and the resultant subgraph is connected and circuit-free which contains all the vertices of *G*.

Subsequently, at the end of the total producure we will obtain a spanning tree.

Hence, it proves that every connected graph has atleast one spanning tree.

Example 14.1: Describe a method of find all spanning tree of a graph.

Solution: Let *G* be a connected graph.

If *G* is a tree then *G* itself will be one and only one spanning tree of *G*.

Now, as shown in the Figure 14. consider a connected graph *G*. It is not a tree because it has one circuit. Let T_1 be a spanning tree of *G* that contains the branches *a, b, c, d*.

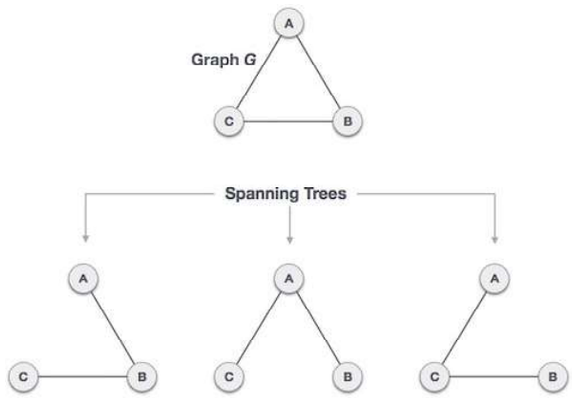


Fig. 14.4 *Finding a Spanning Tree*

Add a chord, say *h,* to the tree which will form a fundamental circuit through *b, c, h, d*. Removal of the branch *c* of T_1 from the fundamental circuit *b, c, h, d* will break the circuit and will create another spanning tree, say T_2.

Instead of deleting *C*, if we delete *d* or *b* then we will obtain two more different spanning trees, namely *a, b, c, h* and *a, d, h, c*. This process generates all possible trees corresponding to the chord *h and associated fundamental circuit.*

Restarting with the initial tree T_1 and repeating the process of deletion or removal with the chord *h*, using another chord *e* or *f* or *g* you can obtain all possible different spanning trees corresponding to each chord addition to T_1.

Therefore, we can obtain all possible spanning trees of a connected graph.

14.5.1 Fundamental Cut Sets

For defining the concept of Cut Set, let us consider a spanning tree *T* of a connected graph *G*. In graph *G* take any branch *b* in *T.* Subsequently (b) is cut set in *T,* therefore (b) partitions all vertices of *T* into two disjoint sets–one at each end of *b.*

Consdider the same partition of vertices in *G*, and the cut set *S* in *G* that corresponds to this partition. Cut set *s* will contain only one branch *b* of *T*, and the rest (if any) of the edges in *S* will be referred as chords with respect to *T*. This cut-set *S* containing exactly one branch of a tree *T* is termed as **Fundamental Cut Set** with respect to *T*. In addition a fundamental cut set is also termed as **Basic Cut Set.**

In Figure 14.5, a spanning tree *t* (shown with dark lines) and all five of the fundamental cut sets with respect to *T* are shown with broken lines cutting through each cut set.

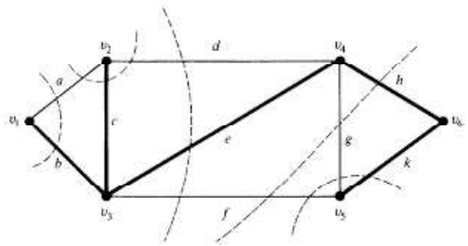


Fig. 14.5 *Fundamental Cut Set of a Graph*

Every chord of a spanning tree defines a **Unique Fundamental Circuit** while every branch of a spanning tree defines a **Unique Fundamental Cut Set**. Remember that the term fundamental cut set can be defined only with respect to a given spanning tree.

The cut sets of a graph can be obtained from a given set of cut sets.

Theorem 14.4: The ring sum of any two cut sets in a graph is either a third cut set or an edge-disjoint union of cut sets.

In the Figure 14.5 consider that ring sums of the following three pairs of cut sets are given:

$\{d, e, f\} \oplus \{f, g, h\} = \{d, e, g, h\}$, Another cut set.

$\{a, b\} \oplus \{b, c, e, f\} = \{a, c, e, f\}$, Another cut set.

$\{d, e, g, h\} \oplus \{f, g, k\} = \{d, e, f, h, k\}$,

$= \{d, e, f\} \cup \{h, k\}$, An edge-disjoint Union of cut sets

14.6 CONNECTIVITY AND SEPARABILITY

In this section, we study the structure of graphs. A walk in a graph *G* is an alternating sequence.

$W : v_0, e_1, v_1, e_2, ..., v_{n-1}, e_n, v_n$ $(n \geq 0)$ of vertices and edges, beginning and ending with vertices, such that $e_i = v_{i-1}\, v_i$, $i = 1, 2, ..., n$. It is denoted by

NOTES

$(v_0 - v_n)$ walk. The number of edges (not necessarily distinct) is called the length of walk. In graph G, u, e_1, v, e_2, w, e_6, x, e_4, u is a walk of length 4 (Refer Figure 14.6).

