ADVANCED DISCRETE MATHEMATICS

M.A./M.Sc. Mathematics (Final)

MM-504 & 505
(Option-P_3)

Directorate of Distance Education
Maharshi Dayanand University
ROHTAK – 124 001

Maharshi Dayanand University
ROHTAK – 124 001

Developed & Produced by EXCEL BOOKS PVT. LTD., A-45 Naraina, Phase 1, New Delhi-110 028

Contents

M.A./M.Sc. Mathematics (Final)

ADVANCED DISCRETE MATHEMATICS

MM- 504 & 505 (P_3)

Max. Marks : 100

Time : 3 Hours

Note: Question paper will consist of three sections. Section I consisting of one question with ten parts covering whole of the syllabus of 2 marks each shall be compulsory. From Section II, 10 questions to be set selecting two questions from each unit. The candidate will be required to attempt any seven questions each of five marks. Section III, five questions to be set, one from each unit. The candidate will be required to attempt any three questions each of fifteen marks.

Unit I

Formal Logic: Statement, Symbolic representation, totologies, quantifiers, pradicates and validity, propositional logic.

Semigroups and Monoids: Definitions and examples of semigroups and monoids (including those pertaining to concentration operations). Homomorphism of semigroups and monoids, Congurence relation and quotient semigroups, sub semigropups and sub monoids, Direct products basic homomorphism theorem.

Lattices: Lattices as partially ordered sets, their properties. Lattices and algebraic systems. Sub lattices, direct products and homomorphism. Some special lattices for example complimented and distributive lattices.

Unit II

Boolean Algebra: Boolean Algebra as Lattices. Various Boolean Identities Join-irreducible elements. Atoms and Minterms. Boolean Forms and their Equivalence. Minterm Boolean Forms, Sum of Products Canonical Forms. Minimization of Boolean Functions. Applications of Boolean Algebra to Switching Theory (using AND, OR and NOT gates). The Karnaugh Map method.

Unit III

Graph Theory – Definition of (undirected) Graphs, Paths, Circuits, Cycles and Subgroups. Induced Subgraphs. Degree of a vertex. Connnectivity. Planar Graphs and their properties. Trees, Duler's Formula for connected Planar Graphs, Complete and Complete Bipartite Graphs. Kurtowski's Theorem (statement only) and its use. Spanning Trees. Cut-sets. Fundamental Cut-sets and Cycles. Minimal Spanning Trees and Kruskal's Algorithum. Matrix Representations of Graphs. Euler's Theorem on the Existence of Eulerian Paths and Circuits. Directed Graphs. Indegree and Outdegree of a Vertex. Weighted undirected Graphs. Dijkstra's Algorithm. Strong Connectivity & Warshall's Algorithm. Directed Trees. Search Trees. Tree Traversals.

Unit IV

Introductory Computability Theory – Finite state machines and their transition table diagrams. Equivalence of finite state machines. Reduced Machines, Homomorphism. Finite automata. Acceptors. Non-deterministic finite automata and Equivalence of its power to that of Deterministic Finite Automata. Moore and Mealy Machines.

Unit V

Grammar and Languages — Phrase Structure Grammars. Rewriting Rules. Derivations Sentential Forms. Language generated by Grammar. Regular, Context Free, and Context Sensitive Grammar and Languages. Regular sets, Regular Expressions and the Pumping Lemma, Kleene's Theorem.

Notions of Syntax Analysis. Polish Notations. Conversion of Infix Expressions to Polish Notations. The Reverse Polish Notation.

Unit-1

Logic, Semigroups & Monoids and Lattices

PART - A : LOGIC

1.1. Logic is a science of the necessary laws of thought, without which no employment of the understanding and the reason takes place.

Consider the following argument:

All mathematicians wear sandals

Anyone who wears sandals is an algebraist

Therefore, all mathematicians are algebraist.

Technically, logic is of no use in determining whether any of these statements is true. However, if the first two statements are true, logic assures us that the statement.

All mathematicians are algebraists is also true.

Example:- which of sentences are true or false (but not both)?

(a) The only positive integers that divide 7 are 1 and 7 itself.
(b) For every positive integer n, there is a prime number larger than n.
(c) Earth is the only planet in the universe that has life.

Solution:- (a) We call an integer n prime if n>1 and the only positive integers that divide n are 1 and n itself. Sentence (a) is another way say that 7 is a prime. Hence sentence (a) is true.
(b) Sentence (b) is another way to say that there are an infinite number of prime. Hence (b) is true.
(d) Sentence (c) is either true or false (but not both) but no one knows which at this time.

Definition:- A declarative sentence that is **either true** or false, but not both is called a **Proposition** (or **statement**).

For example, sentences (a) to (c) in the above example are propositions.

But the sentence
$x + y > 0$

is **not** a statement because for some values of x and y the sentence is true whereas for other values of x and y it is false. For example, if x = 1, y = 3, the sentence is true, but for x = -2, y = 0, it is false.
Similarly, the sentence
Take two crocins is not a statement. It is a command.

The propositions are represented by **lower case letters** such as p, q and r. We use the notation p: 1+1=3 to define p to be the proposition 1+1=3.

Many propositions are composite, that is, composed of subpropositions and various connectives. The "Composite propositions are called **compound propositions."**A proposition which is not compound is said to be **primitive.** Thus, a primitive proposition cannot be broken into simpler propositions.

Example:- The sun is shining and it is cold. This is a compound proposition composed of two propositions
The sun is shining
and
It is cold.
Connected by the connective "and".
On the other hand, the proposition
London is in Denmark
is **primitive** statement.

Definition:- The truth values of a compound statement in terms of its component parts, is called a **truth table.**

1.2. Basic Logical Operations

The three basic logical operations are

1. Conjunction
2. Disjunction
3. Negation

which correspond, respectively, to "and", "or" and "not".

Definition:- The conjunction of two propositions p and q is the proposition

p and q.

It is denoted by $p \wedge q$.

Example:- Let

p : This child is a boy
q : This child is intelligent

Then

$p \wedge q$: This child is a boy and intelligent.

Thus $p \wedge q$ is true, if the child is a boy and intelligent both.
Even if one of the component is false, $p \wedge q$ is false. Thus

"the proposition $p \wedge q$ is true if and only if the proposition p and q are both true".

The truth value of the compound proposition $p \wedge q$ is defined by the truth table:

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

Example:- If

p : 1 +1 = 3

q : A decade is 10 years,

then p is false, q is true and the conjunction

$p \wedge q$: 1 +1 = 3 and a decade is 10 years

is **false**.

Definition:- The **disjunction** of two proposition p and q is the proposition

p or q

It is denoted by $p \vee q$.

The compound statement $p \vee q$ is true if at least one of p or q is true. **It is false when both p and q are false.**

The truth values of the compound proposition $p \vee q$ is defined by the truth table:

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

For example, if

p : 1 + 1 = 3

q : A decade is 10 years,

then p is false, q is true. The disjunction

$p \vee q$: 1 + 1 = 3 or a decade is 10 years

is **true**.

Definition:- If p is a statement, the **negation** of p is the statement not p, denoted by ~p.
Thus ~p is the statement "it is not the case that p".
Hence if p is true than ~p is false and if p is false, then ~p is true.

The truth table for negation is

p	~p
T	F
F	T

Example:- Give the negation of the following statements :

(a) p : 2 + 3 > 1 (b) q : It is cold

Solution:-
(a) ~p : 2 + 3 is not greater than 1. That is, ~p : $2 + 3 \leq 1$.
Since p is true in this case, ~p is false.

(b) ~q : It is not the case that it is cold. More simply, ~q : It is not cold.

Translating from English to Symbols :- We consider

Example:- Write each of the following sentences symbolically, letting p : "It is hot" and q : " It is sunny":

(a) It is not hot but it is sunny

(b) It is neither hot nor sunny.

Solution:- (a) The convention in logic is that the words "but" and "and" **mean the same thing.** Generally, but is used in place of and when the part of the sentence that follows is in some way unexpected.
The given sentence is equivalent to " It is not hot and it is sunny" which can be written symbolically as $\sim p \wedge q$.

(c) The phrase neither A nor B means the same as not A and not B. Thus to say " IT is neither hot nor sunny" means that it is not hot and it is not sunny. Therefore the given sentence can be written symbolically as $\sim p \wedge \sim q$.

Definition:- A "Statement form" or "Propositional form" is an expression made up of **statement variables** (such as ~, $\wedge$, $\vee$) that becomes a statement when actual statements are substituted for the component statement variable. The **truth table** for a given statement form displays the truth values that correspond to the different combinations of truth values for the variables.
Example:- Construct a truth table for the statement form:

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

solution:-The truth table for the given statement form is

p	q	r	P∧q	~r	(p∧q) ∨ ~r
T	T	T	T	F	T
T	T	F	T	T	T
T	F	T	F	F	F
T	F	F	F	T	T
F	T	T	F	F	F
F	T	F	F	T	T
F	F	T	F	F	F
F	F	F	F	T	T

Definition:- Two different compound propositions(or statement forms) are said to **logically equivalent** if they have the same truth value no matter what truth values their constituent propositions have.
We use the symbol ≡ for logical equivalent.

Example:- Consider the statements forms

(a) Dogs bark and cats meow
(b) Cats meow and dogs bark

If we take

p : Dogs bark
q : Cats meow,

then (a) and (b) are in logical expression

(a) $p \wedge q$
(b) $q \wedge p$

If we construct the truth tables for $p \wedge q$ and $q \wedge p$, we observe that $p \wedge q$ and $q \wedge p$ have same truth values.

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

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

Thus $p \wedge q$ and $q \wedge p$ are logically equivalent. That is

$$p \wedge q \equiv q \wedge p$$

Example:- Negation of the negation of a statement is equal to the statement. Thus

$\sim(\sim p) \equiv p$.

Solution:- The truth table of $\sim(\sim p)$ is

p	~p	~(~p)
T	F	T
F	T	F

Thus truth values for p and ~(~p) are same and hence p and ~(~p) are logically equivalent. The logical equivalence $\sim(\sim p) \equiv p$ is called **Involution Law.**

Example:- Show that the statement forms $\sim(p \wedge q)$ and $\sim p \wedge \sim q$ are not logically equivalent.

Solution:- Construct the truth table for both statement forms:

p	q	~p	~q	$p \wedge q$	$\sim(p \wedge q)$		$\sim p \wedge \sim q$
T	T	F	F	T	F		F
T	F	F	T	F	T	$\neq$	F
F	T	T	F	F	T	$\neq$	F
F	F	T	T	F	T		T

Thus we have different truth values in rows 2 and 3 and so $\sim(p \wedge q)$ and $\sim p \wedge \sim q$ are not topologically equivalent.

Remark:- If we consider $\sim p \vee \sim q$, then its truth values shall be

F
T
T
T

and hence $\sim(p \wedge q)$ and $\sim p \wedge \sim q$ are logically equivalent. Symbolically

$$\sim(p \wedge q) \equiv \sim p \vee \sim q \qquad (1)$$

Analogously,

$$\sim(p \vee q) \equiv \sim p \wedge \sim q \qquad (2)$$

The above two logical equivalence are known as **De Morgan's Laws of Logic.**

Example:- Use De Morgan's Laws to write the negation of

p : Jim is tall and Jim is thin.

Solution:-The negation of p is

~p : Jim is not tall or Jim is not thin.

Definition:- A compound proposition which is **always true** regardless of truth values assigned to its component propositions is called a **Tautology.**

Definition:- A compound proposition which is **always false** regardless of truth values assigned to its component propositions is called a **Contradiction**.

Definition:- A compound proposition which can be either true or false depending on the truth values of its component propositions is called a **Contingency.**

Example:- Consider the statement form

$$p \vee \sim p.$$

The truth table for this statement form is

P	~p	$p \vee \sim p$
T	F	T
F	T	T

↑
all T's

Hence $p \vee \sim p$ is a tautology.

Exercise :- Show that $p \wedge \sim p$ is a contradiction.

Remark:- If τ and c denote tautology and contradictions respectively, then we notice that

$$\sim\tau \equiv c \qquad (1)$$

and

$$\sim c \equiv \tau \qquad (2)$$

Also from the above two examples

$$p \vee \sim p \equiv \tau \qquad (3)$$

and

$$p \wedge \sim p \equiv c \qquad (4)$$

the logical equivalence (1), (2), (3) and (4) are known as **Complement Laws.**

Logical Equivalence involving Tautologies and Contradictions

If t is a tautology and c is a contradiction, then the truth tables for $p \wedge \tau$ and $p \wedge c$ are :

p	τ	p ∧ τ
T	T	T
F	T	F

Same truth values and so $p \wedge \tau \equiv p$

p	c	p ∧ c
T	F	F
F	F	F

Same truth values and so $p \wedge c \equiv c$

Similarly, the truth tables for $p \vee \tau$ and $p \vee c$ are

p	τ	p ∨ τ	c	p ∨ c
T	T	T	F	F
F	T	T	F	F

Same truth value and so $p \vee c \equiv p$

Same truth values

So

$p \vee \tau \equiv \tau$

Thus we have the following logical equivalence:

$p \wedge \tau \equiv p$ $\quad$ $p \wedge c \equiv c$

$p \vee \tau \equiv \tau$ $\quad$ $p \vee c \equiv p$ (**universal bound laws**)

These four logical equivalence are known as **Identity Law.**

Example:- (**Idempotent Laws**): Consider the truth tables for $p \wedge p$ and $p \vee p$ given below:

p	p	p ∧ p
T	T	T
F	F	F

p	p	p ∨ p
T	T	T
F	F	F

We note that

(i) $p \wedge p$ and p have same truth values

(ii) $p \vee p$ and p have same truth values

Hence

$p \wedge p \equiv p$ $\quad$ and $p \vee p \equiv p$

These two logical equivalence are known as **Idempotent Laws.**

Exercise :- Show that $p \wedge q \equiv q \wedge p$ and $p \vee q \equiv q \vee p$ (these logical equivalences are known as **Commutative Laws).**

Exercise :- Prove that

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

and

$$p \vee (p \wedge q) \equiv p .$$

(These logical equivalence are known as **Absorption Laws).**

Exercise :- Show that

$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$, $(p \vee q) \vee r \equiv p \vee (q \vee r)$ (**Associative Laws**) and

$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$, $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$ (**Distributive Laws**)

1.3. Conditional Propositions

Definition:- If p and q are propositions, the compound proposition

if p then q or p implies q

is called a **conditional proposition** or **implication** and is denoted by

$p \rightarrow q$.

The proposition p is called the hypothesis or antecedent whereas the proposition q is called the **conclusion** or **consequent.**

The connective if…then is denoted by the symbol $\rightarrow$.

It is false when p is true and q is false, otherwise it is true. In particular, if p is false, then $p \rightarrow q$ is true for any q.

Definition:- A conditional statement that is true by virtue of the fact that its hypothesis is false is called **true by default** or **vacuously true**.
For example, the conditional statement
" If 3 + 3 = 7, then I am the king of Japan" is **true** simply because p : 3 + 3 = 7 is false. So it is not the case that p is true and q is false simultaneously.
Thus the truth values of the conditional proposition $p \rightarrow q$ are defined by the truth table:

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Each of the following expressions is an equivalent form of the conditional statement $p \rightarrow q$:

p implies q
q if p
p only if q

p is sufficient condition for q
q is necessary condition for p.

Example:- Restate each proposition in the form of a conditional proposition:

(a) I will eat if I and hungry

(b) 3 + 5 = 8 if it is snowing

(c) when you sing, my ears hurt

(d) Ram will be a good teacher if he teaches well.

(e) A necessary condition for English to win the world series is that they sign a right handed relief pitcher.

(f) A sufficient condition for Sohan to visit Calcutta is that he goes to Disney land.

Solution:-

(a) If I am hungry, then I will eat

(b) If it is snowing, then 3 + 5 = 8

(c) If you sing, then my ears hurt

(d) If Ram teaches well, then he will be a good teacher

(e) If English win the world series , then they sign a right handed relief pitcher

(f) If Sohan visit Calcutta, then he goes to Disney land.

Representation of "If …..then" as OR.

Lemma:- Show that for proposition p and q,

$$p \rightarrow q \equiv \sim p \vee q$$

Proof:- The truth values for $p \rightarrow q$ and $\sim p \vee q$ are given below:

P	q	$p \rightarrow q$	$\sim p$	$\sim p \vee q$
T	T	T	F	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Same truth values

Hence

$$p \rightarrow q \equiv \sim p \vee q$$

Example:- Rewrite the statement in "If….then" form:

Either you get to work on time or you are fired.

Solution:- Let

$\sim p$: you get to work on time

and

q : you are fired

then the given statement is $\sim p \vee q$. But

p : you do not get to work on time.

Hence according to above lemma, the equivalent "If….then" version of the given statement is

If you do not get to work on time, then you are fired.

Negation of a conditional statement:- We know that $p \rightarrow q$ is false if and only if p is true and its conclusion q is false. Also, we have shown above that

$$p \rightarrow q \equiv \sim p \vee q$$

Taking negation of both sides, we have

$$\begin{aligned} \sim(p \rightarrow q) &\equiv \sim(\sim p \vee q) \\ &\equiv \sim(\sim p) \wedge (\sim q) \quad \text{(De-Morgan's Law)} \\ &\equiv p \wedge \sim q \quad \text{(Double negative Law or Involution Law)} \end{aligned}$$

(This can also be obtained by constructing the truth tables for $\sim(p \rightarrow q)$ and $p \wedge \sim q$; the truth tables would have the same truth values proving the logical equivalence)
Thus

The negation of "If p then q" is **logically equivalent to** "p and not q".

Example:- Write negations for each of the following statements:

(a) If I am ill, then I cannot go to university

(b) If my car is in the repair shop, then I cannot attend the class.

Solution:- We know that negation of " If p then q" is logically equivalent to "p and not q". Using this fact, the negations of (a) and (b) are respectively

(1) I am ill and I can go to university
(2) My car is in the repair shop and I can attend the class.

Remark:- The negation of a "if…..then" proposition does not start with the word if.

Definition:- If $p \rightarrow q$ is an implication, then the **converse** of $p \rightarrow q$ is the implication $q \rightarrow p$.

Definition:- The **contrapositive** of a conditional statement "If p then q" is "If ~q then ~pf".
In symbols,

The contrapositive of $p \rightarrow q$ is $\sim q \rightarrow \sim p$.

Lemma:- A conditional statement is logically equivalent to its contrapositive.

Solution:- The truth tables of $p \rightarrow q$ and $\sim q \rightarrow \sim p$ are:

		$p \rightarrow q$			$\sim q \rightarrow \sim p$		
P	q	$p \rightarrow q$	p	q	~p	~q	$\sim q \rightarrow \sim p$
T	T	T	T	T	F	F	T
T	F	F	T	F	F	T	F
F	T	T	F	T	T	F	T
F	F	T	F	F	T	T	T

Same truth values

Hence

$$p \rightarrow q \equiv \sim q \rightarrow \sim p$$

Example:- Give the converse and contrapositive of the implications

(a) If it is raining, then I use my umbrella.

(b) If today is Monday, then tomorrow is Tuesday.

Solution:- (a) we have

p: It is raining

q : I use my umbrella

The converse is $q \rightarrow p$: If I use my umbrella, then it is raining.
The contrapositive is $\sim q \rightarrow \sim p$: If I do not use my umbrella, then it is not raining.

(b) we have

p : Today is Monday

q : Tomorrow is Tuesday

The converse is $q \rightarrow p$: If Tomorrow is Tuesday, then today is Monday.
The contrapositive is $\sim q \rightarrow \sim p$: If tomorrow is not Tuesday, then today is not Monday.

Definition:- The **inverse** of the conditional statement $p \rightarrow q$ is $\sim p \rightarrow \sim q$.
For example, the inverse of "If today is Easter, then tomorrow is Monday" is "If today is not Easter, then tomorrow is not Monday".
Remark:- If a conditional statement is true, then its converse and inverse may or may not be true. For example, on any Sunday except Ester, the conditional statement is true in the above example yet its inverse is false.

Only if:- " p only if q " means that p can take place only if q takes place also. That is, if q does not take place, then p cannot take place, i.e. $\sim q \rightarrow \sim p$. Therefore equivalence between a statement and its contrapositive imply that " if p occurs, then q must also occur". Hence

If p and q are statements, "p only if" means "if not q, then not p" or equivalently "if p then q".

Remark:- "p only if q" does not mean "p if q".

Example:- Use contrapositive to rewrite the following statement I n" if ….then" form:

"Ram will stand first in the class only if he works twelve hours a day."

Solution:- Version 1: We have

p : Ram will stand first in the class

q: he works twelve hours a day

The contrapositive is $\sim q \rightarrow \sim p$: If Ram does not works twelve hours a day, then he will not stand first in the class.

Version 2 : If Ram stands first in the class, then he will work twelve hours a day.

Definition:- If p and q are statements, the compound statement "p if and only if q" is called a **Biconditional statement** or an **equivalence**. It is denoted by $p \leftrightarrow q$. Observe that $p \leftrightarrow q$ is true only when both p and q are true or when both p and q are false.(i.e. if both p and q have same truth values) and is false if p and q have opposite truth values.

The biconditional statement has the following truth table:

$p \leftrightarrow q$

P	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

Lemma:- Show that

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

Solution:- We know that " p if and only if q" means that both " p if q" and " p only if q" hold. This means $p \leftrightarrow q$ should be logically equivalent to $(p \rightarrow q) \wedge (q \rightarrow p)$. We verify it using the truth table:

P	q	$p \rightarrow q$	$q \rightarrow p$	$p \leftrightarrow q$	$(p \rightarrow q) \wedge (q \rightarrow p)$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	T	F	F	F
F	F	T	T	T	T

Same truth values

Hence

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$

Remark:- It follows there for that biconditional statement can be written as the **conjunction** of two "if……then" statement namely $p \rightarrow q$ and $q \rightarrow p$. Also we know that

$$p \rightarrow q \equiv \sim p \vee q$$

and so

$$q \rightarrow p \equiv \sim q \vee p$$

Hence

$$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$$
$$\equiv (\sim p \vee q) \wedge (\sim q \vee p)$$

Thus the statements having $\rightarrow$ or $\leftrightarrow$ symbol are logically equivalent to statement having $\sim$, $\wedge$ and $\vee$.

Definition:- Let p and q be statements. Then p is a sufficient condition for q means "if p then q" p is a necessary condition for q means " if not p then not q".

The hierarchy of operations of logical connectives : The order of operations of connectives are

$$\sim, \ \wedge, \ \vee, \ \rightarrow, \ \leftrightarrow$$

1.4. Arguments and Their Validity

Definition:- An **argument** is a sequence of statements. **All statements but the final one** are called premises (or assumptions or hypothesis). The **final statement** is called the **conclusion**.

The symbol $\therefore$, read "therefore", is generally placed just before the conclusion.

Logical form of an argument : The logical form of an argument can be obtained from the contents of the given argument. For example, consider the argument:

If a man is a bachelor, he is unhappy

If a man is unhappy, he dies young

$\therefore$ Bachelors die young.

This argument has the abstract form

If p then q

If q then r

$\therefore p \rightarrow r$,

where

p : He is bachelor

q : He is unhappy

r : He dies young

Consider another example:

If Socrates is a human being, then Socrates is mortal

Socrates is a human being

$\therefore$ Socrates is mortal.

The abstract form of this argument is

If p then q

p

$\therefore$ q,

where

p : Socrates is human being

q : he is mortal

Definition:- An argument is said to be **valid** if the conclusion is true whenever all the premises are true.

Definition:- An **argument** which is **not true** is called a **fallacy**.

Method to Test Validity of an Argument

1. Identify the premises and conclusion of the argument

2. Construct a truth table showing the truth values of all the premises and conclusion

3. Find the rows (called **critical rows**) in which all the **premises are true.**

4. In each critical row, determine whether the conclusion of the argument is also true.

(a) If in each critical row the conclusion is also true, then the **argument form is valid**.

(b) If **there is at least one critical row** in which conclusion is **false**, the argument form is **fallacy** (invalid).

Example:- Show that the argument

$$p$$
$$p \rightarrow q$$
$$\therefore q$$

is valid.

Solution:- The premises are p and $p \rightarrow q$. The conclusion is q. The truth table is

		Premises		Conclusion	
p	q	p	$p \rightarrow q$	q	
T	T	T	T	T	$\rightarrow$Critical row
T	F	T	F	F	
F	T	F	T	T	
F	F	F	T	F	

In the first row, all the premises are true. Therefore the first row is critical row. The conclusion in this critical row is also true. Hence the argument is valid.

The argument (discussed above)

$$p$$
$$p \rightarrow q$$
$$\therefore q$$

is known as **Law of Detachment.**

Example:- Consider the following argument form

$$p \rightarrow q$$

$$p$$

$$\therefore q$$

An argument of this type is

$p \rightarrow q$: If the last digit of this number is a 0, then this is divisible by 10

p : The last digit of this number is a 0

$\therefore$ This number is divisible by 10.

The truth table for the premises and conclusion is

			Premises	Conclusion	
P	q	$p \rightarrow q$	p	Q	
T	T	T	T	T	→Critical row
T	F	F	T	F	
F	T	T	F	T	
F	F	T	F	F	

The first row is critical row and the conclusion I the critical row is true. Hence the given argument form is **Valid**.

The fact that this argument form is valid is called **Modus ponens**. This Latin term means "**Method of affirming**" (since the conclusion is an affirmation).

Example:- Consider the argument form

$$p \rightarrow q$$

$$\sim q$$

$$\therefore \sim p$$

An example of this type of argument form is

If Zeus is human, then Zeus is mortal

Zeus is not mortal

$\therefore$ Zeus is not human.

The truth table for the premises and conclusion is

		Premises		Conclusion	
p	q	$p \rightarrow q$	$\sim q$	$\sim p$	
T	T	T	F	F	
T	F	F	T	F	
F	T	T	F	T	
F	F	T	T	T	→Critical row

The last row is critical row and conclusion in this row is also true. Hence the argument form is valid.

The fact that this argument is valid is called **Modus Tollens** which means (**Method of denying**) since the conclusion is denial.
The above example can be solved by "Method of contradiction" also in the following way : Suppose that the conclusion is false, i.e, Zeus is human. Then by the given statement (If…..then) Zeus is mortal. But this contradicts the premises "Zeus is not mortal". Hence the argument is valid and so Zeus is not human.

Exercise :- Using truth table or critical row method, show that the argument

$$p \to q$$

$$q \to r$$

$$\therefore\ p \to r$$

is universally valid. This argument is known as **Rule of Inference** or **Law of Syllogism.**

Example:- Consider the argument

Smoking is healthy

If smoking is healthy, then cigarettes are prescribed by physicians

$\therefore$ Cigarettes are prescribed by physicians.

Solution:- In symbols, the argument is

$$p$$

$$p \to q$$

$$\therefore\ q$$

The argument is of the form **Modus Ponens** (or Law of Detachment) and so is valid. However, the **conclusion is false**. Observe that the first premises, p : " Smoking is healthy", is **false**. The second premises, $p \to q$ is then true and conjunction of the two premises $(p \wedge (p \to q))$ is false.

Example:- Fill in the blanks of the following arguments so that they become valid inferences :

(a) If there are more pigeons than there are pigeonholes, then two pigeons roost in the same hole.
There are more pigeons than there are pigeonholes
$\therefore$ ----------
(b) If this number is divisible by 6, then it is divisible by 2
This number is not divisible by 2
$\therefore$ ----------

Solution:- (a) In logical symbols, the argument is

$p \rightarrow q$

p

$\therefore$ ----------.

Hence, by **Modus ponens**, the answer is q, that is,

Two pigeons roost in the same hole.

(b) In logical symbols, the given premises and conclusion are

$p \rightarrow q$

$\sim q$

$\therefore$ ----------.

Hence, by **Modus tollen**, the answer is ~p, that is,

This number is not divisible by 6.

Example:- Using rules of valid inference solve the problem:

(a) If my glasses are on the kitchen table, then I saw them at breakfast
(b) I was reading the newspaper in the living room or I was reading in the kitchen
(c) If I was reading the newspaper in the living room. Then my glasses are on the coffee table.
(d) I did not see my glasses at breakfast
(e) If I was reading my book in bed, then my glasses are on the bed table.
(f) If I was reading the newspaper in the kitchen, then my glasses are on the kitchen table.

Where are the glasses?

Solution:-Let

p : my glasses are on the kitchen table

q : I saw them at breakfast

r : I was reading the newspaper in the living room

s : I was reading the newspaper in the kitchen

t : my glasses are on the coffee table

u : I was reading my book in bed

v : my glasses are on the bed table.

Then the given statements are

(a) $p \rightarrow q$ (b) $r \vee s$ (c) $r \rightarrow t$

(d) ~q (e) $u \to v$ (f) $s \to p$

The following deductions can be made:(1)

$p \to q$ by (a)
~q by (d)
$\therefore$ ~p by Modus Tollen (2)
$s \to p$ by (f)
~p by the conclusion of (1)
$\therefore$ ~ s by Modus Tollen (3)
$r \vee s$ by (b)
~s by the conclusion of (2)
$\therefore r$ by disjunctive syllogism(4)
$r \to t$ by (c)
r by the conclusion of (3)
$\therefore$ t by Modus Ponens

Hence t is true and the glasses are on the coffee table.

Contradiction Rule:- If the supposition that the statement p is false leads logically to a contradiction, then you can conclude that p is true.
In symbols,
$\sim p \to c$, where c is a contradiction
$\therefore$ p
The truth table for the premise and the conclusion of this argument is given below:

p	~p	c	$\sim p \to c$	p	
T	F	F	T	T	$\to$Critical row
F	T	F	F	F	

The premises and conclusion are both true in the critical row and hence the argument is valid.

Example:- Knights and Knaves (Raymond Smullyan's Description of an island containing two types of people):

This island contains two types of people: knights who always tell the truth and Knaves who always lie. A visitor visits the island and approached two natives who spoke to the visitor as follows:

A says : B is a knight
B says : A and I are of opposite type.

What are A and B?

Solution:- Suppose A is a knight. Because A always tells the truth, it follows that B is a knight.

Therefore what B says is true (by the definition of Knight). Therefore A and B are of opposite type. Thus we arrive at a contradiction: A and B are both Knights and A and B are of opposite type. Therefore supposition is wrong. Hence A is not a Knight. So A is a Knave. Therefore what A says is false. Hence B is not a Knight and so is a Knave. Hence A and B **are both Knaves**.

1.5. Quantifiers

So far we have studied the compound statements which were made of simple statements joined by the connectives $\sim$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$. That study cannot be used to determine validity in the majority of everyday and mathematical situations. For example, the argument
All human being are mortal
Socrates is a human being
$\therefore$ Socrates is mortal
is intuitively correct. Yet its validity cannot be derived using the methods studied so far. To check the validity of such argument it is necessary to separate the statements into parts-subjects and predicates. Also we must analyse and understand the special role played by words denoting quantities such as All or Some.

Definition:- The symbolic analysis of predicates and quantified statements is called the **predicate calculus** whereas the symbolic analysis of ordinary compound statements is called the **Statement Calculus** (or **prepositional calculus**).

In English grammar, the predicate is the part of a sentence that gives information about the subject. For example, in the sentence "Ram is a resident of Karnal", the word Ram is the subject and the phrase "is a resident of Karnal" is the predicate. Thus, **predicate is the part of the sentence from which the subject has been removed**.

In logic, predicates can be obtained by removing any nouns from a statement. For example, if P stands for "is a resident of Karnal" and Q stands for "is a resident of", then both P and Q are predicate symbols. The sentences "x is a resident of Karnal" and "x is a resident of y" are denoted as P(x) and Q(x, y) respectively, where x and y are predicate variables that take values in appropriate sets.

Definition:- A **predicate** is a sentence that contains a finite number of variables and becomes a statement when specific values are substituted for the variables.

The domain of a predicate variable is the set of all values that may be substituted in place of the variables. The predicates are also known as "**propositional functions** or **open sentences**".

Definition:- Let P(x) be a predicate and x has domain D. Then the set

$$\{ x \in D : P(x) \text{ is true}\}$$

is called the **truth set of P(x)**.

For example, let P(x) be " x is an integer less than 8" and suppose the domain os x is the set of all positive integers. Then the truth set of P(x) is {1, 2, 3, 4, 5, 6, 7}

Let P(x) and Q(x) be predicates with common domain D of x. The notation $P(x) \Rightarrow Q(x)$ means that every element in the truth set of P(x) is in the truth set of Q(x).

Similarly $P(x) \Leftrightarrow Q(x)$ means that P(x) and Q(x) have **Identical truth sets**.

For example, let

P(x) be "x is a factor of 8"

Q(x) be "x is a factor of 4"

R(x) be " $x < 5$ and $x \neq 3$"

and let the domain of x be set of positive integers (Zahlen).
Then

Truth set of P(x) is {1, 2, 4, 8}

Truth set of Q(x) is {1, 2, 4}

Since every element in the truth set of Q(x) is in the truth set of P(x), $Q(x) \Rightarrow P(x)$.

Further, truth set of R(x) is {1, 2, 4}, which is identical to the truth set of Q(x). Hence $R(x) \Leftrightarrow Q(x)$.

Definition:- The words that refer to quantities such as "All", or "some" and tell for how many elements a given predicate is true are called **quantifiers**.
By adding quantifier, we can obtain statements from a predicate.

1.6. Universal Quantifiers and Existence Of Quantifiers

Definition:- The symbol $\forall$ denotes " for all" and is called the **Universal quantifier**.

Thus the sentence

All human beings are mortal

Can he written as

$$\forall\ x \in S,\ x \text{ is mortal,}$$

where S denotes the set of all human being.

Definition:- Let P(x) be a predicate and D the domain of x. A statement of the form "$\forall\ x \in D$, P(x)" is called a **universal statement**.
A universal statement P(x) is true if and only if P(x) is true **for every** x in D and a universal statement P(x) is false if and only if P(x) is false **for at least one** $x \in D$.
A value for x for which P(x) is **false** is called a **Counterexample** to the universal statement.]

Example:- Let D = {1, 2, 3, 4} and consider the universal statement

$$P(x) : \forall\ x \in D,\ x^3 \geq x$$

This is true for all values of $x \in D$ since $1^3 \geq 1$, $2^3 \geq 2$ and so on.
But the universal statement

$$Q(x) : \forall\ n \in N,\ n + 2 > 8$$

is not true because if we take n = 6, then 8 > 8 which is absurd.

Definition:- The symbol $\exists$ denotes "there exists" and is called the **existential quantifier**.
For example, the sentence " There is a University in Kurukshetra" can be expressed as

$\exists$ a university u such that u is in Kurukshetra.

or, we can write

$\exists\ u \in U$ such that u is in Kurukshetra, where U is the set of universities.

The words **such that** are inserted just before the predicate.

Definition:- Let P(x) be a predicate and D is the domain of x. a statement of the form "$\exists\ x \in D$ such that P(x)" is called an **Existential Statement**. It is defined to be true if and only if P(x) is true for at least one x in D.
It is false if and only if P(x) is false for all x in D.
For example the existential statement

$$\exists\ n \in N : n + 3 < 9$$

is **true** since the set

$$\{n : n + 3 < 9\} = \{1, 2, 3, 4, 5\}$$

is not empty.

Example:- Let A = {2, 3, 4, 5}, then the existence statement

$$\exists\ n \in A : n^2 = n$$

is false because there is no element in A whose square is equal to itself.

Definition:- A statement of the form

$$\forall\ x, \text{ if } P(x) \text{ then } Q(x)$$

is called **universal conditional statement**.
Consider the statement

$$\forall\ x \in \mathbf{R}, \text{ if } x > 2 \text{ then } x^2 > 4$$

can be written in any of the form

(i) If a real number is greater than 2, then its square is greater than 4
(ii) Whenever a real number is greater than 2, its square is greater than 4
(iii) The square of any real number that is greater than 2 is greater than 4.
(iv) The squares of all real numbers greater than 2 are greater than 4.

On the other hand, consider the statements

(i) All bytes have eight bits

(ii) No fire trucks are green.

These can be written as

(i) $\forall$ x, if x is a byte, then x has eight bits

(ii) $\forall$ x, if x is a fire truck , then x is not green.

Example:- Consider the statement

(i) $\forall$ Polygons p, if p is a square, then p is a rectangle.

This is equivalent to the universal statement

" $\forall$ squares p, p is a rectangle".

(ii) $\exists$ a number n such that n is prime and n is even.

This is equivalent to

"$\exists$ a prime number n such that n is even".

Remark:- Existential quantification can also be implicit. For example, the statement

"The number 24 can be written as a sum of two even integers"

can be expressed as

"$\exists$ even integers m and n such that 24 = m + n".

1. Universal quantification can also be implicit. For example the statement

"If a number is an integer, then it is rational number"

is equivalent to

"$\forall$ real number x, if x is an integer, then it is a rational number."

1.7. Negation of University Statement

Definition:- The negation of a universal statement

$\forall$ x in D, P(x)

is logically equivalent to a statement of the form

$\exists$ x in D such that ~P(x)

Thus

$$\sim(\forall\ x \in D, P(x)) \equiv \exists\ x \in D, \sim P(x)$$

Hence

The negation of a universal statement "all are" is logically equivalent to an existential statement "some are not".
For example, the negation of

(i) " For all positive integer n, we have $n + 2 > 9$"

is

" There exists a positive integer n such that $n + 2 \not> 0$".

(ii) The negation of

" All students are intelligent"

is

"Some students are **not** intelligent"

or

"$\exists$ a student who is **not** intelligent".

(iii) the negation of

" No politicians are honest"

is

"$\exists$ a politician x such that x is honest."

or

"Some politicians are honest".

Definition:- The **negation of a universal conditional statement** is defined by

$$\sim(\forall x, P(x) \rightarrow Q(x)) \equiv \exists\ x \text{ such that } \sim(P(x) \rightarrow Q(x)).$$

Also we know that the negation of if-then statement is

$$\sim (P(x) \rightarrow Q(x)) \equiv P(x) \wedge \sim Q(x).$$

Hence

$$\sim(\forall x, P(x) \rightarrow Q(x)) \equiv \exists \text{ x such that } P(x) \wedge \sim Q(x) ,$$

that is,

$$\sim(\forall x, P(x) \rightarrow Q(x)) \equiv \exists \text{ x such that P(x) and } \sim Q(x).$$

Example:- The negation of

$$\forall \text{ people p, if p is blond then p has blue eyes}$$

is

$$\exists \text{ a person p such that p is blond and p does not have blue eyes.}$$

Example:- Suppose there is a bowl and we have no ball in the bowl. Then the statement

"All the balls in the bowl are blue"

is true "by default" or " Vacuously true" because there is no ball in the bowl which is not blue.

If P(x) is a predicate and the domain of x is D = $\{x_1, x_2, \ldots, x_n)$, then the statement

$$\forall x \in D, P(x)$$

and

$$P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n)$$

Are logically equivalent.
For example, let P(x) be

"x . x = x"

and let D = {0, 1}. Then

$$\forall x \in D, P(x)$$

can be written as

$$\forall \text{ binary digits x , } \quad x . x = x.$$

This is equivalent to

$$0 . 0 = 0 \text{ and } 1 . 1 = 1$$

which can be written as

$$P(0) \wedge P(1)$$

Similarly, if P(x) is a predicate and D = $(x_1, x_2, \ldots, x_n\}$ then the statements

$$\exists x \in D, P(x)$$

and

$$P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n)$$

are logically equivalent.

Definition:- Let

$$\forall\ x \in D, \text{ if } P(x) \text{ then } Q(x)$$

be a statement. Then

(i) **Contrapositive** of this statement is

$$\forall\ x \in D, \text{ if } \sim Q(x) \text{ then } \sim P(x)$$

(ii) **Converse** of this statement is

$$\forall\ x \in D, \text{ if } Q(x) \text{ then } P(x)$$

(iii) **Inverse** of this statement is

$$\forall\ x \in D, \text{ if } \sim P(x) \text{ then } \sim Q(x)$$

1.8. Universal Modus Ponens

The following argument form is valid

Formal Version	**Informal Version**
$\forall$ x if P(x) then Q(x)	If x makes P(x) true, then x makes Q(x) true
P(a) for a particular a	a makes P(x) true
$\therefore$ Q(a)	$\therefore$ a makes Q(x) true.

An argument of this form is called a **Syllogism**. The first and second premises are called its **major premises** and **minor premises** respectively.

Example:- Consider the argument:

If a number is even, then its square is even

K is a particular number that is even

$\therefore$ K^2 is even

The major premises of this argument can be written as

$\forall$ x, if x is even then x^2 is even

Let

P(x) : "x is even"

Q(x) : "x^2 is even"

and let k be an even number. Then the argument is

$\forall$ x, if P(x) then Q(x)

P(k) for k

$\therefore$ Q(k)

This form of argument is valid by universal Modus Ponens.

1.9. Universal Modus Tollens

The following argument form is valid

Formal Version	**Informal Version**
$\forall$ x if P(x) then Q(x)	If x makes P(x) true, then x makes Q(x) true
~Q(a) for a particular a	a does not makes Q(x) true
$\therefore$ ~P(a)	$\therefore$ a does not makes P(x) true.

Example:-

All human being are mortal

Zeus is not mortal

$\therefore$ Zeus is not human

The major premise of this argument can be rewritten as

$\forall$ x, if x is human, then x is mortal

Let

P(x) : x is human

Q(x) : x is mortal

let Z = Zeus

Then we have

$\forall$ x, if P(x) then Q(x)

~Q(Z)

$\therefore$ ~P(Z)

which is valid by Universal Modus Tollens.

Example:- The argument

All professors are absent minded

Tom is not absent minded

$\therefore$ Tom is not a professor.

The major premise can be written as

$\forall$ x, if x is professor, then x is absent minded.

Let

P(x) : x is professor.

Q(x) : x is absent minded.

Z = Tom

Then we have

$\forall$ x, if P(x) then Q(x)

~Q(Z)

$\therefore$ ~P(Z).

Hence, by Universal Modus Tollens, Tom is not a professor.

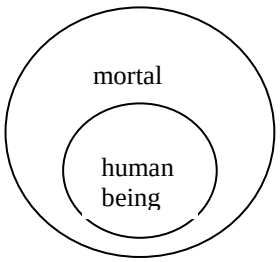
1.10. Use of Diagrams For Validity of Arguments

Consider the argument:

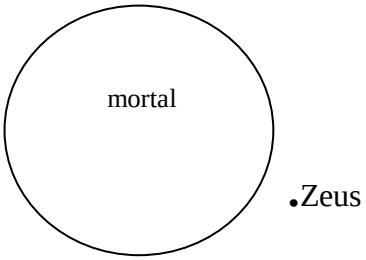
All human beings are mortal

Zeus is not mortal

$\therefore$ Zeus is not a human being.

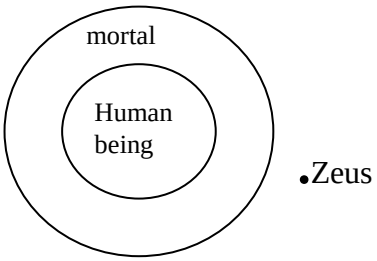


Major premise



Minor premise

The two diagrams fit together in only one way as shown below:



Since Zeus is outside the mortal disc it is necessarily outside the human beings disk. Hence the Conclusion is true.

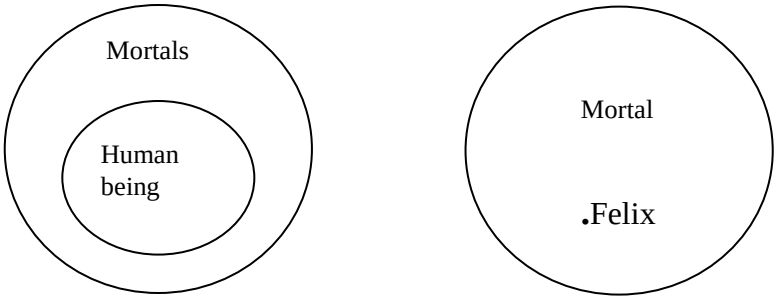
Example:- Use a diagram to show the invalidity of the arguments

All human being are mortal

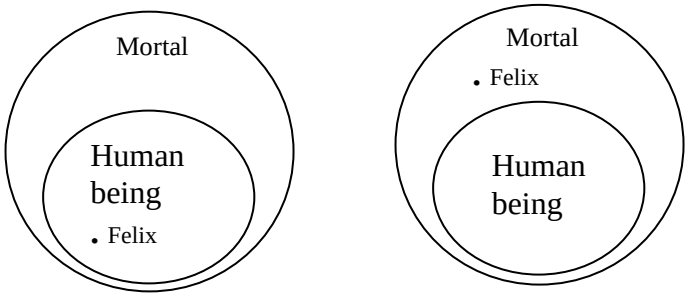
Felix is mortal

$\therefore$ Felix is a human being.

Solution:- The major premise and a minor premise of the arguments are shown in the diagrams below :



There are **two possibilities** to fit these two diagrams into a single one.



(1) (2)

The conclusion "Felix is a human being" is true in the first case but not in the second. Hence the argument is invalid.

PART B : SEMIGROUPS AND MONOIDS

1.11. Binary Operation and its Properties

Definition. Let A be a non-empty set. Then a mapping $f : A \times A \to A$ is called a **binary operation.** Thus, a binary operation is a rule that assigns to each ordered pair $(a, b) \in A \times A$ an element of A.

For the sake of simplicity, we write $a * b$ in place of f(a,b).

Examples. 1. Let **Z** be the set of integers. Then $f : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ defined by $f(a,b) = a * b = a+b$, $a, b \in \mathbf{Z}$ is a binary operation on **Z** because the sum of two integers a and b is again an integer.

Thus, **addition of integers** is a binary operation.

2. Let **N** be the set of positive integers. Then $f : N \times N \to N$ defined by $f(a,b) = a * b = a–b$ is **not a binary operation** because difference of two positive integers need not be positive integer. For example 2-5 is not a positive integer.

3. For the set **N** of positive integers, let $f : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ be defined by $f(a,b) = \frac{a}{b}$. Then f is not a binary operation. For example, if a = 2, b = 7, then $\frac{a}{b} = \frac{2}{7}$ is not a positive integer.

4. Let **Z** be the set of all integers. Then $f : \mathbf{Z} \times \mathbf{Z} \to \mathbf{Z}$ defined by

$$f(a,b) = \max(a, b)$$

is a binary operation. For example,

$$f(2, 4) = 2 * 4 = \max(2,4) = 4 \in \mathbf{Z}.$$

5. Let A = {a, b, c}. Define $*$ by

$$x * y = x, \quad x, y \in A.$$

Then the table given below defines the operation $*$

*	a	b	c
a	a	a	a
b	b	b	b
c	c	c	c

Further, if we define **.** by

$$x.y = y, \quad x, \quad y \in A,$$

then the table given below defines the operation **.**

.	**a**	**b**	**c**
a	a	b	c
b	a	b	c
c	a	b	c

6. If A = {0, 1}. Then the binary operations $\wedge$ and $\vee$ are defined by the following tables :

$\wedge$	**0**	**1**
0	0	0
1	0	0

and

$\vee$	**0**	**1**
0	0	1
1	1	1

Properties of Binary Operation

1. Commutative Law :- A binary operation $*$ on a set A is said to be **commutative if**

$$a * b = b * a$$

for any elements a and b in A.

For example, consider the set **Z** of integers. Since

$$a+b = b+a \quad \text{and } a.b = b.a,$$

for a, b $\in$ **Z**, the addition and multiplication operations on **Z** are commutative.

But, on the other hand, subtraction in **Z** is not commutative since, for example,

$$2 - 3 \neq 3 - 2$$

Example. Fill in the following table so that the binary operation $*$ is commutative.