The following figure illustrates the path and walk in a graph:

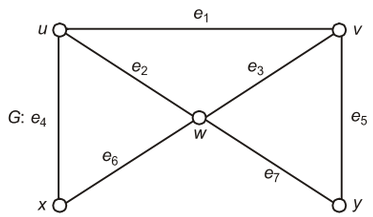


Fig. 14.6 *Path and Walk in a Graph*

A trail is a *walk* in which no edge is repeated and a *path* is a trail in which no vertex is repeated. Thus, a path is a trail, but not every trail is a *path*. In the above graph $G, x, e_6, w, e_3, v, e_1, u, e_2, w, e_7, y$ is a *trail* that is not a path, and $u, e_4, x, e_6, w, e_3, v$ is a path.

Theorem 14.5: *Every (u – v) walk in a graph contains a (u – v) path.*

Proof: Let W be a $(u - v)$ walk in a graph G. If $u = v$, then w is the trail path, i.e., walk of length zero.

Suppose $u \neq v$ and $W : u = u_0, u_1, u_2,..., u_n = v$. If no vertex of G appears in W more than once, then w itself is a $(u - v)$ path. Otherwise, there are vertices of G that occur in w twice or more. Let i and j be distinct positive integers such that $i < j$ with $u_i = u_j$. Then, say, $u_i, u_{i+1},...,u_{j-2}, u_{j-1}$ are removed from w, and the resulting sequence is $(u - v)$ walk w_1 whose length is less than that of w. By induction hypothesis, this w_1 contains a $(u - v)$ path and hence, w has a $(u - v)$ path. If no vertex of G appears more than once in w_1, then w_1 is a $(u - v)$ path. If not, apply the procedure, until we get a $(u - v)$ path.

Cycle: A cycle is a walk. $v_0, v_1,..., v_n$ is a walk in which $n \geq 3$, $v_0 = v_n$ and the 'n'-vertices $v_1, v_2,..., v_n$ are distinct. We say that a $(u - v)$ walk is closed if $u = v$ and open if $u \neq v$.

Connection: Let u and v be vertices in a graph G. We say that u is connected to v if G contains a $(u - v)$ path. The graph G is connected, if u is connected to v for every pair u, v of vertices of G.

Disconnection: A graph G is disconnected, if there exists two vertices u and v for which there is no $(u - v)$ path.

Component: A subgraph H of a graph G is called a component of G if H is a maximal connected subgraph of G and component is denoted by $\omega(G)$.

Note: If $\omega(G) > 1$, then G is disconnected.

NOTES

For example,

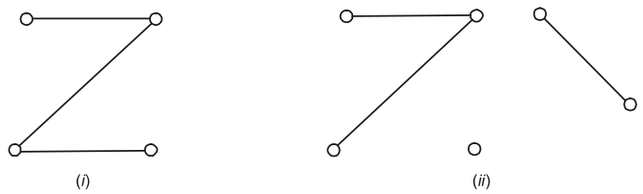


Fig. 14.7 *Connected and Disconnected Graphs*

Graph (*i*) is connected and (*ii*) is disconnected (Refer Figure 14.7).

Note that graph (*ii*) has 3 components.

Connectedness in Directed Graph

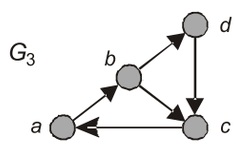
Strongly Connected: A directed graph is strongly connected if there is a path from *u* to *v* and *v* to *u*, whenever *u* and *v* are vertices in the graph.

Weakly Connected: A directed graph is weakly connected if there is a path between any two vertices in the underlying undirected graph.

Unilaterally Connected: A directed graph is said to be unilaterally connected if in the two vertices *u* and *v*, there exists a directed path either from *u* to *v* or from *v* to *u*.

For example, Figure 14.8 represents the various types of connected graphs.

(I) Weakly Connected Directed Graph *(Ii) Unilaterally Connected Directed Graph*



(Iii) Strongly Connected Directed Graph

Fig. 14.8 *Connected Graphs*

G_1 is weakly connected; G_2 unilaterally connected and G_3 is strongly connected.

Cut-Sets and Connectivity of Graphs

Let *G* be a connected graph. Let us recollect the definition of cut-edge (bridge) and cut-vertex. If *G* contains an edge *e* such that *G–e* is disconnected, then *e* is a bridge of *G*. Further, if *G* contains a vertex *v* such that *G–v* is disconnected, then *v* is a cut-vertex of *G*.

NOTES

Definition

Edge Cut-Set: A subset S of the edge set of a connected graph G is called an edge cut-set or cut-set of G if,

(*i*) $G - S$ is disconnected.

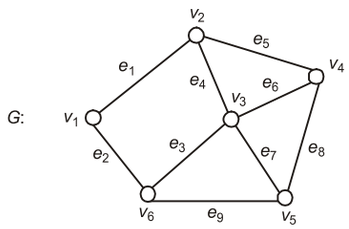
(*ii*) $G - S_1$ is connected for every proper subset S_1 of S.

Vertex Cut-Set: A subset u of the vertex set of G is called a vertex cut-set if,

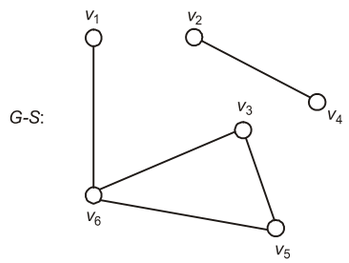
(*i*) $G - u$ is disconnected.

(*ii*) $G - u_1$ is connected for every proper subset u_1 of u.

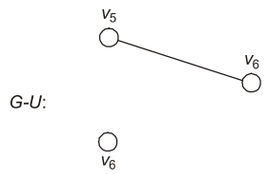
For example,



(*i*) $S = \{e_1, e_4, e_6, e_8\}$ is a cut-set.



(*ii*) $u = \{v_1, v_3, v_5\}$ is a vertex-cut-set.



Note: For a connected graph, there may be more than one cut-set.

For example, consider the above graph G. Some cut-sets of G are,

$$S_1 = \{e_1, e_4, e_6, e_8\}$$

$$S_2 = \{e_1, e_2\}, S_3 = \{e_1, e_3, e_9\}$$

From the above note, we are forced to introduce two more parameters for graphs viz. edge-connectivity $\lambda(G)$ and vertex connectivity $k(G)$.

Edge Connectivity: The edge connectivity $\lambda(G)$ of a graph is the minimum cardinality of a set S of edges of G such that $G - S$ is disconnected, i.e., the edge (line) connectivity of a connected graph is the number of edges in a minimum cut-set in the graph (Refer Figure 14.9).

For example,

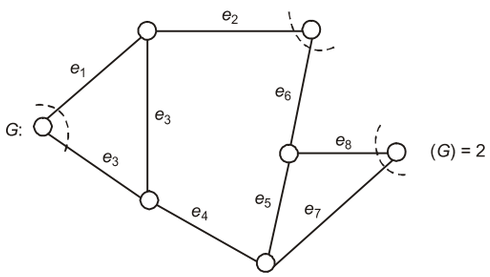


Fig. 14.9 *Edge Connectivity*

Notes:

1. If G is a tree, then $\lambda(G) = 1$.
2. G has $\lambda(G) = 0$ iff G is disconnected or trivial.

Vertex Connectivity: The *vertex connectivity* $K(G)$ of a graph G is the minimum number of vertices whose deletion makes G a disconnected or trivial graph, i.e., the number of vertices in a minimum *vertex* cut is called the connectivity of the graph (Refer Figure 14.10).

For example,

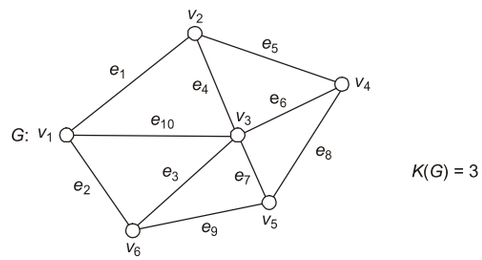


Fig. 14.10 *Vertex Connectivity*

Theorem 14.6: *For every graph G, $K(G) \leq \lambda(G) \leq \delta(G)$.*

Proof: Let v be a vertex of G with minimum degree, i.e., $d(v) = \delta(G)$. Removing $\delta(G)$ edges of G incident with v produces a graph G_1, in which v is isolated. Clearly, G is disconnected or trivial.

$\therefore \lambda(G) \leq \delta(G)$ ...(14.3)

***Corollary*:** $K(G) \leq \delta(G)$

If, $\delta(G) = 0$, then G is disconnected.

$\therefore$ $K(G) = 0$.

NOTES

If $\delta(G) = 1$, then G is a connected graph containing a cut-edge (bridge). Therefore, either G is isomorphic to K_2 or G is a connected graph having atleast one cut-vertex.

$\therefore$ In both cases, $K(G) = 1$.

Now, let us assume that $\lambda(G) \geq 2$. Let S be a cutset of G with $\lambda(G)$ edges and $e = xy$ be an edge in S. If the edges of $S - \{e\}$ are deleted from G, the resulting subgraph H_1 is connected and contains e as a cut-edge. Now select an incident vertex different from x and y for each and every edge in $S - \{e\}$. Remove these vertices from H_1, the resulting subgraph H_2 is disconnected, then

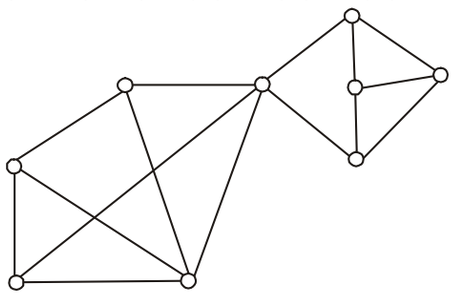
$$K(G) \leq \lambda(G) - 1 < \lambda(G)$$

Suppose the subgraph H_2 is connected, then H_2 is isomorphic to K_2 or the subgraph H_2 has a cut-vertex, since H_2 is an induced subgraph of H_1. In any case, there exists a vertex of H_2 whose removal results in a disconnected graph. Therefore,

$$K(G) \leq \lambda(G) \qquad ...(14.4)$$

From Equations (14.3) and (14.4), $K(G) \leq \lambda(G) \leq \delta(G)$

G:



In this figure, $K(G) = 1$; $\lambda(G) = 3$ and $\delta(G) = 3$.