*	**a**	**b**	**c**
a	b	–	–
b	c	b	a

c	a	–	c

We note that $b * a = c$, therefore, for commutativity we must have $a * b = c$.

Further, $c * a = a$, hence $a * c$ should also be a.

Further, for commutativity we should have

$$\begin{aligned} c * b &= b * c \\ &= a \end{aligned}$$

Thus $c * b$ should be a .

Note that for commutativity of $*$, the entries in the table are symmetric with respect to the main diagonal.

Definition. A binary operation $*$ on a set A is said to be **associative** if for any elements a, b, c in A, we have

$$a * (b * c) = (a * b) * c$$

For example, addition and multiplication of integers are associative. But subtraction of integers is **not** associative. For example,

$$(2-4) - 5 = -7 ,$$

but

$$2-(4-5) = 3$$

Theorem. Let $*$ be a binary operation on a set A. Then any product $a_1 * a_2 * \dots * a_n$ requires no parenthesis, that is, all possible products are equal.

Proof. We shall prove this result by induction on n. Since $*$ is associative, the theorem holds for n = 1, 2 and 3. Suppose $[a_1\ a_2 \dots a_n]$ denote any product and

$$(a_1\ a_2 \dots a_n) = (\dots (a_1\ a_2)a_3\dots)a_n$$

It is sufficient then to show that

$$[a_1a_2\dots a_n] = (a_1a_2 \dots a_n)$$

Since $[a_1\ a_2 \dots a_n]$ denote arbitrary product, there is an $m < n$ such that induction yields

$$\begin{aligned} [a_1\ a_2 \dots a_n] &= [a_1\ a_2 \dots a_m]\ [a_{m+1} \dots a_n] \\ &= [a_1\ a_2 \dots a_m]\ (a_{m+1} \dots a_n) \\ &= [a_1\ a_2 \dots a_m]\ ((a_{m+1} \dots a_{n-1})a_n) \\ &= (\ [a_1\ a_2 \dots a_m]\ (a_{m+1} \,..\, a_{n-1}))a_n \\ &= [\ a_1 \dots a_{n-1}]\ a_n \\ &= (a_1 \dots a_{n-1})a_n \\ &= (a_1\ a_2 \dots a_n) , \end{aligned}$$

which proves the result.

Definition. Let $*$ be a binary operation on a set A. An element e in A is called an **identity** element for $*$ if for any element $a \in A$,

$$a * e = e * a = a.$$

Further e is called right identity if $a * e = a$ and left identity if $e * a = a$ for any $a \in A$.

Let e_1 the left identity and e_2 be the right identity for a binary operation $*$. Then

$$e_1 e_2 = e_2 \quad \text{since } e_1 \text{ is left identity}$$

and

$$e_1 e_2 = e_1 \quad \text{since } e_2 \text{ is right identity}$$

Hence $e_1 = e_2$ and so **identity element for a binary operation is unique**.

Definition. Let $*$ be a binary operation on a set A and let A has identity element e. Then inverse of an element a in A is an element b such that

$$a * b = b * a = e.$$

We shall see later on that if $*$ is associative, then the inverse of an element, if it exits, is unique.

Definition. A binary operation $*$ on a set A is said to satisfy the **left cancellation law** if

$$a * b = a * c \Rightarrow b = c$$

A binary operation $*$ on a set A is said to obey **right cancellation law** if

$$b * a = c * a \Rightarrow b = c$$

Let **Z** be the set of integers. Since

$$a + b = a + c \Rightarrow b = c$$

and

$$b + a = c + a \Rightarrow b = c \text{ for } a, b, c \in \mathbf{Z},$$

it follows that addition of integers in **Z** obeys both cancellation laws.

Similarly multiplication of integers also obey cancellation laws.

On the other hand, matrix multiplication does not obey cancellation laws. **To see it, let**

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \; B = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \; C = \begin{bmatrix} 0 & -3 \\ 1 & 5 \end{bmatrix}.$$

Then

$$AB = AC = \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix}$$

but $B \neq C$.

1.12. Algebraic Systems

Definition. A non-empty set together with a number of binary operations on it is called an **algebraic system.**

In what follows, we shall define some algebraic systems :

Definition. A non-empty set S is said to be a **semigroup** if in S there is defined a binary operation $*$ satisfying the following property :

If $a, b, c \in S$, then

$$a * (b * c) = (a * b) * c \qquad \text{(Associative Law)}$$

Thus

A non-empty set S together with an associative binary operation $*$ defined on S is called a Semi-group.

We denote the semigroup by $(S, *)$.

Definition. A semigroup $(S, *)$ is called **commutative** if the binary operation $*$ is a commutative operation, i.e., if

$$a * b = b * a \quad \text{for } a, b \in S.$$

Examples. 1. Let **Z** be the set of all integers. Then (**Z**, +) is a commutative semigroup. In fact, if $a, b, c \in \mathbf{Z}$, then

(i) $a * b = a+b$ is an integer. Therefore, the operation + on **Z** is a binary operation.

(ii) $a + (b+c) = (a+b) + c$, because associative law holds in the set of integers.

(iii) $a + b = b + a$, because addition in **Z** is commutative.

2. The set **Z** of integers with the binary operation of subtraction is not a semi-group since subtraction is not associative in **Z**.

3. Let S be a finite set and let F(S) be the collection of all functions $f : S \to S$ under the operation of **composition of functions.** We know that composition of functions is associative, i.e.

$$fo(goh) = (fog)oh , \qquad f , g , h \in F(S) .$$

Hence F(s) is a semigroup.

4. The set P(S), where S is a set, together with the operation of union is a commutative semigroup.

5. The integers modulo m, denoted by $\mathbf{Z}_m$, refer to the set

$$\mathbf{Z}_m = \{0, 1, 2,\ldots, m-1\} .$$

The addition in $\mathbf{Z}_m$ is defined as

$$a + b = r,$$

where r is the remainder when a+b is divided by m. The multiplication in $\mathbf{Z}_m$ is defined by

$$a.b = r,$$

where r is the remainder when a+b is divided by m .

For example, consider

$$\mathbf{Z}_4 = \{0, 1, 2,3\}$$

The addition table is

+	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

We note

$$(1+2) + 3 = 3+3 = 2 \quad \text{and} \quad 1+(2+3) = 1+1 = 2$$

Hence

$$(1+2)+3 = 1 + (2+3)$$

In general,

$$(a+b) + c = a + (b+c), \ a, b, c \in \mathbf{Z}_4$$

Hence $\mathbf{Z}_4$ is a semi-group.

Definition. A non-empty set S is said to be a **monoid** if in S there is defined a binary operation $*$ satisfying the following properties :

(i) If a, b, c $\in$ S, then

$$a * (b * c) = (a * b) * c \qquad \text{(Associative Law)}$$

(ii) There exists an element e $\in$ S such that

$$e * a = a * e = a \quad \text{for all } a \in S \quad \text{(Existence of identity element)}$$

Thus :

An algebraic system (S, $*$) is said to be a **monoid** if

(i) $*$ is a binary operation on non-empty set S

(ii) $*$ is an associative binary operation on S

(iii) There exists an identity element e in S.

It, therefore, follows that

A monoid is a semi-group (S, $*$) that has an identity element.

Example. 1. In example 3 above, identity function is an identity element for F(S). Hence F(S) is a monoid.

2. Let **M** be the set of all n × n matrices and let the binary operation $*$ of **M** be taken as addition of matrices. Then (**M**, $*$) is a monoid. In fact,

   (i) The sum of two n × n matrices is again a matrix of order n × n . Thus the operation of matrix addition is a binary operation.

   (ii) If A, B, C ∈ **M**, then

   $$A + (B+C) = (A+B) + C \quad \text{(Associative Law)}$$

   (iii) The zero matrix acts as additive identity of this monoid because

   $$A + 0 = 0 + A = A \quad \text{for } A \in \mathbf{M}.$$

Definition. Let A be a non-empty set. **A word** w on A is a finite sequence of its elements.

For example,

$$w = ab\ ab\ bb = ab\ ab^3$$

is a word on A = {a, b} .

Definition. The number of elements in a word w is called **its length** and is denoted by $l(w)$**.**

For example, length of w in the above example is

$$l(w) = 6$$

Definition. Let u and v be two words on a set A. Then the word obtained by writing down the elements of u followed by the elements of v is called the **concatenation** of the words u and v on A.

For example, if A = {a, b, c} and

$$u = ab\ a\ bbb \quad \text{and } v = a\ c\ b\ a\ b$$

then

$$w = ab\ abbb\ ac\ bab = abab^3acbab$$

is the concatenation of u and v.

Let F(A) denote the collection of all words on A under the operation of concatenation. We note that

$$(u\ v)w = u(v\ w)$$

for u, v, w ∈ F(A). Hence F(A) is a **semigroup** known as **Free semigroup on A.** The elements of A are called the **generator** of F(A).

Also, we note that if u, v are two words, then

$$l(uv) = l(u) + l(v).$$

Further, the empty sequence, denoted by λ, is also considered as a word on A. However, we do not assume that λ belongs to the free semigroup F = F(A). The set of all words on A

including λ is usually denoted by A_*. Thus **A_* is a monoid under concatenation.** It is called the **free monoid** on A.

Definition. Let $(S, *)$ be a semigroup and T be a subset of S. If T is closed under the operation $*$ that is, $a * b \in +$ whenever $a, b \in T$, then $(T, *)$ is called a **subsemigroup** of $(S, *)$.

Definition. Let $(S, *)$ be a monoid with identity e, and let T be a non-empty subset of S. If T is closed under the operation $*$ and $e \in T$, then $(T, *)$ is called a **submonoid** of $(S, *)$.

Clearly, the associative property holds in any subset of a semigroup and so **a subsemigroup $(T, *)$ of a semigroup $(S, *)$ is itself a semigroup.**

Similarly, a submonoid of a monoid is itself a monoid.

Example. 1. Let A be the set of even positive integers. Then (A, **.**), where **.** denotes ordinary multiplication is a subsemigroup of (N, X) since A is closed under multiplication.

Similarly, the set B of odd positive integers form a subsemigroup (B, X) of (N, X).

Also (A, +) is a subsemigroup of (**N**, **.**). But (B, +) is not a subsemigroup of (**N**, +) because B is not closed under addition. For example, 1+3 = 4 which is not odd.

2. Let $(S, *)$ be a semigroup and $a \in S$. If $T = \{a^i : i \in \mathbf{N}\}$, then $(T, *)$ is a subsemigroup of $(S, *)$.

3. Let F(A) be a free semigroup on the set A = {a, b}. Let G consists of all even words, that is, words with even length. The concatenation of two such words is also even. Thus G is a subsemigroup of F(A).

Theorem. The inverse of every element in a **semigroup with identity** e is unique.

Proof. We shall use associativity of the binary operation $*$ to prove the uniqueness of the inverse element.

So, suppose that b and c are two inverses of an element a in a monoid $(S, *)$. Therefore, we have

$$a * b = b * a = e \qquad \text{(i)}$$

$$a * c = c * a = e \qquad \text{(ii)}$$

We note that

$$b * (a * c) = b * e, \quad \text{by (ii)}$$
$$= b, \text{ because e is identity} \qquad \text{(iii)}$$

and

$$(b * a) * c = e * c, \text{ by (i)}$$
$$= c, \text{ because e is identity} \qquad \text{(iv)}$$

But associativity of binary operation $*$ implies

$$b * (a * c) = (b * a) * c$$

Hence, from (iii) and (iv) it follows that

$$b = c \quad ,$$

proving that inverse, if exist, of every element in a monoid is unique.

1.13 Homomorphism of Semigroups

We discuss now a method for comparing the algebraic structures of the two semigroups.

Definition. Let $(S, *)$ and $(T, *')$ be two semigroups. A function $f : S \to T$ is called a **semigroup homomorphism** if

$$f(a * b) = f(a) *' f(b)$$

for all $a, b \in S$.

If, in addition, f is also onto, we say that T is a **homomorphic image** of S.

Definition. Let $(S, *)$ and $(T, *')$ be two semigroups. If $f : S \to T$ is both one-to-one and onto in addition to being a homomorphism**,** then f is called an **isomorphism** from $(S, *)$ to $(T, *')$.

Definition. A homomorphism f from $(S, *)$ to $(T, *')$ is called a **monomorphism if f as a map is injective (one-to-one).**

Definition. A homomorphism f from $(S, *)$ to $(T, *')$ is called an **Epimorphism** if f as a map is surjective (onto).

Thus we may define isomorophism between two semigroups $(S, *)$ and $(T, *')$ as

Definition. Let $(S, *)$ and $(T, *')$ be two semigroups. Then a homomorphism $f : (S, *) \to (T, *')$ is called an **isomorphism** if it is both monomorphism and epimorphism.

OR

Definition. Let $(S, *)$ and $(T, *')$ be two semigroups. Then a mapping $f : S \to T$ is called an **isomorphism** if

(i) $f(a * b) = f(a) *' f(b)$ for all $a, b \in S$ (**semigroup homomorphism**)

(ii) f as a map is bijective.

Definition. Let $(S, *)$ and $(T, *')$ be two semigroups. If $f : S \to T$ is an isomorphism, then the semigroups $(S, *)$ and $(T, *)$ are called **isomorphic**. In such a case $(T, *')$ is called isomorphic image of $(S, *)$.

Examples. 1. Let F(A) be the free semigroup of a set A, and let **Z** be the semigroup of integers under addition. Let

$$f : F(A) \to \mathbf{Z}$$

be defined by

$$f(w) = l(w), \quad w \in F(A)$$

We note that, if $u, v \in F(A)$, then

$$f(uv) = l(uv)$$
$$= l(u) + l(v)$$
$$= f(u) + f(v)$$

Hence f is a **homomorphism.** Here, the operation in F(A) is written multiplicatively, whereas the operation in **Z** is addition.

2. Let **Z** be the set of integers and T be the set of all even integers. Then (**Z**, +) and (T, +) are semigroups. Let

$$f : \mathbf{Z} \to T$$

be defined by

$$f(a) = 2a, \ a \in \mathbf{Z}$$

We note that

(i) $f(a+b)= 2(a+b)$
$= 2a + 2b$
$= f(a) + f(b)$

Thus f is a homomorphism.

(ii) $f(a) = f(b) \Rightarrow 2a = 2b$
$\Rightarrow \quad a = b$

Hence f is one-to-one, that is, f is monomorphism.

(iii) Let b be an even integer. Then $a = \frac{b}{2} \in \mathbf{Z}$ and

$$f(a) = f\left(\frac{b}{2}\right) = 2\left(\frac{b}{2}\right) = b$$

Thus to every $b \in T$, there is an $a \in \mathbf{Z}$ such that $f(a) = b$.

Hence f is onto, i.e., f is epimorphism.

Hence f is an isomorphism.

Theorem. Let $(S, *)$ and $(T, *')$ be monoids with identities e and e′ respectively. Let $F : S \to T$ be a homomorphism from $(S *)$ onto $(T, *')$. Then $f(e) = e'$.

Proof. Let b be any element of T. Since f is surjective, there is an element $a \in S$ such that $f(a) = b$. Since e is identity of S, we have

$$a * e = a = e * a \qquad \text{(i)}$$

and so

$b = f(a)= f(a * e)$, by (i)
$= f(a) *' f(e)$, because f is homomorphism

$$= b *' f(e)$$

Also,

$$b = f(a) = f(e * a)$$
$$= f(e) *' f(a)$$
$$= f(e) *' b$$

Hence

$$b *' f(e) = f(e) *' b = b$$

and so f(e) is identity for T. Thus, $f(e) = e'$.

Theorem. If f is a homomorphism from a commutative semigroup (S, $*$) onto a semigroup (T, $*'$), then (T, $*'$) is also commutative, **that is, homomorphic image of an abelian (commutative) semigroup is abelian.**

Proof. Let $t_1, t_2 \in T$. Since f is onto, there exist $s_1, s_2 \in S$ such that

$$f(s_1) = t_1 \text{ and } f(s_2) = t_2$$

Then

$$\begin{aligned} t_1 *' t_2 &= f(s_1) *' f(s_2) \\ &= f(s_1 * s_2) , \text{ since f is homomorphism} \\ &= f(s_2 * s_1), \text{ since S is abelian} \\ &= f(s_2) *' f(s_1), \text{ since f is homomorphism} \\ &= t_2 *' t_1 . \end{aligned}$$

Hence (T, $*'$) is abelian.

Remark. The converse of the above theorem is not true.

Theorem. Let $f : (S, *) \to (T, *')$ be semigroup homomorphism. If S' is a subsemigroup of (S, $*$), then the image of S' under f is a subsemigroup of (T, $*'$).

Proof. Let f (S') be the image of S' under f and let t_1, t_2 be in f (S'). Then there are s_1 and s_2 in S' such that

$$t_1 = f(s_1) \text{ and } t_2 = f(s_2)$$

We claim that f(S') is closed under the binary operation $*'$. It is sufficient to show that $t_1 *' t_2 \in f(S')$. We have, in this direction,

$$\begin{aligned} t_1 *' t_2 &= f(s_1) *' f(s_2) \\ &= f(s_1 * s_2), \text{ because f is homomorphism.} \end{aligned}$$

Now since S' is a semigroup and $s_1, s_2 \in S'$, we have $s_1 * s_2 \in S'$(due to closeness of the peration $*$). Hence $f(s_1 * s_2) \in f(S')$. It follows, therefore, that $t_1 *' t_2 \in f(S')$.
Further, since the associativity hold in T, it also holds in f(S'). Hence f(S') is a subsemigroup of (T, $*'$).

Theorem. The intersection of two subsemigroups of a semigroup (S, $*$) is subsemigroup of (S, $*$).

Proof. Let $(S_1, *)$ and $(S_2, *)$ be two subsemigroups of the semigroup $(S, *)$. Let $a \in S_1 \cap S_2$ and $b \in S_1 \cap S_2$. Then

$$a \in S_1 \cap S_2 \Rightarrow a \in S_1 \text{ and } a \in S_2$$

$$b \in S_1 \cap S_2 \Rightarrow b \in S_1 \text{ and } b \in S_2$$

Since S_1 is a subsemigroup, therefore, $a, b \in S_1$ implies $a * b \in S_1$. Similarly, since S_2 is a subsemigroup, $a, b \in S_2$ implies $a * b \in S_2$. Hence

$$a * b \in S_1 \cap S_2$$

Hence $S_1 \cap S_2$ is closed under the operation $*$. Further associativity in S_1 and S_2 implies the associativity of $S_1 \cap S_2$ since $S_1 \cap S_2 \subseteq S_1$ and $S_1 \cap S_2 \subseteq S_2$. Hence $S_1 \cap S_2$ is a subsemigroup of $(S, *)$.

Corollary. Intersection of two submonoids of a monoid $(S, *)$ is a semimonoid of $(S, *)$.

Proof follows the same line as that in the above Theorem.

Remark. Union of two subsemigroups of a semigroup $(S, *)$ need not be a subsemigroup of $(S, *)$.

For example,

$$(S_1, *) = \{0, \pm 2, \pm 4, \pm 6, + \ldots.\}$$

and

$$(S_2, *) = \{0, \pm 3, \pm 6, \pm 9, \pm, \ldots\}$$

are subsemigroups of the semigroup $(\mathbf{Z}, +)$ of integers. But

$$S_1 \cup S_2 = \{0, \pm 2, \pm 3, \pm 4, \pm 6, \pm \ldots.\}$$

is not a subsemigroup of $(\mathbf{Z}, +)$, because

$$2 \in S_1 \cup S_2, \; 3 \in S_1 \cup S_2,$$

but $2+3 = 5 \notin S_1 \cup S_2$ showing that $S_1 \cup S_2$ is not closed under addition.

1.14. Quotient Structure

Definition. An equivalence relation R on a semigroup $(S, *)$ is called a **congruence relation** if $a \, R \, a'$ and $b \, R \, b'$ imply $(a * b) \, R \, (a' * b')$.

Examples. 1. Let $(\mathbf{Z}, +)$ be the semigroup of integers. Consider the relation R defined on $\mathbf{Z}$ by

$$A \, R \, b \text{ if and only if } a \equiv b \pmod{m}.$$

We know that $a \equiv b \pmod{m}$ if m divides $a-b$. We note that

(i) For any integer a, we have $a \equiv a \pmod{m}$, i.e., $a \, R \, a$

(ii) If $a \, R \, b$, then $a \equiv b \pmod{m} \Rightarrow m \mid (a-b) \Rightarrow m \mid (b-a)$ and so $b \equiv a \pmod{m}$ which means $b \, R \, a$.

(iii) If a R b and b R c, then

$$a \equiv b(\text{mod } m) \text{ and } b \equiv c(\text{mod } m)$$

$$\Rightarrow m|(a-b) \text{ and } m|(b-c)$$

$$\Rightarrow m\,[(a-b) + (b-c)]$$

$$\Rightarrow m|(a-c)$$

$$\Rightarrow a \equiv c \text{ (mod } m), \text{ which means that } a \text{ R } c.$$

Thus R is reflexive, symmetric and transitive and so is an **equivalence relation**. Further, if

$$a \equiv c \text{ (mod } m) \text{ and } b \equiv d \text{ (mod } m),$$

then

$$m \mid (a-c) \text{ and } m \mid (b-d)$$

$$\Rightarrow \quad m \mid [(a-c) + (b-d)]$$

$$\Rightarrow \quad m|[(a+b) - (c+d)]$$

$$\Rightarrow \quad (a+b) \equiv (c+d) \text{ (mod } m)$$

$$\Rightarrow \quad (a+b) \text{ R } (c+d)$$

Hence R is a congruence relation.

2. Consider the semigroup (**Z**, **.**), where **.** denotes ordinary multiplication. Let us again consider the relation R on **Z** defined by

$$a \text{ R } b \text{ if and only if } a \equiv b \text{ (mod } m).$$

This relation is an equivalence relation. Further if $a \equiv c$ (mod m) and $b \equiv d$ (mod m), then

$$m|(a-c) \text{ and } m|(b-d)$$

$$\Rightarrow \quad m|b(a-c) \text{ and } m|c(b-d)$$

$$\Rightarrow \quad n|(ab-bc) \text{ and } m|(bc-cd)$$

$$\Rightarrow \quad m|[(ab-bc + bc-cd)]$$

$$\Rightarrow \quad m|(ab-cd)$$

$$\Rightarrow \quad ab \equiv cd \text{ (mod } m)$$

Hence the relation is a congruence relation on (**Z**, **.**) .

3. Let F(A) be the free semigroup on a set A. Define u R v if u and v have the same length. We note that

(i) u R u because u has same length as u

(ii) If u R v, then u and v have same length $\Rightarrow$ v and u have same length $\Rightarrow$ v R u

(iii) If u R v and v R w, then u and v have same length and also v and w have same length and so u and w have same length, that is, u R w :

Hence R is an equivalence relation**.** Further, let u R v and u' R v'. Then

$$l(u) = l(v) \text{ and } l(u') = l(v') .$$

Then

$$l(uu') = l(vv') = m + n ,$$

that is

$$l(uu') = l(vv')$$

$$\Rightarrow \quad uv' \ R \ vv'$$

Hence R is a congruence relation on F = F(A).

4. Let (**Z**, +) be the semigroup of integers and let $f(x) = x^2 - x - 2$. Let R be a relation defined on **Z** by

a R b if and only if f(a) = f(b).

It can be shown that R is an equivalence relation. Further we note that

$$f(-1) = f(2) = 0 \quad \text{and so } -\mathbf{1\ R\ 2}$$

$$f(-2) = f(3) = 4 \quad \text{and so } -\mathbf{2\ R\ 3.}$$

But

$$f(-3) = 10 \quad \text{and } f(5) = 18 ,$$

and so

$$-\mathbf{3} \not\mathrel{R} \mathbf{5} .$$

Hence R is not a congruence relation.

1.15 Equivalence Classes

If R is an equivalence relation on the semi-group (S, *) , it will partition S into equivalence classes. Let [a] be the equivalence class containing a in S and let S/R denote the set of all equivalence classes, where R is congruence relation.

We define an operation $\odot$ on the equivalence classes S/R by

$$[a] \odot [b] = [a * b] , \qquad a , b \in S$$

that is $\odot : S/R \times S/R \to S/R$ is defined by

$$\odot ([a], [b]) = [a] \odot [b] = [a * b]$$

Then we have

Theorem. Let R be a congruence relation on the semigroup (S, $*$). Then $\odot : S/R \times S/R \to S/R$ defined by

$$\odot ([a], [b]) = [a] \odot [b] = [a * b] , a, b \in S$$

is a binary operation on S/R and (S/R, $\odot$) is a semigroup.

Proof. Suppose that ([a], [b]) = [a′], [b′]). Then a R a′ and b R b′. Since R is congruence relation, this implies a $*$ b R a′ $*$ b′. Thus [a $*$ b] = [a′ $*$ b′], that is, $\odot$ is a well defined function. Hence $\odot$ is a **binary operation** S/R.

Further we note that

$[a] \odot ([b] \odot [c]) = [a] \odot [b * c]$ (by definition of $\odot$)

$= [a * (b * c)]$ (by definition of $\odot$)

$= [(a * b) * c]$ (Associativity of $*$ in S)

$= [a * b] \odot [c]$ (by definition of $\odot$)

$= ([a] \odot [b]) \odot [c]$ (by definition of $\odot$)

Hence $\odot$ is an associative operation. This implies that (**S/R**, $\odot$) is a **semigroup**.
The operation $\odot$ is called **quotient binary relation** on S/R constructed from the given binary relation $*$ on S by the congruence relation R.
The semigroup (S/R, $\odot$) is called Quotient Semigroup **or** Factor Semigroup **or** the Quotient of S by R.

Theorem. **Let R be the congruence relation on the monoid (S, *), then (S/R, $\odot$) is a monoid.**

Proof. **We have shown above that (S/R, $\odot$) is a semigroup. Further if e is identity element in(S, $*$), then [e] is the identity in (S/R, $\odot$). Thus (S/R, $\odot$) is semigroup having identity element [e] and so is a monoid.**

Theorem. **Let R be a congruence relation on a semigroup (S,$*$) and let (S/R, $\odot$) be the corresponding quotient semigroup. Then the mapping ϕ : S $\rightarrow$ S/R (called the** natural mapping**) defined by**

$$\phi(a) = [a]$$

is an onto homomorphism**, known as** Natural homomorphism**.**

Proof. **According to definition of ϕ, to each [a] in S/R, there is a $\in$ S such that ϕ[a] = [a]. Hence ϕ is subjective. Now let a, b $\in$ S. Then**

$$\phi(a * b) = [a * b]$$

$$= [a] \odot [b]$$

$$= \phi(a) \odot \phi(b)$$

Hence ϕ is homomorphism onto.

Theorem **(**Fundamental Theorem of Semi-group Homomorphism**). Let f : S $\rightarrow$ T be a homomorphism of the semigroup (S, $*$) onto the semigroup (T, $*'$). Let R be the relation on S defined by**

$$a \; R \; b \quad \text{if } f(a) = f(b) \text{ for } a, b \in S$$

Then

(i) R is a congruence relation on S

(ii) (S/R, $\odot$) is isomorphic to (T, $*'$).

(If f is not onto, them (ii) shall be "S/R is isomorphic to f(S)"**.**

Proof. **First we show that R is an equivalence relation. We note that**

(i) Since f (a) = f (a), we have a R a.

(ii) If a R b, then f(a) = f (b) or f (b) = f(a) and hence b R a.

(iii) If a R b and b R c , then

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

and hence

$$f(a) = f(c)$$

and so a R c.

Thus the relation R is reflexive, symmetric and transitive and so an equivalence relation.

Suppose now that

$$a \, R \, a' \quad \text{and} \quad b \, R \, b' .$$

Then

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

Since f is homomorphism,

$$f(a * b) = f(a) *' f(b)$$

$$= f(a') *' f(b')$$

$$= f(a' * b')$$

Hence

$$(a * b) \, R(a' * b')$$

and so R is a congruence relation.

Define

$$\psi : S/R \to T$$

by

$$\psi ([a]) = f(a) .$$

We claim that ψ is well defined. Suppose [a] = [b]. ψ will be well defined if f(a) = f(b). Now [a] = [b] implies a R b, that is, f(a) = f(b). Hence ψ is a function (well defined).

Further, if [a], [b] $\in$ S/R, then

$$\psi ([a] \odot [b]) = \psi ([a * b]), \; a, b \in S$$

$$= f(a * b)$$

$$= f(a) *' f(b), \text{ because f is homomorphism}$$

$$= \psi [a] *' \psi [b]$$

So ψ is semigroup homomorphism.

Also

$$\psi ([a] = \psi ([b]) \Rightarrow f(a) = f(b)$$

$$\Rightarrow a \, R \, b$$

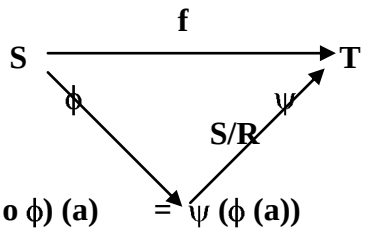
$$\Rightarrow [a] = [b],$$

and so ψ is one – to – one .

Thus ψ, as a map, is bijective and homomorphism. Hence ψ is an isomorphism and

$$S/R \cong T$$

Remark. **We have proved that the mapping $\phi : S \to S/R$ is natural homomorphism. Also, we proved that the mapping $\psi : S/R \to T$ is an isomorphism. Thus diagram of the situation becomes**



Also, we note that

$$(\psi \circ \phi)(a) = \psi(\phi(a)) = \psi([a]) = f(a) \text{ for all } a \in S .$$

Hence

$$\psi \circ \phi = f$$

1.16. Direct Product of Semigroups

Let $(S, *)$ and $(T, *')$ be two semigroups. Consider the cartesian product $S \times T$. Define a binary operation $*''$ on $S \times T$ by

$$(s_1, t_1) *'' (s_2, t_2) = (s_1 * s_2, t_1 *' t_2)$$

In what follows, we prove that $(S \times T, *'')$ is a semigroup.

Theorem. **Let $(S, *)$ and $(T, *')$ be semigroups. Then $(S \times T, *'')$ is a semigroup under the binary operation $*''$ defined by**

$$(s_1, t_1) *'' (s_2, t_2) = (s_1 * s_2, t_1 *' t_2) .$$

Proof. **If (s_1, t_1), (s_2, t_2) and $(s_3, t_3) \in S \times T$, then**

$$[(s_1, t_1) *'' (s_2, t_2)] *'' (s_3, t_3) = (s_1 * s_2, t_1 *' t_2) *'' (s_3, t_3)$$

$$= ((s_1 * (s_2 * s_3), t_1 *' (t_2) *' t_3))$$

$$= (s_1 * (s_2 * s_3), t_1 *' (t_2 *' t_3))$$

$$= (s_1, t_1) *'' (s_2 * s_3, t_2 *' t_3)$$

$$= (s_1, t_1) *'' [(s_2, t_2) *'' (s_3, t_3)]$$

Hence $*''$ is associative and so $(S \times T, *'')$ is a semigroup.

Corollary. If $(S, *)$ and $(T, *')$ are monoids, then $(S \times T, *'')$ is also a monoid.

Proof. **We have proved above that $(S \times T, *'')$ is a semigroup. We further note that if e_S is identity of $(S, *)$ and e_T is identity of $(T, *')$, then for $(s_1, t_1) \in S \times T$, we have**

$$(e_S, e_T) *'' (s_1, t_1) = (e_S * s_1, e_T *' t_1) = (s_1, t_1)$$

and

$$(s_1, t_1) *'' (e_S, e_T) = (s_1 * e_S, t_1 *' e_T) = (s_1, t_1)$$

Thus

$$(s_1, t_1) *'' (e_S, e_T) = (e_S, e_T) *''(s_1, t_1) = (s_1, t_1)$$

showing that (e_S, e_T) is identity element of $(S \times T, *'')$, that is, $(S \times T, *'')$ is a semigroup with identity (e_S, e_T) and hence is a monoid.

PART C : LATTICES

1.17 Definitions and Examples

Definition: A **lattice** is a partially ordered set (L, ≤) in which every subset {a, b} consisting of **two element** has a **least upper bound** and a **greatest lower bound.**

We denote lub({a, b}) by a ∨ b and call it **join** or **sum of a and b.** Similarly, we denote GLB({a, b}) by a ∧ b and call it **meet** or **product of a and b.**

Other symbol used are:

LUB : ⊕ , +, ∪

GLB : ∗, . , ∩

Thus **Lattice is** a mathematical structure with **two binary operations, join and meet.** Lattice structures often appear in computing and mathematical applications.

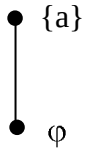
A totally ordered set is obviously a lattice but not all partially ordered sets are lattices.

Example 1. **Let A be any set and P(A) be its power set. The partially ordered set (P(A), ⊆) is a lattice in which the meet and join are the same as the operations ∩ and ∪ respectively. If A has single element, say a, then P(A) = {φ, {a}} and**

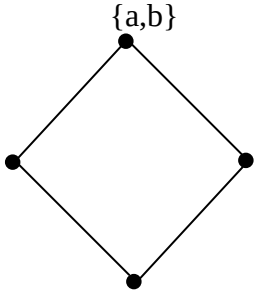
LUB({ φ, {a}) = {a}

GLB({φ, {a}) = φ

The Hasse diagram of (P(A), ⊆) is a chain containing two elements φ and {a} as shown below:



If A has two elements, say a and b. Then P(A) = {φ, {a}, {b}, {a, b}}. The Hasse diagram of {P(A), ⊆) is then as shown below :



{a} {b}

φ

We note that

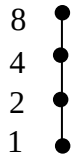
1. LUB exists for every two subsets and is $L \cup M$
2. GLB exists for every two subsets and is in $L \cap M$

for L, M $\in$ P(A). Hence P(A) in a lattice.

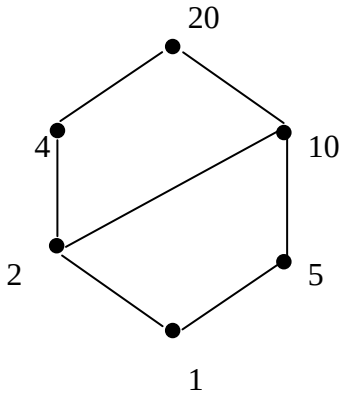
Example 2. Consider the poset (**N**, ≤), where ≤ is relation of divisibility. Then **N** is a lattice in which

join of a and b = a $\vee$ b = L C M(a, b)
meet of a and b = a $\wedge$ b = G C D (a, b) for a, b $\in$ **N**.

Example 3. Let n be a positive integer and let D_n be the set of all positive divisors of n. Then D_n is a lattice under the relation of divisibility**.** The Hasse diagram of the lattices D_8, D_{20} and D_{30} are respectively

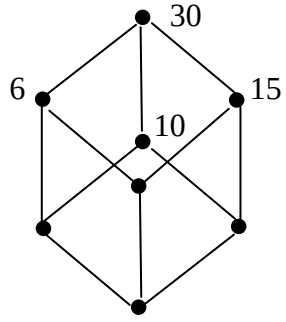


$D_8 = \{1, 2, 4, 8\}$



$D_{20} = \{1, 2, 4, 5, 10, 20\}$

and



3

2 5

1

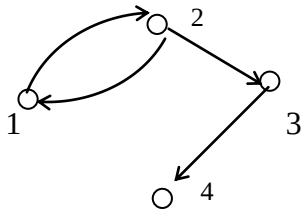
$D_{30} = \{1, 2, 3, 5, 6, 10, 15, 30\}$.

1.18. The Transitive Closure of a Relation

Definition: The **Transitive closure** of a relation R is the smallest transitive relation containing R. It is denoted by R^{∞} .

Example: Let A = {1, 2, 3, 4} and R = [(1, 2), (2, 3), (3, 4), (2, 1)] Find the transitive closure of R.

Solution: The digraph of R is



We note that from vertex 1, we have paths to the vertices 2, 3, 4 and 1. Note that path from 1 to 1proceeds from 1 to 2 to 1. Thus we see that the ordered pairs (1, 1), (1, 2), (1, 3) and (1, 4) are in R^{∞}. Starting from vertex 2, we have paths to vertices 2, 1, 3 and 4 so the ordered pairs (2, 1), (2, 2), (2, 3) and (2, 4) are in R^{∞}. The only other path is from vertex 3 to 4, so we have

$R^{\infty} = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 1), (2, 2), (2, 3), (2, 4), (3,4)\}$

Example: Let R be the set of all equivalence relations on a set A. As such R consists of subsets of $A \times A$ and so R is a partially ordered set under the partial order of set inclusion. If R and S are equivalence relations on A, the same property may be expressed in relational notations as follows:

$$R \subseteq S \text{ if and only if } x\ R\ y \Rightarrow x\ S\ y \text{ for all } x\ y \in A.$$

Then **(**R, $\subseteq$) is a poset. R is a lattice**,** where the meet of the equivalence relations R and S is their intersection $R \cap S$ and their join is $(R \cup S)^{\infty}$, the transitive closure of their union.

Definition: Let (L, ≤) be a poset and let (L, ≥) be the dual poset. If (L, ≤) is a lattice, we can show that (L, ≥) is also a lattice. In fact, for any a and b in L, the L U B of a and b in (L, ≤) is equal to the GLB of a and b in (L, ≥). Similarly, the GLB of a and b in (L, ≤) is equal to L U B in (L, ≥).

The operation $\vee$ and $\wedge$ are called **dual of each other**.

Example: Let S be a set and L = P(S). Then (L, $\subseteq$) is a lattice and its **dual lattice** is (L, $\supseteq$), where $\supseteq$ represents "contains". We note that in the poset (L, $\supseteq$), the join A $\vee$ B is the set A $\cap$ B and the meet A $\wedge$ B is the set A $\cup$ B.

1.19. Cartesian Product of Lattices

Theorem: If (L_1, $\leq$) and (L_2, $\leq$) are lattices, then (L, $\leq$) is a lattice, where $L = L_1 \times L_2$ and the partial order $\leq$ of L is the product partial order.

Proof: **We denote the join and meet in L_1 by $\vee_1$, and $\wedge_1$ and the join and meet in L_2 by $\vee_2$ and $\wedge_2$ respectively. We know that Cartesian product of two posets is a poset. Therefore $L = L_1 \times L_2$ is a poset. Thus all we need to show is that if (a_1, b_1) and $(a_2, b_2) \in$ L, then $(a_1, b_1) \vee (a_2, b_2)$and $(a_1, b_1) \wedge (a_2, b_2)$ exist in L.**

Further, we know that

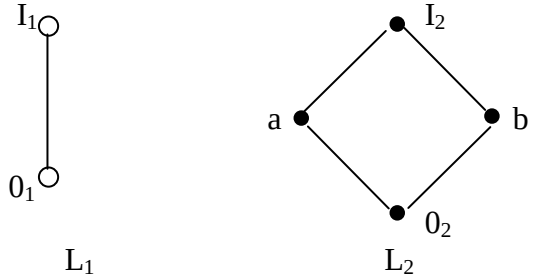
$$(a_1, b_1) \vee (a_2, b_2) = (a_1 \vee_1 a_2, \quad b_1 \vee_2 b_2)$$

and

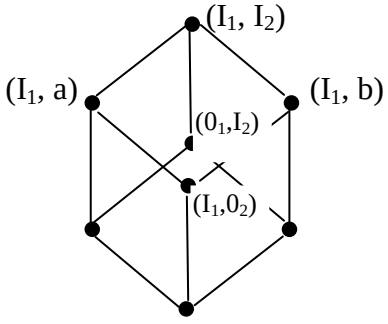
$$(a_1, b_1) \wedge (a_2, b_2) = (a_1 \wedge_1 a_2, \quad b_1 \wedge_2 b_2)$$

Since L_1 is lattice, $a_1 \vee_1 a_2$ and $a_1 \wedge_1 a_2$ exist. Similarly, since L_2 is a lattice, $b_1 \vee_2 b_2$ and $b_1 \wedge_2 b_2$ exist. Hence $(a_1, b_1) \vee (a_2, b_2)$ and $(a_1, b_1) \wedge (a_2, b_2)$ both exist and therefore (L, $\leq$) is a lattice, called **the direct product of (L_1, $\leq$) and (L_2, $\leq$).**

Example: Let L_1 and L_2 be the lattices whose Hasse diagram are given below :



Then $L = L_1 \times L_2$ is the lattice shown in the diagram below:



$(0_1, a)$ $(0_1, b)$

$(0_1, 0_2)$

$L = L_1 \times L_2$

1.20. Properties of Lattices

Let $(L, \leq)$ be a lattice and let $a, b, c \in L$. Then, from the definition of $\vee$ (join) and $\wedge$ (meet) we have

(i) $a \leq a \vee b$ and $b \leq a \vee b$; $a \vee b$ is an upper bound of a and b.
(ii) if $a \leq c$ and $b \leq c$, then $a \vee b \leq c$; $a \vee b$ is the least bound of a and b.
(iii) $a \wedge b \leq a$ and $a \wedge b \leq b$; $a \wedge b$ is a lower bound of a and b.

(iv) if $c \leq a$ and $c \leq b$, then $c \leq a \wedge b$; $a \wedge b$ is the greatest lower bound of a and b

Theorem: Let L be a lattice. Then for every a and b in L,

(i) $a \vee b = b$ if and only if $a \leq b$

(ii) $a \wedge b = a$ if and only if $a \leq b$

(iii) $a \wedge b = a$ if and only if $a \vee b = b$

Proof: (i) Let $a \vee b = b$. Since $a \leq a \vee b$, we have $a \leq b$.

Conversely, if $a \leq b$, then since $b \leq b$, it follows that b is an upper bound of a and b. Therefore, by the definition of least upper bound, $a \vee b \leq b$. Also $a \vee b$ being an upper bound, $b \leq a \vee b$. Hence $a \vee b = b$.

(ii) Let $a \wedge b = a$. Since $a \wedge b \leq b$, we have $a \leq b$. Conversely, if $a \leq b$ and since $a \leq a$, a is a lower bound of a and b and so, by the definition of greatest lower bound, we have

$$a \leq a \wedge b$$

Since $a \wedge b$ is lower bound,

$$a \wedge b \leq a$$

Hence

$$a \wedge b = a.$$

(iii) From (ii)

$$a \wedge b = a \Leftrightarrow a \leq b \text{.....(iv)}$$

From (i)

$$a \leq b \Leftrightarrow a \vee b = b \text{.....(v)}$$

Hence, combining (iv) and (v), we have

$$a \wedge b = a \Leftrightarrow a \vee b = b.$$

Example: Let L be a linearly (total) ordered set. Therefore a, b $\in$ L imply either $a \leq b$ or $b \leq a$. Therefore, the above theorem implies that

$$a \vee b = a$$

$$a \wedge b = a$$

Thus for every pair of elements a, b in L, $a \vee b$ and $a \wedge b$ exist. Hence **a linearly ordered set is a lattice.**

Theorem : Let $(L, \leq)$ be a lattice and let a, b, c $\in$ L. Then we have

L_1 : **Idempotent property**

(i) $a \vee a = a$
(ii) $a \wedge a = a$

L_2 : **Commutative property**

(i) $a \vee b = b \vee a$
(ii) $a \wedge b = b \wedge a$

L_3 : **Associative property**

(i) $a \vee (b \vee c) = (a \vee b) \vee c$
(ii) $a \wedge (b \wedge c) = (a \wedge b) \wedge c$

L_4 : **Absorption property**

(i) $a \vee (a \wedge b) = a$

(ii) $a \wedge (a \vee b) = a$

Proof: L_1 : The idempotent property follows from the definition of LUB and GLB.

L_2 : Commutativity follows from the symmetry of a and b in the definition of LUB and GLB.

L_3 : (i) From the definition of LUB, we have

$$a \leq a \vee (b \vee c) \qquad (1)$$

$$b \vee c \leq a \vee (b \vee c) \qquad (2)$$

Also $b \leq b \vee c$ and $c \leq b \vee c$ and so transitivity implies

$$b \leq a \vee (b \vee c) \qquad (3)$$

and

$$c \leq a \vee (b \vee c) \qquad (4)$$

Now, (1) and (3) imply that $a \vee (b \vee c)$ is an upper bound of a and b and hence by the definition of least upper bound, we have

$$a \vee b \leq a \vee (b \vee c) \qquad (5)$$

Also by (4) and (5), $a \vee (b \vee c)$ is an upper bound of c and $a \vee b$. Therefore

$$(a \vee b) \vee c \leq a \vee (b \vee c) \qquad (6)$$

Similarly

$$a \vee (b \vee c) \leq (a \vee b) \vee c \qquad (7)$$

Hence, by antisymmetry of the relation $\leq$, (6) and (7) yield

$$a \vee (b \vee c) = (a \vee b) \vee c$$

The proof of (ii) is analogous to the proof of part (i).

L_4 : (i) Since $a \wedge b \leq a$ and $a \leq a$, it follows that a is an upper bound of $a \wedge b$ and a. Therefore, by the definition of least upper bound

$$a \vee (a \wedge b) \leq a \qquad (8)$$

On the other hand, by the definition of LUB, we have

$$a \leq a \vee (a \wedge b) \qquad (9)$$

The expression (8) and (9) yields

$$a \vee (a \wedge b) = a.$$

(ii) Since $a \leq a \vee b$ and $a \leq a$, it follows that a is a lower bound of $a \vee b$ and a. Therefore, by the definition of GLB,

$$a \leq a \wedge (a \vee b) \qquad (10)$$

Also, by the definition of GLB, we have

$$a \wedge (a \vee b) \leq a \qquad (11)$$

Then (10) and (11) imply

$$a \wedge (a \vee b) = a$$

and the proof is completed.

In view of L_3, we can write $a \vee (b \vee c)$ and $(a \vee b) \vee c$ as $a \vee b \vee c$. Thus, we can express

LUB $(\{a_1, a_2, \ldots a_n)$ as $a_1 \vee a_2 \vee \ldots\ldots \vee a_n$

GLB $(\{a_1, a_2, \ldots a_n)$ as $a_1 \wedge a_2 \wedge \ldots\ldots \wedge a_n$

Remark: Using commutativity and absorption property, part (ii) of previous Theorem can be proved as follows :

Let $a \wedge b = a$. We note that

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

But

$$b \vee (a \wedge b) = b \quad \text{(Absorption property)}$$

Hence

$$a \vee b = b$$

and so by part (i), $a \leq b$. Hence $a \wedge b = a$ if and only if $a \leq b$.

Theorem: Let $(L, \leq)$ be a lattice. Then for any $a, b, c \in L$, the following properties hold :

1. (**Isotonicity) :** If $a \leq b$, then

   (i) $a \vee c \leq b \vee c$

   (ii) $a \wedge c \leq b \wedge c$

This property is called "Isotonicity".

2. $a \leq c$ and $b \leq c$ if and only if $a \vee b \leq c$

3. $c \leq a$ and $c \leq b$ if and only if $c \leq a \wedge b$

4. If $a \leq b$ and $c \leq d$, then

   (i) $a \vee c \leq b \vee d$

   (ii) $a \wedge c \leq b \wedge d$.

Proof : 1 (i). We know that

$a \vee b = b$ if and only if $a \leq b$.

Therefore, to show that $a \vee c \leq b \vee c$, we shall show that

$(a \vee c) \vee (b \vee c) = b \vee c$.

We note that

$$(a \vee c) \vee (b \vee c) = [(a \vee c) \vee b] \vee c$$
$$= a \vee (c \vee b) \vee c$$
$$= a \vee (b \vee c) \vee c$$
$$= (a \vee b) \vee (b \vee c)$$
$$= b \vee c \quad (\because a \vee b = b \text{ and } c \vee c = c)$$

The part 1 (ii) can be proved similarly.