***n*-edge Connected:** A graph G is n-edge connected ($n \geq 1$) if $\lambda(G) \geq n$ and G is n-connected if $K(G) \geq n$.

14.7 EULER GRAPH

A trail that traverses every edge of G is called an Euler trail of G. A circuit (tour) of G is a closed walk that traverses each edge of G atleast once. An Euler tour is a tour which traverses each edge exactly once. A graph is Eulerian if it contains an Euler tour.

Theorem 14.7: A connected graph is Eulerian iff it has no vertices of odd degree.

Proof: Let G be Eulerian and let C be an Euler tour of G, which begins and ends at some vertex u.

Claim: G has no vertices of odd degree, i.e., to prove that every vertex of G is

NOTES

even. Consider a vertex $w \neq u$. Since w is neither the first nor the last vertex of C, each time w is encountered, it is reached by some edge and left by another edge. Hence each occurrence of w in C contributes 2 to its degree. Thus w is of even degree. This is true for all internal vertices of C. The initial occurrence and final occurrence of the vertex u in C contributes 1 to the degree of u. Therefore, every vertex of G is of even degree.

Conversely, let us assume that every vertex of a connected graph G is even.

Claim: G is Eulerian.

Suppose G be a connected non-Eulerian graph with no vertices of odd degree.

Among such graphs, choose one, say G having least number of edges.

Since, each vertex of G has atleast two edges, G contains a trail. Let C be a closed trail of maximum possible length in G. By assumption, C is not a Euler circuit of G and hence $G - E(C)$ has edges.

Therefore, $G - E(C)$ has some component G' with edges. Since C itself is Eulerian, degree of every vertex in C is even. Hence degree of every vertex in $G - E(C)$ is also even. Therefore degree of every vertex in G' is even. Since $E(G') < E(G)$.

G' is Eulerian and hence G' has an Euler circuit (tour) say C'. Since G is connected, there is a vertex v in $V(C) \cap V(C')$ and we may assume without loss of generality that v is the initial and the terminal vertex of both circuits C and C'. Now $(C \cup C')$ is a closed trail of G with $E(C \cup C') > E(C)$. This contradicts the choice of C. Hence, every non-empty connected graph with no vertices of odd degree is Eulerian.

For example,

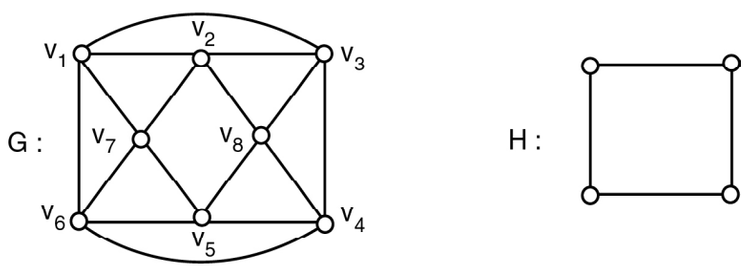


Fig. 14.11 *Eulerian Graphs*

In Figure 14.11 G and H are Eulerian graphs.

Theorem 14.8: A connected graph G has an Eulerian trail iff G has exactly two odd vertices.

Proof: Let G be a connected graph with an Eulerian $(u - v)$ trail. By the similar argument in the previous theorem, we conclude that all the vertices on the trail except u and v, have even degree. Conversely, let G be connected graph with two odd vertices u and v. Let G' be the graph obtain from G by adding a new edge e

= uv between u and v. By applying the previous theorem to G', we can obtained an Eulerian tour in which the edge e is the first edge. Hence, this Eulerian trail of G can be obtained that starts at v and ends at u. Therefore, G has an Eulerian trail.

NOTES

14.7.1 Eulerian Digraphs

An Eulerian trail of a connected directed graph D, is a trail of D that contains all the edges of D; while an Eulerian circuit of D, is a circuit which contains every edges of D. A directed graph that contains an Eulerian circuit is called Eulerian digraph (Refer Figure 14.12).

For example,

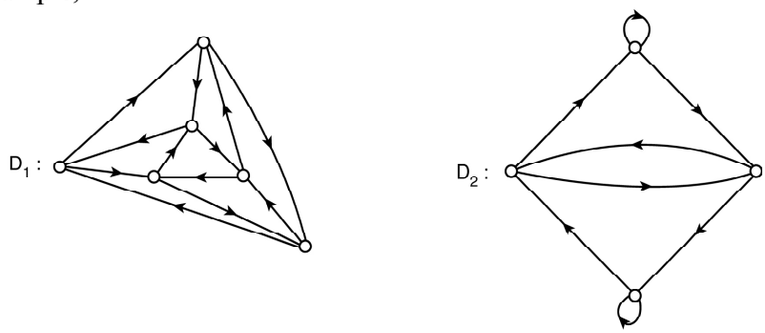


Fig. 14.12 *Eulerian Digraphs*

Theorem 14.9: Let D be a connected directed graph. D is Eulerian iff $d^+(v) = d^-(v)$, $\forall v \in G$, G is called balanced digraph.

Proof: Let D be an Euler directed graph. Then D contains an Euler circuit C with common initial and terminal vertex v. Let b_u be the number of occurrence of an internal vertex u in C.

Whenever C enters u through some edge incident into u, there is another edge incident out of u through which C leaves u. Thus, each occurrence of u contributes one in-degree and one out-degree. Moreover, C contains all the edges of D. Thus,

$d^+(u) = d^-(u) = bu$. Similarly $d^+(v) = d^-(v)$

Hence, $d^+(v) = d^-(v)$, $\forall v \in V(D)$

Conversely, suppose the connected digraph D is balanced. Then, for each vertex u, $d^+(u) = d^-(u) \neq 0$. Start with an arbitrary vertex u_1, $d^+(u_1) \neq 0$. There exists an edge, incident out of u_1. Let u_2 be the terminal vertex of this edge, $d^+(u_2) \neq 0$. Hence, there exists an edge, incident out of u_2. Continuing like this, we reach a vertex which has been traversed directly. Thus, we obtain a directed circuit C_1 in D. If $E(C_1) = E(D)$, then, C_1 is the required Euler circuit. If not, i.e., $E(C_1) \neq E(D)$, then remove all the edge of C_1 from D to obtain a spanning

subgraph D_1. Since D is balanced, D_1 is also balanced. Applying the above process to D_1, we will obtain a circuit C_2 in D_1. If $E(D) = E(C_1) \cup E(C_2)$ and C_1 then C_2 can be combined to obtain an Euler circuit in D_1. Otherwise, we remove the edges of C_2 from D_1 to obtain a spanning subgraph D_2 of D. We repeat the above process in D_2 and after a finite number of steps, we obtain edge disjoint circuits $C_1, C_2, \ldots, C_k$ such that $E(D) = E(C_1) \cup E(C_2) \cup \ldots \cup E(C_k)$. Since D is connected, any two of these cycles have a common vertex. Then the circuits $C_1, C_2, \ldots, C_k$ can be combined to obtain an Euler circuit in D. Hence, D is an Eulergraph.

14.8 HAMILTONIAN CIRCUITS AND PATHS

In 1857, Sir William Rowan Hamilton invented a game called, Around The World. In this game, a solid regular dodecahedron (20 vertices, 30 edges and 12 faces) and a supply of string is given. Every vertex is given the name of an important city. The objective of the game is to find a route along the edges of dodecahedron that visits every city exactly once and terminates where it started.

The graph D is a dodecahedron.

Another famous problem is 'The Knight's Puzzle', Is it possible for a knight to tour the chess board, i.e., visit each square exactly once and return to its initial square?

It can be represented by a graph G, where the vertices u_i corresponds to squares S_i of the chess board and u_j is adjacent to u_i, iff it is possible for a knight to proceed from S_i to S_j in a single move.

To solve 'Around The World' and 'Knight's Puzzle', we must determine if the given graph is Hamiltonian.

A path that contains every vertex of G is called a Hamilton path of G. Similarly, a Hamilton cycle of G is a cycle that contains every vertex of G (in other words spanning cycle). A graph is Hamiltonian, if it contains a Hamilton cycle or spanning cycle.

Example 14.2: Prove that k_n has a Hamiltonian circuit, $\forall n \geq 3$.

Solution: Let us construct the Hamiltonian circuit in $k_n\ (n \geq 3)$ as follows: Choose a vertex arbitrarily in k_n and beginning the Hamiltonian circuit at this vertex. Such a circuit can be built by traversing vertices in any order, as long as the path begins and terminates at the same vertex and visits each other vertex exactly once. This is possible in k_n, since every vertex is adjacent to all other vertices. Thus, k_1, k_2 has only hamiltonian path, not circuit.

Graphical: A sequence $d = (d_1, d_2, \ldots, d_n)$ is graphical if there exists a simple undirected graph on n vertices with the degrees of the vertices $d_1, d_2, \ldots, d_n$, respectively.