2. If $a \leq c$, then 1(i) implies

$$a \vee b \leq c \vee b$$

But

$$b \leq c \Leftrightarrow b \vee c = c$$

$$\Leftrightarrow c \vee b = c \quad \text{(commutativity)}$$

Hence $a \leq c$ and $b \leq c$ if and only if $a \vee b \leq c$

3. If $c \leq a$, then 1(ii) implies

$$c \wedge b \leq a \wedge b$$

But

$$c \leq b \Leftrightarrow c \wedge b = c$$

Hence $c \leq a$ and $c \leq b$ if and only if $c \leq a \wedge b$.

4 (i) We note that 1(i) implies that

if $a \leq b$, then $a \vee c \leq b \vee c = c \vee b$

if $c \leq d$, then $c \vee b \leq d \vee b = b \vee d$

Hence, by transitivity

$$a \vee c \leq b \vee d$$

(ii) We note that 1(ii) implies that

if $a \leq b$, then $a \wedge c \leq b \wedge c = c \wedge b$

if $c \leq d$, then $c \wedge b \leq d \wedge b = b \wedge d$.

Therefore transitivity implies

$$a \wedge c \leq b \wedge d.$$

Theorem: Let $(L, \leq)$ be a lattice. If $a, b, c \in L$, then

$$(1)\ a \vee (b \wedge c) \leq (a \vee b) \wedge (a \vee c)$$

$$(2)\ a \wedge (b \vee c) \geq (a \wedge b) \vee (a \wedge c)$$

These inequalities are called "Distributive Inequalities".

Proof: We have

$$a \leq a \vee b \quad \text{and} \quad a \leq a \vee c \qquad \text{(i)}$$

Also, by the above theorem, if $x \leq y$ and $x \leq z$ in a lattice, then $x \leq y \wedge z$. Therefore (i) yields

$$a \le (a \vee b) \wedge (a \vee c) \qquad \text{(ii)}$$

Also

$$b \wedge c \le b \le a \vee b$$

and

$$b \wedge c \le c \le a \vee c \ ,$$

that is, $b \wedge c \le a \vee b$ and $b \wedge c \le a \vee c$ and so, by the above argument, we have

$$b \wedge c \le (a \vee b) \wedge (a \vee c) \quad \text{(iii)}$$

Also, again by the above theorem if $x \le z$ and $y \le z$ in a lattice, then

$$x \vee y \le z$$

Hence, (ii) and (iii) yield

$$a\, c\, (b \wedge c) \le (a \vee b) \wedge (a \vee c)$$

This proves (1).

The second distributive inequality follows by using the **principle of duality.**

Theorem: (Modular Inequality) : Let $(L, \le)$ be a lattice. If $a, b, c \in L$, then

$$a \le c \text{ if and only if } a \vee (b \wedge c) \le (a \vee b) \wedge c$$

Proof: We know that

$$a \le c \Leftrightarrow a \vee c = c \qquad (1)$$

Also, by distributive inequality,

$$a \vee (b \wedge c) \le (a \vee b) \wedge (a \vee c)$$

Therefore using (1) $a \le c$ if and only if

$$a \vee (b \wedge c) \le (a \vee c) \wedge c,$$

which proves the result.

The modular inequalities can be expressed in the following way also:

$$(a \wedge b) \vee (a \wedge c) \le a \wedge [b \vee (a \wedge c)]$$

$$(a \vee b) \wedge (a \vee c) \ge a \vee [b \wedge (a \vee c)]$$

Example: Let $(L, \le)$ be a lattice and $a, b, c \in L$. If $a \le b \le c$, then

(i) $a \vee b = b \wedge c$, (ii) $(a \wedge b) \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

Solution: (i) We know that

$$a \le b \Leftrightarrow a \vee b = b$$

and

$$b \leq c \Leftrightarrow b \wedge c = b$$

Hence $a \leq b \leq c$ implies

$$a \vee b = b \wedge c.$$

(ii) Since $a \leq b$ and $b \leq c$, we have

$$a \wedge b = a \text{ and } b \wedge c = b$$

Thus

$$(a \wedge b) \vee (b \wedge c) = a \vee b$$

$$= b, \text{ since } a \leq b \Leftrightarrow a \vee b = b.$$

Also, $a \leq b \leq c \Rightarrow a \leq c$ by transitivity. Then

$$a \leq b \text{ and } a \leq c \Rightarrow a \vee b = b, \; a \vee c = c$$

and so

$$(a \vee b) \wedge (a \vee c) = b \wedge c$$

$$= b \text{ since } b \leq c \Leftrightarrow b \wedge c = b.$$

Hence

$$(a \wedge b) \vee (b \wedge c) = b = (a \vee b) \wedge (a \vee c),$$

which proves (ii).

1.21. Lattices as Algebraic System

Definition. A **Lattice** is an algebraic system $(L, \vee, \wedge)$ with two binary operations $\vee$ and $\wedge$, called **join** and **meet** respectively, on a non-empty set L which satisfy the following axioms for $a, b, c \in L$:

1. Commutative Law :

$$a \vee b = b \vee a \quad \text{and} \quad a \wedge b = b \wedge a.$$

2. Associative Law :

$$(a \vee b) \vee c = a \vee (b \vee c)$$

and

$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$

3. Absorption Law :

(i) $a \vee (a \wedge b) = a$

(ii) $a \wedge (a \vee b) = a$

We note that Idempotent Law follows from axiom 3 above. In fact,

$$a \vee a = a \vee [a \wedge (a \vee b)] \quad \text{using 3(ii)}$$

$$= a \quad \text{using 3(i)}$$

The proof of $a \wedge a = a$ follows by principle of duality.

1.22 Partial Order Relations on a Lattice

A partial order relation on a lattice (L) follows as a consequence of the axioms for the binary operations $\vee$ and $\wedge$.

We define a relation $\leq$ on L such that for $a, b \in L$,

$$a \leq b \Leftrightarrow a \vee b = b$$

or analogously,

$$a \leq b \Leftrightarrow a \wedge b = a .$$

We note that

(i) For any $a \in L$

$$a \vee a = a \quad \text{(idempotent law),}$$

therefore $a \leq a$ showing that $\leq$ is **reflexive**.

(ii) Let $a \leq b$ and $b \leq a$. Therefore

$$a \vee b = b$$

$$b \vee a = a$$

But

$$a \vee b = b \vee a \text{ (Commutative Law in lattice)}$$

Hence

$$a = b ,$$

showing that $\leq$ is **antisymmetric.**

(iii) Suppose that $a \leq b$ and $b \leq c$. Therefore $a \vee b = b$ and $b \vee c = c$. Then

$$a \vee c = a \vee (b \vee c)$$

$$= (a \vee b) \vee c \quad \text{(Associativity in lattice)}$$

$$= b \vee c$$

$$= c ,$$

showing that $a \leq c$ and hence $\leq$ is transitive.

This shows that a **lattice is a partially ordered set**

1.23 Least Upper Bounds and Latest Lower Bounds in a Lattice

Let $(L, \vee, \wedge)$ be a lattice and let $a, b \in L$. We now show that LUB of $\{a, b\} \subseteq L$ with respect to the partial order introduced above is $a \vee b$ and GLB of $\{a, b\}$ is $a \wedge b$.

From absorption law

$$a \wedge (a \vee b) = a$$

$$b \wedge (a \vee b) = b$$

Therefore $a \leq a \vee b$ and $b \leq a \vee b$, showing that $a \vee b$ is upper bound for $\{a, b\}$. Suppose that there exists $c \in L$ such that $a \leq c$, $b \leq c$. Thus we have

$$a \vee c = c \text{ and } b \vee c = c$$

and then

$$(a \vee b) \vee c = a \vee (b \vee c) = a \vee c = c \ ,$$

implying that $a \vee b \leq c$. Hence $a \vee b$ is the least upper bound of a and b.

Similarly, we can show that $a \wedge b$ is GLB of a and b.

The above discussion shows that the two definitions of lattice given so far are equivalent.

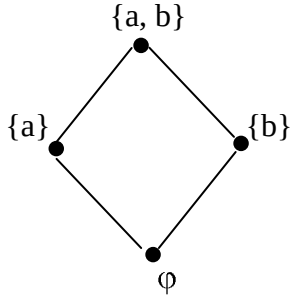
Example: Let $\hat{C}$ be collection of sets with binary operations Union and Intersection of sets. Then $(\hat{C}, \cup, \cap)$ is a lattice. In this lattice, the partial order relation is **set inclusion.** In fact, for $A, B \in \hat{C}$,

$A \subseteq B$ iff $A \cup B = B$

Or

$$A \subseteq B \text{ iff } A \cap B = A.$$

For example, the diagram of lattice of subsets of {a, b} is



1.24. Sublattices

Definition: Let (L, $\leq$) be a lattice. A non-empty subset S of L is called a **sublattice** of L if a $\vee$ b $\in$ S and a $\wedge$ b $\in$ S whenever a $\in$ S, b $\in$ S.

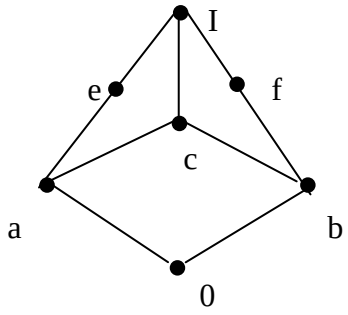
Or

Let (L, $\vee$, $\wedge$) be a lattice and let S $\subseteq$ L be a subset of L. Then (S, $\vee$, $\wedge$) is called a sublattice of (L, $\vee$, $\wedge$) if and only if S is closed under both operations of join($\vee$) and meet($\wedge$).

From the definition it is clear that **sublattice itself is a lattice.**

However, **any subset of L which is a lattice need not be a sublattice.**

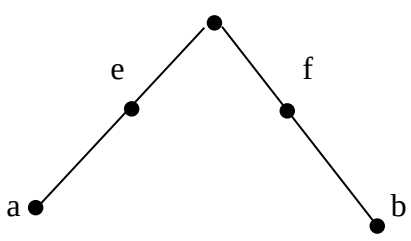
For example, consider the lattice shown in the diagram:



L

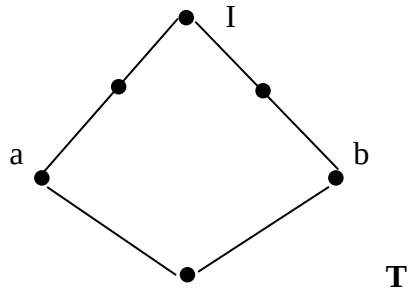
We note that

(i) the subset S shown by the diagram below is not a sublattice of L, since a $\wedge$ b $\notin$ S and a $\vee$ b $\notin$ S.



S

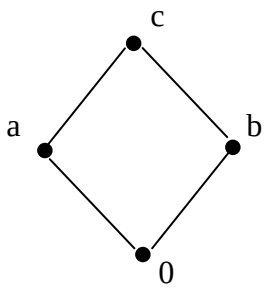
(ii) the set T shown below is not a sublattice of L since a $\vee$ b $\notin$ T.



T

However, T is a lattice when considered as a poset by itself.

(iii) the subset $\cup$ of L shown below is a sublattice of L:



U

Example: Let A be any set and P(A) its power set. Then (P(A), $\vee$, $\wedge$) is a lattice in which join and meet are union of sets and intersection of sets respectively.

A family $\hat{C}$ of subsets of A such that S $\cup$ T and S $\cap$ T are in $\hat{C}$ for S, T $\in \hat{C}$ is a sublattice of (P(A), $\vee$, $\wedge$). **Such a family $\hat{C}$ is called a ring of subsets of A and is denoted by (R(A), $\vee$, $\wedge$)** (This is not a ring in the sense of algebra). **Some author call it lattice of subsets.**

Example: The lattice (D_n, $\leq$) is a sublattice of (**N**, $\leq$), where $\leq$ is the relation of divisibility.

1.25 Lattice Isomorphism

Definition: Let (L_1, $\vee_1$, $\wedge_1$) and (L_2, $\vee_2$, $\wedge_2$) be two lattices. A mapping f : $L_1 \to L_2$ is called a **lattice homomorphism** from the lattice the lattice (L_1, $\vee_1$, $\wedge_1$) to (L_2, $\vee_2$, $\wedge_2$) if for any a, b $\in L_1$,

$$f(a \vee_1 b) = f(a) \vee_2 f(b) \text{ and } f(a \wedge_1 b) = f(a) \wedge_2 f(b)$$

Thus, here both the binary operations of join and meet are preserved. **There may be mapping which preserve only one of the two operations. Such mapping are not lattice homomorphism.**

Let $\leq_1$ and $\leq_2$ be partial order relations on (L_1, $\vee_1$, $\wedge_1$) and (L_2, $\vee_2$, $\wedge_2$) respectively. Let f : $L_1 \to L_2$ be lattice homomorphism. If a, b $\in L_1$, then

$$a \leq_1 b \Leftrightarrow a \vee_1 b = b$$

and so

$$f(b) = f(a \vee_1 b)$$
$$= f(a) \vee_2 f(b)$$
$$\Leftrightarrow f(a) \leq_2 f(b)$$

Thus

$$a \leq_1 b \Leftrightarrow f(a) \leq_2 f(b)$$

Thus order **relations are also preserved** under lattice homomorphism.

If a lattice homomorphism f: $L_1 \rightarrow L_2$ is one-to-one and onto, then it is called **lattice isomorphism.**
If there exists an isomorphism between two lattices, then the lattices are called **isomorphic**.

Since lattice isomorphism preserves order relation, therefore isomorphic lattices can be represented by the same diagram in which nodes are replaced by images.

Theorem: Let $A = \{a_1, a_2,\ldots,a_n\}$ and $B = \{b_1, b_2,\ldots\ldots b_n\}$ be any two finite sets with n elements. Then the lattices $(P(A), \subseteq)$ and $(P(B), \subseteq)$ are isomorphic and so have identical Hasse-diagram.

Proof: Consider the mapping

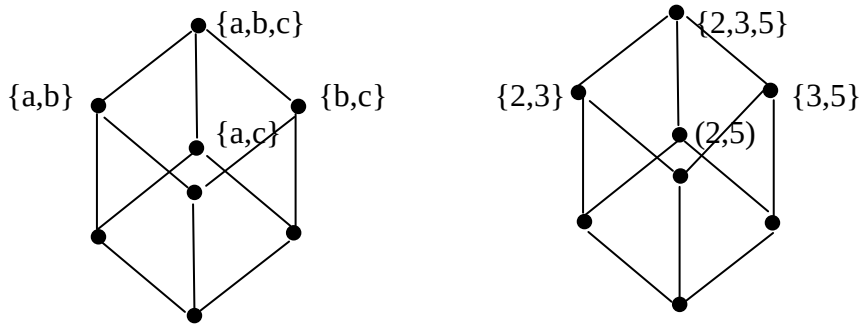
$$f : P(A) \rightarrow P(B)$$

defined by

$$f(\{a_n\} = \{b_n\}, f(\{a_1, a_2,\ldots,a_m\}) = \{b_1, b_2,\ldots\ldots b_n\} \text{ for } m \leq n .$$

Then f is bijective mapping and $L \subseteq M \Leftrightarrow f(L) \subseteq f(M)$ for subsets L and M of P(A). Hence P(A) and P(B) are isomorphic.

For example, let A = {a, b, c}, B = {2, 3, 5}. The Hasse-diagram of P(A) and P(B) are then given below:



{b} {3}

{a} {c} {2} {5}

φ φ

Define a mapping f : P(A) → P(B) by

$f(\phi) = \varphi$, f({a}) = {2}, f({b}) = {3}, f({c}) = {5}

f({a, b}) = {2, 3}, f({b, c}) = {3, 5}, f({a, c}) = {2, 5}

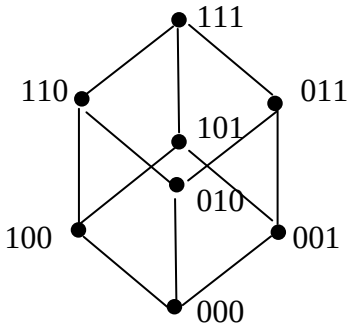
and

f({a, b, c}) = {2, 3, 5}.

This is a bijective mapping satisfying the condition that if S and T are subsets of A, then $S \subseteq T$ if and only if $f(S) \subseteq f(T)$. Hence f is isomorphism and (P(A), $\subseteq$) and (P(B), $\subseteq$) are isomorphic.

Thus, for each n = 0, 1, 2,…., there is only one type of lattice and this lattice depends only on n, the number of elements in the set A, and not on A. It has 2^n elements. Also, we know that if A has n elements, then all subsets of A can be represented by sequences of 0's and 1's of length n. We can therefore label the Hasse diagram of a lattice (P(A), $\subseteq$) by such sequence of 0's and 1's.

For example, lattices of P(A) and P(B) of the last example can be labeled as below:



The lattice so obtained is named B_n. The properties of the partial order in B_n can be described directly as follows:

Let $x = a_1\, a_2 \ldots a_n$ and $y = b_1\, b_2 \ldots b_n$ be any two elements of B_n. Then

(1) $x \leq y$ if and only if $a_k < b_k$, k = 1, 2,…..,n, where a_k and b_k are 0 or 1.

(2) $x \wedge y = c_1\, c_2 \ldots c_n$, where $c_k = \min(a_k, b_k)$.
(3) $x \vee y = d_1\, d_2 \ldots d_n$, where $d_k = \max(a_k, h_k)$.

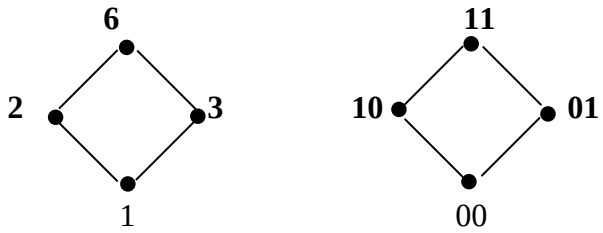
(4) x has a complement $x' = z_1\, z_2 \ldots\ldots z_n$ where $z_k = 1$ if $x_k = 0$ and $z_k = 0$ if $x_k = 1$.

Remark: **$(B_n, \leq)$ under the partial order $\leq$ defined above is isomorphic to $(P(A), \subseteq)$, when A has n elements. In such a case $x \leq y$ corresponds to $S \subseteq T$, $x \vee y$ corresponds to $S \cup T$ and x' corresponds to A^c.**

Example : **Let $D_6 = \{1, 2, 3, 6\}$, set of divisors of 6. Then D_6 is isomorphic to B_2. In fact $f : D_6 \to B_2$ defined by**

$f(1) = 00, f(2) = 10, f(3) = 01, f(6) = 11$

is an isomorphism.



D_6 B_2

Example: Let $A = \{a, b\}$ and $P(A) = \{\varphi, \{a\}, \{a, b\}\}$ then the lattice $(P(A), \subseteq)$ is isomorphic to the lattice $(D_6, 1)$ with divisibility as the partial order relation. In fact, we define a mapping $f : D_6 \to P(A)$ by

$$f(1) = \varphi, f(2) = \{a\}, f(3) = \{b\}, f(6) = \{a, b\},$$

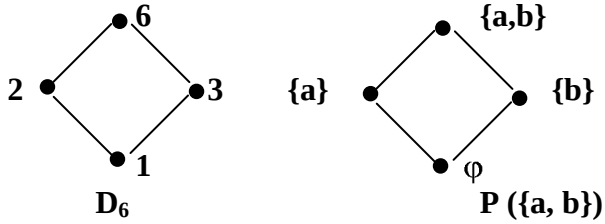
then f is bijective and we note that

$$1|2 \Leftrightarrow \{\varphi\} \subseteq \{a\} \Leftrightarrow f(1) \subseteq f(2)$$

$$2|6 \Leftrightarrow \{a\} \subseteq \{a, b\} \Leftrightarrow f(2) \subseteq f(6)$$

and so on.

Hence f is isomorphism.



D_6 **P ({a, b})**

Definition: **Let $(L, \vee, \wedge)$ be a lattice. Then lattice homomorphism $f : L \to L$ is called an** endomorphism.

Definition: **Let $(L, \vee, \wedge)$ be lattice. Then the lattice isomorphism $f: L \to L$ is called an** automorphism.

If f : L $\rightarrow$ L is an endomorphism, then the image set of f is sublattice of L.

Definition: **Let (A, $\leq$) and (B, $\leq'$) be two partially ordered sets. A mapping f : A $\rightarrow$ B is called order preserving relative to the ordering $\leq$ in A and $\leq'$ in B iff for a, b $\in$ A,**

$$\mathbf{a \leq b \Rightarrow f(a) \leq' f(b)}$$

If A and B are lattices and f : A $\rightarrow$ B is a lattice homomorphism, then f is order preserving.

Definition: **Two partially ordered sets (A, $\leq$) and (B, $\leq'$) are said to be** order isomorphic **if there exists a mapping f : A $\rightarrow$ B which is bijective and both f and f^{-1} are order preserving.**

For lattices (A, $\leq$) and (B, $\leq'$), an order isomorphism is equivalent to lattice isomorphism. Hence lattices which are order-isomorphic as partially ordered sets are isomorphic.

Let (L, $\vee$, $\wedge$) be a lattice and let S = {a_1, a_2….,a_n} be a finite subset of L. Then

LUB of S is represented by $a_1 \vee a_2 \vee \ldots\ldots \vee a_n$

GLB of S is represented by $a_1 \wedge a_2 \wedge \ldots\ldots \wedge a_n$

Definition: **A lattice is called** complete **if each of its non-empty subsets has a least upper bound and a greatest lower bound.**

Obviously, every finite lattice is complete.

Also every complete lattice must have a least element, denoted by 0 and a greatest element, denoted by I. The least and greatest elements if exist are called bound (units, universal bounds) **of the lattice.**

1.26 Bounded, Complemented and Distributive Lattices

Definition: **A lattice L is said to be** bounded **if it has a greatest element I and a least element 0.**

For the lattice (L, $\vee$, $\wedge$) with L = {a_1, a_2,….,a_n},

$$\mathbf{a_1 \vee a_2 \vee \ldots\ldots \vee a_n = I \text{ and } a_1 \wedge a_2 \wedge \ldots\ldots \wedge a_n = 0}\ .$$

Example : **The lattice Z^+ of all positive integers under partial order of divisibility** is not a bounded lattice **since it has a least element (the integer 1) but no** greatest element.

Example: **The lattice** Z **of integers under partial order** $\leq$ **(less than or equal to) is** not bounded since it has neither a greatest element nor a least element.

Example: **Let A be a non-empty set. Then the lattice (P(A),** $\subseteq$**)** is bounded. **Its greatest element is A and the least element is empty set** ϕ**.**

If (L, $\leq$**) is a bounded Lattice, then for all a** $\in$ **L**

$$0 \leq a \leq I$$

$$a \vee 0 = a,\ a \wedge 0 = 0$$

$$a \vee I = I,\ a \wedge I = a$$

Thus 0 acts as identity of the operation $\vee$ **and I acts as identity of the operation** $\wedge$**.**

Definition: **Let (L** $\vee$, $\wedge$, **0, I) be a bounded lattice with greatest element I and the least element 0. Let a** $\in$ **L. Then an element b** $\in$ **L is called a** complement **of a if**

$$a \vee b = I \text{ and } a \wedge b = 0$$

It follows from this definition that

0 and I are complement of each other.

Further, I is the only complement of 0. For suppose that c $\neq$ **I is a complement of 0 and c** $\in$ **L, then**

$$0 \vee c = I \text{ and } 0 \wedge c = 0$$

But 0 $\vee$ **c = c. Therefore c = I which contradicts c** $\neq$ **I.**

Similarly, 0 is the only complement of I.

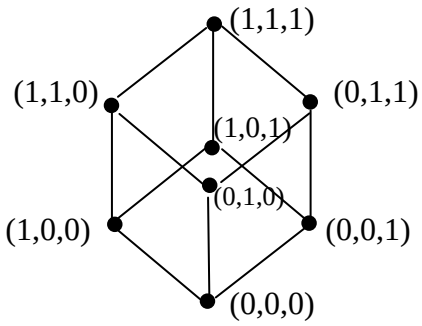
Definition: **A lattice (L,** $\vee$, $\wedge$**, 1, 0) is called** complemented **if it is bounded and if every element of L has at least one complement.**

Example: **The lattice (P(A),** $\subseteq$**) of the power set of any set A is a bounded lattice, where meet and join operations on e(A) are** $\cap$ **and** $\cup$ **respectively. Its bounds are** φ **and A. The lattice (P(A),** $\subseteq$**) is complemented in which the complement of any subset B of A is A – b.**

Example: **Let** L^n **be the lattice of n tuples of 0 and 1, where partial ordering is defined for** $a = (a_1, a_2, \ldots, a_n)$ **,** $b = (b_1, b_2, \ldots, b_n) \in L^n$ **by**

$$a \leq_n b \Leftrightarrow a_i \leq b_i \qquad \text{for all } i = 1, 2, \ldots, n ,$$

where $\leq$ means less than or equal to. Then $(L^n, \leq_n)$ is lattice which is bounded. For example, the bounds are (0, 0, 0) and (1, 1, 1) for L^3.



The complement of an element of L^n can be obtained by interchanging 1 by 0 and 0 by 1 in the n-tuple representing the element. For example, complement of (1, 0, 1) in L^3 is (0, 1, 0).

Definition: **A lattice $(L, \vee, \wedge)$ is called a** distributive lattice **if for any elements a, b and c in L,**

$$\textbf{(1) } a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$\textbf{(2) } a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Properties (1) and (2) are called distributive properties.

Thus, in a distributive lattice, the operations $\wedge$ and $\vee$ are distributive over each other.

We further note that, by the principle of duality, the condition (1) holds if and only if (2) holds. Therefore it is sufficient to verify any one of these two equalities for all possible combinations of the elements of a lattice.

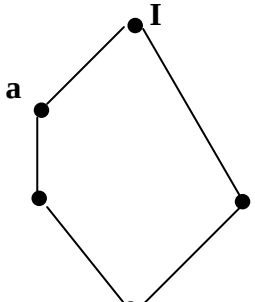
If a lattice L is not distributive, we say that L is non-distributive.

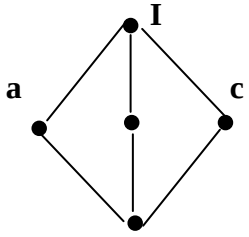
Example: **For a set S, the lattice $(P(S), \subseteq)$ is distributive. The meet and join operation in P(S) are $\cap$ and $\cup$ respectively. Also we know, by set theory, that for A, B, C $\in$ P(S),**

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

Example: **The** five elements **lattices given in the following diagrams are** non distributive.





b

b **c**

o

(ii)

o **(i)**

In fact for the lattice (i), we note that

$$\mathbf{a \wedge (b \vee c) = a \wedge I = a ,}$$

while

$$\mathbf{(a \wedge b) \vee (a \wedge c) = b \vee 0 = b}$$

Hence

$$\mathbf{a \wedge (b \vee c) \neq (a \wedge b) \vee (a \wedge c) ,}$$

showing that (i) is non-distributive.

For the lattice (ii) , we have

$$\mathbf{a \wedge (b \vee c) = a \wedge I = a ,}$$

while

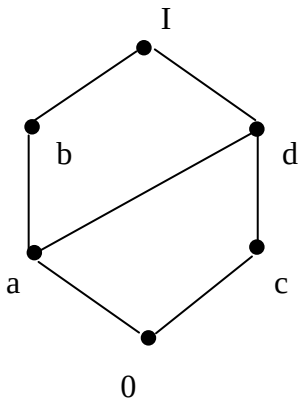
$$\mathbf{(a \wedge b) \vee (a \wedge c) = 0 \vee 0 = 0 .}$$

Hence

$$\mathbf{a \wedge (b \vee c) \neq (a \wedge b) \vee (a \wedge c) ,}$$

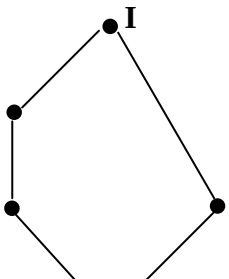
showing that (ii) is also non-distributive

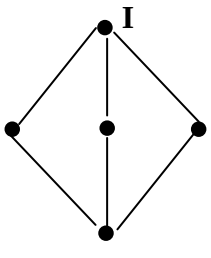
Example: **The lattice shown in the diagram below is** distributive:



The distributive properties are satisfied for any ordered triplet chosen from the given elements.

Theorem: **A lattice L is** non distributive **if and only if it contains a sublattice isomorphic to any one of the following two five-element lattices:**





a **a** **c**

b

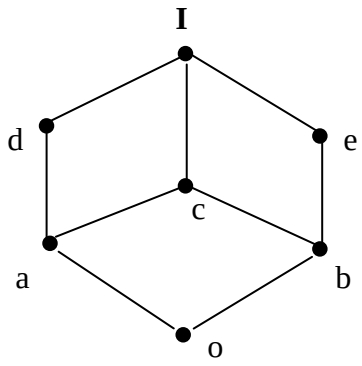
b **c**

o

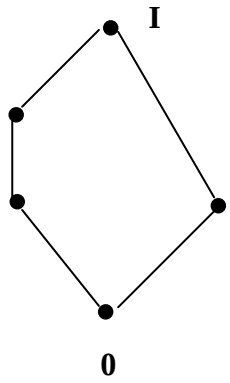
o

(The Proof of this theorem is out of the scope of this book)

Example: **Is the following lattice a distributive lattice ?**



Solution: **The given lattice is** not distributive **since {0, a, d, e, I} is a sublattice which is isomorphic to the five-element lattice shown below :**



Theorem: **Every chain is a distributive lattice.**

Proof: **Let (L, ≤) be a chain and a, b, c ∈ L. We shall show that distributive law holds for any a, b, c ∈ L. Two cases arise :**

Case 1. **Let a ≤ b or a ≤ c. In this case**

$$a \wedge (b \vee c) = a$$

and

$$(a \wedge b) \vee (a \wedge c) = a$$

and hence

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

Also, by Principle of Duality

$$a \wedge (b \vee c) = (a \vee b) \wedge (a \vee c)$$

Case II. **Let $b \leq a$ or $c \leq a$. Then we have**

$$a \wedge (b \vee c) = (b \vee c)$$

and

$$(a \wedge b) \vee (a \wedge c) = (b \vee c)$$

Hence

$$a \wedge (b \vee c) = (b \vee c)$$

Hence distributive law holds for any $a, b, c \in L$.

Theorem: **The direct product of any two distributive lattices is a distributive lattice.**

Proof: **Let $(L_1, \leq_1)$ and $(L_2, \leq_2)$ be two lattices in which meet and join are $\wedge_1$, $\vee_1$ and $\wedge_2$, $\vee_2$ respectively. Then meet and join in $L_1 \times L_2$ are defined by**

$$(a_1, b_1) \wedge (a_2, b_2) = (a_1 \wedge_1 a_2, b_1 \wedge_2 b_2) \qquad (1)$$

and

$$(a_1, b_1) \vee (a_2, b_2) = (a_1 \vee_1 a_2, b_1 \vee_2 b_2) \qquad (2)$$

Since L_1 is distributive,

$$a_1 \wedge_1 (a_2 \vee_1 a_3) = (a_1 \wedge_1 a_2) \vee_1 (a_1 \wedge_1 a_3) \qquad (3)$$

Since L_2 is distributive,

$$b_1 \wedge_2 (b_2 \vee_2 b_3) = (b_1 \wedge_2 b_2) \vee_2 (b_1 \wedge_2 b_3) \qquad (4)$$

Therefore

$$(a_1, b_1) \wedge [(a_2, b_2) \vee (a_3, b_3)]$$

$$= (a_1, b_1) \wedge [(a_2 \vee_1 a_3, b_2 \vee_2 b_3)]$$

$$= [(a_1 \wedge_1 (a_2 \vee_1 a_3), b_1 \wedge_2 (b_2 \vee_2 b_3)]$$

$$= [(a_1 \wedge_1 a_2) \vee_1 (a_1 \wedge_1 a_3), (b_1 \wedge_2 b_2) \vee_2 (b_1 \wedge_2 b_3)]$$

(using (3) and (4))

and using (1) and (2), we have

$$[(a_1, b_1) \wedge (a_2, b_2)] \vee [(a_1, b_1) \wedge (a_3, b_3)]$$

$$= (a_1 \wedge_1 a_2, b_1 \wedge_2 b_2) \vee (a_1 \wedge_1 a_3, b_1 \wedge_2 b_3)$$

$$= [(a_1 \wedge_1 a_2) \vee_1 (a_1 \wedge_1 a_3), (b_1 \wedge_2 b_2) \vee_2 (b_1 \wedge_2 b_3)]$$

Hence

$$(a_1, b_1) \wedge [(a_2, b_2) \vee (a_3, b_3)] = [(a_1, b_1) \wedge (a_2, b_2)] \vee [(a_1, b_1) \wedge (a_3, b_3)],$$

proving that $L_1 \times L_2$ is distributive.

Theorem: **Let L be a bounded distributive lattice. If a complement of any element exists, it is unique.**

Proof: **Suppose on the contrary that b and c are complements of the element $a \in L$. Then**

$$a \vee b = I \qquad a \vee c = I$$

$$a \wedge b = 0 \qquad a \wedge c = 0$$

Using distributive law, we have

$$b = b \vee 0$$
$$= b \vee (a \wedge c)$$
$$= (b \vee a) \wedge (b \vee c)$$
$$= (a \vee b) \wedge (b \vee c)$$
$$= I \wedge (b \vee c)$$
$$= b \vee c$$

Similarly,

$$c = c \vee 0$$
$$= c \vee (a \wedge b)$$
$$= (c \vee a) \wedge (c \vee b)$$
$$= (a \vee c) \wedge (c \vee b)$$
$$= I \wedge (c \vee b)$$
$$= I \wedge (b \vee c)$$
$$= b \vee c$$

Hence b = c.

Definition: **Let (L, $\wedge$, $\vee$) be a lattice. An element a $\in$ L is said to be** join-irreducible **if it cannot be expressed as the join of two distinct elements of L.**

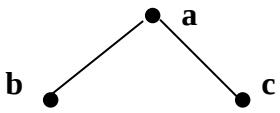
In other words, a $\in$ L is join-irreducible if for any b, c $\in$ L

$a = b \vee c \Rightarrow a = b$ or $a = c$.

For example, prime number under multiplication have this property. In fact if p is a prime number, then $p = a\,b \Rightarrow p\,a$ or $p = b$.

Clearly 0 is join – irreducible.

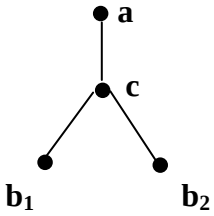
Further, if a has at least two immediate predecessors, **say b and c as in the diagram below:**



Then $a = b \vee c$ and so a is not join – irreducible.

On the other hand if a has a unique immediate predecessor c, then

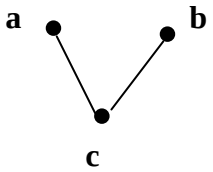
$a \neq \sup(b_1, b_2) = b_1 \vee b_2$ for any other elements b_1 and b_2 because c would lie between b_1, b_2 and a.



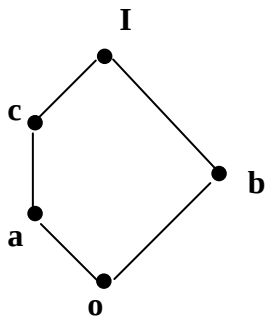
In other words, a $\neq$ 0 is join irreducible if and only if a has a unique predecessor.

Definition: **Those elements, which immediately succeed 0, are called** atoms.

From the above discussion, it follows that the atoms are join-irreducible**.**



However, lattices can have other join-irreducible elements. For example, the element c in five-element lattice is not an atom, even then it is join irreducible because it has only one immediate predecessor, namely a.



Let a be an element of a finite lattice which is not join irreducible, then we can write

$$a = b \vee c$$

If b and c are not join irreducible, then we can write them as the join of other elements. Since L is finite we shall finally have

$$a = d_1 \vee d_2 \vee d_3 \vee \ldots\ldots \vee d_n \ , \qquad (1)$$

where d_i, i = 1, 2, …,n are join-irreducible. If d_i precedes d_j, then $d_i \vee d_j = d_j$, so we delete d_i from the expression. Thus d's are irredundant, i.e., no d precedes any other d.

The expression (1) need not be unique. **For example, in lattice shown above**

$$I = a \vee b \text{ and } I = b \vee c \ .$$

Theorem: **Let (L, $\wedge$, $\vee$) be a finite distributive lattice. Then every a in L can written uniquely (except for order) as the join of irredundant join irreducible elements.**

Proof: **Let a $\in$ L. Since L is finite, we can express a as the join of irredundant join irreducible elements (as discussed above). To prove uniqueness let**

$$a = b_1 \vee b_2 \vee \ldots.. \vee b_n = c_1 \vee c_2 \vee \ldots.. \vee c_m \ ,$$

where b_i are irredundant join-irrducible and c_i are irrdundant and join-irreducible. For any given i, we have

$$b_i \leq (b_1 \vee b_2 \vee \ldots.. \vee b_n) = c_1 \vee c_2 \vee \ldots.. \vee c_m \ ,$$

Hence

$$b_i = b_i \wedge (c_1 \vee c_2 \vee \ldots.. \vee c_m)$$

$$= (b_i \wedge c_1) \vee (b_i \wedge c_2) \vee \ldots\ldots\ldots \vee (b_i \wedge c_m)$$

Since b_i is join-irreducible, there exists j such that $b_i = b_i \wedge c_j$ and so $b_i \leq c_j$.

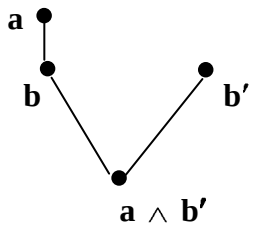
Similarly, for c_i there exists a b_k such that $c_j \leq b_k$. Hence

$$b_i \leq c_j \leq b_k \ ,$$

which gives $b_i = c_j = b_k$ since b_i are irredundant. Hence b_i and c_i may be paired off. Hence the representation for a is unique except for order.

Theorem: **Let L be a complemented lattice with unique complements. Then the join irreducible elements of L, other than 0, are its atoms.**

Proof: **Suppose a is join irreducible and is not an atom. Then a has a unique immediate predecessor $b \neq 0$. Let b' be the complement of b (complement exists since L is complemented). Since $b \neq 0$, $b' \neq I$. If a precedes b', then $b \leq a \leq b'$, and so $b \vee b' = b'$ which is impossible since $b \vee b' = I$. Thus a does not precede b' and so $a \wedge b'$ must strictly precede a. Since b is the unique immediate predecessor of a, we also have that $a \wedge b'$ precedes b. But $a \wedge b'$ precedes b'.**



Hence

$$a \wedge b' \leq \inf(b, b') = b \wedge b' = 0$$

Thus $a \wedge b' = 0$. Since $a \vee b = a$, we also have

$$a \vee b' = (a \vee b) \vee b' = a \vee (b \vee b')$$

$$= a \vee I = I$$

Therefore b' is a complement of a. Since complements are unique, a = b. This contradicts the assumption that b is an immediate predecessor of a. Thus the only join irreducible elements of L are its atoms.

Combining this result with the above-proved theorems, we have

Theorem: **Let L be a finite complemented distributive lattice. Then every element a in L is the join of a unique set of atoms.**

Unit-2
Boolean Algebra

2.1. Definitions and Examples

Definition: A non-empty set B with two binary operations $\vee$ and $\wedge$, a unary operation $'$, and two distinct elements 0 and I is called a **Boolean Algebra** if the following axioms holds for any elements a, b, c $\in$ B:

[B_1]: **Commutative Laws:**

$$a \vee b = b \vee a \quad \text{and} \quad a \wedge b = b \wedge a$$

[B_2]: **Distributive Law:**

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

[B_3]: **Identity Laws:**

$$a \vee 0 = a \quad \text{and} \quad a \wedge I = a$$

[B_4]: **Complement Laws:**

$$a \vee a' = I \quad \text{and} \quad a \wedge a' = 0$$

We shall call 0 as zero element, 1 as unit element and a' the complement of a.

We denote a Boolean Algebra by (B, $\vee$, $\wedge$, ~, 0, I).

Example 1. Let A be a non-empty set and P(A) be its power set. Then the set algebra (P(A), $\cup$, $\cap$, –, ϕ, A) is a Boolean algebra.

Example 2 : Let B = {0, 1} be the set of bits (binary digits) with the binary operations $\vee$ and $\wedge$ and the unary operation $'$ defined by the following tables:

$\vee$	1	0
0	1	0
1	1	1

$\wedge$	1	0
1	1	0
0	0	0

	$'$
1	0
0	1

Here the operations $\vee$ and $\wedge$ are logical operations and complement of 1 is 0 whereas complement of 0 is 1. Then (B, $\vee$, $\wedge$, $'$, 0, 1) is a Boolean Algebra. It is the simplest example of a two-element algebra.

Further, a two element Boolean algebra is the only Boolean algebra whose diagram is a chain.

Example 3 : Let B_n be the set of n tuples whose members are either 0 or 1. Let $a = (a_1, a_2,\ldots,a_n)$ and $b = (b_1, b_2,\ldots,b_n)$ be any two members of B_n. Then we define

$$a \vee_1 b = (a_1 \vee b_1, a_2 \vee b_2,\ldots..,a_n \vee b_n)$$

$$a \wedge_1 b = (a_1 \wedge b_1, a_2 \wedge b_2,\ldots..,a_n \wedge b_n) \ ,$$

where $\vee$ and $\wedge$ are logical operations on {0, 1}, and

$$a' = (\sim a_1, \sim a_2,\ldots, \sim a_n) \ ,$$

where $\sim 0 = 1$ and $\sim 1 = 0$.

If 0_n represents (0, 0,…..,0) and $1_n = (1, 1,\ldots\ldots,1)$, then $(B_n, \vee_1, \wedge_1, ', 0_n, 1_n)$ is a Boolean algebra.

This algebra is known as Switching Algebra and represents a switching network with n inputs and one output.

Example 4. The poset $D_{30} = \{1, 2, 3, 5, 6, 10, 15, 30\}$ has eight element. Define $\vee$, $\wedge$ and $'$ on D_{30} by

$$a \vee b = \text{lcm}(a, b) \ , \quad a \wedge b = \gcd(a, b) \qquad \text{and} \qquad a' = \frac{30}{a}.$$

Then D_{30} is a Boolean Algebra with 1 as the zero element and 30 as the unit element.

Example 5: Let S be the set of statement formulas involving n statement variables. The algebraic system (S, $\wedge$, $\vee$, ~, F, T) is a Boolean algebra in which $\wedge, \vee$, ~ denotes the operations of conjunction, disjunction and negation respectively. The element F and T denotes the formulas which are contradictions and Tautologies respectively. The partial ordering corresponding to $\wedge$, $\vee$ is implication $\Rightarrow$.

We have seen that B_n is a Boolean algebra. Using this fact, we can also define Boolean algebra as follows:

Definition: A finite lattice is called a **Boolean Algebra** if it is isomorphic with B_n for some non-negative integer n.

For example, D_{30} is isomorphic to B_3. In fact, the mapping $f: D_{30} \to B_3$ defined by

$$f(1) = 000, \quad f(2) = 100, \quad f(3) = 010, \quad f(5) = 001,$$

$$f(6) = 110, \quad f(10) = 101, \quad f(15) = 011, \quad f(30) = 111$$

is an isomorphism. Hence D_{30} is a Boolean algebra.

If a finite L does not contain 2^n elements for some non-negative integer n, then L cannot be a Boolean Algebra.

For example, consider $D_{20} = \{1, 2, 4, 5, 10, 20\}$ that has 6 elements and $6 \neq 2^n$ for any integer $n \geq 0$. Therefore, D_{20} is not a Boolean algebra.

If $|L| = 2^n$, then L may or not be a Boolean Algebra. If L is isomorphic to B_n, then it is Boolean algebra, otherwise it is not.

For large value of n, we use the following theorem for determining whether D_n is a Boolean Algebra or not.

Theorem: Let

$$n = p_1 p_2 \ldots\ldots p_k,$$

where p_i are distinct primes, known as set of atoms**.** Then D_n is a Boolean algebra.

Proof: Let $A = \{p_1, p_2, \ldots, p_k\}$. If $B \subseteq A$ and a_B is the product of primes in B, then $a_B | n$. Also any divisor of n must be of the form a_B for some subset B of A, where we assume that $a_\varphi = 1$. Further, if C and B are subsets of A, then $C \subseteq B$ if and only if $a_C | a_B$. Also

$$a_{C \cap B} = a_C \wedge a_B = \gcd(a_C, a_B)$$

and

$$a_{C \cup B} = a_C \vee a_B = \text{lcm}(a_C, a_B)$$

Thus the function $f : P(A) \rightarrow D_n$ defined by

$$f(B) = a_B$$

is an isomorphism. Since P(A) is a Boolean algebra, it follows that D_n is also a Boolean algebra.

For example, consider D_{20}, D_{30}, D_{210}, D_{66}, D_{646}. We notice that

(i) 20 cannot be represented as product of distinct primes and so D_{20} is not a Boolean algebra.

(ii) 30 = 2.3.5, where 2, 3, 5 are distinct primes. Hence D_{30} is a Boolean Algebra.

(iii) 210 = 2.3.5.7 (all distinct primes) and so D_{210} is a Boolean algebra.

(iv) 66 = 2.3.11 (product of distinct primes) and so D_{66} is a Boolean algebra.

(v) 646 = 2.17.19 (product of distinct primes) and so D_{646} is a Boolean Algebra.

Duality: The **dual of any statement** in a Boolean algebra B is obtained **by interchanging** $\vee$ and $\wedge$ and interchanging the zero element and unit element in the original statement.

For example, the dual of $a \wedge 0 = 0$ is $a \wedge I = I$

Principle of duality: The dual of any theorem in a Boolean Algebra is also a theorem.
(Thus, dual theorem is proved by using the **dual of each step of the proof of the original statement).**

2.2 Properties of a Boolean Algebra

Theorem: Let a, b and c be any elements in a Boolean algebra (B, $\vee$, $\wedge$,′, 0, I). Then

1. Idempotent Laws:

(i) $a \vee a = a$ (ii) $a \wedge a = a$

2. Boundedness Laws:

(i) $a \vee I = I$ (ii) $a \wedge 0 = 0$

3. Absorption Laws:

(i) $a \vee (a \wedge b) = a$ (ii) $a \wedge (a \vee b) = a$

4. Associative Laws:

(i) $(a \vee b) \vee c = a \vee (b \vee c)$ (ii) $(a \wedge b) \wedge c = a \wedge (b \wedge c)$

Proof: **It is sufficient to prove first part of each law since second part follows from the first by principle of duality.**
1. (i). We have

$$\mathbf{a = a \vee 0 \text{ (by identity law in a Boolean algebra)}}$$
$$\mathbf{= a \vee (a \wedge a') \text{ (by complement law)}}$$
$$\mathbf{= (a \vee a) \wedge (a \vee a') \text{ (by distributive law)}}$$
$$\mathbf{= (a \vee a) \wedge I \text{ (complement law)}}$$
$$\mathbf{= a \vee a \text{ (identity law) ,}}$$

which proves 1(i).
2(i) : We have

$$\mathbf{a \vee I = (a \vee I) \wedge I \text{ (identity law)}}$$
$$\mathbf{= (a \vee I) \wedge (a \vee a') \text{ (complement law)}}$$
$$\mathbf{= a \vee (I \wedge a') \text{ (Distributive law)}}$$

$= a \vee a'$ (identity law)
$= I$ (complement law).