For example, A graph with degree sequence (4, 4, 3, 2, 2, 1)

NOTES

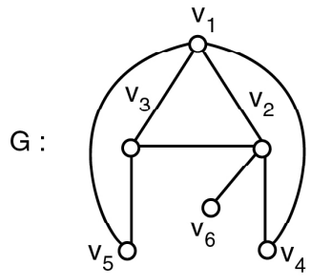


Fig. 14.13 *Graph with Degree Sequence Construction*

Closure: A closure (CG) of a n-vertex graph G is a graph from G by recursively joining pairs of non-adjacent vertices whose degree sum is atleast n until no such pair remains.

For example,

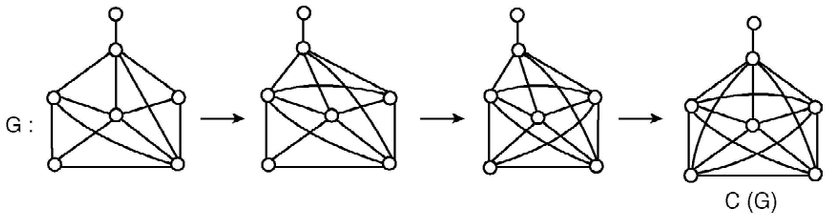


Fig. 14.14 *Constructing a closure of a graph*

Important Theorems

Theorem 14.10: Let G be a n-vertex graph. Suppose G_1 and G_2 are two graphs obtained from G by recursively joining pairs of non-adjacent vertices whose degree sum is atleast n. Then, $G_1 = G_2$. In other words, $C(G)$, the closure of a graph G is unique.

Proof: Let $e_1, e_2,\ldots,e_k$ and $f_1,f_2,f_3,\ldots,f_e$ be the edges added to G to obtain G_1 and G_2, respectively. We have to prove that every e_i $(1 \le i \le k)$ is an edge of G_2 and f_j $(1 \le j \le l)$ is an edge of G_1. Suppose that some edge in the sequence $e_1, e_2,...,e_k$ does not belong to G_2. Let p be the smallest positive integer, such that e_{p+1} is not an edge of G_2. Let $e_{p+1} = uv$. Let $H = G + \{e_1, e_2,...,e_p\}$. Then H is a sub-graph of G_1 and G_2. By the construction of G_1,

$$d_H(u) + d_H(v) \ge n$$

Therefore, $d_{G_2}(u) + d_{G_2}(v) \ge d_H(u) + d_H(v) \ge n$

This is a contradiction, since u and v are non-adjacent in G_2. Therefore, each e_i is an edge of G_2. Similarly, each f_j belongs to G_1. Hence, $G_1 = G_2$.

Theorem 14.11: A graph G is Hamiltonian, iff its closure $C(G)$ is Hamiltonian.

Proof: Let $e_1, e_2,\ldots,e_n$ be edges, added to G, to obtain its closure $C(G)$. Let G_i be the graph obtained from G by adding the edge e_i.

By repeated application of Example 14.2.

G is Hamiltonian $\Leftrightarrow C(G)$ is Hamiltonian.

Corollary 1: Let G be a graph with atleast 3 vertices. If $C(G) \cong k_n$, $(n \geq 3)$ then G is Hamiltonian.

Proof: By result 1, k_n is Hamiltonian. Since $C(G) \cong k_n, C(G)$ is Hamiltonian and hence G is Hamiltonian.

Corollary 2: Let G be a graph with atleast 3 vertices if $d(u) + d(v) \geq n$ $(n \geq 3)$, for all pairs u and v of non-adjacent vertices of G, then G is Hamiltonian.

Proof: Let G be a graph with atleast 3 vertices. Given that $d(u) + d(v) \geq n$ $(n \geq 3)$ for all pairs of non-adjacent vertices of G. Hence we can add edges between such pair of vertices to obtain $C(G)$. Since $C(G)$ is complete by Corallary 1, G is Hamiltonian.

For example,

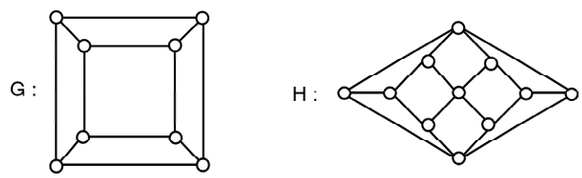


Fig. 14.15 *Hamiltonian and Non-Hamiltonian Graphs*

In Figure 14.15, G Hamiltonian graph and H non-Hamiltonian graph.

Theorem 14.12: If G is Hamiltonian then, for every non-empty proper subset S of V, $w(G - s) \leq |s|$

Proof: Let G be a Hamiltonian graph and S be a proper subset of V. Since G is Hamiltonian, G has a Hamiltonian cycle C. Suppose $\omega(G - S) = n$, where G_1, G_2,...,G_n are the components of G – S. Let u_i $(1 \leq i \leq v)$ be the last vertex of C that belongs to G_i and let v_i be the vertex that immediately follows u_i on C. Clearly $v_i \in S$ for each i and $v_j \neq v_k$ for $j \neq k$. Hence, there are atleast as many vertices in S as components in $G - S$.

i.e., $\omega(G - S) \leq |S|$.

Weight Graph: A graph G is called a weight graph if every edge of G is assigned with a real number.

Travelling Salesmen Problem (TSP)

Suppose that a salesman is expected to take a trip through a given collection of n cities $(n \geq 3)$. What route should he take to minimize the total distance travelled? This can be represented as a weight graph. Let G be a connected weight graph whose vertices represent the cities to be visited and let the weight of an edge $v_i v_j$ be the distance between the cities v_i and v_j. Now TSP is equivalent to finding a minimum Hamiltonian cycle in a connected weight graph.

NOTES

Check Your Progress

8. Interpret the fundamental circuits.
9. Define the cut-sets and connectivity of graph.
10. Illustrate the Euler graph.
11. Describe the Eulerian diagraphs.
12. Explain the Hamiltonian circuits and paths.

14.9 ANSWERS TO CHECK YOUR PROGRESS QUESTIONS

1. In graph theory, a cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut, and in some cases cuts are identified with their cut-sets rather than with their vertex partitions.
2. Cut-Vertex: A vertex v in a graph G is said to be a cut-vertex if $w(G-v) > w(G)$, where $w(G)$ is the component of G and component is a maximal connected subgraph of G, i.e., a vertex v of a connected graph is a cut-vertex, iff $(G-v)$ is disconnected.
3. Cut-Edge: An edge e in a graph G is said to be a Cut-edge, if $(G-e)$ is disconnected.
4. Edge Cut-Set: A subset S of the edge set of a connected graph G is called an edge cut-set or cut-set of G if

   (i) $G-S$ is disconnected.

   (ii) $G-S_1$ is connected for every proper subset S_1 of S.

5. Vertex Cut-Set: A subset u of the vertex set of G is called a vertex cut-set if

   (i) $G-u$ is disconnected

   (ii) $G-u_1$ is connected for every proper subset u_1 of u.

6. Edge Connectivity: The edge connectivity $\lambda(G)$ of a graph is the minimum cardinality of a set S of edges of G such that $G–S$ is disconnected, i.e., the edge (line) connectivity of a connected graph is the number of edges in a minimum cut-set in the graph.
7. Vertex Connectivity: The vertex connectivity K(G) of a graph G is the minimum number of vertices whose deletion makes G a disconnected or

trivial graph, i.e., the number of vertices in a minimum vertex cut is called the connectivity of the graph.

8. Let T be a spanning tree in a connected graph G. When a chord is added to a spanning tree T then it forms exactly one circuit. Such a circuit is termed as a fundamental circuit.
9. Let G be a connected graph. Let us recollect the definition of cut-edge (bridge) and cut-vertex. If G contains an edge e such that $G–e$ is disconnected, then e is a bridge of G. Further, if G contains a vertex v such that $G–v$ is disconnected, then v is a cut-vertex of G.
10. A trail that traverses every edge of G is called an Euler trail of G. A circuit (tour) of G is a closed walk that traverses each edge of G atleast once. An Euler tour is a tour which traverses each edge exactly once. A graph is Eulerian if it contains an Euler tour.
11. An Eulerian trail of a connected directed graph D, is a trail of D that contains all the edges of D; while an Eulerian circuit of D, is a circuit which contains every edges of D. A directed graph that contains an Eulerian circuit is called Eulerian digraph.
12. A path that contains every vertex of G is called a Hamilton path of G. Similarly, a Hamilton cycle of G is a cycle that contains every vertex of G (in other words spanning cycle). A graph is Hamiltonian, if it contains a Hamilton cycle or spanning cycle.

14.10 SUMMARY

- In graph theory, a cut is a partition of the vertices of a graph into two disjoint subsets. Any cut determines a cut-set, the set of edges that have one endpoint in each subset of the partition. These edges are said to cross the cut. In a connected graph, each cut-set determines a unique cut, and in some cases cuts are identified with their cut-sets rather than with their vertex partitions.
- Cut-Vertex: A vertex v in a graph G is said to be a cut-vertex if $w(G-v) > w(G)$, where $w(G)$ is the component of G and component is a maximal connected subgraph of G, i.e., a vertex v of a connected graph is a cut-vertex, iff $(G-v)$ is disconnected.
- Cut-Edge: An edge e in a graph G is said to be a Cut-edge, if $(G-e)$ is disconnected.
- Edge Cut-Set: A subset S of the edge set of a connected graph G is called an edge cut-set or cut-set of G if

  (i) $G-S$ is disconnected.

  (ii) $G-S_1$ is connected for every proper subset S_1 of S.