3(i) : we note that

$$a \vee (a \wedge b) = (a \wedge I) \vee (a \wedge b) \text{ (identity law)}$$
$$= a \wedge (I \vee b) \text{ (distributive law)}$$
$$= a \wedge (b \vee I) \text{ (commutativity)}$$
$$= a \wedge I \text{ (Identity law)}$$
$$= a \text{ (identity law)}$$

4(i) Let

$$L = (a \vee b) \vee c, \qquad R = a \vee (b \vee c)$$

Then

$$a \wedge L = a \wedge [(a \vee b) \vee c]$$
$$= [a \wedge (a \vee b)] \vee (a \wedge c) \text{ (distributive Law)}$$
$$= a \vee (a \wedge c) \text{ (absorption law)}$$
$$= a \text{ (absorption law)}$$

and

$$a \wedge R = a \wedge [a \vee (b \vee c)]$$
$$= (a \wedge a) \vee (a \wedge (b \vee c)] \text{ (distributive law)}$$
$$= a \vee (a \wedge (b \vee c)] \text{ (idempotent law)}$$
$$= a \text{ (absorption Law)}$$

Thus $a \wedge L = a \wedge R$ and so, by duality, $a \vee L = a \vee R$.

Further,

$$a' \wedge L = a' \wedge [(a \vee b) \vee c]$$
$$= [a' \wedge (a \vee b)] \vee (a' \wedge c) \text{ (distributive law)}$$
$$= [(a' \wedge a) \vee (a' \wedge b)] \vee (a' \wedge c) \text{ (distributive law)}$$
$$= [0, \vee (a' \wedge b)] \vee (a' \wedge c) \text{ (complement Law)}$$
$$= (a' \wedge b)] \vee (a' \wedge c) \text{ (Identity law)}$$
$$= a' \wedge (b \vee c) \text{ (distributive law)}$$

On the other hand,

$$a' \wedge R = a' \wedge [a \vee (b \vee c)]$$
$$= (a' \wedge a) \vee [a' \wedge (b \vee c)] \text{ (distributive law)}$$
$$= 0 \vee [a' \wedge (b \vee c)] \text{ (complement law)}$$
$$= a' \wedge (b \vee c)] \text{ (identity law)}$$

Hence

$$a' \wedge L = a' \wedge R \text{ and so by duality } a' \vee L = a' \vee R$$

Therefore

$$L = (a \vee b) \vee c$$

$$= 0 \vee [(a \vee b) \vee c] = 0 \vee L \text{ (identity law)}$$

$$= (a \wedge a') \vee [(a \vee b) \vee c] = (a \wedge a') \vee L \text{ (complement law)}$$

$$= (a \vee L) \wedge (a' \vee L) \text{ (distributive law)}$$

$$= (a \vee R) \wedge (a' \vee R) \text{ (using } A \vee L = a \vee R \text{ and } a' \vee L = a' \vee R]$$

$$= (a \wedge a') \vee R \text{ (distributive law)}$$

$$= 0 \vee R \text{ (complement law)}$$

$$= R \text{ (identity law)}$$

Hence

$$(a \vee b) \vee c = a \vee (b \vee c),$$

which completes the proof of the theorem.

Theorem: **Let a be any element of a Boolean algebra B. Then**
(i) Complement of a is unique (uniqueness of complement)
(ii) $(a')' = a$ (Involution law)
(iii) $0' = 1$ **and** $1' = 0$

Proof: **(i) Let** a' **and x be two complements of a** ε **B. Then**

$a \vee a' = I$ **and** $a \wedge a' = 0$ **(i)**
$a \vee x = I$ **and** $a \wedge x = 0$ **(ii)**

and we have

$a' = a' \vee 0$ **(Identity law)**
$= a' \vee (a \wedge x)$ **by (ii)**
$= (a' \vee a) \wedge (a' \vee x)$ **(Distributive law)**
$= I \wedge (a' \vee x)$ **by (i)**
$= a' \vee x$ **[Identity law]**

Also

$x = x \vee 0$ **(Identity law)**
$= x \vee (a \wedge a')$ **, by (i)**
$= (x \vee a) \wedge (x \vee a')$ **[Distributive law]**
$= I \wedge (x \vee a')$ **, (by (ii))**
$= x \vee a' = a' \vee x$ **(Identity and commutative law)**

Hence $a' = x$ **and so complement of any element in B is unique.**

(ii) Let a' **be a complement of a. Then**

$a \vee a' = I$ **and** $a \wedge a' = 0$

or , by commutativity ,

$a' \vee a = I$ **and** $a' \wedge a = 0$

This implies that a is complement of a'**, that is,**

$a = (a')'$.

(iii) By boundedness law,

$0 \vee 1 = 1$

and by identity law

$$0 \wedge 1 = 0$$

These two relations imply that 1 is the complement of 0, that is $1 = 0'$.
By principle of duality, we have then

$$0 = 1'.$$

Theorem: **Let a, b be elements of a Boolean Algebra. Then $(a \vee b)' = a' \wedge b'$ and $(a \wedge b)' = a' \vee b'$.**

Proof: **we have**

$$(a \vee b) \vee (a' \wedge b') = (b \vee a) \vee (a' \wedge b') \quad \text{(commutative)}$$

$$= b \vee (a \vee (a' \wedge b')) \quad \text{(associative)}$$

$$= b \vee [(a \vee a' \wedge (a \vee b')] \quad \text{(distributive)}$$

$$= b \vee [I \wedge (a \vee b') \quad \text{(complement)}$$

$$= b \vee (a \vee b') \quad \text{(identity)}$$

$$= b \vee (b' \vee a) \quad \text{(commutative)}$$

$$= (b \vee b') \vee a \quad \text{(associative law)}$$

$$= I \vee a \quad \text{(complement law)}$$

$$= I \quad \text{(Identity law)}$$

Also

$$(a \vee b) \wedge (a' \wedge b') = [(a \vee b) \wedge a'] \wedge b' \quad \text{(associativity)}$$

$$= [a \wedge a') \vee (b \wedge a')] \wedge b' = [0 \vee (b \wedge a')] \wedge b'$$

(complement) (distributive)

$$= (b \wedge a') \wedge b' \quad \text{(identity)}$$

$$= b \wedge b' \wedge a' = 0 \wedge a' = 0$$

Hence $a' \wedge b'$ is complement of $a \vee b$, i.e. $(a \vee b)' = a' \wedge b'$.
The second part follows by principle of duality.
We have proved already that Boolean algebra $(B, \vee, \wedge, ', 0, I)$ satisfies associative laws, commutative law and absorption law. Hence every Boolean algebra is a lattice with join as $\vee$ and meet as $\wedge$. Also boundedness law hold in a Boolean algebra. Thus Boolean algebra becomes a bounded lattice. Also Boolean algebra obeys distributive law and is complemented. Conversely, every bounded, distributive and complemented lattice satisfied all the axiom of a Boolean algebra. Hence we can define a Boolean algebra as

Definition: **A Boolean Algebra is a** bounded distributive and complemented lattice.

Now, being a lattice, a Boolean algebra must have a partial ordering. Recall that in case of lattice we had defined partial ordering $\leq$ by $a \leq b$ if $a \vee b = b$ or $a \wedge b = a$.

The following result yields much more than these required conditions:

Theorem: **If a, b are in a Boolean algebra, then the following are equivalent:**

(1) $a \vee b = b$

(2) $a \wedge b = a$

(3) $a' \vee b = I$

(4) $a \wedge b' = 0$

Proof: **(1) $\Leftrightarrow$ (2) already proved.**

(1) $\Rightarrow$ (3) : Suppose $a \vee b = b$, then

$$a' \vee b = a' \vee (a \vee b)$$

$$= (a' \vee a) \vee b \qquad \text{(associativity)}$$

$$= I \vee b = I \qquad \text{(complement \& boundedness)}$$

Conversely, suppose $a' \vee b = I$, then

$$a \vee b = 1 \wedge (a \vee b) = (a' \vee b) \wedge (a \vee b) \qquad \text{(by assumption of (3))}$$

$$= (a' \wedge a) \vee b \qquad \text{(distributivity)}$$

$$= 0 \vee b = b \qquad \text{(complement \& identity)}$$

Thus (1) $\Leftrightarrow$ (3).

Now we show that (3) $\Leftrightarrow$ (4).

Suppose first that (3) holds. Then, using De-Morgan Law and involution, we have

$$0 = I' = (a' \vee b')' = a'' \wedge b'$$

$$= a \wedge b' \qquad \text{(Involution)}$$

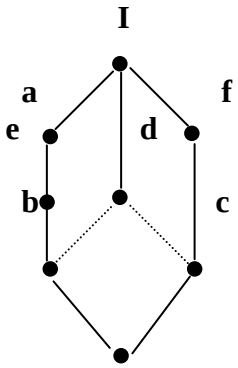
Conversely, if (4) holds, then

$$1 = 0' = (a \wedge b')' = a' \vee b'' = a' \vee b$$

Thus (3) $\Leftrightarrow$ (4)

Hence all the four condition are equivalent.

Example: **Show that the lattice whose diagram is**



0

is not a Boolean algebra.

Solution: **Elements a and e are both complements of c since c $\vee$ a = I, c $\wedge$ a = 0 and c $\vee$ e = I, c $\wedge$ e = 0**

But in a Boolean algebra complement of an element is unique. Hence the given lattice is not a Boolean algebra.

Definition: **Let (B, $\vee$, $\wedge$, $'$, 0, 1) be a Boolean algebra and S $\subseteq$ B. If S contains the elements 0 and 1 and is closed under the operation $\vee$, $\wedge$ and 1, then (S, $\wedge$, $\vee$, $'$, 0, 1) is called** Sub-Boolean Algebra**.**

In practice, it is sufficient to check closure with respect to the set of operations ($\wedge$, $'$) or ($\vee$, $'$) for proving a subset S of B as the sub-Boolean algebra.

The definition of sub-Boolean implies that it is a Boolean algebra.

But a subset of Boolean algebra can be a Boolean algebra, but not necessarily a Boolean subalgebra because it is not closed with respect to the operations in B.

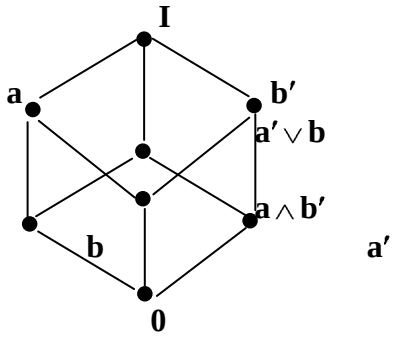
For any Boolean algebra (B, $\wedge$, $\vee$, $'$, 0, 1), the subsets {0, 1} and the set B are both sub-Boolean algebras.

In addition to these sub-Boolean algebras, consider now any element a $\in$ B such that a $\neq$ 0 and a $\neq$ 1 and consider the set {a, a$'$, 0, 1}. Obviously this set is a sub-Boolean algebra of the given Boolean algebra.

For example D_{70} = {1, 2, 5, 7, 10, 14, 35, 70} is a Boolean algebra and {1, 2, 35, 70} is a subalgebra of D_{70}.

Every element of a Boolean algebra generates a sub-Boolean algebra, More generally, any subset of B generates a sub-Boolean algebra.

Example: **Consider the Boolean algebra given in the diagram below:**



Verify whether the following subsets are Boolean algebras or not :

S_1 = {a, a$'$, 0, 1}

S_2 = { a$'$ $\vee$ b , a $\wedge$ b$'$, 0, 1}

S_3 = {a $\wedge$ b$'$, b$'$, a, 1}

S_4 = {b$'$, a $\wedge$ b$'$, a$'$, 0}

$$S_5 = \{a, b', 0, 1\}$$

Solution: **The subset S_1 and S_2 are sub-Boolean algebras. The subsets S_3 and S_4 are Boolean algebras but not sub-Boolean algebras of the given Boolean algebra. The subset S_5 is not even a Boolean algebra.**

Definition: **Let $(B_1, \wedge_1, \vee_1, ', 0_1, 1_1)$ and $(B_1, \wedge_2, \vee_2, '', 0_2, 1_2)$ be two Boolean algebras. The** Direct Product **of the two Boolean algebras is defined to be a Boolean algebra, denoted by, $(B_1 \times B_2, \wedge_3, \vee_3, ''', 0_3, 1_1)$ in which the operations are defined for any (a_1, b_1) and $(a_2, b_2) \in B_1 \times B_2$ as**

$$(a_1, b_1) \wedge_3 (a_2, b_2) = (a_1 \wedge_1 a_2, b_1 \wedge_2 b_2)$$
$$(a_1, b_1) \vee_3 (a_2, b_2) = (a_1 \vee_1 a_2, b_1 \vee_2 b_2)$$
$$(a_1, b_1)''' = (a_1', b_1'')$$
$$0_3 = (0_1, 0_2) \text{ and } I_3 = (I_1, I_2)$$

Thus, from a Boolean algebra B, we can generate $B^2 = B \times B$, $B^3 = B \times B \times B$ etc.

2.3 Boolean Homomorphism

Definition: **Let $(B, \wedge, \vee, ', 0, 1)$ and $(P, \cap, \cup, —, \alpha, \beta)$ be two Boolean Algebras. A mapping $f : B \to P$ is called a** Boolean Homomorphism **if all the operations of the Boolean Algebra are preserved , that is , for any $a, b \in B$**

$$f(a \wedge b) = f(a) \cap f(b)$$
$$f(a \vee b) = f(a) \cup f(b)$$
$$f(a') = \overline{f(a)}$$
$$f(0) = \alpha$$
$$f(1) = \beta$$

The above definition of homomorphism can be simplified by asserting that $f : B \to P$ preserves either the operations $\wedge$ and $'$ or the operations $\vee$ and $'$.

We now consider a mapping $g : B \to P$ in which the operations $\wedge$ and $\vee$ are preserved. Thus g is a lattice homomorphism. Naturally g preserves the order and hence it maps the bounds 0 and I into the least and the greatest element respectively of the image set $g(B) \subseteq P$. It is however, not necessary that $g(0) = \alpha$ and $g(1) = \beta$. The complements, if defined in terms of g(0) and g(1) in g(B), are preserved, and $(g(B), \cap, \cup, —, g(0), g(1))$ is a Boolean algebra. Note that $g : B \to P$ is not a a Boolean homomorphism, although $g : B \to g(B)$ is a Boolean homomorphism.

In any case, for any mapping from a Boolean Algebra which preserves the operations $\wedge$ and $\vee$, the image set **is a Boolean algebra.**

A Boolean homomorphism is called Boolean isomorphism **if it is bijective.**

2.4. Representation Theorem

Let B be a finite **Boolean algebra. We know that an element a in B is called an** atom (or min term) **if a immediately succeed the** least element **0 . Let A be the set of atoms of B and let P(A) be the Boolean algebra of all subsets of the set A of atoms. Then (as proved in chapter on lattices) each $x \neq 0$ in B can be expressed uniquely (except for order) as the join of atoms (i.e. elements of A). So, let**

$$x = a_1 \vee a_2 \vee \ldots\ldots \vee a_n$$

Consider the function

$$f : B \to P(A)$$

defined by

$$f(x) = \{a_1, a_2, \ldots\ldots, a_n\}$$

for each $x = a_1 \vee a_2 \vee \ldots. \vee a_n$.

Stone's Representation Theorem: **Any Boolean Algebra is isomorphic to a power set algebra (P(S), $\cap$, $\cup$, ~, ϕ, S) for some set S.**

Restricting our discussion to finite Boolean Algebra B, the representation theorem can be stated as :

Theorem: **Let B be a finite Boolean Algebra and let A be the set of atoms of B. If P(A) is the Boolean Algebra of all subsets of the set A of atoms, then the mapping $f : B \to P(A)$ is an isomorphism.**

Proof: **Suppose B is finite Boolean algebra and P(A) is the Boolean algebra of all subsets of the set A of atoms of B. Consider the mapping**

$$f : B \to P(A)$$

defined by

$$f(x) = \{a_1, a_2, \ldots., a_n\} \ ,$$

where $x = a_1 \vee a_2 \vee \ldots. \vee a_n$ is the unique representation of $x \in B$ as the join of atoms $a_1, a_2, \ldots., a_n \in A$. If a_i are atoms, then we know that $a_i \wedge a_i = a_i$ but $a_i \wedge a_j = 0$ for $a_i \neq a_j$.

Let x and y are in the Boolean algebra B and suppose

$$x = a_1 \vee \ldots.. \vee a_r \vee b_1 \vee \ldots.. \vee b_s$$
$$y = b_1 \vee \ldots.. \vee b_s \vee c_1 \vee \ldots.. \vee c_t,$$

where

$$A = \{ a_1, a_2, \ldots, a_r, b_1, b_2, \ldots, b_s, c_1, \ldots, c_t, d_1 \ldots, d_k\}$$

is the set of atoms of B. Then

$$x \vee y = a_1 \vee \ldots. \vee a_r \vee b_1 \vee \ldots. \vee b_s \vee c_1 \ldots \vee c_t$$
$$x \wedge y = b_1 \vee \ldots.. \vee b_s$$

Hence

$$f(x \vee y) = \{ a_1, a_2, \ldots., a_r, b_1, b_2, \ldots.., b_s, c_1, c_2 \ldots., c_t\}$$
$$= \{ a_1, \ldots., a_r, b_1, \ldots.., b_s\} \cup \{b_1, b_2, \ldots.., b_s, c_1, c_2 \ldots., c_t\}$$
$$= f(x) \cup f(y)$$

and

$$f(x \wedge y) = \{b_1, \ldots\ldots, b_s\}$$
$$= \{ a_1, a_2 \ldots., a_r, b_1, \ldots.., b_s\} \cap \{b_1, \ldots.., b_s, c_1, \ldots., c_t\}$$
$$= f(x) \cap f(y)$$

Let

$$y = c_1 \vee \ldots\ldots \vee c_t \vee d_1 \vee \ldots\ldots \vee d_k$$

Then

$x \vee y = I$ and $x \wedge y = 0$

and so $y = x'$. Thus

$f(x') = f(y) = \{c_1 \ldots\ldots c_t, d_1 \ldots\ldots d_k\}$

$= \{a_1, a_2 \ldots., a_r, b_1, b_2 \ldots.., b_s\}^c$

$= (f(x))^c$.

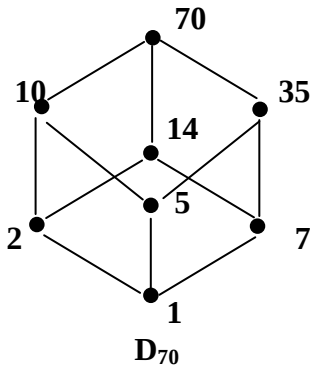
Since the representation is unique, f is one-to-one and onto. Hence f is a Boolean algebra isomorphism. Thus, every finite Boolean algebra is structurally the same as a Boolean algebra of sets.

If a set A has n elements, then its power set P(A) has 2^n elements. Thus we have

Corollary: **A finite Boolean algebra has 2^n elements for some positive integer n.**

Example: **Consider the Boolean algebra**

$D_{70} = \{1, 2, 5, 7, 10, 14, 35, 70\}$



D_{70}

Then the set of atoms of D_{70} is

$A = \{2, 5, 7\}$

The unique representation of each non-atom by atoms is

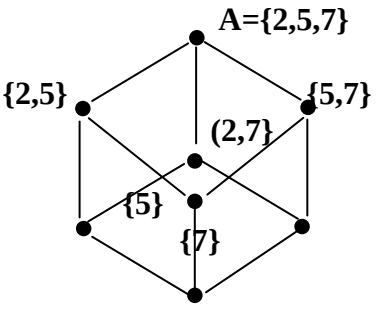
$10 = 2 \vee 5$

$14 = 2 \vee 7$

$35 = 5 \vee 7$

$70 = 2 \vee 5 \vee 7$

The diagram of the Boolean algebra of the power set e(A) of the set A of atoms is given below :



φ

P(A)

We note that the diagram for D_{70} and P(A) are structurally the same.

2.5. Boolean Expressions

Definition: **Let $x_1, x_2,\ldots,x_n$ be a set of n variables (or letters or symbols).** A Boolean Polynomial (Boolean expression, Boolean form or Boolean formula) **$p(x_1, x_2, \ldots, x_n)$ in the variables $x_1, x_2, \ldots, x_n$ is defined recursively as follows:**

1. The symbols 0 to 1 are Boolean polynomials

2. $x_1, x_2, \ldots, x_n$ are all Boolean polynomials

3. if $p(x_1, x_2, \ldots, x_n)$ and $q(x_1, x_2, \ldots, x_n)$ are two Boolean polynomials, then so are

$$p(x_1, x_2, \ldots, x_n) \vee q(x_1, x_2, \ldots, x_n)$$

and

$$p(x_1, x_2, \ldots, x_n) \wedge q(x_1, x_2, \ldots, x_n)$$

4. If $p(x_1, x_2, \ldots, x_n)$ is a Boolean polynomial, then so is

$$(p(x_1, x_2, \ldots, x_n))'$$

5. There are no Boolean polynomials in the variables $x_1, x_2, \ldots, x_n$ other than those obtained in accordance with rules 1 to 4.

Thus, Boolean expression is an expression built from the variables given using Boolean operations $\vee$, $\wedge$ and $'$.

For example, for variables x, y, z , the expressions

$$p_1(x, y, z) = (x \vee y) \wedge z$$

$$p_2(x, y, z) = (x \vee y') \vee (y \wedge 1)$$

$$p_3(x, y, z) = (x \vee (y' \wedge z)) \vee (x \wedge (y \wedge 1))$$

are Boolean expressions.

Notice that a Boolean expression is n variables may or may not contain all the b variables. Obviously, an infinite number of Boolean expressions may be constructed in n variables.

Definition: **A** literal **is a variable or complemented variable such as x, x′, y, y′, and so on.**

Definition: **A** fundamental product **is a literal or a product of two or more literal in which no two literals involve the same variable.**

For example,

$$x \wedge z', x \wedge y' \wedge z, x, y', x' \wedge y \wedge z$$

are fundamental products whereas

$$x \wedge y \wedge x' \wedge z \text{ and } x \wedge y \wedge z \wedge y$$

are not fundamental products.

Remark: **Fundamental product is also called a** minterm **or** complete product. **In what follows we shall denote x $\wedge$ y by xy.**

Any product of literals can be reduced to either 0 or a fundamental product.

For example, consider x y x′ z. Since x $\wedge$ x′ = 0 by complement law, we have xyx′z = 0.

Similarly, if we consider x y z y, then since y $\wedge$ y = y (idempotent law), we have xyzy = xyz, which is a fundamental product.

Definition: **A fundamental product P_1 is said to be** contained in **(or included in) another fundamental Product P_2 if the literals of P_1 are also literals of P_2.**

For example, x′ z is contained in x′ yz but x′ z is not contained in x y′ z since x′ is not a literal of xy′z.

Observe that if P_1 is contained in P_2, say $P_2 = P_1 \wedge Q$, then, by the absorption law,

$$P_1 \vee P_2 = P_1 \vee (P_1 \wedge Q) = P_1$$

For example,

$$x' z \vee x' y z = x' z$$

Definition: **A Boolean expression E is called** a sum-of-products expression(disjunctive Normal Form or D NF) **if E is a fundamental product or the sum (join) of two or more fundamental products none of which is contained in another.**

Definition: **Two Boolean expression $P(x_1, x_2,\ldots,x_n)$ and $Q(x_1, x_2,\ldots,x_n)$ are called** equivalent (or equal) **if one can be obtained from the other by a finite number of applications of the identities of a Boolean algebra.**

Definition: **Let E be any Boolean expression. A** sum of product form **of E is an equivalent Boolean sum of products expression.**

Example: **Consider the expression**

$$E_1(x, y, z) = x z' + y' z + x y z'$$

Although the expression E_1 is a sum of products, it is not a sum-of-products expression because, the product x z′ is contained in the product x y z′. But, by absorption law, E_1 can be expressed as

$$E_1(x, y, z) = x z' + y' z + x y z' = x z' + x y z' + y' z = x z' + y' z ,$$

which is a sum-of-product form for E_1.

2.6. Algorithm for Finding Sum-of-Products Forms

The input is a Boolean expression E. The output is a sum-of-products expression equivalent to E.

Step 1. **Use De Morgan's Law and involution to move the complement operation into any parenthesis untill finally the complement operation only applies to variables. Then E will consists only sums and products of literals.**

Step 2. **Use the distributive operation to next transform E into a sum of products.**

Step 3. **Use the commutative, idempotent, and complement laws to transform each product in E into 0 or a fundamental product.**

Step 4. **Use the absorption law and identity law to finally transform E into a sum of products expression.**

For example, we apply the above Algorithm to the Boolean expression.

$$E = ((x\ y)'\ z)'\ ((x' + z)\ (y' + z'))'$$

Step 1. **Using De Morgan's laws and involution, we obtain**

$$E = ((x\ y)'' \vee z')\ ((x' \vee z)' \vee (y' \vee z'))'$$
$$= (x\ y \vee z') \wedge [(x \wedge z') \vee (y \wedge z)]$$

Thus E consists only of sum and products of literals.

Step 2. **Using the distributive laws, we obtain**

$$E = (x\ y + z')\ x\ z' + (x\ y + z')\ y z$$
$$= x\ y\ x\ z' + x\ z'\ z' + x\ y\ y\ z + y\ z\ z'$$

Thus E is now a sum of products.

Step 3. **Using commutative, idempotent and complement law, we obtain**

$$E = x\ y\ z' + x\ z' + x\ y\ z + 0$$

Thus each term in E is a fundamental product or 0.

Step 4. **Using absorption law**

$$x\ z' + x\ y\ z' = x\ z' + (x\ z' \wedge y)$$

$$= x\ z'$$

Hence

$$E = x\ z' + x\ y\ z + 0$$

Step 5. **Now using identity law**

$$E = x\ z' + x\ y\ z\ ,$$

which is the required sum-of-products expression.

2.7 Complete Sum-of-Product Expression

Definition: **A Boolean expression E ($x_1, x_2, \ldots, x_n$) is said to be a** complete sum-of-product expression (or full disjunctive normal form or disjunctive canonical form, or the minterm canonical form) **if E is a sum-of-products expression where each product involves all the n variables.**

A fundamental product which involves all the variables is called a minterm **and there is a maximum of 2^n such products for n variables.**

It can be seen that "every non-zero Boolean expression $E(x_1, x_2, \ldots, x_n)$ is equivalent to a complete sum-of-product expression and such a representation is unique.**"**

ALGORITHM FOR OBTAINING COMPLETE SUM OF PRODUCT EXPRESSION

The input is a Boolean sum-of-products expression $E(x_1, x_2, \ldots, x_n)$. The output is a complete sum-of-products expression equivalent to E.

Step 1. **Find a product P in E which does not involve the variable x_i and then multiply P by $(x_i + x_i')$ deleting any repeated products (This is possible since $x + x' = 1$ and $P + P = P$).**

Step 2. **Repeat step 1 until every product in E is a minterm, i.e. every product P involve all the variables.**

Example: Express $x_1 \vee x_2$ in its complete sum-of-products form in three variables x_1, x_2, x_3.

Solution: We have, using the above stated algorithm,

$x_1 + x_2 = [x_1(x_2 + x_2')] + [x_2 (x_1 + x_1')]$

$= x_1 x_2 + x_1 x_2' + x_2 x_1 + x_1' x_2$

$= x_1 x_2 + x_1 x_2' + x_1' x_2$

$= x_1 x_2(x_3 + x_3') + x_1 x_2'(x_3 + x_3') + x_1' x_2(x_3 + x'_3)$

$= x_1 x_2 x_3 + x_1 x_2 x_3' + x_1 x_2' x_3 + x_1 x_2' x_3' + x_1' x_2 x_3 + x_1' x_2 x_3'$,

which is the complete sum-of-products form in x_1, x_2, x_3.

2.8 Minimal Sum-of-Products

Consider a Boolean sum-of-products expression E. Let E_L denote the number of literals in E (counted according to multiplicity) and let E_S denote the number of summands in E. For example, let

$$E = x y z' + x' y' z + x y' z' t + x' y z t.$$

Then

$$E_L = 3 + 3 + 4 + 4 = 14 \text{ and } E_S = 3.$$

Let E and F be equivalent Boolean sum-of-products expressions. Then E is called **simpler** than F if

(i) $E_L < F_L$ and $E_S \leq F_L$

or

(ii) $E_L \leq F_L$ and $E_S < F_L$

Definition : A Boolean sum-of-product expression is called **minimal** if there is no equivalent sum-of-product expression which is simpler than E.
There can be more than one equivalent minimal sum-of-products expressions.

Definition : A fundamental product P is called **prime implicants** of a Boolean expression E if P + E = E but no other fundamental product contained in P has this property.

For example, suppose

$$E = x y' + x y z' + x' y z'$$

Then, we find first the complete-sum-of-products form of x z′. Towards this end, we have

$$x\,z' = x\,z'(y + y')$$

$$= x\,z'\,y + x\,z'\,y' \qquad (1)$$

Also we know that the complete sum-of-products form is unique, A + E = E, where A ≠ 0 if and only if the summands in the complete sum-of-products form for A are among the summands in the complete sum-of-products form for E. We observe that summands x y z′ and x y′ z′ in (1) are in the complete form of E given below:

$$\mathbf{E = x\,y'(z + z') + x\,y\,z' + x'\,y\,z'}$$

$$= x\,y'\,z + x\,y'\,z' + x\,y\,z' + x'\,y\,z'$$

Therefore, by the above argument,

$$x\,z' + E = E$$

Also, the complete sum-of-products form of x is

$$x = x(y + y')\,(z + z')$$

$$= (x\,y + x\,y')\,(z + z')$$

$$= x\,y\,z + x\,y\,z' + x\,y'\,z + x\,y'\,z'$$

The summand x y z of x is not a summand of E. Hence

$$x + E \neq E$$

Similarly, the complete sum-of-product form of z′ is

$$z' = z'(x + x')\,(y + y')$$

$$= (z'\,x + z'\,x')\,(y + y')$$

$$= z'\,x\,y + z'\,x\,y' + z'\,x'\,y + z'\,x'\,y'$$

The summand x′ y′ z′ of z′ is not a summand of E. Hence

$$z' + E \neq E.$$

Thus the fundamental products x and z′ contained in x z′ do not have the property P + E = E where as x z′ has this property. Hence x z′ is a prime implicant of E**.**

It can be seen "a minimal sum-of-products form for a Boolean expression E is a sum of prime implicants of E**"**

2.9. Consensus of Fundamental Products

Let P_1 and P_2 be fundamental products such that exactly one variable say x_k appears uncomplemented in **one of** P_1 and P_2 and complemented in the other. Then the **consensus of P_1 and P_2** is the product (without repetitions) of the literals of P_1 and P_2 after x_k and x_k' are deleted. (**we do not define the consensus of $P_1 = x$ and $P_2 = x'$**)

Lemma: Suppose Q is the consensus of P_1 and P_2. Then $P_1 + P_2 + Q = P_1 + P_2$.

Proof: Since the literals commute, we can assume without loss of generally that

$$P_1 = a_1\, a_2 \ldots\ldots a_r\, t, \qquad P_2 = b_1\, b_2 \ldots\ldots b_s\, t'$$

$$Q = a_1\, a_2 \ldots\ldots a_r\, b_1\, b_2 \ldots\ldots b_s$$

Now $Q = Q(t + t') = Q\,t + Q\,t'$. Because $Q\,t$ contains P_1, $P_1 + Q\,t = P_1$; and because $Q\,t'$ contain P_2,

$$P_2 + Qt' = P_2.$$

Hence

$$\begin{aligned} P_1 + P_2 + Q &= P_1 + P_2 + Q\,t + Q\,t' \\ &= (P_1 + Q\,t) + (P_2 + Q\,t') \\ &= P_1 + P_2. \end{aligned}$$

Example : **Find the consensus Q of P_1 and P_2, where**

(i) $P_1 = x\,y\,z'\,s$, $P_2 = x\,y'\,t$

(ii) $P_1 = x\,y'$, $P_2 = y$

(iii) $P_1 = x'\,y\,z$, $P_2 = x'\,y\,t$

(iv) $P_1 = x'\,y\,z$, $P_2 = x\,y\,z'$.

Solution: **(i) $P_1 = x\,y\,z'\,s$, $P_2 = x\,y'\,t$**

Delete y and y' and then multiply the literals of P_1 and P_2 (without repetition) to obtain

$Q = x\,z'\,s\,t$

(ii) $P_1 = x\,y'$, $P_2 = y$

Delete y and y' then multiply the literal of P_1 and P_2 (without repetition) to obtain

$Q = x$

(iii) $P_1 = x'\,y\,z$, $P_2 = x'\,y\,t$

In this case, no variable appears uncomplemented in one of the products and complemented in the other. Hence P_1 and P_2 have no consensus.

(iv) $P_1 = x' y z$, $P_2 = x y z'$.

Each x and z appear complemented in one of the products and uncomplemented in the other. Hence P_1 and P_2 have no consensus.

CONSENSUS METHOD FOR FINDING PRIME IMPLICANTS

The following algorithm, known as consensus Method **is used to find the prime implicants of a Boolean expression.**

ALGORITHM (CONSENSUS METHOD)

The input is a Boolean expression $E = P_1 + P_2 + \ldots + P_m$, where P_m are fundamental products. The output expresses E as a sum of its prime implicants.

Step 1. **Delete any fundamental product P_i which includes any other fundamental product P_j (this is permissible by the absorption law)**

Step 2. **Add the consensus of any P_i and P_j providing Q does not include any of the P_i (this is permissible by the lemma $P_1 + P_2 + \ldots + P_n + Q = P_1 + \ldots + P_n$.)**

Step 3. **Repeat Step 1/or Step 2 untill neither can be applied.**

Example : **Let**

$E(x, y, z) = x y z + x' z' + x y z' + x' y' z + x' y z'$

Then

$E = x y z + x' z' + x y z' + x' y' z$ ($\because x' y z'$ include $x' z'$)

$= x y z + x' z' + x y z' + x' y' z + x y$ (consensus xy of xyz , xyz' added)

$= x' z' + x' y' z + x y$ ($\because x y z$ and $x y z'$ include x y)

$= x' z' + x' y' z + x y + x' y'$ (consensus $x' y'$ of $x' z'$ and $x' y' z$ added)

$= x' z' + x y + x' y'$ ($\because x' y' z$ include $x' y'$)

$= x' z' + x y + x' y' + y z'$ (consensus of $x'z'$ and xy, which is yz', added)

After this none of the step in the consensus method will change E. Thus E is the sum of its prime implicants $x' z'$, x y, $x' y'$ and $y z'$.

Use of Consensus method for finding Minimal Sum-of-Products Form

We have seen that consensus method can be used to express a Boolean expression E as a sum of all its prime implicants. Using such a sum, we can find a minimal sum-of-products form for E as follows:

Algorithm: **The input is a Boolean expression $E = P_1 + P_2 + \ldots\ldots + P_m$, where P_i are all prime implicants of E. The output expresses E as a minimal sum-of-products.**

Step 1. **Express each prime implicant P as a complete sum-of-products.**

Step 2. **Delete one by one those prime implicants whose summands appear among the summands of the remaining prime implicants.**

Example: **Consider Boolean expression E expressed as the sum of prime implicants in the above example. We have**

$$E = x' z' + x y + x' y' + y z'$$

We first convert each prime implicant into complete sum-of-products form. We have

$$x' z' = x' z'(y + y') = x' z' y + x' z' y'$$
$$x y = x y(z + z') = x y z + x y z'$$
$$x' y' = x' y'(z + z') = x' y' z + x' y' z'$$
$$y z' = y z'(x + x') = y z' x + y z' x'$$

The summands of $x' z'$ appear in the summands of $x' y'$ and $y z'$. So we delete $x' z'$ and get

$$E = x y + x' y' + y z' \qquad (1)$$

The summands of no other prime implicant appear among the summands of the remaining prime implicants. Hence expression (1) is a minimal sum-of-product form for E. In other words, none of the remaining prime implicants is superfluous, that is, none can be deleted without changing E.

2.10 Logic Gates And Circuits

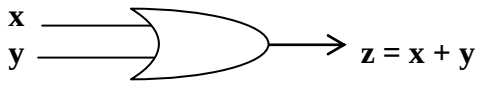
Definition: Logic circuit **(or** logic networks**) are structures which are built up from certain elementary circuit called** logical gates.

LOGIC GATES

There are three basic logic gates. The lines (wires) entering the gate symbol from the left are input lines and the single line on the right is the output line.

1. OR Gate: **An OR gate has input x and y and output $z = x \vee y$ or $z = x + y$, where addition (or Join) is defined by the truth table. In this case the output z = 0 only when inputs x = 0 and y = 0.**

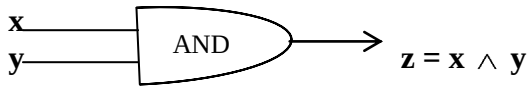
The symbol and the truth table for OR gate are shown in the diagram below:



x	y	x + y
1	**1**	**1**
1	**0**	**1**
0	1	1
0	0	0

(Truth Table for OR gate)

2. AND Gate: **In this gate the inputs are x and y and output is $x \wedge y$ or x.y or xy, where multiplication is defined by the truth table.**



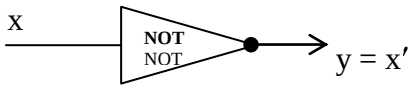
x	**y**	**z = x ∧ y**
1	**1**	**1**
1	**0**	**0**
0	1	0
0	0	0

(Truth Table for AND gate)

Thus output is 1 only when x = 1 , y = 1, otherwise it is zero.

The AND gate may have more than two inputs. The output in such a case will be 1 if all the inputs are 1.

3. **NOT Gate (inverter):** The diagram below shows NOT gate with input x and output y = x′, where inversion, denoted by the prime, is defined by the truth table:



(NOT gate)

x	y = x′
1	0
0	1

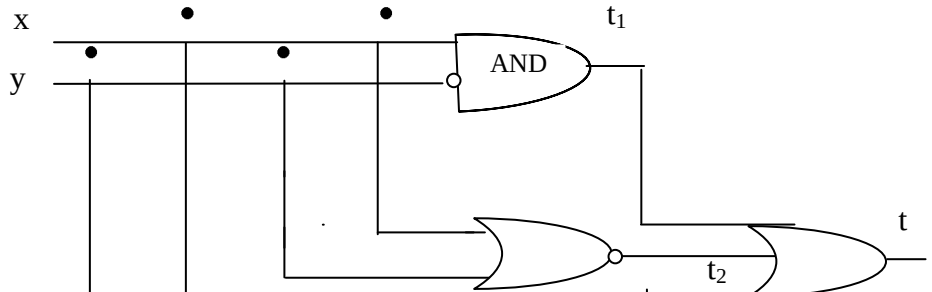
Truth Table for NOT gate

For example, if x = 10101, then output x′ in NOT gate shall be

x′ = 01010

Exercise : Draw logic circuit for a b′ + ab

Logic circuits as a Boolean Algebra: **The truth tables for OR, AND and NOT gates are respectively identical to the truth tables for the propositions p ∨ q (disjunction, "p or q"), p ∧ q(Conjunction, "p and q") and ~ p (negation, "not p"). The only difference is that 0 and 1 are used instead of F (contradiction) and T (tautology). Thus the logic circuits satisfy the same laws as do propositions and hence they form a Boolean Algebra. Hence, we have established the following:**
Theorem: Logic circuits form a Boolean Algebra.
Example: Express the output of the logic circuit below as a Boolean expression. **(Here small circle represents complement (NOT))**



Solution: **We note that**

$$t_1 = xy'$$
$$t_2 = (x+y)'$$
$$t_3 = (x'y)'$$

and so we have

$$t = t_1 + t_2 + t_3$$
$$= x\,y' + (x + y)' + (x'\,y)'$$

NAND and NOR Gates

NAND and NOR gates are frequently used in computers.

NAND gate: **It is equivalent to AND gate followed by a NOT gate. Its symbol is**

NAND gate

Its truth table is

x	y	x y	z = (x y)′
1	**1**	**1**	**0**
1	**0**	**0**	**1**
0	1	0	1
0	0	0	1

Thus, the **output of a NAND gate is 0 if and only if all the inputs are 1.**

NOR gate: This gate is equivalent to OR gate followed by a NOT gate. Its symbol is

NOR Gate

Its truth table is as shown as:

x	y	x + y	(x + y)′
1	**1**	**1**	**0**
1	**0**	**1**	**0**
0	1	1	0
0	0	0	1

Thus, **the output of NOR gate is 1 if and only if all inputs are 0.**

2.11 Boolean Function

We know that ordinary polynomials could produce functions by substitution. For example, the polynomial x y + y z^3 produces a function f : $R^3 \rightarrow R$ **by letting f(x,y,z) = xy +yz^3. Thus f (3, 4, 2) = 3. 4 + 4. 2^3 = 44. In a similar way, Boolean polynomials involving n variables produce functions from B_n to B.**

Definition: **Let (B, . , +, ′, 0, 1) be a Boolean algebra. A function f :** $B_n \rightarrow B$ **which is associated with a Boolean expression (polynomial) is n variables is called a** Boolean function.

Thus a Boolean function is completely determined by the Boolean expression α **(**$x_1, x_2, \ldots, x_n$**) because it is nothing but the evaluation function of the expression. It may be mentioned here that every function g :** $B_n \rightarrow B$ **needs not be a Boolean function.**

If we assume that the Boolean algebra B is of order 2^m **for m** $\geq$ **1, then the number of function from** B_n **to B is greater than** 2^{2n} **showing that there are functions from** B_n **to B which are not Boolean functions. On the other hand, for m = 1, that is, for a two element Boolean algebra, the number of function from** B_n **to B is** 2^{2n} **which is same as the number of distinct Boolean expressions in n variable. Hence every function from** B_n **to B in this case is a Boolean function.**

Example: **Show that the following Boolean expression are equivalent to one-another. Obtain their sum-of-product canonical form.**

(a) (x + y)(x′ + z)(y + z)
(b) (x.z) + (x′y) + (yz)
(c) (x + y)(x′ + z)
(d) x z + x′y

Solution: **The binary valuation of the expression are**

x	y	z	x+y	x′+z	y+z	(a)	(c)	xz	x′y	yz	(b)	(d)
0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	0	0	0	0	0	0
0	1	0	1	1	1	1	1	0	1	0	1	1
0	1	1	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	0	0	1	1
1	1	0	1	0	1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	0	1	1	1

Since the values of the given Boolean expression are equal over every triple of the two element Boolean algebra, they are equal.

To find the sum-of-product canonical (complete) **form, we note that (d) is in sum-of-product form. Therefore to find complete sum-of-product form, we have**

$$(d) = (x\ z) + (x'\ y)$$
$$= x\ z(y + y') + (x'\ y)\ (z + z')$$
$$= x\ z\ y + x\ z\ y' + x'\ y\ z + x'\ y\ z'$$

METHOD TO FIND TRUTH TABLE OF A BOOLEAN FUNCTION

Consider a logic circuit consisting of 3 input devices x, y, z. Each assignment of a set of three bits to the input x, y, z yield an output bit for z. There are $2^n = 2^3 = 8$ possible ways to assign bits to the input as follows:

000, 001, 010, 011, 100, 101, 110, 111.

The assumption is that the sequence of first bits is assigned to x, the sequence of second bits to y, and the sequence of third bits to z. Thus the above set of inputs may be rewritten in the form

x = 00001111, y = 00110011, z = 01010101

These three sequences (of 8 bits) contain the eight possible combination of the input bits.

The truth table T = T(L) of the circuit L consists of the output t that corresponds to the input sequences x, y, z.

The truth table is same as we generally have written in vertical columns. The difference is that here we write x, y, z and t horizontally.

Consider a logic circuit L with n input devices. There are many ways to form n input sequences x_1, x_2,….,x_n so that they contain 2^n different possible combinations of the input bits (Each sequence must contain 2^n bits).

The assignment scheme is:

x_1 : Assign 2^{n-1} bits which are 0 followed by 2^{n-1} bits which are 1.

x_2 : Assign 2^{n-2} bits which are 0 followed by 2^{n-2} bits which are 1.

x_3 : Assign 2^{n-3} bits which are 0 followed by 2^{n-3} bits which are 1.

and so on.

The sequence obtained in this way is called "Special Sequence**". Replacing 0 by 1 and 1 by 0 in the special sequences yield the** complements **of the special sequences.**

Example: **Suppose a logic circuit L has n = 4 input devices x, y, z, t. Then $2^n = 2^4 = 16$ bit special sequences for x, y, z, t are**

x = 0000000011111111 (2^3 = 8 zeros followed by 8 ones)

y = 0000111100001111 ($2^{n-2} = 2^{4-2}$ = 4 zeros followed by 4 ones)

z = 0011001100110011 ($2^{n-3} = 2^{4-3}$ = 2 zeros followed by 2 ones)

t = 0101010101010101 ($2^{n-4} = 2^{4-4} = 2^0$ = 1 zeros followed by 1 one)

ALGORITHM FOR FINDING TRUTH TABLE FOR A LOGIC CIRCUIT LWHERE OUTPUT T IS GIVEN BY A BOOLEAN SUM-OF-PRODUCT EXPRESSION IN THE INPUTS.

The input is a Boolean sum-of-products expression t(x_1, x_2,….,..).

Step 1. **Write down the special sequences for the inputs x_1, x_2,….and their complements**

Step 2. **Find each product appearing in $t(x_1, x_2,\ldots.)$ keeping in mind that $x_1, x_2,\ldots.=1$ is a position if and only if all $x_1, x_2\ldots..$have 1 in the position.**
Step 3. **Find the sum t of the products keeping in mind that $x_1 + x_2 + \ldots..= 0$ in a position if and only if all $x_1, x_2,\ldots..$have 0 in the position.**

2.12 Representation of Boolean Functions using Karnaugh Map

Karnaugh Map is a graphical procedure to represent Boolean function as an "or" combination of minterms where minterms are represented by squares. This procedure is easy to use with functions f: $B_n \rightarrow B$, if n is not greater than 6. We shall discuss this procedure for n = 2, 3, and 4.
A Karnaugh map structure is an area which is subdivided into 2^n cells, one for each possible input combination for a Boolean function of n variables. Half of the cells are associated with an input value of 1 for one of the variables and the other half are associated with an input value of 0 for the same variable. This association of cell is done for each variable, with the splitting of the 2^n cells yielding a different pair of halves for each distinct variable.
Case of 1 variable: **In this case, the Karnaugh map consists of $2^1 = 2$ squares.**

0	1
x′	x

The variable x is represented by the right square and its complement x′ by the left square.
Case of 2 variables: **For n = 2, the Boolean function is of two variable, say x and y. We have $2^2 = 4$ squares, that is, a 2×2 matrix of squares. Each squares contains one possible input from B_2.**
The variable x appears in the first row of the matrix as x′ whereas x appears in the second row as x. Similarly y appears in the first column as y′ and as y in the second column.

	0	1
0	00	01
1	10	11

x′ \ y′	y′	y
x′	x′y′	x′y
x	xy′	xy

(2 variable Karnaugh Map)

In this case, x is represented by the points in lower half of the map and y is represented by the points in the right half of the map.
Definition: **Two fundamental products are said to be** adjacent **if they have the same variables and if they differ in exactly one literal**. **Thus there must**

be an uncomplemented variable in one product which is complemented in the other.

For example, if $P_1 = x\ y\ z'$ and $P_2 = x\ y'\ z'$, then they are adjacent.

The sum of two such adjacent products will be a fundamental product with one less literal.

For example, in the case of above mentioned adjacent products,

$$P_1 + P_2 = x\ y\ z' + x\ y'\ z' = x\ z'(y + y') = x\ z'\ (1) = x\ z'.$$

We note that two squares in Karnaugh map above are adjacent if and only if squares are geometrically adjacent, that is, have a side in common.