NOTES

- Vertex Cut-Set: A subset u of the vertex set of G is called a vertex cut-set if

  (i) $G - u$ is disconnected

  (ii) $G - u_1$ is connected for every proper subset u_1 of u.
- Edge Connectivity: The edge connectivity $\lambda(G)$ of a graph is the minimum cardinality of a set S of edges of G such that $G–S$ is disconnected, i.e., the edge (line) connectivity of a connected graph is the number of edges in a minimum cut-set in the graph.
- Vertex Connectivity: The vertex connectivity K(G) of a graph G is the minimum number of vertices whose deletion makes G a disconnected or trivial graph, i.e., the number of vertices in a minimum vertex cut is called the connectivity of the graph.
- Let T be a spanning tree in a connected graph G. When a chord is added to a spanning tree T then it forms exactly one circuit. Such a circuit is termed as a fundamental circuit.
- Let G be a connected graph. Let us recollect the definition of cut-edge (bridge) and cut-vertex. If G contains an edge e such that $G–e$ is disconnected, then e is a bridge of G. Further, if G contains a vertex v such that $G–v$ is disconnected, then v is a cut-vertex of G.
- A trail that traverses every edge of G is called an Euler trail of G. A circuit (tour) of G is a closed walk that traverses each edge of G atleast once. An Euler tour is a tour which traverses each edge exactly once. A graph is Eulerian if it contains an Euler tour.
- An Eulerian trail of a connected directed graph D, is a trail of D that contains all the edges of D; while an Eulerian circuit of D, is a circuit which contains every edges of D. A directed graph that contains an Eulerian circuit is called Eulerian digraph.
- A path that contains every vertex of G is called a Hamilton path of G. Similarly, a Hamilton cycle of G is a cycle that contains every vertex of G (in other words spanning cycle). A graph is Hamiltonian, if it contains a Hamilton cycle or spanning cycle.

14.11 KEY WORDS

- **Cut set:** In graph theory, a cut is a partition of the vertices of a graph into two disjoint subset. Any cut determines a cut-set, the set of degree that have one endpoint in each subset of the partition.
- **Cut-vertex:** A vertex v in a graph G is said to be a cut-vertex if $\omega(G–v) > \omega(G)$, where $\omega(G)$ is the component of G and component is a maximal connected subgraph of G, i.e., a vertex v of a connected graph is a cut-vertex, iff $(G–v)$ is disconnected.

NOTES

- **Cut-edge:** An edge e in a graph G is said to be a cut-edge, if $(G–e)$ is disconnected.
- **Vertex connectivity:** The vertex connectivity $K(G)$ of a graph G is the minimum number of vertices whose deletion makes G a disconnected or trivial graph, i.e., the number of vertices in a minimum vertex cut is called the connectivity of the graph.

14.12 SELF ASSESSMENT QUESTIONS AND EXERCISES

Short-Answer Questions

1. Define the cut set.
2. What do you understand by the cut vertex?
3. Explain the cut edge.
4. E laborate on the edge cut-set.
5. Analyse the vertex cut-set.
6. Illustrate the fundamental circuit.
7. Interpret the cut-sets and connectivity of graph.
8. Explain the Euler graph.
9. Define the Eulerian diagraphs.
10. State the Hamiltonian circuits and paths.

Long-Answer Questions

1. Discuss briefly cut set. Give appropriate example.
2. Explain the properties of a cut sets.
3. Elaborate on the all cut set in a graph.
4. Describe briefly the fundamental circuits and cut set.
5. Define connectivity and separability.
6. Illustrate the Euler graph.
7. Analyse the Hamiltonian circuits and paths.

14.13 FURTHER READINGS

Venkataraman, Dr. M. K., Dr. N. Sridharan and N. Chandrasekaran. 2004. *Discrete Mathematics*. Chennai: The National Publishing Company.

Tremblay, Jean Paul and R Manohar. 2004. *Discrete Mathematical Structures With Applications To Computer Science*. New York: McGraw-Hill Interamericana.

NOTES

Arumugam, S. and S. Ramachandran. 2001. *Invitation to Graph Theory*. Chennai: Scitech Publications (India) Pvt. Ltd.

Balakrishnan, V. K. 2000. *Introductory Discrete Mathematics*. New York: Dover Publications Inc.

Choudum, S. A. 1987. *A First Course in Graph Theory*. New Delhi: MacMillan India Ltd.

Haggard, Gary, John Schlipf and Sue Whiteside. 2006. *Discrete Mathematics for Computer Science*. California: Thomson Learning (Thomson Brooks/ Cole).

Kolman, Bernand, Roberty C. Busby and Sharn Cutter Ross. 2006. *Discrete Mathematical Structures*. London (UK): Pearson Education.

Johnsonbaugh, Richard. 2000. *Discrete Mathematics*, 5th Edition. London (UK): Pearson Education.

Iyengar, N. Ch. S. N., V. M. Chandrasekaran, K. A. Venkatesh and P. S. Arunachalam. 2007. *Discrete Mathematics*. New Delhi: Vikas Publishing House Pvt. Ltd.

Mott, J. L. 2007. *Discrete Mathematics for Computer Scientists*, 2nd Edition. New Delhi: Prentice Hall of India Pvt. Ltd.

Liu, C. L. 1985. *Elements of Discrete Mathematics*, 2nd Edition. New York: McGraw-Hill Higher Education.

Rosen, Kenneth. 2007. *Discrete Mathematics and Its Applications*, 6th Edition. New York: McGraw-Hill Higher Education.