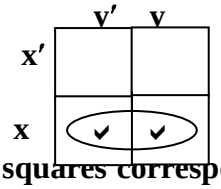
We know that a complete sum-of-products Boolean expression E(x, y) is a sum of minterms and hence can be represented in the Karnaugh map by placing checks in the appropriate square. A prime implicant of E(x, y) will be either a pair of adjacent squares **in E or an isolated square (a square which is not adjacent to other square of E(x, y)). A minimal sum of products for E(x, y) will consists of a minimal number of prime implicants which cover all the square of E(x, y).**

Example : **Find the prime implicants and a minimal sum-of-products form from each of the following complete sum-of-products Boolean expression:**

(a) $E_1 = x\ y + x\ y'$ (b) $E_2 = x\ y + x'\ y + x'\ y'$

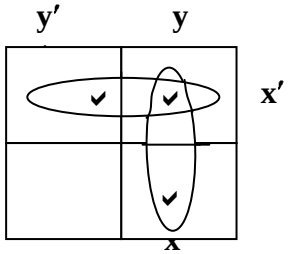
(c) $E_3 = x\ y + x'\ y'$.

Solution: **(a) The Karnaugh map for E_1 is**



Check the squares corresponding to x y and x y′. We note that E_1 consists of one prime implicant, the two adjacent square designated by the loop. The pair of adjacent square represents the variable x. So x is the only prime implicant of E_1. Consequently $E_1 = x$ is its minimal sum.

(b) **The Karnaugh map for E_2 is**

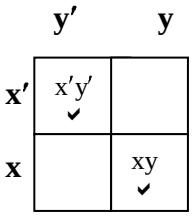


Check the squares corresponding to x y, x′ y, x′ y′. The expression E_2 contains two pairs of adjacent squares (designated by two loops) which include all the squares of E_2. The vertical pair represents y and the horizontal pair x′. Hence y and x′ are the prime implicants of E_2. Thus

$$E_2(x, y) = x' + y$$

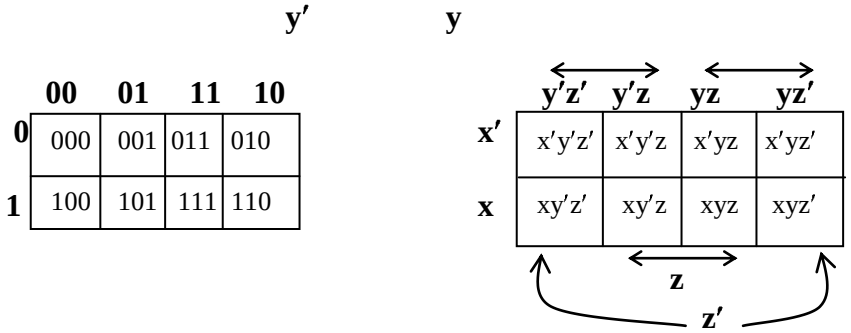
is minimal sum.

(c) **The Karnaugh map for E_3 is**



Check (tick) the squares corresponding to x y and x′ y′. The expression E_3 consists of two isolated squares which represent x y and x′ y′. Hence and x y and x′ y′ are the prime implicants of E_3 and so E_3 = x y + x′ y′ is its minimal sum.

Case of 3 variables: **We now turn to the case of a function f: $B_3 \rightarrow B$ which is function of x, y and z. The Karnaugh map corresponding to Boolean expression E(x, y, z) is shown in the diagram below:**



Here x, y, z are respectively represented by lower half, right half and middle two quarters of the map.

Similarly, x′, y′, z′ are respectively represented by upper half, left half and left and right quarter of the map.

Definition: **By a** Basic Rectangle **in the Karnaugh map with three variables, we mean a square, two adjacent squares or four squares which form a one-by four, or a two by-two rectangle**. **These basic rectangles corresponds to fundamental products of three, two and one literal respectively.**

Further, the fundamental product represented by a basic rectangle is the product of just those literals that appear in every square of the rectangle.

Let a complete sum of products Boolean expression E(x, y, z) is represented in the Karnaugh map by placing checks in the appropriate squares. A prime implicant of E will be a maximal basic rectangle of E,

i.e., a basic rectangle contained in E which is not contained in any larger basic rectangle in E.

A minimal sum-of-products form for E will consist of a minimal cover of E, i.e., a minimal number of maximal basic rectangles of E which together include all the squares of E.

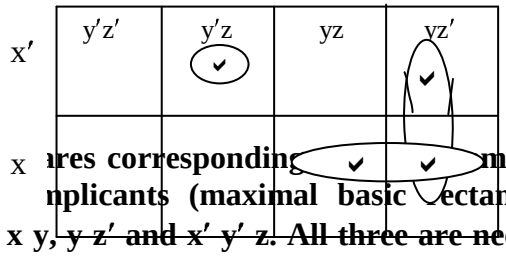
Example: Find the prime implicants and a minimal sum-of-products form for each of the following complete sum of products Boolean expressions :

(a) $E_1 = x\,y\,z + x\,y\,z' + x'\,y\,z' + x'\,y'\,z$

(b) $E_2 = x\,y\,z + x\,y\,z' + x\,y'\,z + x'\,y\,z + x'\,y'\,z$

(c) $E_3 = x\,y\,z + x\,y\,z' + x'\,y\,z + x'\,y'\,z$

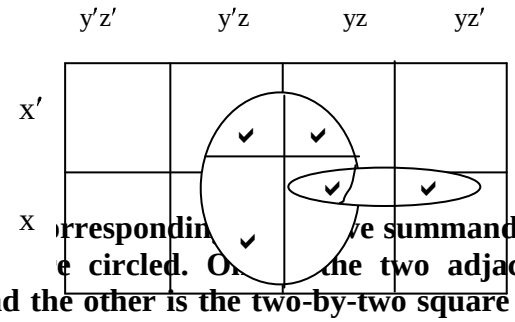
Solution: (a) **The Karnaugh map for E_1 is**



We check the four squares corresponding to the summands in E_1. Here E_1 has three prime implicants (maximal basic rectangles) which are encircled. These are x y, y z′ and x′ y′ z. All three are needed to cover E_1. Hence minimal sum for E_1 is

$$E_1 = x\,y + y\,z' + x'\,y'\,z.$$

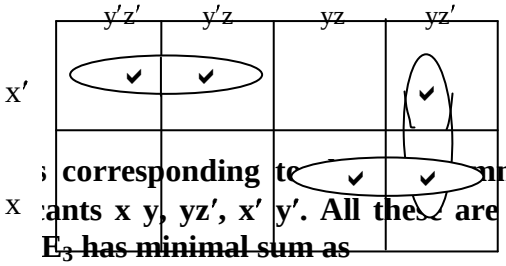
(b) The Karnaugh map for E_2 is



Check the squares corresponding to the summands. E_2 has two prime implicants which are circled. One is the two adjacent squares which represent x y, and the other is the two-by-two square which represents z. Both are needed to cover E_2 so the minimal sum for E_2 is

$E_2 = x\,y + z$

(c) The Karnaugh map for E_3 is



Check the squares corresponding to the summands. Here E_3 has three prime implicants x y, yz′, x′ y′. All these are needed in a minimal cover of E_3. Hence E_3 has minimal sum as

$$E_3 = x\,y + y\,z' + x'\,y'$$

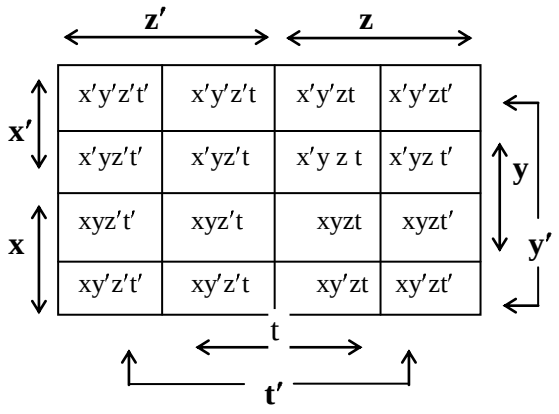
Remark : **To find the fundamental product represented by a basic rectangle, find literals which appear in all the squares of the rectangle.**

Case of 4 Variables: **We consider a Boolean function f : $B_4 \rightarrow B$, considered as a function of x, y, z and t. Each of the 16 squares (2^4) corresponds to one of the minterms with four variables.**

x y z t, x y z t′,……..,x′ y z′ t

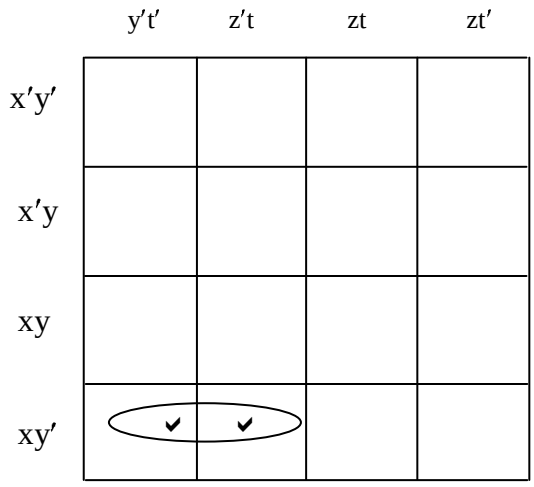
We consider first and last columns to be adjacent, and first and last rows to be adjacent, both by Wrap around, and we look for rectangles with sides of length some power of 2, so the length is 1, 2 or 4. The expression for such rectangles is given by intersecting the large labelled rectangles.

	00	**01**	**11**	**10**
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010



A basic rectangle in a four variable Karnaugh map is a square, two adjacent squares, four squares which form a one-by-four or two by two rectangle or eight square squares which form a two by four rectangle. These rectangle correspond to fundamental product with four, three, two and one literal respectively. Maximal basic rectangles are prime implicants.

Example: **Find the fundamental product P represented by the basic rectangle in the Karnaugh map given below :**



Solution: **We find the literals which appear in all the squares of the basic rectangle. Then P will be the product of such literals.**
Here x, y′, z′ appear in both squares. Hence

$$P = x\,y'\,z'$$

is the fundamental product represented by the basic rectangle in question.

Unit-3
Graph Theory

3.1. Definitions and Examples

Definition: A **graph** G = (V,E) is a mathematical structure consisting of two finite sets V and E. The elements of V are called **Vertices (or nodes)** and the elements of E are called Edges. Each edge
is associated with a set consisting of **either one** or **two vertices** called its **endpoints**.

The correspondence from edges to endpoints is called **edge-endpoint function**. This function is generally denoted by γ. Due to this function, some author denote graph by G = (V, E, γ).

Definition: A graph consisting of one vertex and no edges is called a **trivial graph**.

Definition: A graph whose vertex and edge sets are empty is called a **null graph**.

Definition: An edge with just one end point is called a **loop** or a **self loop**.
Thus, a loop is an edge that joins a single endpoint to itself.

Definition: An edge that is not a self-loop is called a **proper edge**.

Definition: If two or more edges of a graph G have the same vertices, then these edges are said to be
parallel or **multi-edges**.

Definition: Two vertices that are connected by an edge are called **adjacent**.

Definition: An endpoint of a loop is said to be **adjacent to itself**.

Definition: An edge is said to be **incident** on each of its endpoints.

Definition: Two edges incident on the same endpoint are called **adjacent edges**.

Definition: The number of edges in a graph G which are incident on a vertex is called the degree of
that **vertex**.

Definition: A vertex of degree zero is called an **isolated vertex**.
Thus, a vertex on which no edges are incident is called isolated.
Definition: A graph without multiple edges **(parallel edges)** and loops is called **Simple graph**.
Notation: In pictorial representations of a graph, the vertices will be denoted by dots and edges by line segments.

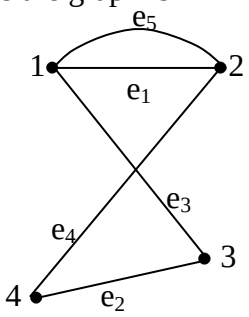
Example: 1. Let

$V = \{1, 2, 3, 4\}$ and $E = \{e_1, e_2, e_3, e_4, e_5\}$.

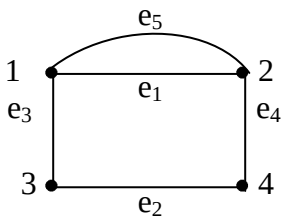
Let γ be defined by

$\gamma(e_1) = \gamma(e_5) = \{1, 2\}$
$\gamma(e_2) = \{4, 3\}$
$\gamma(e_3) = \{1, 3\}$
$\gamma(e_4) = \{2, 4\}$

We note that both edges e_1 and e_5 have same endpoints $\{1, 2\}$. The endpoints of e_2 are $\{4, 3\}$, the endpoints of e_3 are $\{1, 3\}$ and endpoints of e_4 are $\{2, 4\}$. Thus the graph is

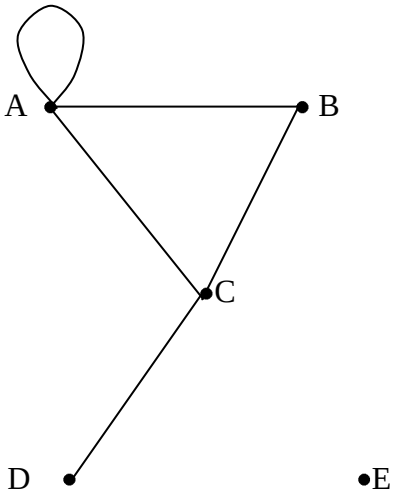


or



The edges e_2 and e_3 are adjacent edges because they are incident on the same vertex B.

2. Consider the graph with the vertices A, B , C, D and E pictured in the figure below.



In this graph, we note that

No. of edges = 5

Degree of vertex A = 4

Degree of vertex B = 2

Degree of vertex C = 3

Degree of vertex D = 1

Degree of vertex E = 0

Sum of the degree of vertices = 4 + 2 + 3 + 1 + 0 =10
Thus, we observe that

$$\sum_{i=1}^{5} \deg(v_i) = 2e \ ,$$

where $\deg(v_i)$ denotes the degree of vertex v_i and e denotes the number of edges.

Euler's Theorem: (The First Theorem of Graph Theory): The sum of the degrees of the vertices of a graph G is equal to twice the number of edges in G.

(Thus, total degree of a graph is even)

Proof: Each edge in a graph contributes a count of 1 to the degree of two vertices (end points of
the edge), That is, each edge contributes 2 to the degree sum. Therefore the sum of degrees of the
vertices is equal to twice the number of edges.

Corollary: There must be an even number of vertices of odd degree in a given graph G.
Proof: We know, by the Fundamental Theorem, that

$$\sum_{i=1}^{n} \deg(v_i) = 2 \times \text{no. of edges}$$

Thus the right hand side is an even number. Hence to make the left-hand side an even number there
can be only even number of vertices of odd degree.

Remarks: (i) A vertex of degree d is also called a **d-valent vertex**.
(ii) The degree (or valence) of a vertex v in a graph G is the number of proper edges incident on v
plus twice the number of self- loops.

Theorem: A non-trivial simple graph G must have at least one pair of vertices whose degrees are
equal.

Proof: Let the graph G has n vertices. Then there appear to be n possible degree values, namely 0, 1, ….,n – 1. But there cannot be both a vertex of degree 0 and a vertex of degree n – 1 because if there is a vertex of degree 0 then each of the remaining n − 1 vertices is adjacent to atmost n–2 other

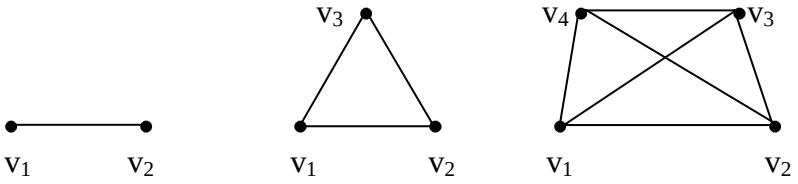
vertices. Hence the n vertices of G can realize atmost n–1 possible values for their degrees. Hence the pigeonhole principle implies that at least two of the vertices have equal degree.

Definition: A graph G is said to **simple** if it has no parallel edges or loops. In a simple graph, an edge with endpoints v and w is denoted by {v, w}.
Definition: For each integer $n \geq 1$, let D_n denote the graph with **n vertices** and **no edges**. Then D_n is called the **discrete graph on n vertices**.
For example, we have

D_3 D_5

Definition: Let $n \geq 1$ be an integer. Then a simple graph with n vertices in which there is an edge between each pair of distinct vertices is called the **complete Graph** on n vertices. It is denoted by K_n.

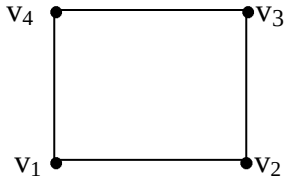
For example, the complete graphs K_2, K_3 and K_4 are shown in the figures below:



K_2 K_3 K_4

Definition: If each vertex of a graph G has the same degree as every other vertex, then G is called a **regular graph**.
A **k-regular graph** is a regular graph whose common degree is k.

For example, consider K_3. The degree of each vertex in K_3 is 2. Hence K_3 is regular. Similarly K_4 is regular. Also the graph shown below is regular because degree of each vertex here is 2 .

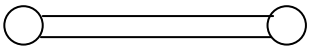


2- regular graph

But this graph is not complete because v_2 and v_4 have not been connected through an edge. Similarly, v_1 and v_3 are not connected by any edge.
Thus

A Complete graph is always regular but **a regular graph need not be complete**.

Example: The oxygen molecule O_2, made up of two oxygen atoms linked by a double bond can be represented by the regular graph shown below:



Definition: Let $n \geq 1$be an integer. Then a graph L_n with n vertices $\{v_1, v_2,\ldots,v_n\}$ and with edges
$\{v_i, v_{i+1}\}$ for $1 \leq i < n$ is called a **Linear Graph on n vertices**.

For example, the linear graphs L_2 and L_4 are shown in the figure below.

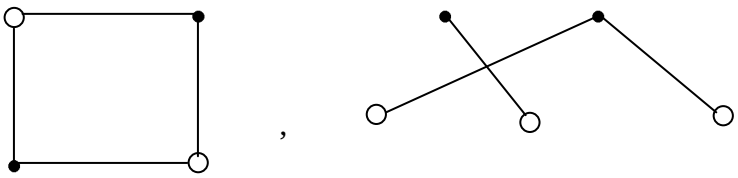
L_2**(or P_2)** L_4**(or P_4)**

It is also called **Path Graph** denoted by P_n.

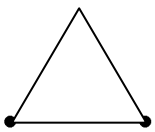
Definition: A **bipartite graph** G is a graph whose vertex set V can be partitioned into two subsets U and W, such that each edge of G has one endpoint in U and one endpoint in W.

The pair (U, W) is called a **Vertex bipartition of G** and U and W are called the bipartition subsets. Obviously, a bipartite graph cannot have any self loop.

Example: 1. If Vertices in U are solid vertices and vertices in W are hollow vertices, then the following graphs are bipartite graphs:



2. The **smallest possible simple graph that is not bipartite is the complete graph K_3 shown below :**



K_3

Definition: A complete bipartite graph G is a simple graph whose vertex set V can be partitioned into two subsets $U = \{v_1, v_2,\ldots,v_m)$ and $W = \{w_1, w_2,\ldots,w_n\}$such that for all i, k in $\{1, 2,\ldots,m\}$and j, *l* in $\{1, 2,\ldots n\}$

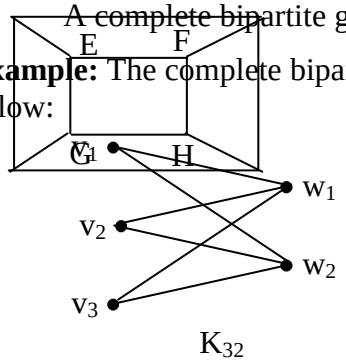
(i) there is an edge from each vertex v_i to each vertex w_j.

(ii) there is not an edge from any vertex v_i to any other vertex v_k.

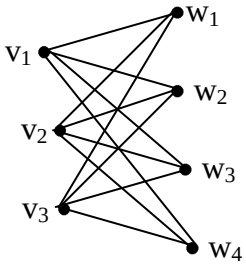
(iii) there is not an edge from any vertex w_j to any other vertex w_l.

A complete bipartite graph on (m, n) vertices is denoted by $K_{m,n}$.

Example: The complete bipartite graphs K_{32} and K_{34} are shown in the figure below:



K_{32}



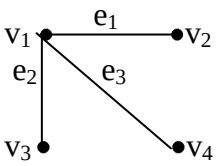
K_{34}

3.2. Subgraphs

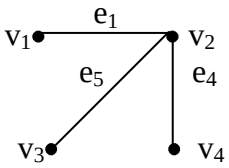
Definition: A graph H is said to be a subgraph of a graph G if and only if every vertex in H is also a vertex in G, every edge in H is also an edge in G and every edge in H has the same endpoints as in G.

We may also say that G is a supergraph of H.

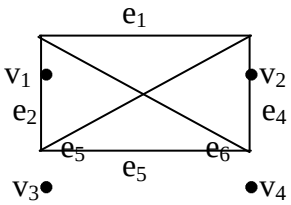
For example,



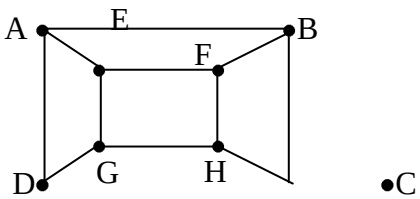
and



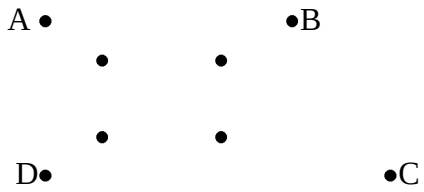
are subgraphs of the graph given below:



Similarly, the graph



is a subgraph of the graph given below:



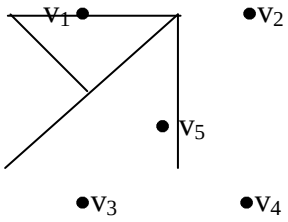
Definition: A subgraph H is said to be a **proper subgraph** of a graph G if vertex set V_H of H is a proper subset of the vertex set V_G of G or edge set E_H is a proper subset of the edge set E_G.

For example, the subgraphs in the above examples are proper subgraphs of the given graphs.

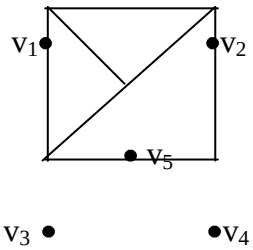
Definition: A subgraph H is said to **span** a graph G if $V_H = V_G$.

Thus H is a spanning sub graph of graph G if it contains all the vertices of G.

For example the subgraph

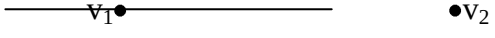


spans the graph

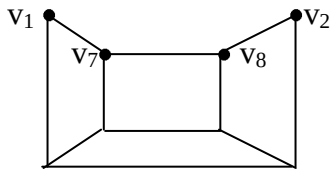


Definition: Let G = (V, E) be a graph. Then the **complement of a subgraph** G´ = (V´, E´) with respect to the graph G is another subgraph G´´ = (V´´, E´´) such that E´´ = E – E´ and V´´ contains only the vertices with which the edges in E´´ are incident.

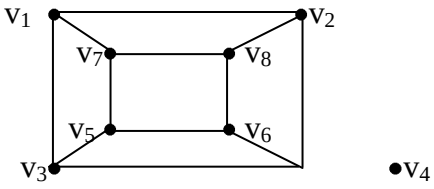
For example, the subgraph



is the complement of the subgraph



with respect to the graph G shown in the figure below:

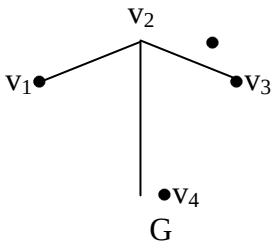


Definition: If G is a simple graph, the **complement of G**, (**Edge complement**), denoted by G′ or G^c is a graph such that

(i) The vertex set of G′ is identical to the vertex set of G, that is $V_{G'} = V_G$

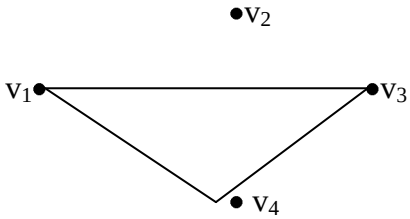
(ii) Two distinct vertices v and w of G′ **are connected by** an edge if and only if v and w **are not connected** by an edge in G.

For example, consider the graph G



G

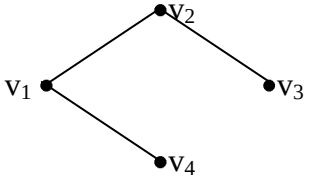
Then complement G′ of G is the graph



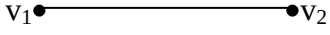
G′

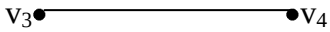
Example: Find the complement of the graphs:

(a)

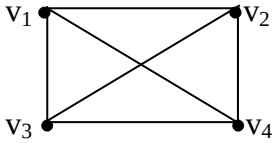


(b)

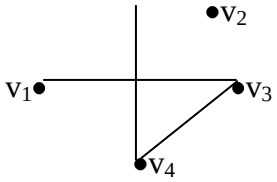




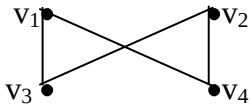
(c) Complete graph K_4 :



Solution: (a)

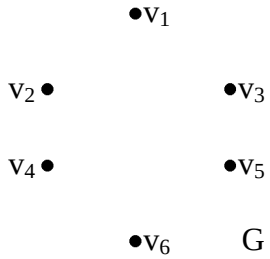


(b)



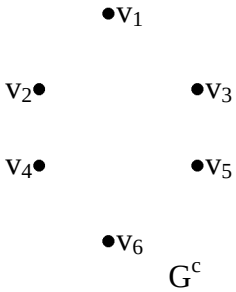
(c) Null graph.

Example: Find the edge complement of the graph G shown below:



G

Solution: The edge complement of G is the following graph G^c

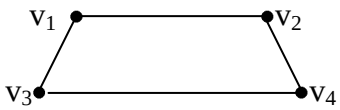


G^c

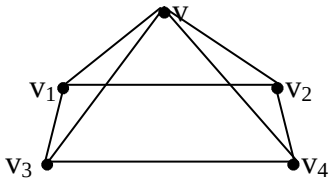
Definition: If a new vertex v is joined to each of the pre-existing vertices of a graph G, then the resulting graph is called the **Join of G and v** or the **suspension of G from v**. It is denoted by G + v.

Thus, A graph obtained by joining a new vertex to each of the vertices of a given graph G is called the **Join of G and v** or the **suspension of G from v**. It is denoted by G + v.

For example, let G be a graph



and let v be a vertex. Then

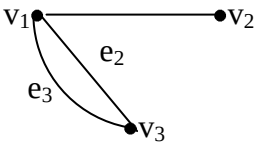


G + v

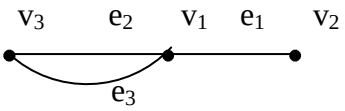
is the join of G to v.

3.3. Isomorphisms of Graphs

We know that shape or length of an edge and its position in space are not part of specification of a graph. For example, the figures



and



represent the same graph.

Definition: Let G and H be graphs with vertex sets V(G) and v(H) and Edge sets E(G) and E(H) respectively. Then **G is said to isomorphic to H** iff there exist one-to-one correspondences g : V(G) $\rightarrow$ v(H) and h : E(G) $\rightarrow$ E(H) such that for all v $\in$ V(G) and e $\in$ E(G),

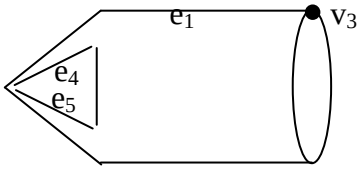
v is an endpoint of e $\Leftrightarrow$ g(v) is an endpoint of h(e).

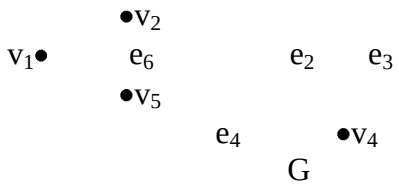
Definition: The property of mapping endpoints to endpoints is called **preserving incidence** or **the continuity rule** for graph mappings.

As a consequence of this property, a self-loop must map to a self-loop.

Thus, two isomorphic graphs are same except for the labeling of their vertices and edges.

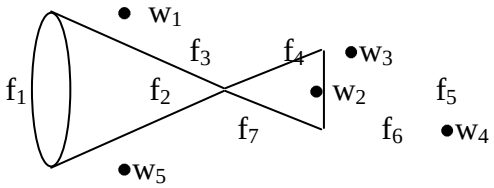
Example: Show that the graphs





G

and



H

are isomorphic.

Solution: To solve this problem, we have to find g: V(G) $\rightarrow$ V(H) and h : E(G) $\rightarrow$ E(H) such that for all v $\in$ V(G) and e $\in$ E(G),

$$v \text{ is an endpoint of } e \Leftrightarrow g(v) \text{ is an endpoint of } h(e).$$

Since e_2 and e_3 are parallel (have the same endpoints), $h(e_2)$ and $h(e_3)$ must also be parallel. Thus we have

$$h(e_2) = f_1 \text{ and } h(e_3) = f_2 \text{ or } h(e_2) = f_2 \text{ and } h(e_3) = f_1.$$

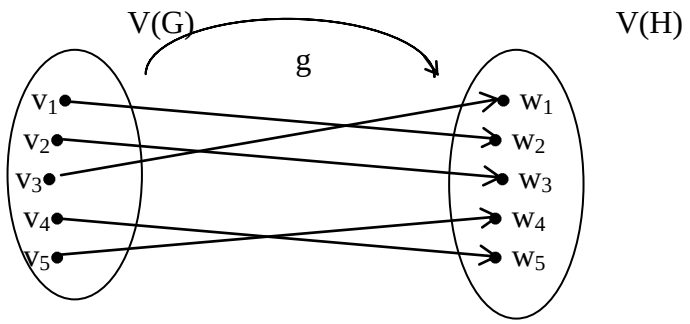
Also the endpoints of e_2 and e_3 must correspond to the endpoints of f_1 and f_2 and so

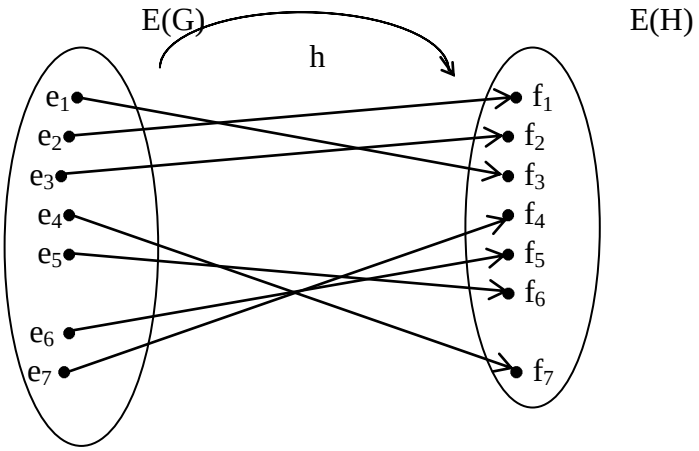
$$g(v_3) = w_1 \text{ and } g(v_4) = w_5 \text{ or } g(v_3) = w_5 \text{ and } g(v_4) = w_1.$$

Further, we note that v_1 is the endpoint of four distinct edges e_1, e_7, e_5 and e_4 .and so $g(v_1)$ should be the endpoint of form distinct edges. We observe that w_2 is the vertex having four edges and so $g(v_1) = w_2$. If $g(v_3) = w_1$, then since v_1 and v_3 are endpoints of e_1 in G, $g(v_1) = w_2$ and $g(v_3) = w_1$ must be endpoints of $h(e_1)$ in H. This implies that $h(e_1) = f_3$.

Continuing in this way we can find g and h to define the isomorphism between G and H.

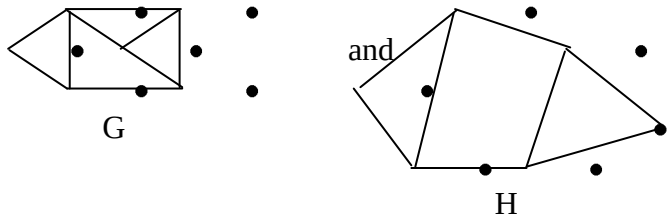
One such pair of functions (of course there exist several) is shown below:



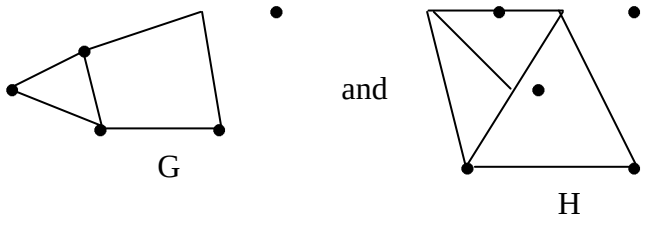


Remark: Each of the following properties is invariant under graph isomorphism, where n, m and h are all non-negative integers:
1. has n vertices
2. has m edges
3. has a vertex of degree k
4. has m vertices of degree k

Example: Examine for isomorphism
(a)



(b)



Solution: (a) G has nine edges whereas H has only eight edges. Hence G is not isomorphic to H.
(b) G has a vertex v of degree 4, whereas H has no vertex of degree 4. Hence G is not isomorphic to H

3.4 Walks, Paths and Circuits

Definition: In a graph G, **a walk from vertex v_0 to vertex v_n** is a finite alternating sequence:

$$\{v_0, e_1, v_1, e_2, \ldots\ldots, v_{n-1}, e_n, v_n\}$$

of vertices and edges such that v_{i-1} and v_i are the endpoints of e_i.

The **trivial walk** from a vertex v to v consists of the single vertex v.

Definition: In a graph G, a **path** from the vertex v_0 to the vertex v_n is a walk from v_0 to v_n that does not contain a repeated edge.

Thus a **path** from v_0 to v_n is a walk of the form

$$\{v_0, e_1, v_1, e_2, v_2, \ldots\ldots, v_{n-1}, e_n, v_n\},$$

where all the edges e_I are distinct.

Definition: In a graph, a simple path from v_0 to v_n is a path that does not contain a repeated vertex.

Thus a simple path is a walk of the form

$$\{v_0, e_1, v_1, e_2, v_2, \ldots\ldots, v_{i-1}, e_n, v_n\},$$

where all the e_i are distinct and all the v_i are distinct.

Definition: A walk in a graph G that starts and ends at the same vertex is called a **closed walk**.

Definition: A closed walk that does not contain a repeated edge is called a **circuit**.

Thus, closed a closed path is called a circuit (or a cycle) and so a circuit is a walk of the form

$$\{v_0, e_1, v_1, e_2, v_2, \ldots\ldots, v_{n-1}, e_n, v_n\} \ ,$$

where $v_0 = v_n$ and all the e_i are distinct.

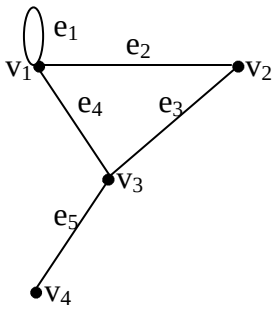
Definition: A **simple circuit** is a circuit that does not have any other repeated vertex except the first and the last.

Thus, a simple circuit is a walk of the form

$$\{v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n\},$$

where all the e_I are distinct and all the v_j are distinct except that $v_0 = v_n$.

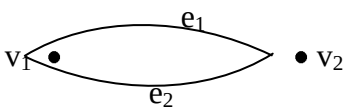
Example: Consider the graph shown below



We note that e_3, e_5 is a path. The walk e_1, e_2, e_3, e_5 is a path but it is not a simple path because the vertex v_1 is repeated (e_1 being a self-loop). The walk e_2, e_3, e_4 is a circuit. The walk e_2, e_3, e_4, e_1 is a circuit but it is not simple circuit because vertex v, repeats twice (or we may write that v_1 met twice).

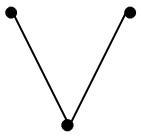
Definition: In a graph the number of edges in the path $\{v_0, e_1, v_1, e_2, \ldots\ldots, e_n, v_n\}$ from v_0 to v_n is called the **length of the path**.

Definition: A cycle with k-edges is called a **k-cycle** or **cycle of length k**.
For example, loop is a cycle of length 1. On the other hand, a pair of parallel edges e_1 and e_2, shown below, is a cycle of length 2

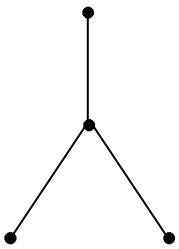


Definition: A graph is said to be **acyclic** if it contains no cycle.

For example, the graphs



and



are acyclic.

Theorem: If there is a path from vertex v_1 to v_2 in a graph with n vertices, then there does not exist a path of more than n-1 edges from vertex v_1 to v_2.

Proof: Suppose there is a path from v_1 to v_2. Let

$$v_1,\ldots\ldots,v_i,\ldots\ldots,v_2$$

be the sequence of vertices which the path meets between the vertices v_1 and v_2. Let there be m edges in the path. Then there will be m + 1 vertices in the sequence. Therefore if m > n–1, then there will be more than n vertices in the sequence. But the graph is with n vertices. Therefore some vertex, say v_k, appears more than once in the sequence. So the sequence of vertices shall be

$$v_1,\ldots\ldots,v_i,\ldots\ldots,v_k,\ldots..,v_k,\ldots\ldots,v_2.$$

Deleting the edges in the path that lead v_k back to v_k we have a path from v_1 to v_2 that has less edges than the original one. This argument is repeated untill we get a path that has n-1 or less edges.

Definition: Two vertices v_1 and v_2 of a graph G are said to be **connected** if and only if there is a walk from v_1 to v_2.

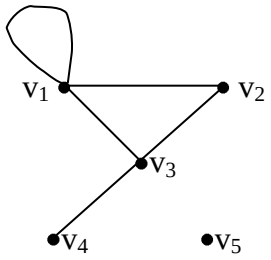
Definition: A graph G is said to be **connected** if and only if given any two vertices v_1 and v_2 in G, there is a walk from v_1 to v_2.

Thus, a graph G is connected if there exists a walk between every two vertices in the graph.

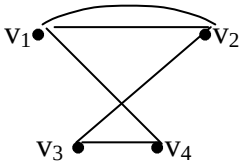
Definition: A graph which is not connected is called **Disconnected Graph.**

Example: Which of the graph below are connected?

(a)



(b)

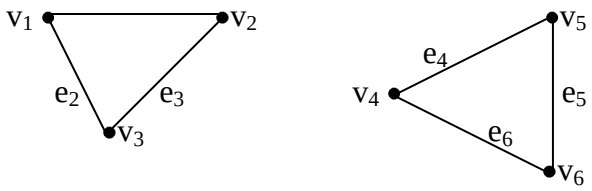


Solution: Graph (a) is not connected as there is no walk from any of v_1, v_2, v_3, v_4 to the vertex v_5.

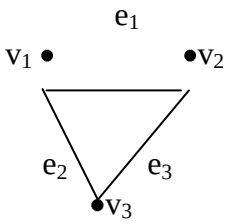
The graph (b) is clearly connected.

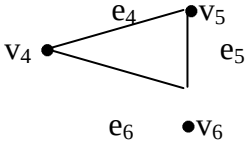
Definition: If a graph G is disconnected, then the various connected pieces of G are called the **connected components of the graph**.

Example: Consider the graph given below:

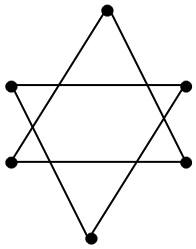


This graph is disconnected and have two connected components:

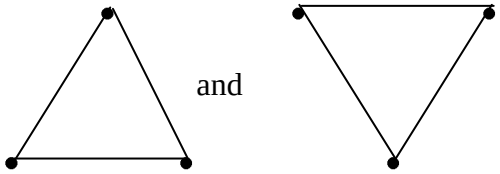
H_1 :  with vertex set $\{v_1, v_2, v_3\}$ and edge set $\{e_1, e_2, e_3\}$

H_2 :  with vertex set $\{v_4, v_5, v_6\}$ and edge set $\{e_4, e_5, e_6\}$.

Example: Find the number of connected components in the graph



Solution: The connected components are :



Remark: If a connected component has n vertices, then degree of any vertex cannot exceed n-1.

3.5. Eulerian Paths And Circuits

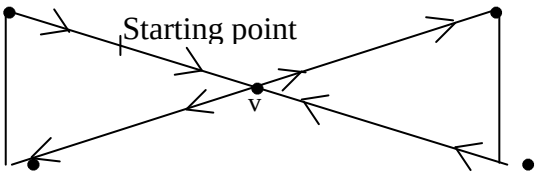
Definition: A path in a graph G is called an **Euler Path** if it includes **every edge exactly once**.

Definition: A circuit in a graph G is called an **Euler Circuit** if it includes every edge exactly once. Thus, an Euler circuit (Eulerian trail) for a graph G is a sequence of adjacent vertices and edges in G that starts and ends at the same vertex, uses every vertex of G at least once, and uses **every edge of G exactly once**.

Definition: A graph is called **Eulerian graph** if there exists a Euler circuit for that graph.

Theorem 1. If a graph has an Euler circuit, then every vertex of the graph has even degree.

Proof: Let G be a graph which has an Euler circuit. Let v be a vertex of G. We shall show that degree of v is even. By definition, Euler circuit contains every edge of graph G. Therefore the Euler circuit contains all edges incident on v. We start a journey beginning in the middle of one of the edges adjacent to the start of Euler circuit and continue around the Euler circuit to end in the middle of the starting edge. Since Euler circuit uses every edge exactly once, the edges incident on v occur



in entry / exist pair and hence the degree of v is a multiple of 2. Therefore the degree of v is even. This completes the proof of the theorem.

We know that contrapositive of a conditional statement is logically equivalent to statement. Thus the above theorem is equivalent to:

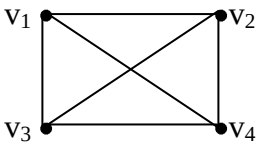
Theorem:2. If a vertex of a graph is not of even degree, then it does not have an Euler circuit.

or

"If some vertex of a graph has odd degree, then that graph does not have an Euler circuit".

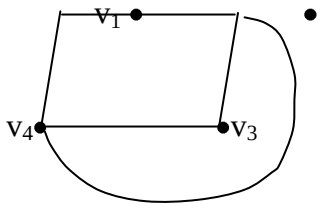
Example: Show that the graphs below do not have Euler circuits.

(a)



(b)

v_2



Solution: In graph (a), degree of each vertex is 3. Hence this **does not** have a Euler circuit.

In graph (b), we have

$$\deg(v_2) = 3$$

$$\deg(v_4) = 3$$

Since there are vertices of odd degree in the given graph, therefore it **does not** have an Euler circuit.

Remark: The converse of Theorem 1 is not true. There exist graphs in which every vertex has even degree but the Euler circuits do not exist.

For example,

and

are graphs in which each vertex has degree 2 but these graphs do not have Euler circuits since there is no path which uses each vertex at least once.

Theorem 3. If G is a connected graph and every vertex of G has even degree, then G has an Euler circuit.

Proof: Let every vertex of a connected graph G has even degree. If G consists of a single vertex, the trivial walk from v to v is an Euler circuit. So suppose G consists of more than one vertices. We start from any verted v of G. Since the degree of each vertex of G is even, if we reach each vertex other than v by travelling on one edge, the same vertex can be reached by travelling on another previously unused edge. Thus a sequence of distinct adjacent edges can be produced indefinitely as long as v is not reached. Since number of edges of the graph is finite (by definition of graph), the sequence of distinct edges will terminate. Thus the sequence must return to the starting vertex. We thus obtain a sequence of adjacent vertices and edges starting and ending at v without repeating any edge. Thus we get a circuit C.

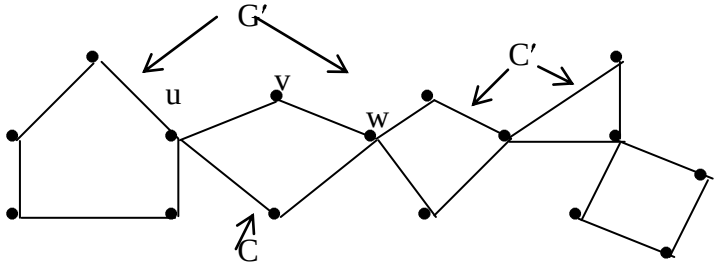
If C contains every edge and vertex of G, then C is an Eular circuit.

If C does not contain every edge and vertex of G, remove all edges of C from G and also any vertices that become isolated when the edges of C are removed. Let the resulting subgraph be G′. We note that when we removed edges of C, an even number of edges from each vertex have been removed. Thus degree of each remaining vertex remains even.

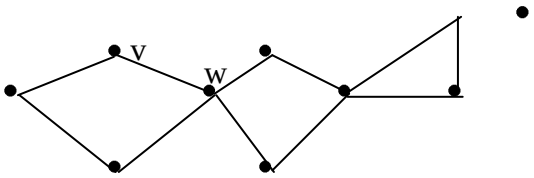
Further since G is connected, there must be at least one vertex common to both C and G′. Let it be w(in fact there are two such vertices). Pick any sequence of adjacent vertices and edges of G′ starting and ending at w without repeating an edge. Let the resulting circuit be C′.

Join C and C′ together to create a new circuit C″. Now, we observe that if we start from v and follow C all the way to reach w and then follow C′ all the way to reach back to w. Then continuing travelling along the untravelled edges of C, we reach v.

If C″ contains every edge an vertex of C, then C″ is an Euler circuit. If not, then we again repeat our process. Since the graph is finite, the process must terminate.
The process followed has been described in the graph G shown below:



G



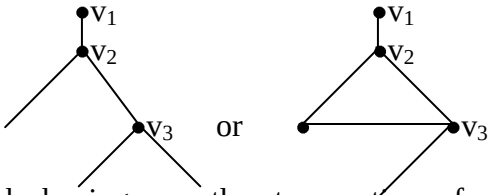
C″

Theorems 1 and 3 taken together imply :

Theorem 4. (Euler Theorem) A finite connected graph G has an Euler circuit if and only if every vertex of G has even degree.
Thus finite connected graph is Eulerian if and only if each vertex has even degree.

Theorem 5. If a graph G has more than two vertices of odd degree, then there can be no Euler path in G.
Proof : Let v_1, v_2 and v_3 be vertices of odd degree. Since each of these vertices had odd degree, any possible Euler path must leave (arrive at) each of v_1, v_2, v_3 with no way to return (or leave). One vertex of these three vertices may be the

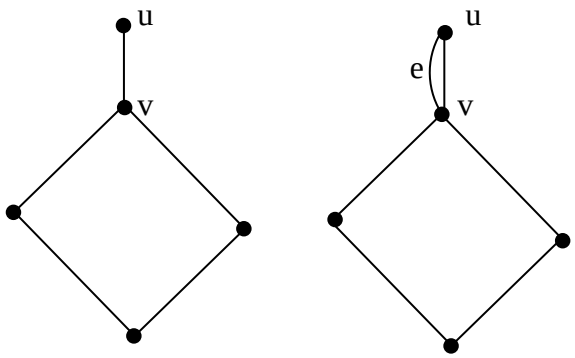
beginning of Euler path and another the end but this leaves the third vertex at one end of an untravelled edge. Thus there is no Euler path.



(Graphs having more than two vertices of odd degree).

Theorem 6. If G is a connected graph and has exactly two vertices of odd degree, then there is an Euler path in G. Further, any Euler path in G must begin at one vertex of odd degree and end at the other.

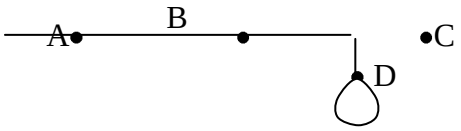
Proof: Let u and v be two vertices of odd degree in the given connected graph G.



G G′

If we add the edge e to G, we get a connected graph G′ all of whose vertices have even degree. Hence there will be an Euler circuit in G′. If we omit e from Euler circuit, we get an Euler path beginning at u(or v) and edning at v(or u).

Examples. Has the graph given below an Eulerian path?



Solution: In the given graph,

deg(A) = 1

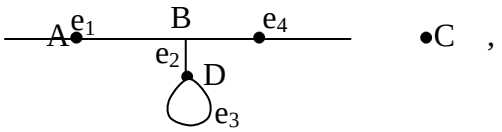
deg(B) = 2

deg(C) = 2

deg(D) = 3

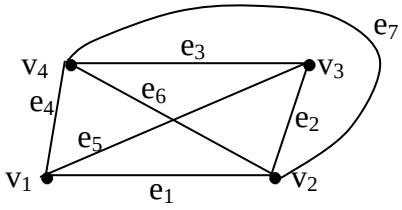
Thus the given connected graph has exactly two vertices of odd degree. Hence, it has an Eulerian path.

If it starts from A(vertex of odd degree), then it ends at D(vertex of odd degree). If it starts from D(vertex of odd degree), then it ends at A(vertex of odd degree).
But on the other hand if we have the graph as given below :



then deg(A) = 1, deg(B) = 3 deg(C) = 1, degree of D = 3 and so we have four vertices of odd degree. Hence it does not have Euler path.

Example: Does the graph given below possess an Euler circuit?



Solution: The given graph is connected. Further

$$\deg(v_1) = 3$$

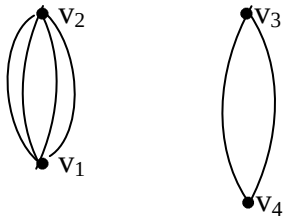
$$\deg(v_2) = 4$$

$$\deg(v_3) = 3$$

$$\deg(v_4) = 4$$

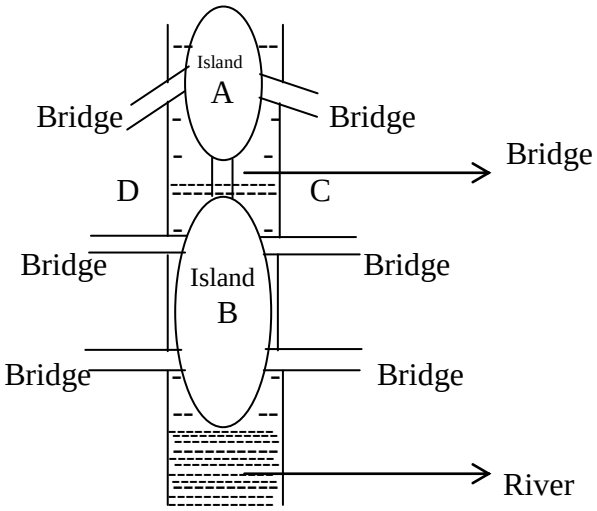
Since this connected graph has vertices with odd degree, it cannot have Euler circuit. But this graph has Euler path, since it has exactly two vertices of odd degree. For example, $v_3\, e_2\, v_2\, e_7\, v_4\, e_6\, v_2\, e_1\, v_1\, e_4\, v_4\, e_3\, v_3\, e_5\, v_1$

Example: Consider the graph



Here, $\deg(v_1) = 4$, $\deg(v_2) = 4$, $\deg(v_3) = 2$, $\deg(v_4) = 2$. Thus degree of each vertex is even. But the graph is not Eulerian since it is **not connected**.

Example 4:. **The bridges of Konigsberg:** The graph Theory began in 1736 when Leonhard Euler solved the problem of seven bridges on Pregel river in the town of Konigsberg in Prussia (now Kaliningrad in Russia). The two islands and seven bridges are shown below:



The people of Konigsgerg posed the following question to famous Swiss Mathematician Leonhard Euler:
"Beginning anywhere and ending any where, can a person walk through the town of Konigsberg crossing all the seven bridges exactly once?

Euler showed that such a walk is impossible. He replaced the islands A, B and the two sides (banks) C and D of the river by vertices and the bridges as edges of a graph. We note then that

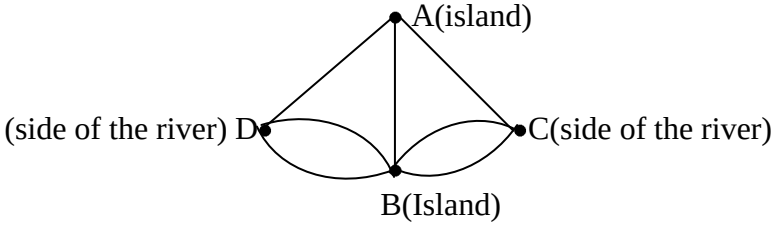
$$\deg(A) = 3$$

$$\deg(B) = 5$$

$$\deg(C) = 3$$

$$\deg(D) = 3$$

Thus the graph of the problem is



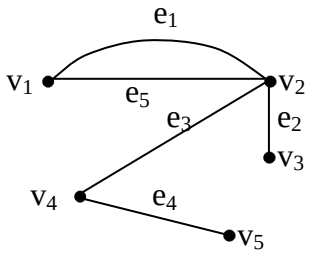
(Euler's graphical representation of seven bridge problem)

The problem then reduces to

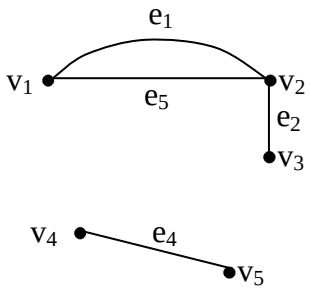
"Is there any Euler's path in the above diagram?".

To find the answer, we note that there are more than two vertices having odd degree. Hence there exist no Euler path for this graph.

Definition: An edge in a connected graph is called a **Bridge** or a **Cut Edge** if deleting that edge creates a disconnected graph.
For example, consider the graph shown below:



In this graph, if we remove the edge e_3, then the graph breaks into two Connected Component given below:

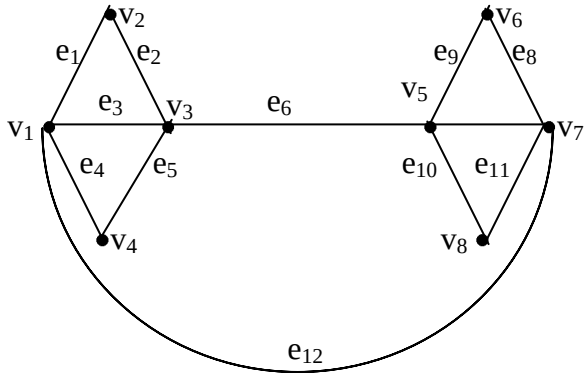


Hence the edge e_3 is a bridge in the given graph.

METHOD FOR FINDING EULER CIRCUIT

We know that if every vertex of a non empty connected graph has even degree, then the graph has an Euler circuit. We shall make use of this result to find an Euler path in a given graph.

Consider the graph



We note that

$$\deg(v_2) = \deg(v_4) = \deg(v_6) = \deg(v_8) = 2$$

$$\deg(v_1) = \deg(v_3) = \deg(v_5) = \deg(v_7) = 4$$

Hence all vertices have even degree. Also the given graph is connected. Hence the given has an Euler circuit. We start from the vertex v_1 and let C be

$$C : v_1\ v_2\ v_3\ v_1$$

Then C is not an Euler circuit for the given graph but C intersect the rest of the graph at v_1 and v_3. Let C′ be

$$C' : v_1 v_4\ v_3\ v_5\ v_7\ v_6\ v_5\ v_8\ v_7\ v_1$$

(In case we start from v_3, then C′ will be $v_3\ v_4\ v_1\ v_7\ v_6\ v_5\ v_7\ v_8\ v_5$)
Path C′ into C and obtain

$$C'' : v_1 v_2\ v_3\ v_1\ v_4\ v_3\ v_5\ v_7\ v_6\ v_5\ v_8\ v_7\ v_1$$

Or we can write

$$C'' : e_1 e_2\ e_3\ e_4\ e_5\ e_6\ e_7\ e_8\ e_9\ e_{10}\ e_{11}\ e_{12}$$

(If we had started from v_2, then $C'' : v_1 v_2\ v_3\ v_4\ v_1\ v_7\ v_6\ v_5\ v_7\ v_8\ v_5\ v_3\ v_1$ **or** $e_1 e_2\ e_5\ e_4\ e_{12}\ e_8\ e_9\ e_7\ e_{11}\ e_{10}\ e_6\ e_3$)

In C″ all edges are covered exactly once. Also every vertex has been covered at least once. Hence C″ is a Euler circuit.

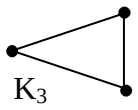
3.6. Hamiltonian Circuits

Definition: A **Hamiltonian Path** for a graph G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once.

Definition: A **Hamiltonian Circuit** for a graph G is a sequence of adjacent vertices and distinct edges in which every vertex of G appears exactly once, except for the first and the last which are the same.

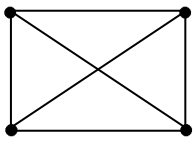
Definition: A graph is called **Hamiltonian** if it admits a Hamiltonian circuit.

Example 1 : A complete graph K_n has a Hamiltonian Circuit. In particular the graphs



K_3

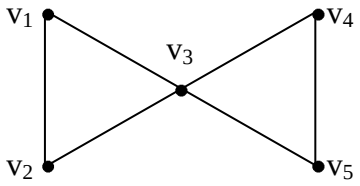
and



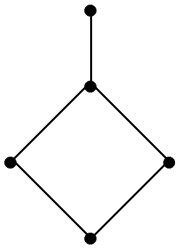
K_4

are Hamiltonian.

Example 2: The graph shown below does not have a Hamiltonian circuit.



Example 3 : The graph



does not have a Hamiltonian circuit.

Remark: It is clear that **only connected graphs can have Hamiltonian circuit**. However, there is no simple criterion to tell us whether or not a given graph has Hamiltonian circuit. The following results give us some sufficient conditions for the existence of Hamiltonian Circuit/path.

Theorem: Let G be a linear graph of n vertices. If the sum of the degrees for each pair of vertices in G is greater than or equal to n –1, then there exists a Hamiltonian path in G.

Theorem: Let G be a connected graph with n vertices. If $n \geq 3$ and $\deg(v) \geq n$ for each vertex v in G, then G had a Hamiltonian Circuit.

Theorem: Let G be a connected graph with n vertices and let u and v be two vertices of G that are not adjacent. If

$$\deg(u) + \deg(v) \geq n,$$

then G has a Hamiltonian circuit.

Corollary : Let G be a connected graph with n vertices. If each vertex has degree greater than or equal to n/2, then G has a Hamiltonian circuit.

Proof: It is given that degree of each vertex is greater than or equal to n/2. Hence the sum of the degree of any two vertices is greater than or equal to n/2 + n/2 = n. So, by the above theorem, the graph G has a Hamiltonian circuit.

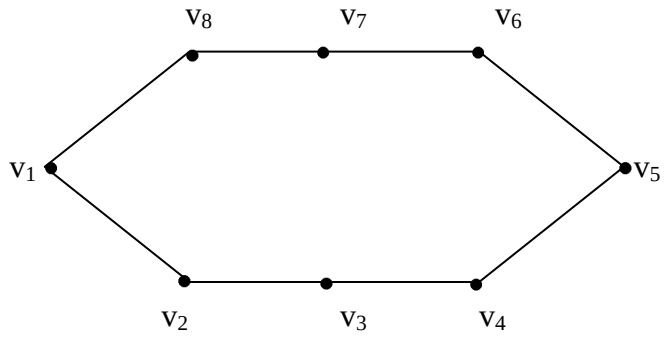
Theorem: Let n be the number of vertices and m be the number of edges in a connected graph G. If

$$m \geq \frac{1}{2}(n^2 - 3n + 6),$$

then G has a Hamiltonian circuit.

The following example shows that the above conditions are not necessary for the existence of Hamiltonian path.

Example : Let G be the connected graph shown in the figure below:



We note that

No. of Vertices in G (n) = 8

No. of Edges in G (m) = 8

Degree of each vertex = 2

Thus, if u and v are non-adjacent vertices, then

$$\deg u + \deg v = 2 + 2 = 4 \not\geq 8$$

Also

$$\frac{1}{2}(n^2 - 3n + 6) = 1/2\ (64 - 24 + 6) = 23$$

Clearly

$$m \not\geq \frac{1}{2}\ (n^2 - 3n + 6)$$

Therefore the above two theorems fail. But the given graph has Hamiltonian circuit. For example,

$$v_1\ v_2\ v_3\ v_4,\ v_5,\ v_6,\ v_7,\ v_8,\ v_1$$

is an Hamiltonian circuit for the graph.

Proposition: Let G be a graph with at least two vertices. If G has a Hamiltonian circuit, then G has a subgraph H with the following properties:

(1) H contains every vertex of G

(2) H is connected

(3) H has the same number of edges as vertices

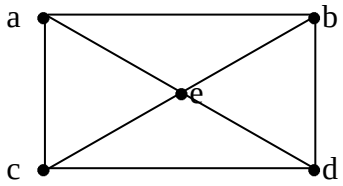
(4) Every vertex of H has degree 2.

The contrapositive of this proposition is

"If a graph G with at least two vertices does not have a subgraph H satisfying (1) – (4), then G does not have a Hamiltonian circuit".

Also we know that contrapositive of a statement is logically equivalent to the statement. Therefore the above result can be used to show non-existence of a Hamiltonian Circuit.

Example 1: Does the graph G given below have Hamiltonian circuit?



Solution: The given graph has

No. of vertices (n) = 5

No. of edges (m) = 8

deg(a) = deg(b) = deg(c) = deg(d) = 3

deg(e) = 4

We observe that

(i) degree of each vertex is greater than n/2

(ii) The sum of any non-adjacent pair of vertices is greater than n

(iii) $\frac{1}{2}(n^2 - 3n + 6) = \frac{1}{2}(25 - 15 + 6) = 8$

Thus the condition

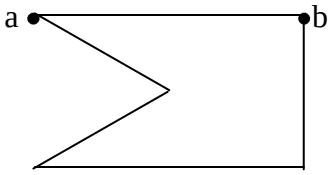
$$m \geq \frac{1}{2}(n^2 - 3n + 6)$$

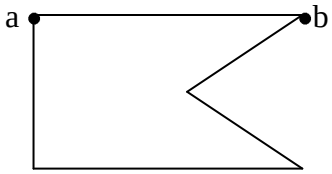
is satisfied.

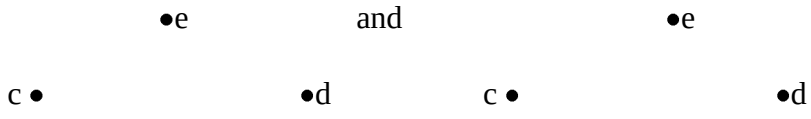
(iv) The sum of degrees of each pair of vertices in the given graph is greater than n–1 = 5–1 = 4.

Thus four sufficiency condition are satisfied (whereas one condition out of these four conditions is sufficient for the existence of Hamiltonian path/graph). Hence the graph has a Hamiltonian Circuit.

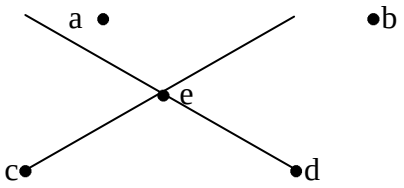
For example, the following circuits in G are Hamiltonian:







Example 2 : Does the graph shown below has Hamiltonian circuit?



Solution: Here

No. of vertices (n) = 5

No. of edge (m) = 4

deg(a) = deg(b) = deg(c) = deg(d) = 1

deg(e) = 4

We note that

(i) $\deg(a) = \deg(b) = \deg(c) = \deg(d) \not\geq \frac{5}{2}$

(ii) deg(a) + deg(b) = 2 $\not\geq$ 5, that is sum of any non-adjacent pair of vertices is not greater than 5

(iii) $\frac{1}{2}(n^2 - 3n + 6) = \frac{1}{2}(25 - 15 + 6) = 8$. Therefore the condition

$$m \geq 1/2\,(n^2 - 3n + 6)$$

is **not satisfied**.

(iv) deg(a) + deg(b) = 2 $\not\geq$ 4, i.e., the condition that sum of degrees of each pair of vertices in the graph is not greater than or equal to n–1.

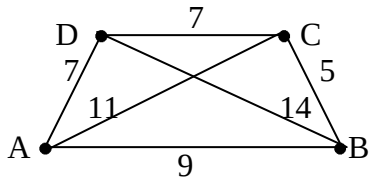
Hence no sufficiency condition is satisfied. So we try the proposition stated above.

Suppose that G has a Hamiltonian circuit, then G should a subgraph which contains every vertex of G, and number of vertices and no. of edges in H should be same. Thus H should have 5 vertices a, b, c, d, e and 5 edges. Since G has only 4 edges, H cannot have more than 4 edges. Hence no such subgraph is possible. Hence, the given graph does not have Hamiltonian circuit.

3.7. Weighted Graphs

Definition: A **weighted graph** is a graph for which each edge or each vertex or both is (are) labeled with a numerical value, called its **weight**.

For example, if vertices in a graph denote recreational sites of a town and weights of edges denote the distances in kilometers between the sites, then the graph shown below is a weighted graph.



Definition: The **weight of an edge** (v_i, v_j) is called **distance between the vertices v_i and v_j**.

Definition: A vertex u is a **nearest neighbour** of vertex v in a graph if u and v are adjacent and no other vertex is joined to v by an edge of lesser weight than (u, v).

For example, in the above example, B is the nearest neighbour of C, whereas A and C are both nearest neighbour of the vertex D. **Thus nearest neighbour of a set of vertices is not unique.**

Definition: A vertex u is a nearest neighbour of a set of vertices $\{v_1, v_2, \ldots, v_n\}$ in a graph if u is adjacent to some member v_i of the set and no other vertex adjacent to member of the set is joined by an edge of lesser weight then (u, v_i).

In the above example if we have set of vertices as {B, D}, C is the neatest neighbour of {B, D} because the edge (C, B) has weight 5 and no other vertex adjacent to {B, D} is linked by an edge of lesser weight than (C, B).

Definition: The **length of a path** in a graph is the sum of lengths of edges in the path.

Definition: Let G (V, E) be a graph and let l_{ij} denote the length of edge (v_i, v_j) in G. Then a **shortest path** from v_i to v_k is a path such that the sum of lengths of its edges

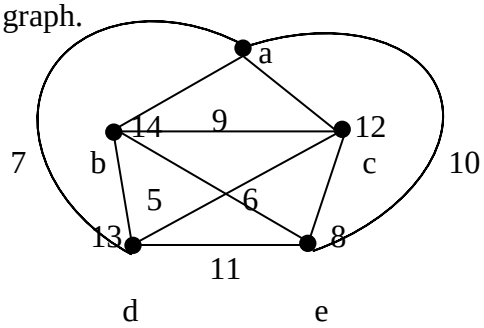
$$l_{12} + l_{23} + \ldots\ldots\ldots + l_{k-1,k}$$

is **minimum**, that is, total edge weight is minimum.

TRAVELLING SALESPERSON PROBLEM

This problem requires the determination of a **shortest Hamiltonian circuit** in a given graph of cities and lines of transportation to minimize the total fare for a travelling person who wants to made a tour of n cities visiting each city exactly once before returning home.
The weighted graph model for this problem consists of vertices representing cities and edges with weight as distances (fares) between the cities. The salesman starts and end his journey at the same city and visits each of n –1 cities once and only once. We want to find minimum total distance.

We discuss the case of five cities and so consider the following weighted graph.



We shall use **Nearest Neighbour** algorithm to solve the problem:

Algorithm: Nearest Neighbour (closest insertion)

Input: a weighted complete graph G
Output: a sequence of labeled vertices that forms a Hamiltonian cycle.

Start at any vertex v.

Initialize l(v) = 0

Initialize i = 0

While there are unlabeled vertices

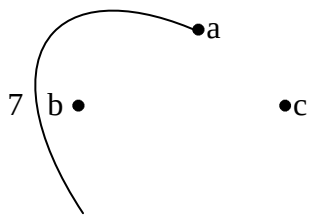
i : = i + 1

Traverse the cheapest edge that join v to an unlabeled vertex, say w

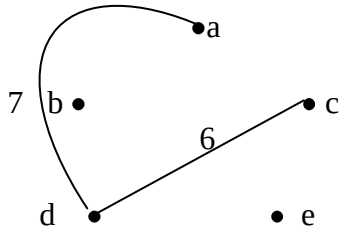
Set l(w) = i

v : = w.

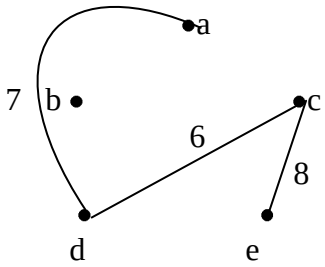
For the present example,
(i) Let us choose a as the starting vertex. Then d is the nearest vertex and then (a, d) is the corresponding edge. Thus we have the figure



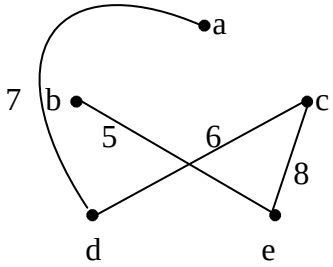
(ii) From d, the nearest vertex is c, so we have a path shown below:



(iii) From c, the nearest vertex is e. So we have the path as show below:

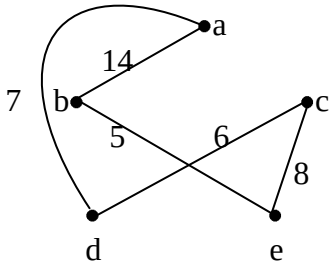


(iv) From e, the nearest vertex is b and so we have the path

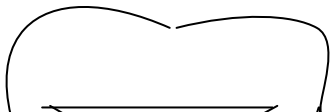


(v) Now, from b, the only vertex to be covered is a to **form Hamiltonian circuit.** Thus we have a Hamiltonian circuit as given below. The length of this Hamiltonian circuit is

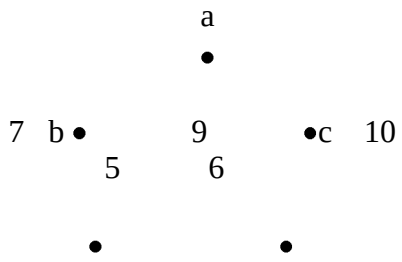
$$7 + 6 + 8 + 5 + 14 = 40.$$



However, **this is not Hamiltonian circuit of minimal length.**



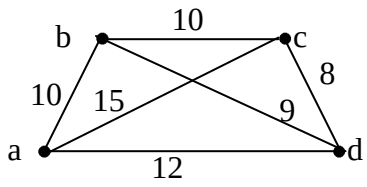
The **total distance** of a **minimum Hamiltonian** circuit (shown below) **is 37**.



Total length = 7 + 6 + 9 + 5 + 10 = 37

Remark: Unless otherwise stated, try to start from a vertex of largest weight.

Example 2: Find a Hamiltonian circuit of minimal weight for the graph shown below:



Solution: Starting from the point a and using nearest neighbour method, we have the required Hamiltonian circuit as

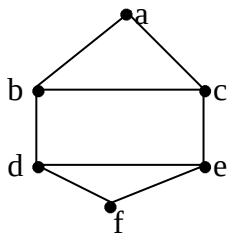
a b c d a

with total length as

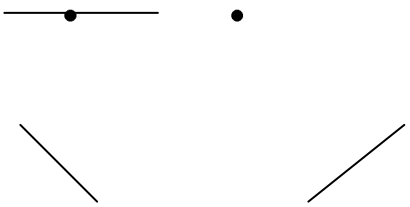
10 + 10 + 8 + 12 = 40

Definition: A **k-factor of a graph** is a spanning subgraph of the graph **with the degree of its vertices being k.**

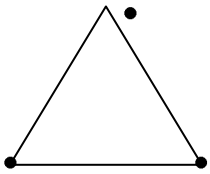
Consider the graph

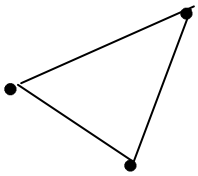


Then



shows a 1-factor of the given graph.
Also then,





is a 2-factor of the given graph.

3.8. Matrix Representation of Graphs

A graph can be represented inside a computer by using the adjacency matrix or the incidence matrix of the graph.

Definition: Let G be a graph with n ordered vertices $v_1, v_2,\ldots\ldots,v_n$. Then the **adjacency matrix of G** is the $n \times n$ matrix $A(G) = (a_{ij})$ over the set of non-negative integers such that

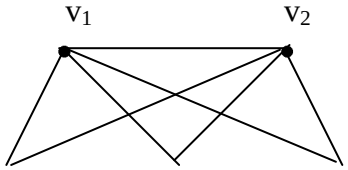
a_{ij} = the number of edges connecting v_i and v_j for all i, j = 1, 2,…,n.

We note that if G has no loop, then there is no edge joining v_i to v_i, i = 1, 2,…,n. Therefore, in this case, all the entries on the main diagonal will be 0.

Further, if G has no parallel edge, then the entries of A(G) are either 0 or 1.
It may be noted that adjacent matrix of a graph is symmetric.

Conversely, given a $n \times n$ symmetric matrix $A(G) = (a_{ij})$ over the set of non-negative integers, we can associate with it a graph G, whose adjacency matrix is A(G), by letting G have n vertices and joining v_i to vertex v_j by a_{ij} edges.

Example 1: Find the adjacency matrix of the graph shown below:



v_3 • • v_4 • v_5

Solution: The adjacency matrix $A(G) = (a_{ij})$ is the matrix such that

a_{iJ} = No. of edges connecting v_i and v_j.

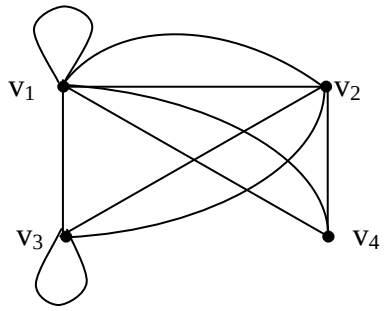
So we have for the given graph

$$A(G) = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

Example 2 : Find the graph that have the following adjacency matrix

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

Solution: We note that there is a loop at v_1 and a loop at v_3. There are parallel edges between v_1, v_2; v_1, v_4; v_2, v_1; v_2 , v_3, v_3, v_2 **;** v_4, v_1. Thus the graph is

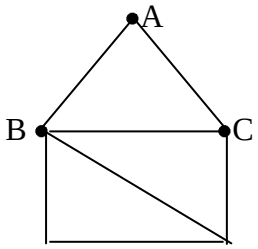


The following theorem is stated without proof.

3.9. Planar Graphs

Definition: A graph which can be drawn in the plane so that its edges do not cross is said to be **planar.**

For example, the graph shown below is planar :



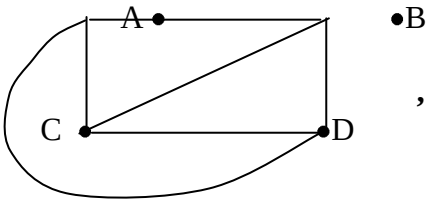
D • •E

Also the complete graph K_4 shown below is planar.

A • •B

C • •D

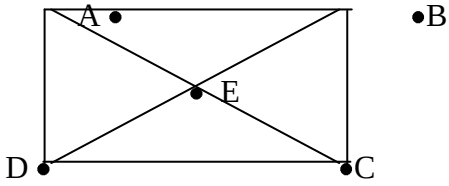
In fact, it can be redrawn as



,

so that no edges cross.

But the complete map K_5 is not planar because in this case, the edges cross each others.

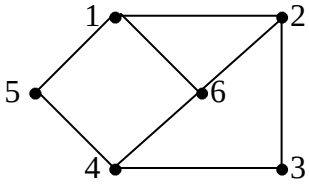


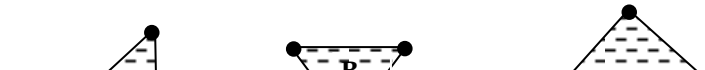
K_5

Definition: An area of the plane that is bounded by edges of the planar graph is not further subdivided into subareas is called a **region** or **face** of a planar graph.
A face is characterised by the cycle that forms its boundary.

Definition: A region is said to be finite if its area is finite and infinite if its area is infinite. Clearly a planar graph has exactly one infinite region.

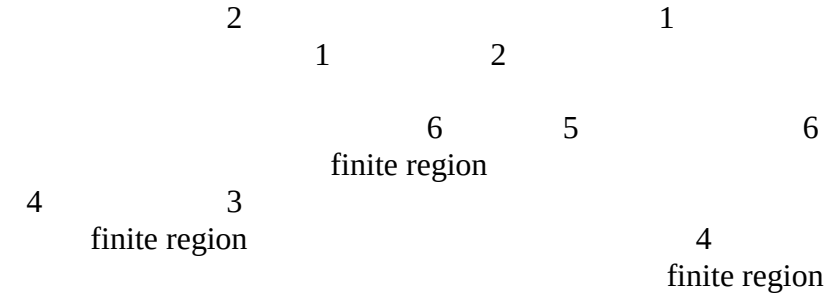
For example, consider the graph :



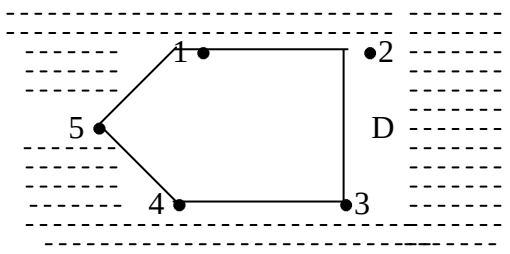


G_2

In graph G_2, there are four region **A, B, C, D**



and



infinite region

Definition: Let f be a face (region) in a planar graph. The length of the cycle (or closed walk) which borders f is called the **degree of the region f**. It is denoted by deg(f).

In a planar graph we note that **each edge either borders two regions or is contained in a region and will occur twice in any walk along the border of the region.** Thus we have

Theorem: The sum of the degrees of the regions of a map is equal to twice the number of edges.
For example, in the graph G_2, discussed above, we have

$$\deg(A) = 4, \deg(B) = 3, \deg(C) = 4, \deg(d) = 5$$

The sum of degrees of all regions = 4 + 3 + 4 + 5 = 16

No. of edges in $G_2 = 8$

Hence

"sum of degrees of region is twice the number of edges".

Theorem (Euler's formula for connected planar graphs): If G is a connected planar graph with e edges, v vertices and r regions, then

$$\mathbf{v - e + r = 2}$$

Proof: We shall use induction on the number of edges. Suppose that e = 0. Then the graph G consists of a single vertex, say P. Thus G is as shown below:

and we have

$$e = 0, v = 1, r = 1$$

Thus

$$1 - 0 + 1 = 2$$

and the formula holds in this case.

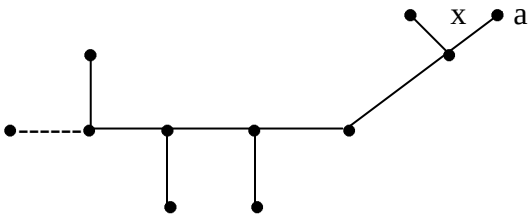
Suppose that e = 1. Then the graph G is one of the two graphs shown below:

,

e = 1, v = 2, r = 1 e = 1, v = 1, r = 2

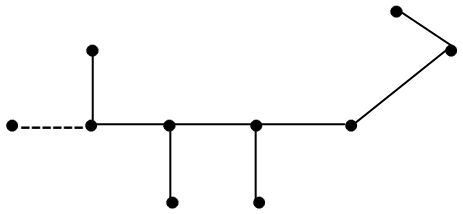
We see that, in either case, the formula holds.

Suppose that the formula holds for connected planar graph with n edges. We shall prove that this holds for graph with n + 1 edges. So, let G be the graph with n + 1 edges. Suppose first that G contains no cycles. Choose "a" vertex v_1 and trace a path starting at v_1. Ultimately, we will reach a vertex a with degree 1, that we cannot leave.



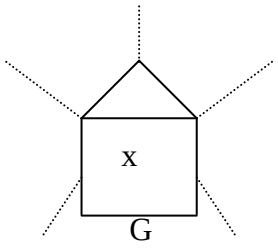
G

We delete "a" and the edge x incident on "a" from the graph G. The resulting graph G′ has n edges and so by induction hypothesis, the formula holds for G′. Since G has one more edge than G′,one more vertex than G′ and the same number of faces as G′, it follows that the formula $v - e + r = 2$ holds also for G.



G′

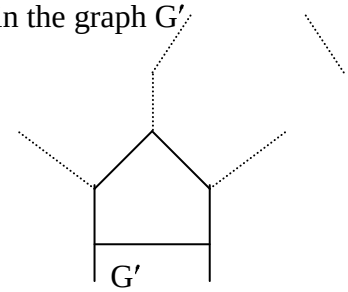
Now suppose that G contains a cycle. Let x be an edge in a cycle.



G

Now the edge x is part of a boundary for two faces. We delete the edge x but no vertices to obtain the graph G′

.

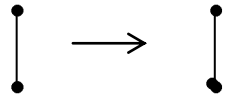


G′

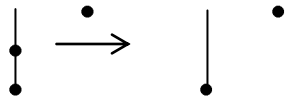
Thus G′ has n edges and so by induction hypothesis the formula holds. Since G has one more face (region) than G′, one more edge than G′ and the same number of vertices as G′, it follows that the formula v – e + r = 2 also holds for G. Hence, by Mathematical Induction, the theorem is true.

Remark: Planarity of a graph is not affected if

(i) an edge is divided into two edges by the insertion of new vertex of degree 2.



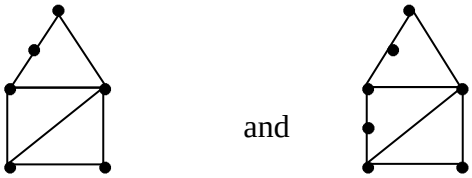
(ii) two edges that are incident with a vertex of degree 2 are combined as a single edge by the removal of that vertex.



Definition: Two graphs G_1 and G_2 are said to be **isomorphic to within vertices of degree 2 (or homeomorphic)** if they are isomorphic or if they can be transformed into isomorphic graphs by repeated insertion and / or removal of vertices of degree 2.

Definition : The repeated insertion/removal of vertices of degree 2 is called **sequence of series reduction.**

For example, the graphs

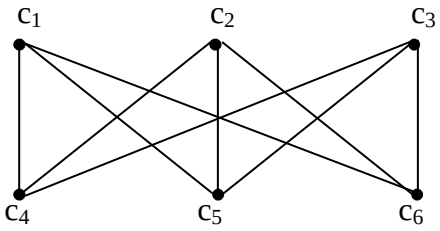


and

are isomorphic to within vertices of degree 2.

If we define a relation R on the set of graphs by G_1 R G_2 if G_1 and G_2 are homeomorphic, then R is an equivalence relation. Each equivalence class consists of a set of mutually homeomorphic graphs.

Example: Show that the graph K_{33}, given below, is not planar.



$K_{3,3}$

A problem based on this example can be stated as "Three cities c_1, c_2 and c_3 are to be directly connected by express ways to each of three cities c_4, c_5 and c_6. Can this road system be designed so that the express ways do not cross? This example shows that it cannot be done.

Solution: Suppose that $K_{3,3}$ is planar. Since every cycle has at least four edges, each face (region) is bounded by at least four edges. Thus the number of edges that bound regions is at least 4r. Also, in a planar graph each edge belongs to atmost two bounding cycles. Therefore,

$2e \geq 4r$ (sums of degrees of region is equal to twice the number of edges)

But, by Euler's formula for planar graph,

$$r = e - v + 2$$

Hence

$$2e \geq 4(e - v + 2) \quad (1)$$

In case of $K_{3,3}$ we have

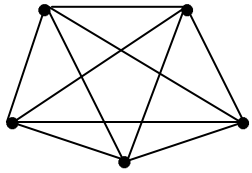
$$e = 9, v = 6$$

and so (1) yields

$$18 \geq 4\,(9 - 6 + 2) = 20\ ,$$

which is a contradiction. Therefore $K_{3,3}$ is not planar.

Remark: By a argument similar to the above example, we can show that the graph K_5 (given below) is not planar.



(non-planar graph K_5)

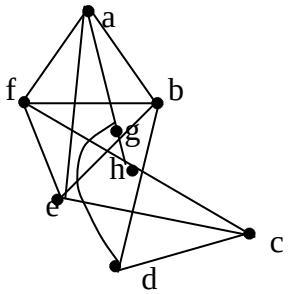
We observe that if a graph contains $K_{3,3}$ or K_5 as a subgraph, then it cannot be planar.

The following theorem, which we state without proof, gives necessary and sufficient condition for a graph to be planar.

Kuratowski's Theorem: A graph G is planar if and only if G does not contain a **subgraph** homeomorphic to $K_{3,3}$ or K_5.

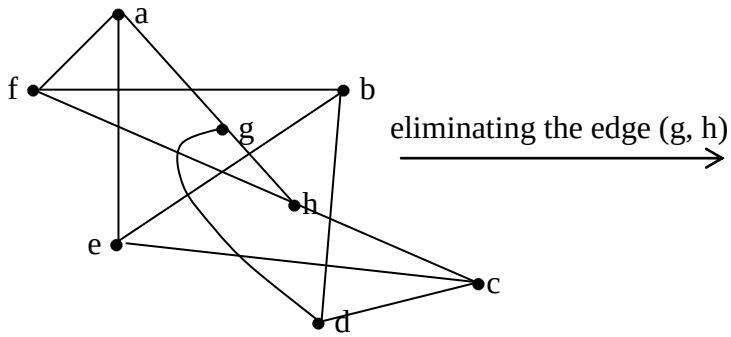
The complete graph K_5 and the complete bipartite graph $K_{3,3}$ are called the **Kuratowski graphs.**

Example: Using Kuratowski's Theorem, show that the graph G, shown below, is not planar

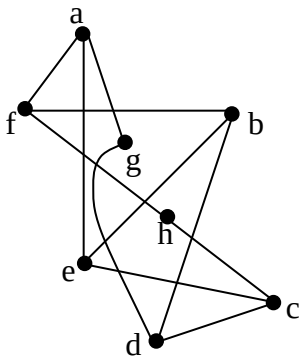


G

Solution: Let us try to find $K_{3,3}$ in the graph G. We know that in $K_{3,3}$, each vertex has degree 3. But we note that in G, the degree of a, b, f and e each is 4. So we eliminate the edges (a,b) and (f, e) so that all vertices have degree 3. If we eliminate one more edge, we will obtain two vertices of degree 2 and we can then carry out series reduction. The resulting graph will have nine edges
Also we know that **$K_{3,3}$ has nine edges**. So this approach seems promising. Using trial and error, we find that the edge (g, h) should be removed. Then g and h have degree 2.

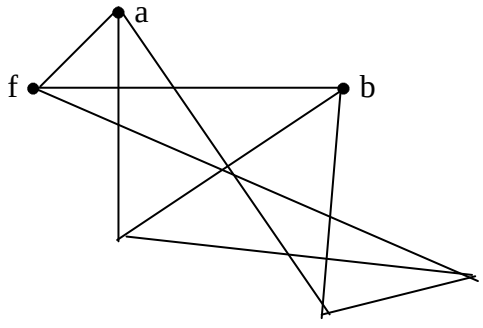


(Graph obtained by deleting edges (a, b) and (f, e)).



(Graph obtained by eliminating the edge (g,h).

Performing series reduction now, we obtain an isomorphic copy of $K_{3,3}$.



e •

•c

• d

$K_{3,3}$(obtained by series reduction)

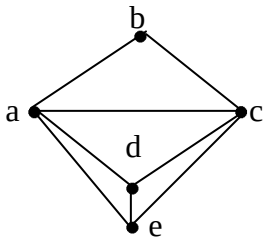
Hence, by Kurtowski's Theorem, the given graph G is not planar.

3.10. Colouring of Graph

Definition: Let G be a graph. The assignment of colours to the vertices of G, one colour to each vertex, so that the adjacent vertices are assigned different colours is called **vertex colouring** or **colouring of the graph G.**
Definition: A graph G is **n-colourable** if there exists a colouring of G which uses n colours.

Definition: The minimum number of colours required to paint (colour) a graph G is called the **chromatic number of G** and is denoted by χ **(G).**

Example: Find the chromatic number for the graph shown in the figure below:



G

Solution: The triangle a b c needs three colours. Suppose that we assign colours c_1, c_2, c_3 to a, b and c respectively. Since d is adjacent to a and c, d will have different colour than c_1 and c_3. So we paint d by c_2. Then e must be painted with a colour different from those of a, d and c, that is, we cannot colour e with c_1, c_2 or c_3. Hence, we have to give e a fourth colour c_4. Hence

$$\chi (G) = 4.$$

3.11 Directed Graphs

Definition: A **directed graph** or **digraph** consists of two finite sets:

(i) A set V of vertices (or nodes or points)

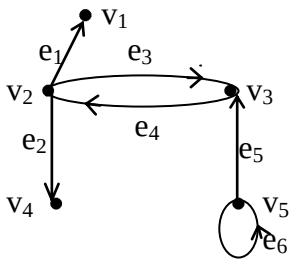
(ii) A set E of directed edges (or arcs), where each edge is associated with an ordered pair (v, w) of vertices called its endpoints. If edge e is associated with the ordered pair (v, w), then e is said to be **directed edge from v to w.**
The directed edges are indicated by arrows.

We say that edge e = (v, w) is incident from v and is incident into w.
The vertex v is called **initial vertex** and the vertex w is called the **terminal vertex** of the
directed edge (v, w).

Definition: Let G be a directed graph. The **outdegree of a vertex v of G** is the number of edges beginning at v. It is denoted by outdeg(v).

Definition: Let G be a directed graph. The **indegree of a vertex v of G** is the number of edges ending at v. It is denoted by indey(v).

Example: Consider the directed graph shown below:



Here edge e_1 is (v_2, v_1) whereas e_6 is denoted by (v_5, v_5) and is called a loop. The indegree of v_2 is 1, outdegree of v_2 is 3.

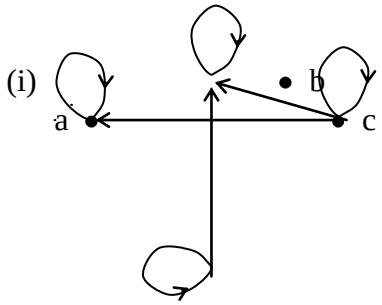
Definition: A vertex with 0 indegree is called a **source**, whereas a vertex with 0 outdegree is called a **sink**.
For instance, in the above example, v_1 is a sink.

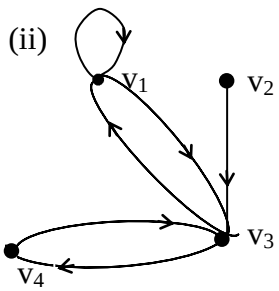
Definition: If the edges and/or vertices of a directed graph G are labeled with some type of data, then G is called a **Labeled Directed Graph.**

Definition: Let G be a **directed graph** with ordered vertices $v_1, v_2, \ldots, v_n$. **The adjacency matrix of G is the matrix** A = (a_{ij}) over the set of non-negative integers such that

a_{ij} = the number of arrows from v_i to v_j, i, j = 1, 2, ….,n.

Example 1: Find the adjacency matrices for the graphs given below:





• d

Solution: (i) The edges in the directed graph are (a, a), (b, b), (c, c), (d, d), (c, a), (c, b) and (d, b). Therefore the adjacency matrix $A = (a_{ij})$ is

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

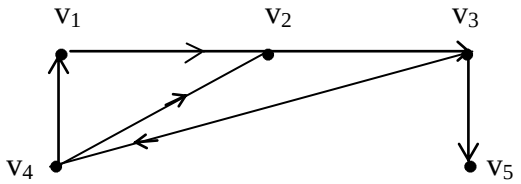
(ii) The edges in the graph in (ii) are (v_2, v_3), (v_1, v_1), (v_1, v_3), (v_3, v_1), (v_3, v_4), (v_4, v_3). Hence the adjacency matrix is

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

Example 2: Find the directed graph represented by the adjacency matrix:

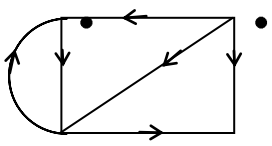
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Solution: we observe that $a_{12} = 1$, $a_{23} = 1$, $a_{34} = 1$, $a_{35} = 1$, $a_{41} = 1$, $a_{42} = 1$. Hence the digraph is as shown below:



Definition: In a directed graph, if there is no more than one directed edge in a particular direction between a pair of vertices, then it is called **simple directed graph.**

For example



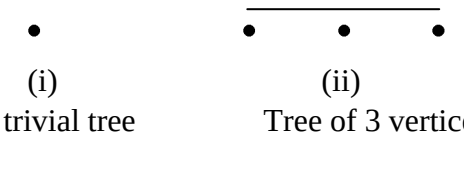
is a simple directed graph.
A directed graph which is not simple is called **directed multigraph**.

3.12. Trees

Definition: A graph is said to be a **Tree** if it is a connected acyclic graph.

A **trivial tree** is a graph that consists of a single vertex. An **empty tree** is a tree that does not have any vertices or edges.

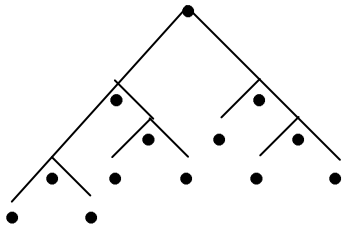
For example, the graphs shown below are all trees.



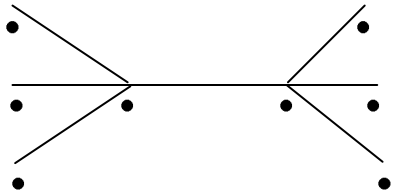
(i)
trivial tree

(ii)
Tree of 3 vertices

(iii)
Tree of 4 vertices



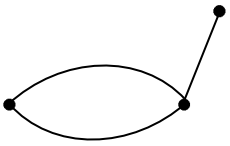
(iv)
Tree of 13 vertices



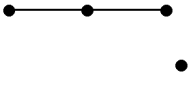
(v)
Tree of 8 vertices

But the graphs shown below are not trees:

(i)
Has a cycle
and so is not a tree
so is not a tree



(ii)
has a cycle
and so is not a tree



(iii)
Disconnected graph
and so is not a tree

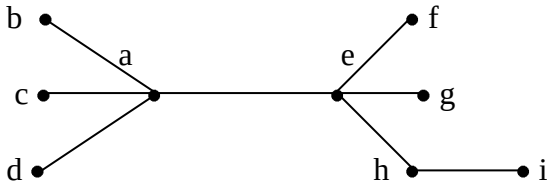
Definition: A collection of disjoint trees is called a **forest.**

Thus a graph is a forest if and only if it is circuit free.

Definition: A vertex of degree 1 in a tree is called a **leaf** or a **terminal node** or a **terminal vertex**.

Definition: A vertex of degree greater than 1 in a tree is called a **Branch node** or **Internal node** or **Internal vertex**.

Consider the tree shown below:



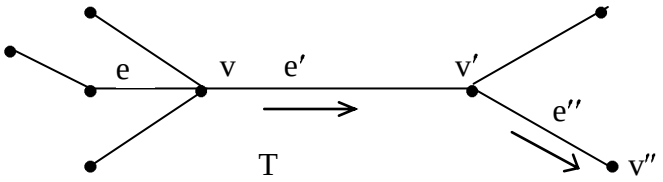
In this tree the vertices b, c, d, f, g, and i are leaves whereas the vertices a, e, h are branch nodes.

CHARACTERIZATION OF TREES

We have the following interesting characterization of trees:

Lemma 1: A tree that has more than one vertex has at least one vertex of degree 1.

Proof: Let T be a particular but arbitrary chosen tree having more than one vertex.



1. Choose a vertex v of T. Since T is connected and has at least two vertices, v is not isolated and there is an edge e incident on v.

2. If deg (v) > 1, there is an edge e′ ≠ e because there are, in such a case, at least two edges incident on v. Let v′ be the vertex at the other end of e′. This is possible because e′ is not a loop by the definition of a tree.
3. If deg(v′) > 1, then there are at least two edges incident on v′. Let e″ be the other edge different from e′ and v″ be the vertex at other end of e″. This is again possible because T is acyclic.

4. If deg(v″) > 1, repeat the above process. Since the number of vertices of a tree is finite and T is circuit free, the process must terminate and we shall arrive at a vertex of degree 1.

Remark: In the proof of the above lemma, after finding a vertex of degree 1, if we return to v and move along a path outward from v starting with e, we shall reach to a vertex of degree 1 again. Thus it follows that **"Any tree that has more than one vertex has at least two vertices of degree 1".**

Lemma 2: There is a unique path between every two vertices in a tree.

Proof: Suppose on the contrary that there are more than one path between any two vertices in a given tree T. Then T has a cycle which contradicts the definition of a tree because T is acyclic. Hence the lemma is proved.

Lemma 3: The number of vertices is one more than the number of edges in a tree.

Or

For any positive integer n, a tree with n vertices has n-1 edges.

Proof: We shall prove the lemma by mathematical induction.

Let T be a tree with **one** vertex. Then T has no edges, that is, T has 0 edge. But $0 = 1 - 1$. Hence the lemma is true for $n = 1$.
Suppose that the lemma is true for $k > 1$. We shall show that it is then true for $k + 1$ also. Since the lemma is true for k, the tree has k vertices and k-1 edges. Let T be a tree with $k+1$ vertices. Since k is +ve, $k+1 \geq 2$ and so T has more than one vertex. Hence, by Lemma 1, T has a vertex v of degree 1. Also there is another vertex w and so there is an edge e connecting v and w. Define a subgraph T′ of T so that

$$V(T') = V(T) - \{v\}$$

$$E(T') = E(T) - \{e\}$$

Then number of vertices in $T' = (k+1) - 1 = k$ and since T is circuit free and T′ has been obtained on removing one edge and one vertex, it follows that T′ is acyclic. Also T′ is connected. Hence T′ is a tree having k vertices and therefore by induction hypothesis, the number of edges in T′ is k-1. But then

No. of edges in T = number of edges in T′ + 1

$$= k - 1 + 1 = k$$

Thus the Lemma is true for tree having $k + 1$ vertices. Hence the lemma is true by mathematical induction.

Corollary 1. Let C(G) denote the number of components of a graph. Then a forest G on n vertices has $n - C(G)$ edges.

Proof: Apply Lemma 3 to each component of the forest G.

Corollary 2. Any graph G on n vertices has at least $n - C(G)$ edges.

Proof: If G has cycle-edges, remove them one at a time until the resulting graph G* is acyclic. Then G* has n – C(G*) edges by corollary 1. Since we have removed only circuit, C(G*) = C(G). Thus G* has n – C(G) edges. Hence G has at least n – C(G) edges.

Lemma 4: A graph in which there is a unique path between every pair of vertices is a tree

(This lemma is converse of Lemma 2).

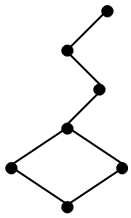
Proof: Since there is a path between every pair of points, therefore the graph is connected. Since a path between every pair of points is unique**,** there does not exist any circuit because existence of circuit implies existence of distinct paths between pair of vertices. Thus the graph is connected and acyclic and so is a tree.

Lemma 5. (converse of Lemma 3) A connected graph G with e = v – 1 is a tree

Proof: The given graph is connected and

$$e = v - 1.$$

To prove that G is a tree, it is sufficient to show that G is acyclic. Suppose on the contrary that G has a cycle. Let m be the number of vertices in this cycle. Also, we know that **number of edges in a cycle is equal to number of vertices in that cycle**. Therefore number of edges in the present case is m. Since the graph is connected, every vertex of the graph which is not in cycle must be connected to the vertices in the cycle.



Now each edge of the graph that is not in the cycle can connect only one vertex to the vertices in the cycle. There are v-m vertices that are not in the cycle. So the graph must contain at least v – m edges that are not in the cycle. Thus we have

$$e \geq v - m + m = v,$$

which is a contradiction to our hypothesis. Hence there is no cycle and so the graph in a tree.

Second proof of Lemma 5: We shall show that a connected graph with v vertices and v – 1 edges is a tree. It is sufficient to show that G is acyclic. Suppose on the contrary that G is not circuit free and has a non trivial circuit C. If we remove one edge of C from the graph G, we obtain a graph G′ which is connected.

G G′

If G′ still has a nontrivial circuit, we repeat the above process and remove one edge of that circuit obtaining a new connected graph. Continuing this process, we obtain a connected graph G* which is circuit free. Hence G* is a tree. Since no vertex has been removed, the tree G* has v vertices. Therefore, by Lemma 3, G* has v-1 edges. But at least one edge of G has been removed to form G*. This means that G* has not more than v – 1 – 1 = v – 2 edges. Thus we arrive at a contradiction. Hence our supposition is wrong and G has no cycle. Therefore G is connected and cycle free and so is a tree.

Lemma 6: A graph G with e = v – 1, that has no circuit is a tree.

Proof: It is sufficient to show that G is connected. Suppose G is not connected and let G′, G″….. be connected component of G. Since each of G′, G″,…. is connected and has no cycle, they all are tree. Therefore, by Lemma 3,

$$e' = v' - 1$$

$$e'' = v'' - 1$$

---

where e′, e″, … are the number of edges and v′, v″,… are the number of vertices in G′, G″, …respectively. We have, on adding

$$e' + e'' + \ldots\ldots = (v' - 1) + (v'' - 1) + \ldots\ldots$$

Since

$$e = e' + e'' + \ldots..$$

$$v = v' + v'' + \ldots.$$,

we have

$$e < v - 1$$,

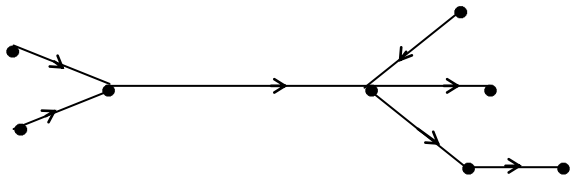
which contradicts our hypotheses. Hence G is connected. So G is connected and acyclic and is therefore a tree.

Example: Construct a graph that has 6 vertices and 5 edges but is not a tree.

Solution: We have, No. of vertices = 6, No. of edges = 5 . So the condition e = v – 1 is satisfied. Therefore, to construct graph with six vertices and 5 edges that is not a tree, we should keep in mind that the graph should not be connected. The graph shown below has 6 vertices and 5 edges but is not connected.

.

Definition: A directed graph is said to be a directed tree if it becomes a tree when the direction of edges are ignored.
For example, the graph shown below is a directed tree.



Definition: A directed tree is called a **rooted tree** if there is exactly one vertex whose incoming degree is 0 and the incoming degrees of all other vertices are 1.
The vertex with incoming degree 0 is called the **root** of the rooted tree.
A tree T with root v_0 will be denoted by (T, v_0).

Definition: In a rooted tree, a vertex, whose outgoing degree is 0 is called a **leaf** or **terminal node**, whereas a vertex whose outgoing degree is non - zero is called a **branch node** or an **internal node**.
Definition: Let u be a branch node in a rooted tree. Then a vertex v is said to be **child** (**son** or **offspring**) of u if there is an edge from u to v. In this case u is called **parent** (**father**) of v.

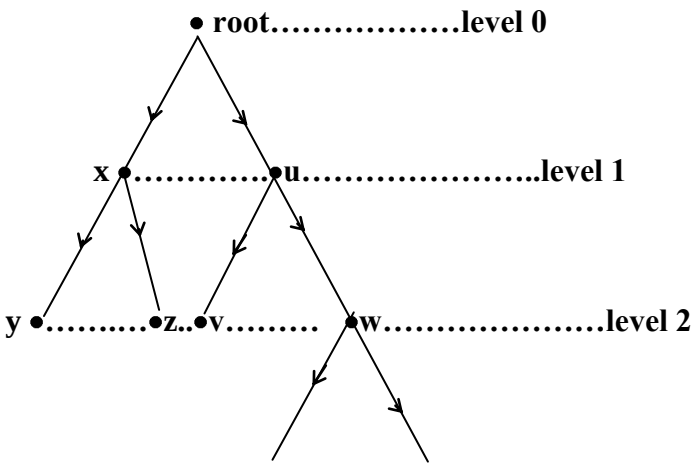
Definition: Two vertices in a rooted tree are said to be **siblings** (**brothers**) if they are both children of same parent.

Definition: A vertex v is said to be a **descendent** of a vertex u if there is a unique directed path from u to v.
In this case u is called the **ancestor** of v.

Definition: The **level** (or **path length**) of a vertex u in a rooted tree is the number of edges along the unique path between u and the root.

Definition: The **height** of a rooted tree is the maximum level to any vertex of the tree.

As an example of these terms consider the rooted tree shown below:

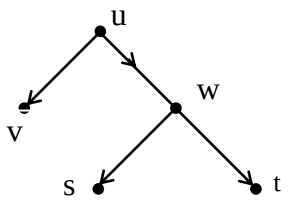


t•…… …•s……level 3

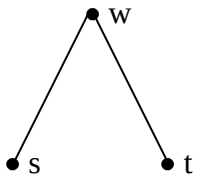
Here y is a child of x; x is the parent of y and z. Thus y and z are siblings. The descendents of u are v, w, t and s. Levels of vertices are shown in the figure. The height of this rooted tree is 3.

Definition: Let u be a branch node in the tree T = (V, E). Then the subgraph T′ = (V′, E′) of T such that the vertices set V′ contains u and all of its descendents and E′ contains all the edges in all directed paths emerging from u is called a **subtree** with u as the root.

Definition: Let u be a branch node. By a subtree of u, we mean a subtree that has child of u as root.
In the above example, we note that the figure shown below is a subtree of T,



where as the figure shown below is a subtree of the branch node u .



is a subtree of the branch node u.

Example. Let

$$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$$

and let

$$E = (\{v_2, v_1), (v_2, v_3), (v_4, v_2), (v_4, v_5), (v_4, v_6), (v_6, v_7), (v_5, v_8)\}.$$

Show that (V, E) is rooted tree. Identify the root of this tree.

Solution: We note that

Incoming degree of $v_1 = 1$

Incoming degree of $v_2 = 1$

Incoming degree of $v_3 = 1$

Incoming degree of $v_4 = 0$

Incoming degree of $v_5 = 1$

Incoming degree of $v_6 = 1$

Incoming degree of $v_7 = 1$

Incoming degree of $v_8 = 1$

Since incoming degree of the vertex v_4 is 0, it follows that v_4 is root.

Further,

Outgoing degree of $v_1 = 0$

Outgoing degree of $v_3 = 0$

Outgoing degree of $v_7 = 0$

Outgoing degree of $v_8 = 0$

Therefore v_1, v_2, v_7, v_8 are leaves. Also ,

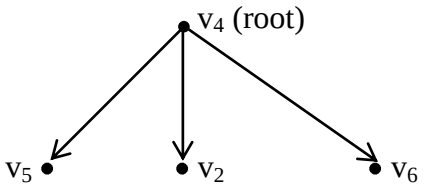
Outgoing degree of $v_2 = 2$

Outgoing degree of $v_4 = 3$

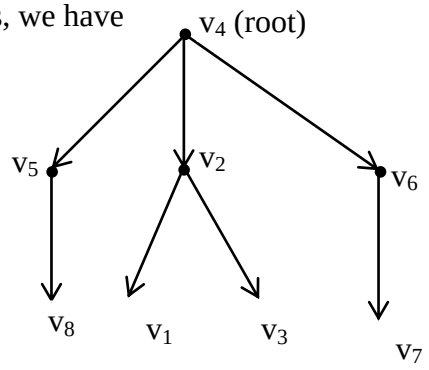
Outgoing degree of $v_5 = 1$

Outgoing degree of $v_6 = 1$

Now the root v_4 is connected to v_2, v_5 and v_6. So, we have



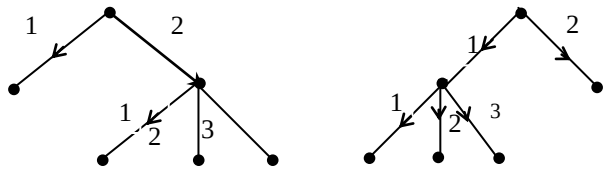
Now v_2 is connected to v_1 and v_3, v_5 is connected to v_8, v_6 is connected to v_7. Thus, we have



We thus have a connected acyclic graph and so (V, E) is a rooted tree with root v_4.

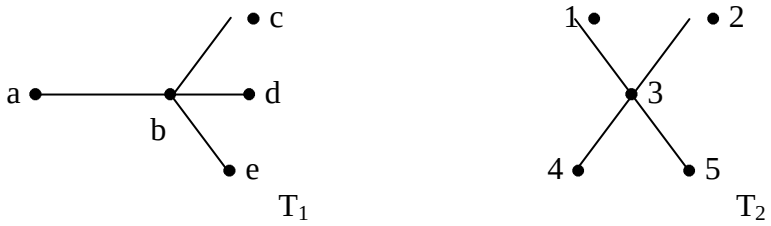
Definition: A rooted tree in which the edges incident from each branch node are labeled with integers 1, 2, 3,…. is called an **ordered tree**.

Definition: Two ordered trees are said to be **isomorphic** if (i) there exists a one-to-one correspondence between their vertices and edges and that preserves the incident relation (ii) labels of the corresponding edges match.
In view of this definition, the ordered trees



are not isomorphic.

Example: Show that the tree T_1 and T_2 shown in the diagram below are isomorphic.



Solution: We observe that in the tree T_1,

$$\deg(b) = 4$$

In the tree T_2,

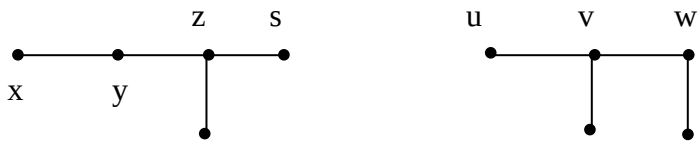
$$\deg(3) = 4$$

Further deg(a) = deg(1) = 1, deg c = deg(2), deg(d) = deg(4) = deg(e) = 1 = deg(5). Thus we may define a function f from the vertices of T_1 to the vertices of T_2 by

$$f(a) = 1,\ f(b) = 3,\ f(c) = 2,\ f(d) = 4,\ f(e) = 5$$

This is a one-to-one and onto function. Also adjacency relation is preserved because if v_i and v_j are adjacent vertices in T_1, then $f(v_i)$ and $f(v_j)$ are adjacent vertices in T_2. Hence T_1 is isomorphic to T_2.

Example: Show that the tree T_1 and T_2, shown in the figure below are isomorphic



t *l* m

T_1 T_2

Solution: Let f be a function defined by

$f(z) = v$

$f(b) = w$

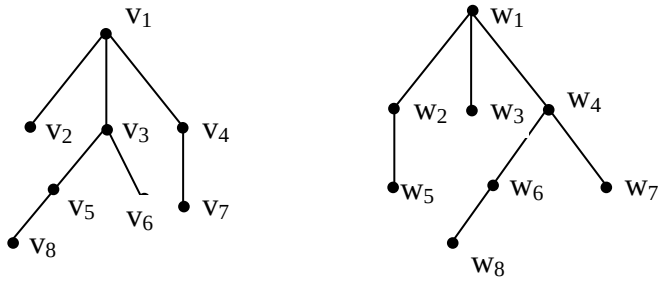
$f(x) = m$

$f(s) = u$

$f(t) = l$.

Then f is an one to one onto mapping which preserves adjacency. HenceT_1 and T_2 are isomorphic.

Definition: Let T_1 and T_2 be rooted tree with roots r_1 and r_2 respectively. Then T_1 and T_2 are **isomorphic** if there exists a one-to-one, onto function f from the vertex set of T_1 to the vertex set of T_2 such that
(i) Vertices v_i and v_j are adjacent in T_1 if and only if the vertices $f(v_i)$ and $f(v_j)$ are adjacent in T_2.
(ii) $f(r_1) = r_2$
The function is then called an isomorphism.
Example: Show that the tree T_1 and T_2 are isomorphic.



Solution: We observe that T_1 and T_2 are rooted tree.
Define f: (Vertex set of T_1) $\rightarrow$ (Vertex set of T_2) by

$f(v_1) = w_1$

$f(v_2) = w_3$

$f(v_3) = w_4$

$f(v_4) = w_2$

$f(v_5) = w_6$

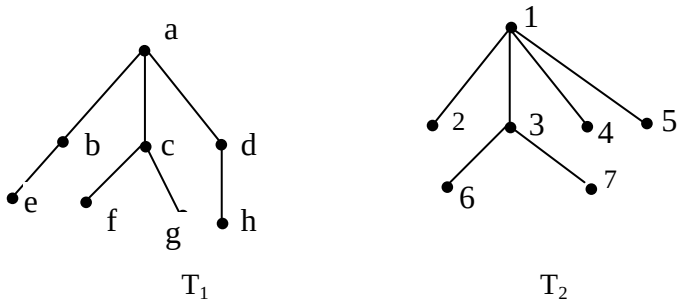
$f(v_6) = w_7$

$f(v_7) = w_5$

$f(v_8) = w_8$

Then f is one-to- one and adjacency relation is preserved. Hence f is an isomorphism and so the rooted tree T_1 and T_2 are isomorphic

Example: Show that the rooted tree shown below are not isomorphic:



T_1 T_2

Solution: We observe that the degree of root in T_1 is 3, whereas the degree of root in T_2 is 4. Hence T_1 is not isomorphic to T_2.

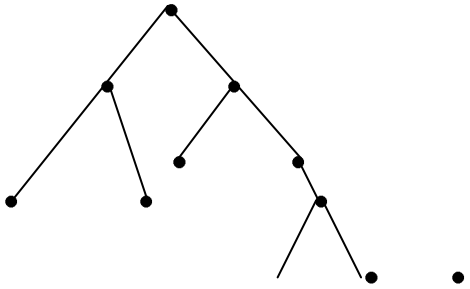
Definition: An ordered tree in which every branch node has atmost n offspring's is called a **n-ary tree** (or **n-tree**).

Definition: An n-ary tree is said to be **fully n-ary tree** (**complete n-ary tree** or **regular n ary tree**) if every branch node has exactly n offspring.

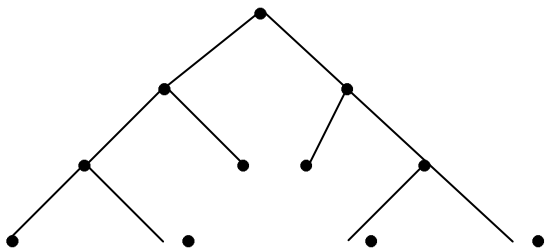
Definition: An ordered tree in which every branch node has almost 2 offsprings is called a **binary tree** (or **2 - tree**).

Definition: A binary tree in which every branch node (internal vertex) has exactly two offspring's is called a **fully binary tree**.

For example, the tree given below is a binary tree,

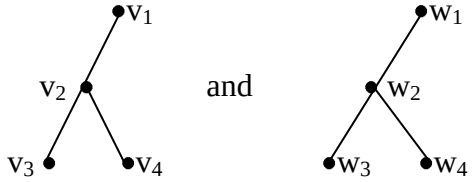


whereas the tree shown below is a fully binary tree.



Definition: Let T_1 and T_2 be binary trees roots r_1 and r_2 respectively. Then T_1 and T_2 are **isomorphic** if there is a one to one, onto function f from the vertex set of T_1 to the vertex set of T_2 satisfying
(i) Vertices v_i and v_j are adjacent in T_1 if and only if the vertices $f(v_i)$ and $f(v_j)$ are adjacent in T_2.
(ii) $f(r_1) = r_2$
(iii) v is a left child of w in T_1 if and only if f(v) is a left child of f(w) in T_2
(iv) v is a right child of w in T_1 if and only if f(v) is a right child of f(w) in T_2.
The function f is then called an **isomorphism** between binary tree T_1 and T_2

Example: Show that the trees given below are isomorphic.



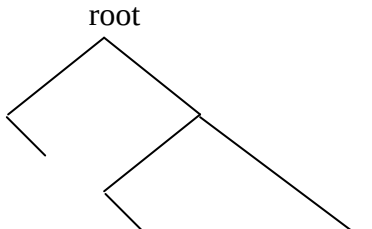
Solution: Define f by $f(v_i) = w_i$, i = 1, 2, 3, 4. Then f satisfies all the properties for isomorphism. Hence T_1 and T_2 are isomorphic.

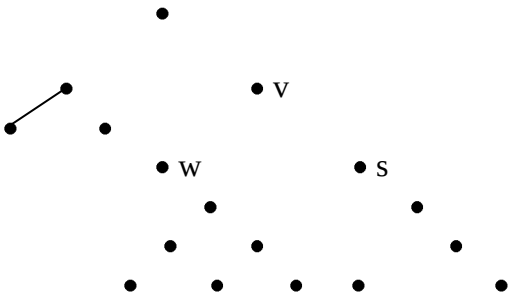
Example: Show that the trees given below are not isomorphic.

Solution: Since the root v_1 in T_1 has a left child but the root w_1 in T_2 has no left child, the binary trees are not isomorphic.

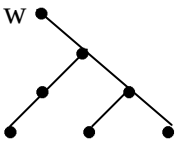
Definition: Let v be a branch node of a binary tree T. The left subtree of v is the binary tree whose root is the left child of v, whose vertices consists of the left child of v and all its descendents and whose edges consists of all those edges of T that connects the vertices of the left subtree together.
The **right subtree** can be defined analogously.

For example, the left subtree and the right subtree of v in the tree (shown below) :



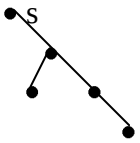


are respectively



(left subtree of v)

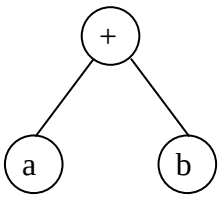
and

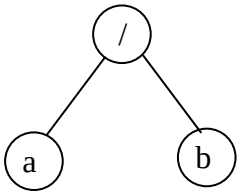


(right subtree of v)

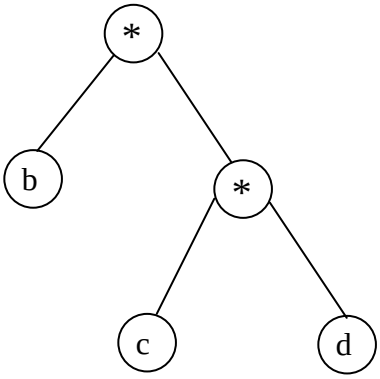
3.13 Representation of Arithmetic/Algebraic Expressions by Binary Trees

Binary trees are used in computer science to represent algebraic expressions involving parentheses. For example, the binary trees





and



represent the expressions

$$a + b \quad , \quad a/b$$

and

$$b * (c * d)$$

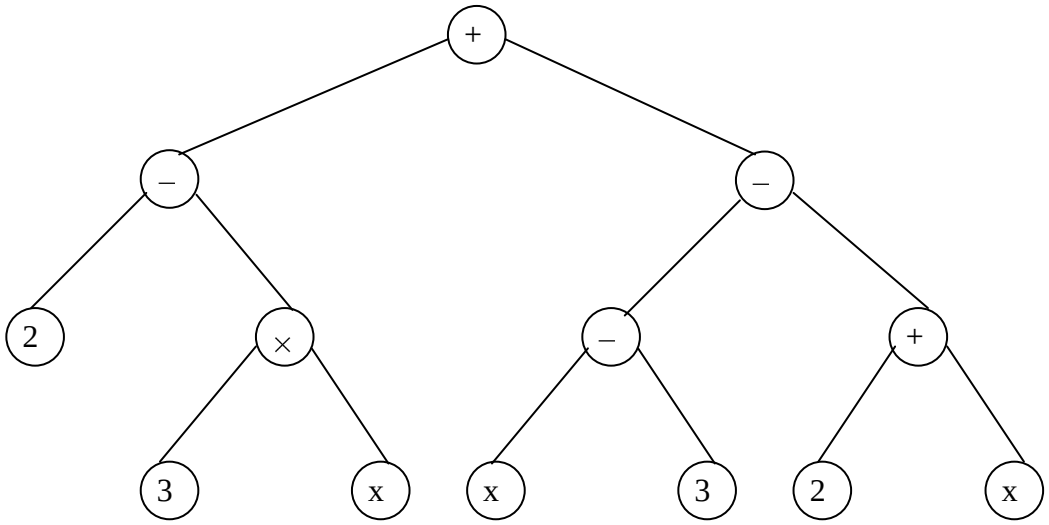
respectively.
Thus, the central operator acts as root of the tree.

Example 1: Draw a binary tree to represent

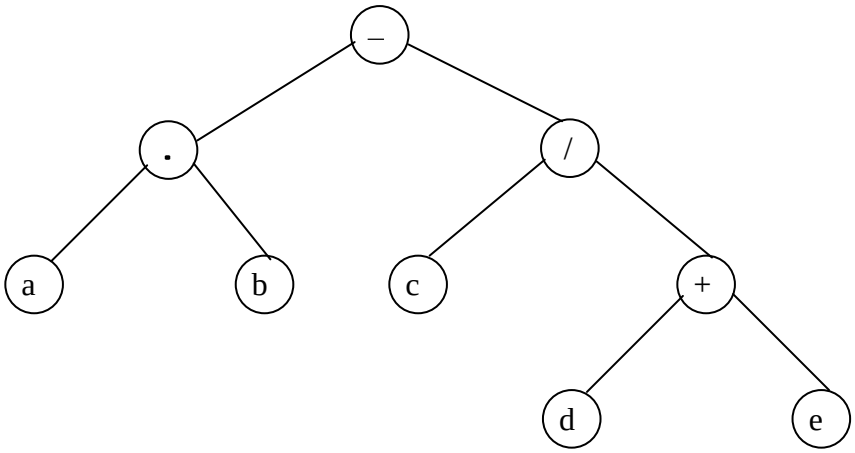
(i) $(2 - (3 \times x) + ((x - 3) - (2 + x)$

(ii) $a.b - (c/(d + e))$.

Solution: (i) In this expression + is the central operator. Therefore the root of tree is +. The binary tree is



(ii) Here the central operator is —. Therefore it is the root of the tree. We have the following binary tree to represent this expression.



To derive this formula we first prove the following result :

Theorem: If T is a full binary tree with i internal vertices, then T has i+1 terminal vertices (leaves) and 2i+1 total vertices.

Proof: The vertices of T consists of the vertices that are children (of some parent) and the vertices that are not children (of any parent). There is nonchild – the root, Since there are i internal vertices, each having two children, there are 2i children. Thus the total number of vertices of T is 2i+1 and the number of terminal vertices is

$$(2i + 1) – i = i + 1$$

This completes the proof.

In the context of above example, we have

$$\text{No. of leaves} = p = i + 1$$

Or

$$i = p – 1$$

Remark: In case of full n-ary tree, if i denotes the number of branch nodes, then total number of vertices of T is $ni + 1$ and the number of terminal vertices is

$$n\,i + 1 – i = i(n - 1) + 1$$

If p is the number of terminal vertices, then

$$p = i(n – 1) + 1$$

or

$$(n – 1)\,i = p – 1$$

Example 1: Find the minimum number of extension cords, each having 4 outlets, required to connect 22 bulbs to a single electric outlet.

Solution: Clearly, the graph of the problem is a regular quaternary tree with 22 leaves.
Let i denote the internal vertices and p denote the number of leaves, then using

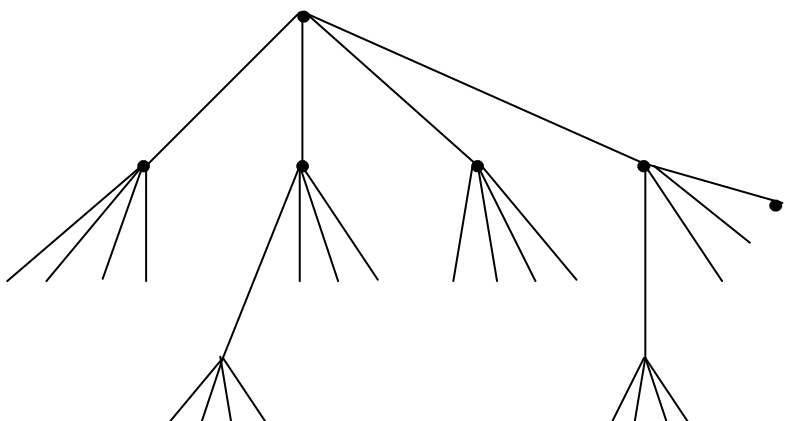
$$(n – 1)\,i = p – 1 \quad ,$$

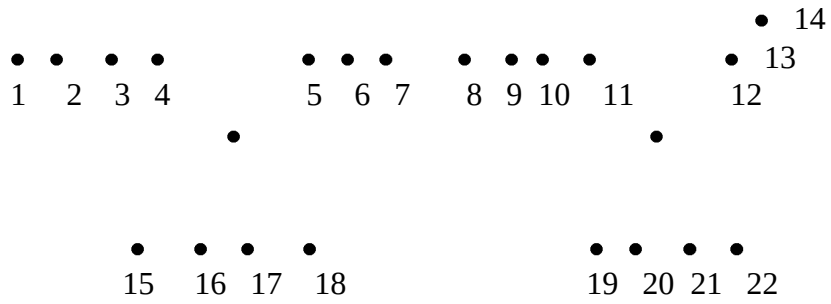
we have

$$(4 – 1)\,i = 22 – 1$$

$$\text{or} \qquad i = \frac{21}{3} = 7.$$

Thus 7 extension cords as shown below are required.





Example: Does there exist a full binary tree with 12 internal vertices and 15 leaves?

Solution: We know that if i is the number of branch nodes in a full binary tree, then the number of leaves is i + 1. Therefore for a tree with 12 branch nodes, the number of leaves should be 13 and not 15. Hence such tree does not exist.

Theorem: The number b_n of different trees on n vertices is

$$b_n = \frac{1}{n+1}\left(\frac{2n}{n}\right)$$

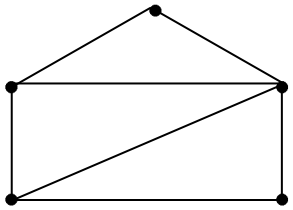
Definition: Let G be a graph, then a subgraph of G which is a tree is called **tree of the graph.**

Definition: A **spanning tree** for a graph G is a subgraph of G that contains every vertex of G and is a tree.

Or

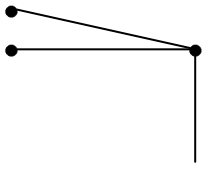
"A **spanning tree** for a graph G is a spanning subgroup of G which is a tree".

Example: Determine a tree and a spanning tree for the connected graph given below:

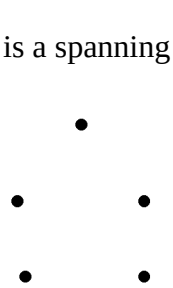


G

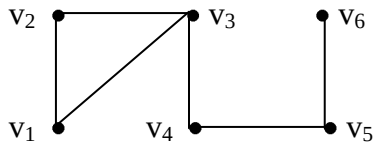
Solution: The given graph G contains circuits and we know that removal of the circuits gives a tree. So, we note that the figure below is a tree.



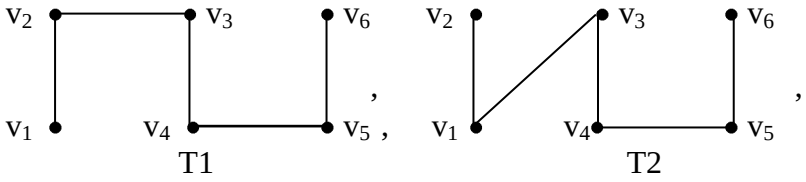
And the figure below is a spanning tree of the graph G.

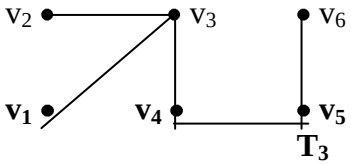


Example: Find all spanning trees for the graph G shown below:



Solution: The given graph G has a circuit $v_1\ v_2\ v_3\ v_1$. We know that removal of any edge of the circuit gives a tree. So the spanning trees of G are





Remark**: We know that a tree with n vertices has exactly n – 1 edges. Therefore if G is a connected graph with n vertices and m edges, a spanning tree of G must have n – 1 edges. Hence the number of edges that must be removed before a spanning tree is obtained must be**

$$m - (n - 1) = m - n + 1.$$

For Illustration, in the above example, n = 6, m = 6, so, we had to remove one edge to obtain a spanning tree.

Definition: **A** branch **of a tree is an edge of the graph that is in the tree.**

Definition: **A** chord **(or a** link**) of a tree is an edge of the graph that is not in the tree.**

It follows from the above remark that the number of chords in a tree is equal to m – n + 1, where n is the number of vertices and m is the number of edges in the graph related to the tree.

Definition: **The set of the chords of a tree is called the** complement of the tree**.**

Example: **Consider the graph discussed in the above example. We note that the edge (v_2, v_3) is a branch of the tree T_1, whereas (v_1, v_3) is a chord of the tree T_1.**

Theorem: **A graph G has a spanning tree if and only if G is connected.**

Proof: **Suppose first that a graph G has a spanning tree T. If v and w are vertices of G, then they are also vertices in T and since T is a tree there is a path from v to w in T. This path is also a path in G. Thus every two vertices are connected in G. Hence G is connected.**

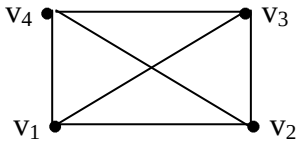
Conversely, suppose that G is connected. If G is acyclic, then G is its own spanning tree and we are done. So suppose that G contains a cycle C_1. If we remove an edge from the cycle, the subgraph of G so obtained is also connected. If it is acyclic, then it is a spanning tree and we are done. If not, it has at least one circuit, say C_2 . Removing one edge from C_2, we get a subgraph of G which is connected. Continuing in this way, we obtain a connected circuit free subgraph T of G. Since T contains all vertices of G, it is a spanning tree of G.

Cayley's Formula : **The number of spanning trees of the complete graph K_n, $n \geq 2$ is n^{n-2}.**

(Proof of this formula is out of scope of this book)

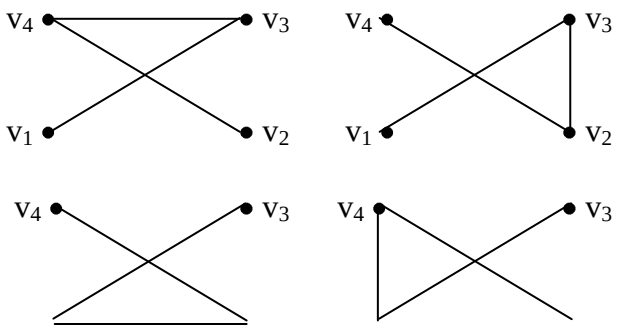
Example: **Find all the spanning trees of K_4.**

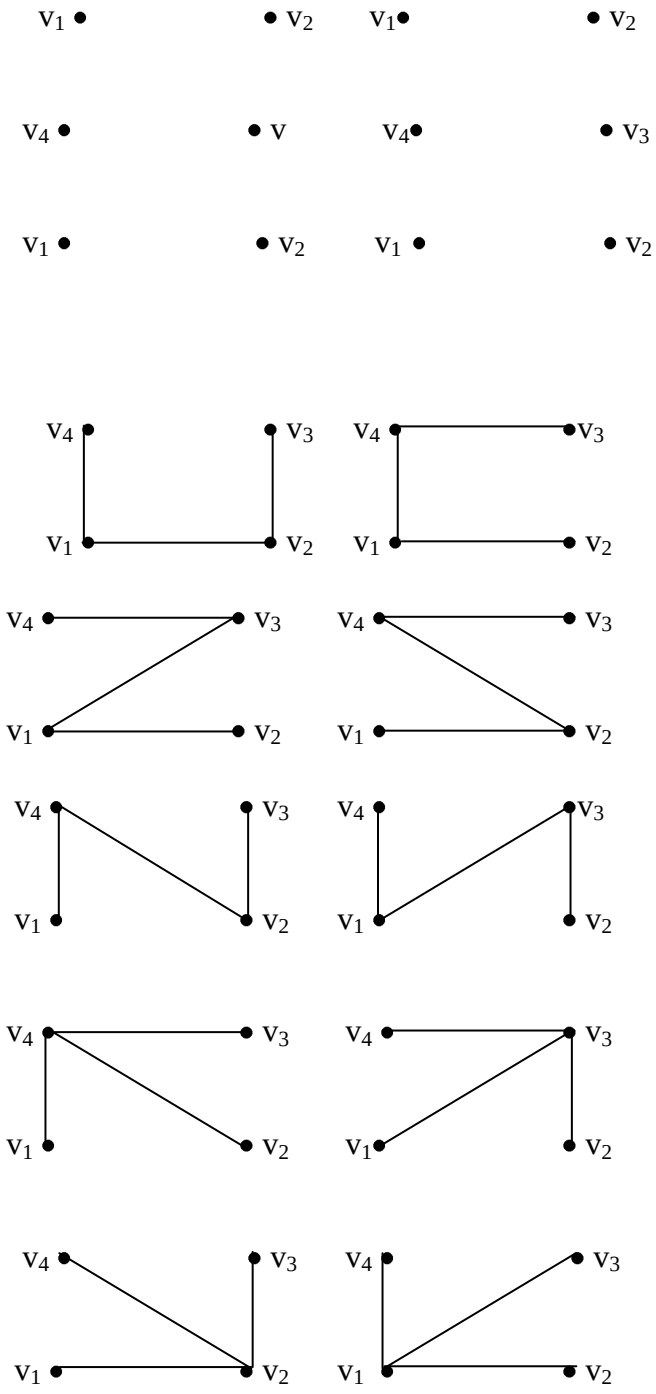
Solution: **According to Cayley's formula, K_4 has $4^{4-2} = 4^2 = 16$ different spanning trees.**



K_4

Here n = 4, so the number of edges in any tree should be n – 1 = 4 – 1 = 3. But here number of edges is equal to 6. So to get a tree, we have to remove three edges of K_4. The 16 spanning trees so obtained are shown below:





3.14. Shortest Path Problem

Let s and t be two vertices of a connected weighted graph G. Shortest Path problem is to find a **path from s to t whose total edge weight is minimum.** We now discuss Algorithm due to E. W. Dijkstra which efficiently solve the shortest path problem. The idea is to grow a Disjkstra tree, starting at the

vertex s, by adding, at each iteration, a frontier edge, whose non-tree end point is as close as possible to s. The algorithm involves assigning labels to vertices. For each tree vertex x, let dist [x] denote the distance from vertex s to x and for each edge e in the given weighted graph G, let w(e) be its edge – weight.
After each iteration, the vertices in the Dijkstra tree (the labeled vertices) are those to which the shortest paths from s have been found.

Priority of the Frontier Edges : Let e be a frontier edge and let its P - value be given by

$$P(e) = dist [x] + w(e),$$

where x is the labeled end point of e and w(e) is the edge – weight of e. Then

(i) The edge with the smallest P – value is given the **highest priority.**

(ii) The P – value of this highest priority edge e gives the distant from the vertex s to the unlabeled endpoint of e.

We are now in a position to describe Dijkstra shortest path algorithm.

DIJKSTRA'S SHORTEST PATH ALGORITHM

Input : A connected weighted graph G with non-negative edge-weights and a vertex s of G.

Output : A spanning tree T of G, rooted at the vertex s**,** whose path from s to each vertex v is a shortest path from s to v in G and a vertex labeling giving the distance from s to each vertex.

Initialize the Dijkstra tree T as vertex s.
Initialize the set of frontier edges for the tree T as empty.

$$dist : [s] = 0.$$

Write label 0 on vertex s.

While Dijkstra tree T does not yet span G.

For each frontier edge e for T,

Let x be the labeled endpoint of edge e.

Let y be the unlabeled endpoint of edge e.

Set $P(e) = dist [x] + w(e)$

Let e be a frontier edge for T that has smallest P – value

Let x be the labeled endpoint of edge e

Let y be the unlabeled endpoint of edge e

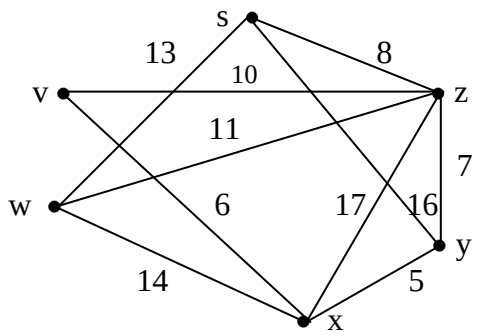
Add edge e (and vertex y) to tree T

dist [y] : P(e)

Write label dist [y] on vertex y.

Return Dijkstra tree T and its vertex labels.

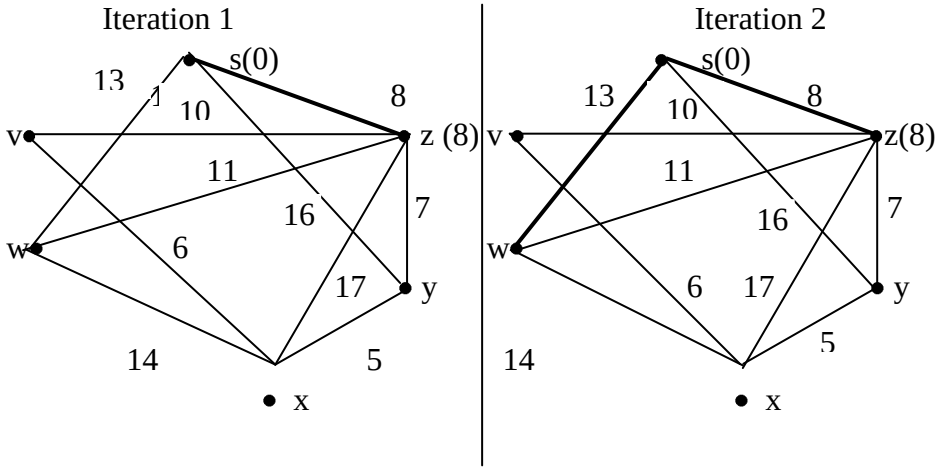
Example : Apply Dijkstra algorithm to find shortest path from s to each other vertex in the graph given below :



If t is the labeled endpoint of edge e, then P – values are given by

P(e) = dist [t] + w(e),

where dist [t] = distance from s to t and w(e) is the edge weight of edge e. For each vertex v, dist [v] appears in the parenthesis. Iteration tree at the end of each iteration is **drawn in dark line**



dist [s] = 0 P(sw) = 13 (minimum)

dist [z] = 8 (minimum) P(zy) = 8 + 7 = 15

P(sy) = 16

P(zv) = 8 + 10 = 18

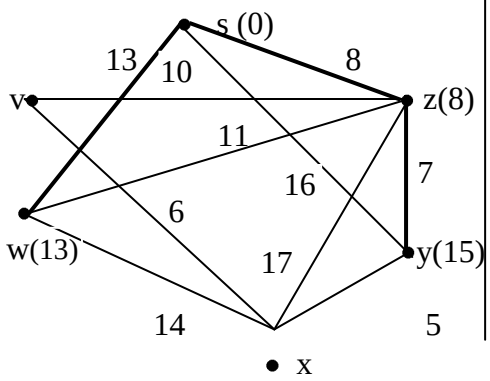
dist [s] = 0 P(zy) = 8 + 7 =15

dist [z] = 8 P(zx) = 8 + 17 = 25

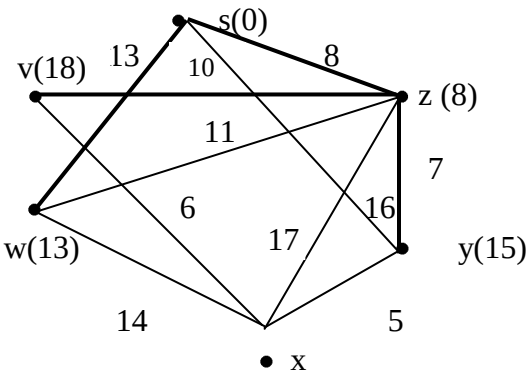
dist [w] = 13 P(zv) = 8 + 10 = 18

P(zw) = 8 + 11 = 19
P(zx) = 8 + 17 = 25
P(sy) = 16
P(wx) = 13 + 14 = 27

Iteration 3



Iteration 4



dist [s] = 0 P(zv) = 18 (minimum)
(minimum)
dist [z] = 8 P(zx) = 8 + 17 = 15
dist [w]= 13 P(wx) = 13 + 14 = 27
14
dist [y] = 15 P(yx) = 15 + 5 = 20
27

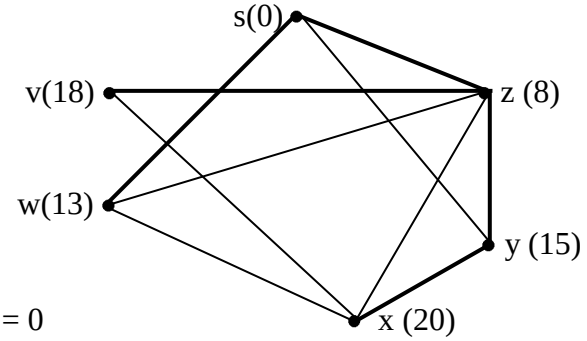
Dist [s] = 0 P(yx) = 20

Dist [z] = 8 P(zx) = 8 + 17 = 25
Dist [w] = 13 P(vx) = 18 + 6 =

Dist [y] = 15 P(wx) = 13 + 14 =

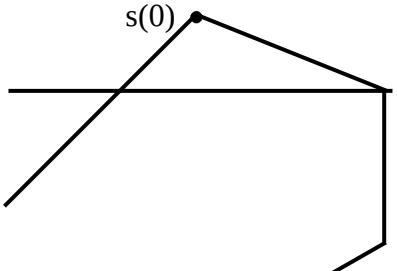
Dist [v] = 18

Iteration 5



dist [s] = 0
dist [z] = 8
dist [w] =13
dist [y] =15
dist [x] = 20

which are the required shortest paths from s to any other point. The Dijkstra tree is shown in dark lines.



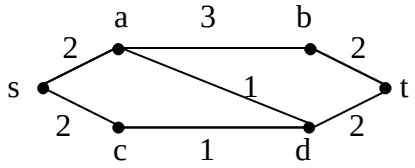
v(18) • • z (8)

w(13) •

• y (15)

• x (20)

Example: Find a shortest path from s to t and its length for the graph given below:



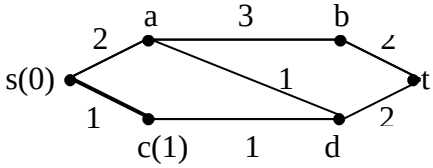
Solution: Let x be the labeled endpoint of edge e, then P-values are given by

$$P(c) = \text{dist}[x] + w(e),$$

where dist [x] denotes the distance from s to x and w(e) is the weight of the edge e.

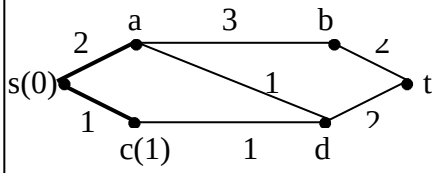
For each vertex v, dist [v] appears in the bracket. Iteration tree at the end of each iteration is shown in dark lines.

Iteration 1



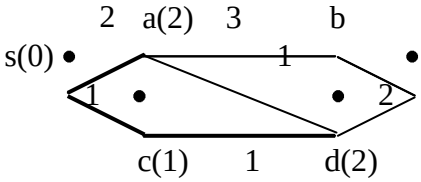
dist [s] = 0 P(c d) = 2
dist [c] = 1 P(s a) = 2 (minimum)
P(s b) = ∞
P(s t) = ∞
P(d t) = ∞
P(b t) = ∞

Iteration 2



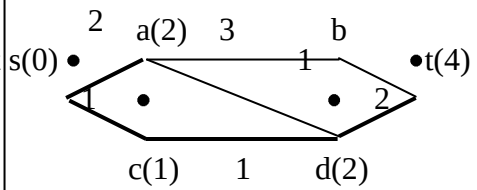
dist [s] =0 P(a b) = 2 + 3 = 5
dist [c] = 1 P(c d) = 1 + 1 = 2 (minimum)
dist [a] = 2 P(d t) = ∞
P(b t) = ∞
P(a d) = 3

Iteration 3



dist [s] = 0 P(d t) = 2 (minimum)

Iteration 4

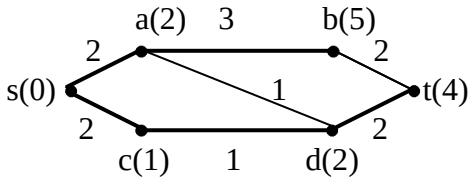


dist [s] =0 P(a b) = 5

dist [c] = 1 P(a b) = 5
dist [a] = 2 P(b t) = ∞
dist [d] = 2

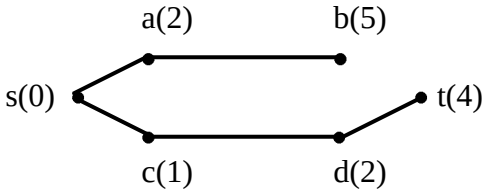
dist [c] = 1 P(b t) = ∞
dist [a] = 2
dist [d] = 2
dist [t] =4

Iteration 5



dist [s] =0
dist [c] =1
dist [a] = 2
dist [d] = 2
dist [t] = 4
dist [b] = 5

Thus, the Dijkstra tree is



Thus the shortest path is scdt and its length is 4.

3.15. Shortest Path if all Edges Have Length 1

If all edges in a connected graph G have length 1, then a **shortest path** $v_1 \rightarrow v_k$ is the path that has the smallest number of edges among all paths v1 $\rightarrow$ vk in the given graph G.

Moore's Breadth First Search Algorithm

This method of finding shortest path in a connected graph G from a vertex s to a vertex t is used when all edges have length 1.

Input : Connected graph G = (V, E), in which one vertex is denoted by s and one by t and each edge (v_i, v_j) has length 1.

Initially all vertices are unlabeled.

Output : A shortest path s $\rightarrow$ t in G = (V, E).

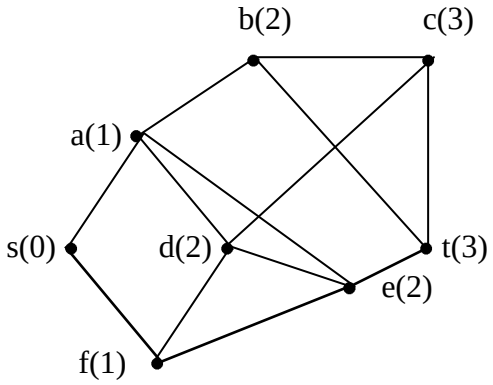
1. Label s with 0.
2. Set $v_i = 0$
3. Find all unlabeled vertices **adjacent to a vertex labeled v_i.**

4. Label the vertices just found with v_{i+1}
5. If vertex t is labeled, then "**back tracking**" gives the shortest path. If k is level of t(i.e., $t = v_k$), then
Output : $v_k, v_{k-1}, \ldots\ldots\ldots v_1, 0$.

Else increase i by 1. Go to step 3.
End Moore.

Remark : There could be several shortest path from s to t.

Example : Use B F S algorithm to find shortest path from s to t in the connected graph G given below:



Solution : Label s with 0 and then label the adjacent vertices with 1. Thus two vertices have been labeled by 1. Now Label the adjacent vertices of all vertices labeled by 1 with label 2. Thus three vertices have been labeled with 2. Label the vertices adjacent to these vertices (labeled by 2) with 3. Thus two vertices have been labeled with 3. We have reached t. Now back tracking yields the following shortest paths

t(3), e(2), f(1), s(0), that is, s f e t

or

t(3), b(2), a(1), s(0), that is s a b t

or

t(3), e(2), a(1), s(0), that is, s a e t

Thus there are three possible shortest paths of length 3.

3.16 Minimal Spanning Tree

Definition : Let G be a weighted graph. A spanning tree of G with minimum weight is called **minimal spanning tree of G**.

We discuss two algorithms to find a minimal spanning tree for a weighted graph G.

PRIM ALGORITHM

Prim algorithm builds a minimal spanning tree T by expanding outward in connected links from some vertex. In this algorithm one edge and one vertex are added at each step. The edge added is the one of least weight that connects the vertices already in T with those not in T.

Input **:** A connected weighted graph G with n vertices

Output **:** The set of edges E in a minimal spanning tree.

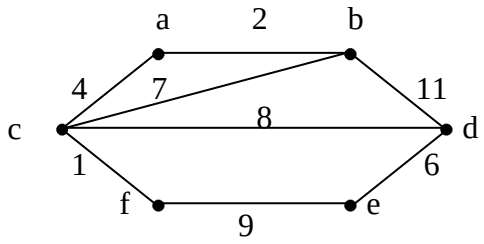
1. Choose a vertex v_1 of G. Let V = $\{v_1\}$ and E = { }.
2. Choose a nearest neighbour v_i of V that is adjacent to v_j **,** $v_j \in V$ and for which the edge (v_i, v_j) does not form a cycle with member of E. Add v_i to V and add (v_i, v_j) to E.

3. Repeat step 2 till number of edges in T is n – 1. Then V contains all n vertices of G and E contains the edges of a minimal spanning tree for G.

Definition: A **greedy algorithm** is an algorithm that optimizes the choice at each iteration without regard to previous choices.

For example**, Prim algorithm is a greedy algorithm.**

Example: Find a minimal spanning tree for the graph shown below :



Solution: We shall use **Prim algorithm to find the required minimal spanning tree.** We note that number of vertices in this connected weighted graph is 6. Therefore the tree will have 5 edges.
We start with any vertex, say c. The nearest neighbour of c is f and (c f) does not form a cycle. Therefore (c, f) is the first edge selected.
Now we consider the set of vertices V = {c, f}. The vertex a is nearest neighbour to V = {c, f} and the edge (c, a) does not form a cycle with the member of set of edges selected so far. Thus

E = {(c, f) , (c, a)} and V = {c, f, a}.

The vertex b is now nearest neighbour to V = {c, f, a} and the edge (a, b) do not form a cycle with the member of E = {(c, f), (c, a)}. Thus

E = {(c, f), (c, a), (a, b)} and V = {c, f, a, b}

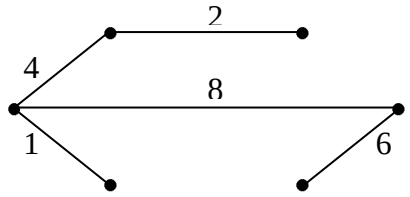
Now the edge (b, c) cannot be selected because it forms a cycle with the members of E. We note that d is the nearest point to V = {c, f, a, b} and (c, d) is the edge which does not form a cycle with members of E = {(c, f), (c, a), (a, b)}. Thus we get

$$E = \{(c, f), (c, a), (a, b)\}, (c, d)\}, \quad V = \{c, f, a, b, d\}$$

The nearest vertex to V is now e and (d, e) in the corresponding edge. Thus

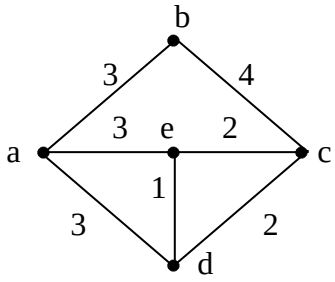
$$E = \{(c, f), (c, a), (a, b), (d, e), V = \{c, f, a, b, d, e\}$$

Since number of edges in the Prim Tree is 5, the process is complete. The minimal spanning tree is shown below :



The length of the tree is 1 + 4 + 2 + 8 + 6 = 21

Example : Using Prim algorithm, find the minimal spanning tree of the following graph :



Solution : Pick up the vertex a. Then

$$E = \{\ \} \quad \text{and} \quad V = \{a\}.$$

The nearest neighbour of V is b or d and the corresponding edges are (a, b) or (a, d). We choose arbitrarily (a, b) and have

$$E = \{(a, b)\}, \quad V = \{a, b\}$$

Now d is the nearest neighbour of V = {a, b} and the corresponding edge (a, d) does not form cycle with (a, b). Thus we get

$$E = \{(a, b), (a, d)\}, \quad V = \{a, b, d\}.$$

Now e is the nearest neighbour of {a, b, d} and (d, e) does not form cycle with {(a, b), (c, d)}. Hence

$$E = \{(a, b), (a, d), (d, e)\}, \quad V = \{a, b, d, e\}$$

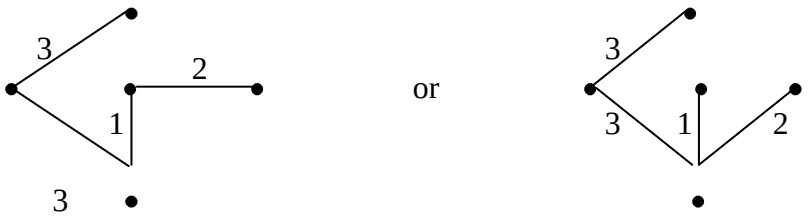
Now c is the nearest neighbour of V = {a, b, d, e} and the corresponding edges are (e, c), (d, c). Thus we have, choosing (e, c) ,

$$E = \{(a, b), (c, d), (d, e), (e, c)\}, \quad V = \{a, b, d, e, c\}$$

Total weight = 3 + 3 + 1 + 2 = 9

(If we choose (d, c), then total weight is 3 + 3 + 1 + 2 = 9.)

The minimal tree is



KRUSKAL'S ALGORITHM

In Kruskal's algorithm, the edges of a weighted graph are examined one by one in order of increasing weight. At each stage an edge with least weight out of edge-set remaining at that stage is added provided this additional edge does not create a circuit with the members of existing edge set at that stage. After n – 1 edges have been added, these edges together with the n vertices of the connected weighted graph form a minimal tree.

ALGORITHM

Input : A connected weighted graph G with n vertices and the set $E = \{e_1, e_2, \ldots\ldots\ldots, e_k\}$ of weighted edges of G.

Output : The set of edges in a minimal spanning tree T for G.

Step 1. Initialize T to have all vertices of G and no edges.

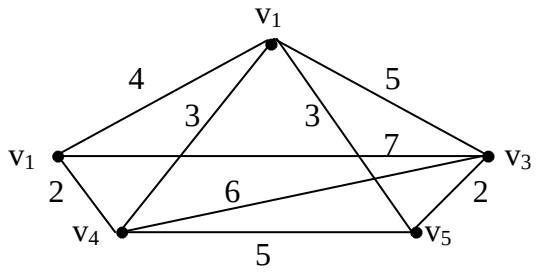
Step 2. Choose an edge e_1 in E of least weight. Let

$$E^* = \{e_1\}, \; E = E - \{e_1\}$$

Step 3. Select an edge e_i in E of least weight that does not form circuit with members of E*. Replace E* by $E^* \cup \{e_i\}$ and E with $E - \{e_i\}$.

Step 4. Repeat step 3 until number of edges in E* is equal to n – 1.

Example : Use Kruskal's algorithm to determine a minimal spanning tree for the connected weighted graph G shown below :



Solution : The given weighted graph has five vertices. The minimal spanning tree would have therefore 4 edges.
Let

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

The edges (v_2, v_4) and (v_3, v_5) have minimum weight. We choose arbitrarily one of these, say
(v_2, v_4). Thus

$$E^* = \{(v_2, v_4)\},$$
$$E = E - \{(v_2 , v_4)\}.$$

The edge (v_3, v_5) has minimum weight, so we pick it up. We have thus

$$E^* = \{(v_2, v_4), (v_3, v_5)\},$$
$$E = E - \{(v_2, v_4), (v_3, v_5)\}$$

The edges (v_1, v_4) and (v_1, v_5) have minimum weight in the remaining edge set. We pick (v_1, v_4) say, as it does not form a cycle with E*. Thus

$$E^* = \{(v_2, v_4), (v_3, v_5), (v_1, v_4),$$
$$E = E - \{(v_2, v_4), (v_3, v_5), (v_1, v_4)\}$$

Now the edge (v_1, v_5) has minimum weight in $E \setminus \{(v_1, v_4), (v_3, v_5), (v_1, v_4)\}$ and it does not form a cycle with E*. So, we have

$$E^* = \{(v_2, v_4), (v_3, v_5), (v_1, v_4), (v_1, v_5)\}$$

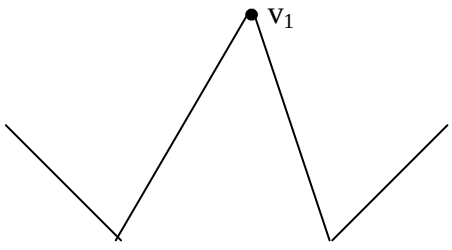
and

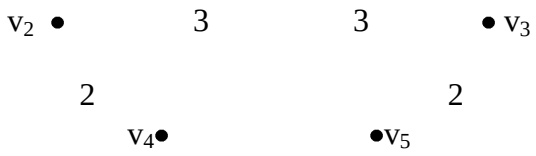
$$E = E - \{(v_2, v_4), (v_3, v_5), (v_1, v_4), (v_1, v_5)\}$$

Thus all the four edges have been selected. The minimal tree has the edges.

$$(v_2, v_4), (v_3, v_5), (v_1, v_4), (v_1, v_5)$$

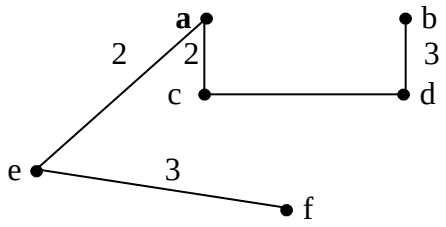
and is shown below :





(Minimal Spanning tree)

Remark : In the above example, if we had chosen (e, f) in place of (c, f) in the last step, then the minimal spanning tree would have been



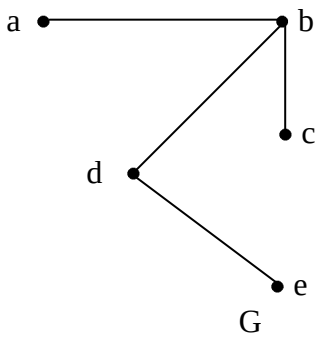
(minimal spanning tree)

3.17 Cut Sets

Let G be a connected graph. We know that the distance between two vertices v_1 and v_2, denoted by $d(v_1, v_2)$, is the **length of the shortest path.**

Definition: The **diameter** of a connected graph G, denoted by diam (G), is the maximum distance between any two vertices in G.

For example, in graph G shown below, we have



G

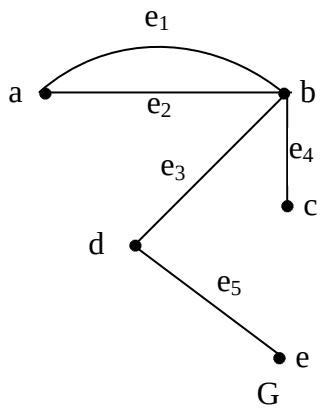
d(a, e) = 3, d(a, c) = 2, d(b, e) = 2 and diam (G) = 3.

Definition: A vertex in a connected graph G is called a **cut point** if G – v is disconnected, where G – v is the graph obtained from G by deleting v and all edges containing v.

For example, in the above graph, d is a cut point.

Definition: An edge e of a connected graph G is called a **bridge** (or cut edge) if G – e is disconnected, where G – e is the graph obtained by deleting the edge e.

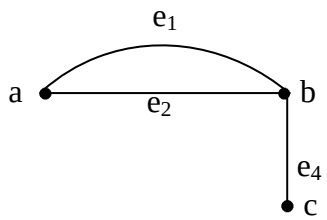
For example, consider the graph G shown below :

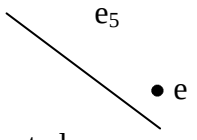


We observe that G – e_3 is disconnected. Hence the edge e_3 is a bridge.

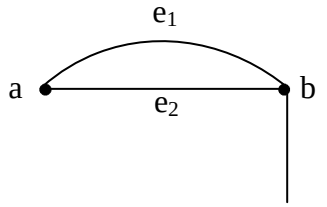
Definition: A minimal set C of edges in a connected graph G is said to be a **cut set (**or **minimal edge – cut)** if the subgraph G – C has more connected components than G has.

For example, in the above graph, if we delete the edge (b, d) = e_3, the resulting subgraph G – e_3 is as shown below :

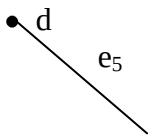


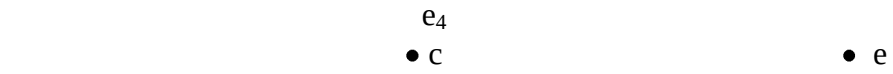


Thus G – e_3 has two connected components



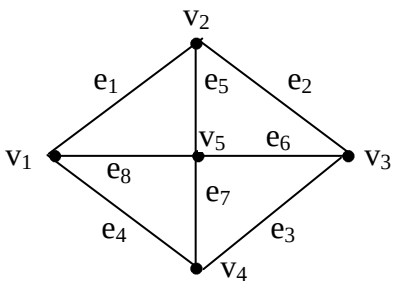
and



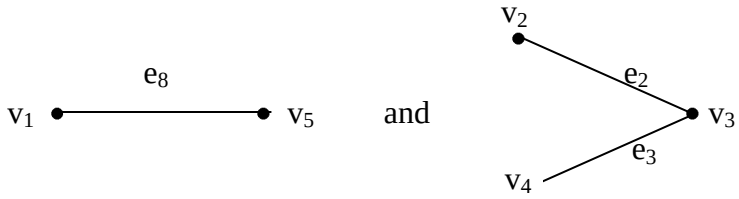


So, in this example, the cut set consists of single edge (b, d) = e_3, which is called edge or bridge.

Example: Find a cut set for the graph given below:



Solution : The given graph is connected. It is sufficient to reduce the graph into two connected components. To do so we have to remove the edges e_1, e_4, e_5, e_6, e_7. The two connected components are

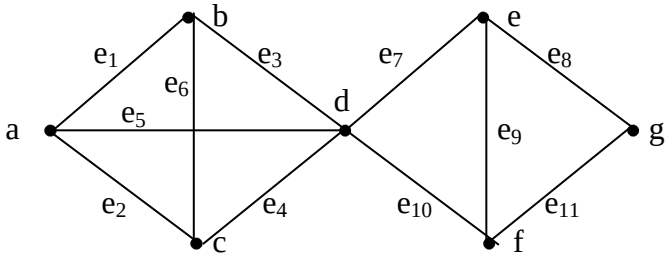


But, if we remove any proper subset of $\{e_1, e_4, e_5, e_6, e_7\}$, then there is no increase in connected components of G.
Hence

$$\{e_1, e_4, e_5, e_6, e_7\}$$

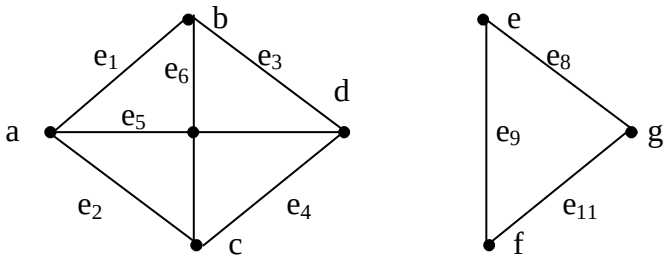
is a cut set.

Example: Find a cut set for the graph



G

Solution: The given graph is a connected graph. We note that removal of the edges e_7 and e_{10} creates two connected components of G shown below:



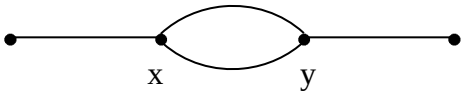
Hence the set $\{e_7, e_{10}\}$ is a cut set for the given graph G.

Theorem: Let G be a connected graph with n vertices. Then G is a tree if and only if every edge of G is a bridge (cut edge).

(This theorem asserts that every edge in a tree is a bridge).

Proof: Let G be a tree. Then it is connected and has n – 1 edges (proved already). Let e be an arbitrary edge of G. Since G – e has n – 2 edges, and also we know that a graph G with n vertices has at least n – c(G) edges, it follows that $n - 2 \geq n - c(G - e)$. Thus G – e has at least two components. Thus removal of the edge e created more components than in the graph G. Hence e is a cut edge. This proves that every edge in a tree is a bridge.

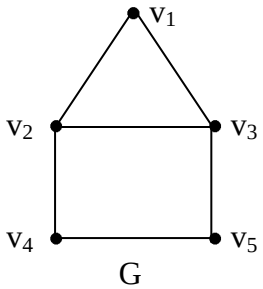
Conversely, suppose that G is connected and every edge of G is a bridge. We have to show that G is a tree. To prove it, we have only to show that G is circuit – free. Suppose on the contrary that there exists a cycle between two points x and y in G. Then any edge on this cycle is



not a cut edge which contradicts the fact that every edge of G is a cut edge. Hence G has no cycle. Thus G is connected and acyclic and so is a tree.

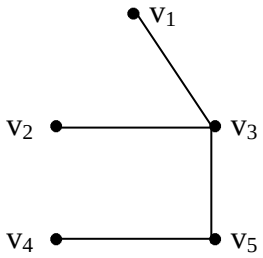
3.18 Relation Between Spanning Trees, Circuits and Cut Sets

A spanning tree contains a unique path between any two vertices in the graph. Therefore, addition of a chord to the spanning tree yields a subgraph that contains exactly one circuit. For example, consider the graph G shown below:



G

For this graph, the figure given below is a spanning tree :



(Spanning tree)

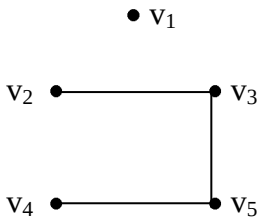
The chords of this tree are (v_1, v_2) and (v_2, v_4). If we add (v_1, v_2) to this spanning tree, we get a circuit $v_1\ v_1\ v_2\ v_3\ v_1$. Similarly addition of (v_2, v_4) gives one more circuit $v_2\ v_3\ v_5\ v_4\ v_2$. If there are v vertices and e edges in a graph, then there are $e - v + 1$ chords in a spanning tree. Therefore, if we add all the chords to the spanning tree, there will be $e - v + 1$ circuits in the graph.

Definition: Let v be the number of vertices and e be the number of edges in a graph G. Then the set of $e - v + 1$ circuits obtained by adding $e - v + 1$ chords to a spanning tree of G is called the **fundamental system of circuits relative to the spanning tree.**

A circuit in the fundamental system is called a **fundamental circuit.**

For example, $\{v_1, v_2, v_3, v_1\}$ is the fundamental circuit corresponding to the chord (v_1, v_2).

On the other hand, since each branch of a tree is cut edge, removal of any branch from a spanning tree breaks the spanning tree into two trees. For example, if we remove (v_1, v_3) from the above figured spanning tree, the resulting components are shown in the figure below :



Thus, **to every branch in a spanning tree, there is a corresponding cut set.** But, in a spanning tree, there are $v - 1$ branches. Therefore, **there are $v - 1$ cut sets corresponding to $v - 1$ branches.**

Definition: The set of $v - 1$ cut sets corresponding to $v - 1$ branches in a spanning tree of a graph with v vertices is called the **fundamental system of cut sets relative to the spanning tree.**

A cut – set in the fundamental system of cut – sets is called a **fundamental cut set.**

For example, the fundamental cut – sets in the spanning tree (figured above) is

$\{(v_1, v_2), (v_1, v_3)\}$, $\{(v_1, v_3), (v_2, v_3), (v_3, v_4)\}$,

$\{(v_3, v_5), (v_4, v_5)\}$, $\{v_2, v_4), (v_4, v_5)\}$.

Theorem: A circuit and the complement of any spanning tree must have at least one edge in common.

Proof: We recall that the set of all chords of a tree is called the complement of the tree. Suppose on the contrary that a circuit has no common edge with the complement of a spanning tree. This means the circuit is wholly contained in the spanning tree. This contradicts the fact that a tree is acyclic (circuit – free). Hence a circuit has at least one edge in common with complement of a spanning tree.

Theorem: A cut – set and any spanning tree must have at least one edge in common.

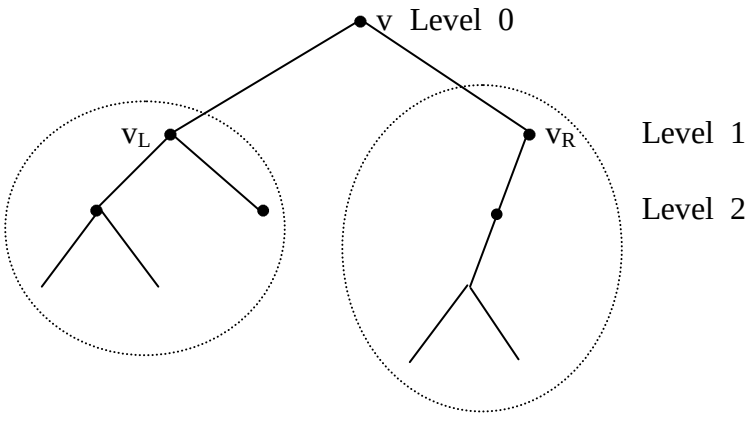
Proof: Suppose on the contrary that there is a cut set which does not have a common edge with a spanning tree. Then removal of cut set has not effect on the tree, that is, the cut set will not separate the graph into two components. But this contradicts the definition of a cut set. Hence the result.

Theorem: Every circuit has an even number of edges in common with every cut – set.

Proof: We know that a cut – set divides the vertices of the graph into two subsets each being set of vertices in one of the two components. Therefore a path connecting two vertices in one subset must traverse the edges in the cut set an even number of times. Since a circuit is a path from some vertex to itself, it has an even number of edges in common with every cut – set.

3.19 Tree Searching

Let T be a binary tree of height $h \geq 1$ and root v. Since $h \geq 1$, v has at least one child : v_L and / or v_R. Now v_L and v_R are the roots of the left and right subtrees of v called T_L and T_R respectively.



Left subtree T_L (dotted circle)

Right subtree T_R (dotted circle)

Definition: Performing appropriate tasks at a vertex is called **visiting the vertex.**

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

We now discuss methods of searching a tree.

PREORDER SEARCH METHOD

Input : the root v of a binary tree.

Output : Vertices of a binary tree using pre-order traversal

1. Visit v
2. If v_L (left child of v) exists, then apply the algorithm to $(T(v_L), v_L)$
3. If v_R (right child of v) exists, then apply this algorithm to $(T(v_R), v_R)$.

End of Algorithm preorder.

In other words, preorder search of a tree consists of the following steps:

Step 1. Visit the root
Step 2. Search the left subtree if it exists

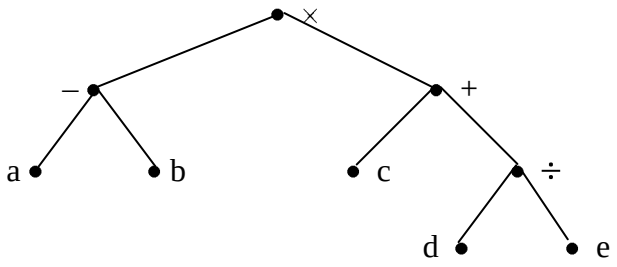
Step 3. Search the right subtree if it exists.

Example 1: Find binary tree representation of the expression

$$(a - b) \times (c + (d \div e))$$

and represent the expression in string form using pre-order traversal.

Solution: In the given expression, $\times$ is the central operator and therefore shall be the root of the binary tree. Then the operator – acts as v_L and the operator + acts as v_R. Thus the tree representation of the given expression is



The result of the pre-order traversal to this binary tree is the string

$$\times \; - \; a \; b \; + \; c \; \div \; d \; e$$

This form of the expression is called **prefix form** or **polish form** of the expression

$$(a - b) \times (c + (d \div e))$$

In a polish form, the variables a, b, c,…are called operands and –, +, ×, ÷ are called **operators**. We observe that, **in polish form, the operands follow the operator.**

PROCEDURE TO EVALUATE AN EXPRESSION GIVEN IN POLISH FORM

To find the value of a polish form, we proceed as follows:

Move from left to right until we find a string of the form K x y, where K is operator and x, y are operands.

Evaluate x K y and substitute the answer for the string K x y. Continue this procedure until only one number remains.

Example: Find parenthesized form of the polish expression

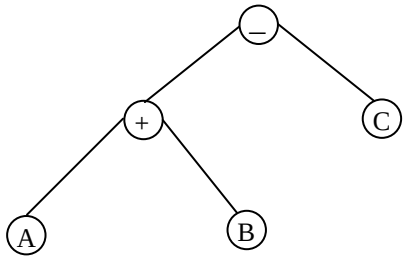
$$- \; + \; A \; B \; C$$

Solution: The parenthesized form of the given polish expression is derived a follows:

$$- \; (A + B) \; C$$

$$(A + B) - C$$

The corresponding binary tree is



POSTORDER SEARCH METHOD

Algorithm

Step 1. Search the left subtree if it exists
Step 2. Search the right subtree if it exists
Step 3. Visit the root

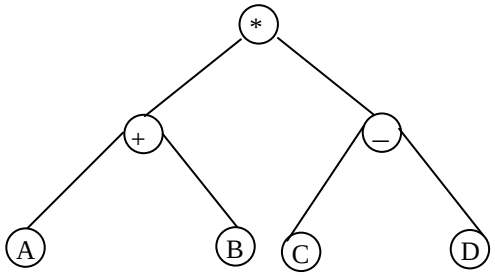
End of algorithm

Example: Represent the expression

$$(A + B) * (C – D)$$

as a binary tree and write the result of postorder search for that tree.

Solution: The binary tree expression (as shown earlier) of the given algebraic expression is



The result of postorder search of this tree is

$$A\ B + C\ D – *$$

This form of the expression is called postfix form of the expression or reverse polish form of the expression.

In postfix form, the operator follows its operands.

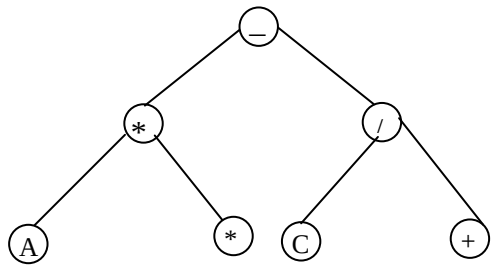
Example: Find the parenthesized form of the postfix form

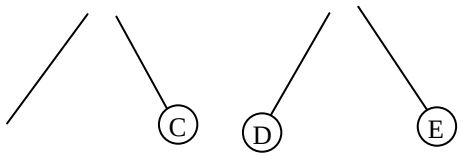
$$A\ B\ C\ *\ *\ C\ D\ E\ +\ /\ –$$

Solution: We have

1. A B C * * C D E + / –
2. A (B * C) * C (D + E) / -
3. (A * (B * C)) (C / (D + E)) –
4. (A * (B * C)) – (C / (D + E))

The corresponding binary tree is





Example: Evaluate the postfix form

$$21 - 342 \div + \times$$

Solution: We have

$$21 - 342 \div + \times$$

$$= (2 - 1)\ 342 \div + \times$$

$$= 13\ (4 \div 2) + \times$$

$$= 132 + \times$$

$$= 1\ (3 + 2) \times$$

$$= 15 \times$$

$$= 1 \times 5$$

$$= 5\ .$$

Unit-4
Computability Theory

4.1. Finite State Machine

Definition: A **finite – state machine** (or complete sequential machine) is an abstract model of a machine with a primitive internal memory. A finite state machine M consists of
(1) A finite set I of **input** symbols

(2) A finite set S of "**internal**" states

(3) A finite set O of **output** symbols

(4) An initial stage s_0 in S

(5) A next – stage function $f : S \times I \to S$

(6) An output function $g : S \times I \to O$

A finite state machine M is denoted by

$M = M (I, S, O, s_0, f, g)$.

Example: 1. Let us take

$I = \{a, b\}$

$S = \{s_0, s_1, s_2)$

$O = \{x, y, z\}$

Initial State is s_0

Next state function $f : S \times I \to S$ defined by

$f (s_0, a) = s_1$, $f (s_1, a) = s_2$, $f (s_2, a) = s_0$

$f (s_0, b) = s_2$, $f (s_1, b) = s_1$, $f (s_2, b) = s_1$

Output function $g : S \times I \to O$ defined by

$g (s_0, a) = x$, $g(s_1, a) = x$, $g(s_2, a) = z$

$g (s_0, b) = y$, $g(s_1, b) = z$, $g(s_2, b) = y/$

Then $M = M(I, S, O, s_0, f, g)$ in a finite state machine.

TRANSITION (STATE) TABLE AND TRANSITION (STATE) DIAGRAM

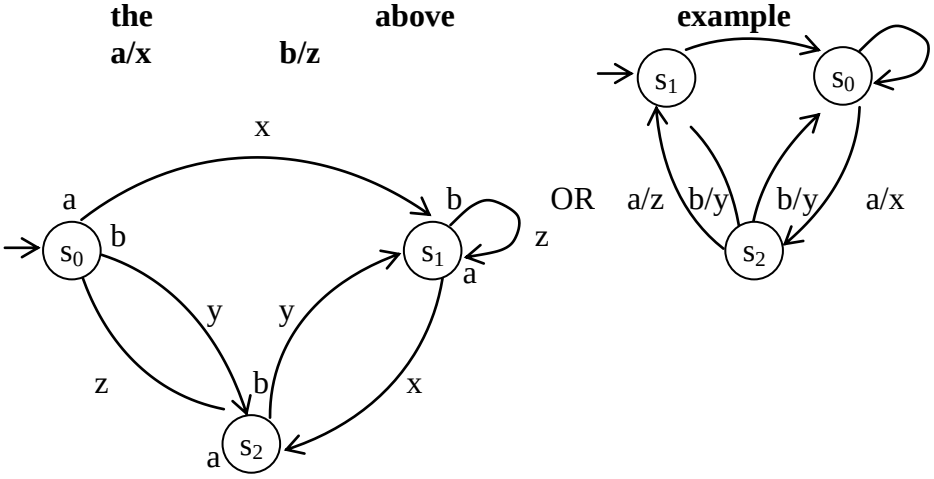
There are two ways of representing a finite state machine M in a compact form:

(A) : **Transition (State) Table :** In this table the functions f and g are represented by a table.
Thus, in case of the above example, the transition table is

	f		g	
I / S	a	b	a	b
s_0	s_1	s_2	x	y
s_1	s_2	s_1	x	z
s_2	s_0	s_1	z	y

(B) **Transition (State) diagram:** A transition diagram of a finite state machine M is a labeled directed graph in which there is a node for each state symbol in S and each node is lebeled by a state symbol with which it is associated. The initial stage is indicated by an arrow. Moreover, if $f(s_i, a_j) = s_k$ and $g(s_i, a_j) = O_r$, then there is an arrow (arc) from s_i to s_k which is labeled with the pair a_j, O_r. We usually put the input symbol a_j near the base of the arrow (near s_i) and the output symbol O_r near the centre of the arrow. **(Also, we can represent it by a_i/O_i near the centre of the arrow)**

Thus, the transition diagram of the finite state machine in the above example is



Example: Let I = {a, b}, O = {0, 1} and S $\{s_0, s_1\}$. Let so be the initial state.

Define $f : S \times I \to S$ by

$$f(s_0, a) = s_0,\ f(s_0, b) = s_1,\ f(s_1, a) = s_1,\ f(s_1, b) = s_1$$

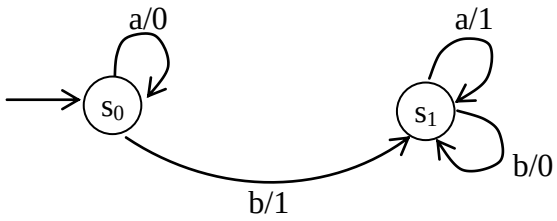
and define $g : S \times I \to O$ by

$$g(s_0, a) = 0,\ g(s_0, b) = 1,\ g(s_1, a) = 1,\ g(s_1, b) = 0$$

Then M = M(I, S, O, s_0, f, g) is a finite state machine. Its transition table representation is given below:

	f	G
I / S	a b	a b
s_0	s_0 s_1	0 1
s_1	s_1 s_1	1 0

The transition diagram for this finite state machine is



Remark: We can regard the finite state machine M = M(I, S, O, s_0, f, g) as a simple computer. We begin in state S, input a string over I, and produce a string of output.

INPUT AND OUTPUT STRINGS

Let M = M(I, S, O, s_0, f, g) be a finite state machine. An **input string** for M is a string over I.
The string

$$y_1\ y_2\ \ldots\ldots\ldots y_n$$

is the **output string** for M corresponding to the input string

$$x_1\ x_2\ldots\ldots\ldots x_n$$

if there exists states $s_0, s_1, \ldots\ldots, s_n \in S$ such that

$$s_i = f(s_{i-1}, x_i) \quad \text{for } i = 1, 2, \ldots\ldots, n$$

$$y_i = g(s_{i-1}, x_i) \quad \text{fo } i = 1, 2, \ldots\ldots\ldots, n$$

Example: In the above example, we had taken

$$I = \{a, b\},\ O = \{0, 1\} \text{ and } S\ \{s_0, s_1\}$$

with

$$f(s_0, a) = s_0,\ f(s_0, b) = s_1,\ f(s_1, a) = s_1,\ f(s_1, b) = s_1$$

and

$$g(s_0, a) = 0,\ g(s_0, b) = 1,\ g(s_1, a) = 1,\ g(s_1, b) = 0$$

We had shown that M = M(I, S, O, s_0, f, g) is a F S M. We want to find the output string to the input string

a a b a b b a

for this machine.

Initially we are in a state s_0. The first symbol input is a. Therefore the output is $g(s_0, a) = 0$. The edge points out to S_0. Next symbol input is again a. So again we have $g(s_0, a) = 0$ as the output and the edge points out to s_0 . Next b is the input symbol and so $g(s_0, b) = 1$ as the output and there is a state of change s_1. Next symbol is a, so $g(s_1, a) = 1$ as the output and the state is s_1. Now b is input and so $g(s_1, b) = 0$ as the output. Again b is input and s_1 is the state, so $g(s_1, b) = 0$. The last input symbol is a and the state is s_1. Therefore $g(s_1, a) = 1$ as the output symbol.

Thus the output string is

0 0 1 1 0 0 1

Example: Consider the F S M of example …. Let the input string be

a b a a b.

we begin by taking S_0 as the initial stage. Using State diagram we have

$$s_0 \xrightarrow{a, x} s_1 \xrightarrow{b, z} s_1 \xrightarrow{a, x} s_2 \xrightarrow{a, z} s_0 \xrightarrow{b, y} s_2$$

Hence the output string is

x z x z y

BINARY ADDITION

We want to describe a finite state machine M which can perform binary addition. Suppose that the machine is given the input

$$\begin{array}{r} 1\,1\,0\,1\,0\,1\,1 \\ +\,0\,1\,1\,1\,0\,1\,1 \end{array} \quad ,$$

then we want to have the output to be the binary sum

1 0 1 0 0 1 1 0

Thus the input is the string of pairs of digits to be added:

11, 11, 00, 11, 01, 11, 10, b ,

where b denotes blank spaces and the output should be the string

0, 1, 1, 0, 0, 1, 0, 1

We also want the machine to enter a state called “stop” when the machine finishes the addition.
The input symbols are

I = {00, 01, 10, 11, b}

and the output symbols are

O = {0, 1, b}

The machine that we construct will have three states:

S = {carry(c), no carry (nc), stop(s)}.

In this case nc is the initial state.

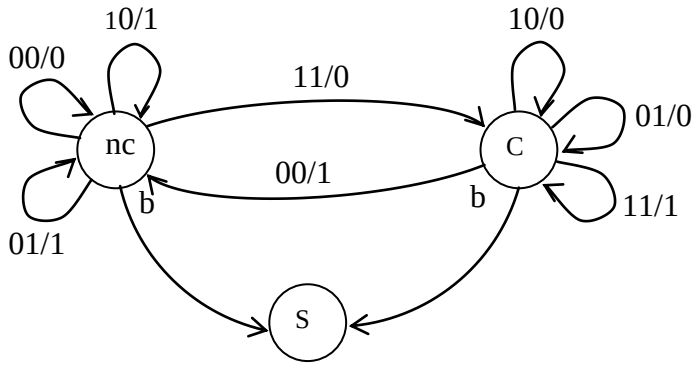
In fact, given an input x y, we take one of three actions :

(A) we add x and y if carry bit is 0

(B) we add x, y and 1 if carry bit is 1

(C) we stop

Next, we consider the possible inputs at each vertex. For examples if 00 is input to nc, we should output 0 and remain in the state nc. Thus nc has a loop labeled 00/0. As another example, if 11 is input to c, we compute 1 + 1 = 11. In this case we output 1 and remain in the state C. Thus C has a loop labeled 11/1. As a final example, if we are in state NC and 11 is input, we should output 0 and move to the state G. By considering all possibilities, we arrive at the transition diagram given below:



Limitation of Machines: **There is no finite state machine which can perform binary multiplication.**

Generalization of f and g in the definition of F S M: **Consider a sequence $x_0\ x_1$ …… of input symbols. Let s_0 be the initial stage. Then the next state s_1 of the machine for the input x_0 is given by $s_1 = f(s_0, x_0) = f_1(s_0, x_0)$ say, where $f = f_1$: $S \times I \to S$. Next consider the change in state due to second input symbol x_1 and the next state is $s_2 = f(s_1, x_1) = f(f_1(s_0, x_0), x_1) = f_2(s_0, x_0\ x_1)$, where f_2: $S \times I^2 \to S$. The next state due to third input symbol x_2 is $s_3 = f(s_2, x_2) = f(f_2(s_0, x_0\ x_1), x_2) = f_3(s_0\ x_0\ x_1\ x_2)$, where f_3 : $S \times I^3 \to S$. Continuing in this fashion, we can define a function**

$$f_n : S \times I^n \rightarrow S \text{ such that}$$

$$s_n = f(f_{n-1}(s_0, x_0\, x_1 \ldots x_{n-2}), x_{n-1}) = f_n(s_0, x_0\, x_1 \ldots .x_{n-1})$$

Similarly, the output symbol 0_0, 0_1, ….. can be described with the help of g as shown below :

$$o_0 = g(s_0, x_0) = g_1(s_0, x_0)$$

$$o_1 = g(s_1, x_1) = g(f_1(s_0, x_0), x_1) = g_2(s_0, x_0, x_1)$$

$$o_{n-1} = g(s_{n-1}, x_{n-1}) = g_n(s_0, x_0\, x_1 \ldots x_{n-1}) .$$

4.2. Equivalence of Finite State Machines

The aim of this section is to obtain an equivalent minimal machine for some given machine. First we treat equivalent states. Intuitively, two states are equivalent if and only if they produce the same output for any input sequence. Thus we can make the following definition:

Definition: **Let M = M{I, S, O, s_0, f, g) be a finite state machine. Two states $s_i, s_j \in S$ are said to be** equivalent, **written as $s_i \equiv s_j$, if and only if**

$$g(s_i, x) = g(s_j, x) \text{ for every word } x \in I^*,$$

where I* denotes the set of words on the input alphabets.

It can be seen that the relation $\equiv$ is an equivalence relation.

Theorem: **Let s be any state in a finite – state machine and let x and y be any words. Then**

$$f(s, x\, y) = f(f(s, x), y)$$

and

$$g(s, x\, y) = g(f(s, x), y).$$

Proof: **We shall prove the theorem by induction on length of y. Let y = a. Then**

$$f(s, x\, a) = f(f(s, x), a)$$

Assume that the equation is true for any y of length n, that is,

$$f(s, x\, y) = f(f(s, x), y)$$

We want to show that it is true for y having n + 1 symbols.
From the generalized definition, we can write

$$f(s, x\, y\, a) = f(f(s, x\, y), a) = f(f(f(s, x), y), a)$$

by the induction hypothesis. Taking $s' = f(s, x)$,we have

$$f(f(f(S, x), y), a) = f(f(s', y), a)$$
$$= f(s', y\ a)$$
$$= f(f(s, x), y\ a)$$

The result regarding g may be established similarly.

Theorem: **Let $M = M(I, S, O, s_0, f, g)$ be a finite state machine. If the states s_i and s_j are equivalent, then for any input sequence x,**

$$f(s_i, x) = f(s_j, x),$$

that is, if two states are equivalent, then their next states are also equivalent.

Proof: **Since $s_i \equiv s_j$, it follows by definition that**

$$g(s_i, x\ y) = g(s_j, x\ y) \qquad (1)$$

for any input word x y. Then, by the above theorem, (1) reduces to

$$g(f(s_i, x)\ y) = g(f(s_j, x), y)$$

for any y belonging to the set of words I*, which in term of definition of equivalence of states implies

$$f(s_i, x) \equiv f(s_j, x),$$

that is the next states are equivalent.

Definition: **Let $M = (I, S, O, s_0, f, g)$ be a finite state machine. Then for some positive integer k, s_i is said to be** k – equivalent **to s_j if and only if**

$$g(s_i, x) = g(s_j, x) \text{ for all x such that } |x| \leq k.$$

Obviously, equivalence of states is a generalization of k –equivalence of states for all k, that is,

$$s_i \equiv s_j \Rightarrow s_i \overset{k}{\equiv} s_j$$

but not conversely.

Definition: Let $M = (I, S, O, s_i, f, g)$ and $M' = (I, S', O, s_i', f', g')$ be finite – state machines. Then M is said to be equivalent to M′, written as $M \equiv M'$ if and only if for all $s_i \in S$, there exists an $s'_j \in S'$ such that

$$s_i \equiv s_j$$

and for all $s_j \in S'$, there exists an $s_i \in S$ such that

$$s_i \equiv s'_j.$$

The relation $\equiv$ is an equivalence relation.

For example, consider two finite state machines whose transition tables are

	f		g	
S \ I	0	1	0	1
s_0	s_5	s_3	0	1
s_1	s_1	s_4	0	0
s_2	s_1	s_3	0	0
s_3	s_1	s_2	0	0
s_4	s_5	s_2	0	1
s_5	s_4	s_1	0	1

M(I, S, O, S_i, f, g)

and

	f′		g′	
I′ \	0	1	0	1
s_0	s_3'	s_2'	0	1
s_1	s_1'	s_0'	0	0
s_2	s_1'	s_2'	0	0
s_3	s_0'	s_1'	0	1

M′(I, S′, O, S′, f ′, g′)

Observe that s_0' in M′ is equivalent to s_0 and s_4 in M; s_1' in M′ is equivalent to s_1 in M; s_2' in M′ is equivalent to s_2 and s_3 in M, and s_3' in M′ is equivalent to s_5 in M. Also note that the functions g and g′ are same for the indicated correspondence, but this is only a necessary condition for equivalence, not a sufficient one.

Definition: A finite state machine M = (I, S, O, s_i, f, g) is said to be **reduced** if and only if $s_i \equiv s_j$ implies that $s_i = s_j$ for all states $s_i, s_j \in S$.

Thus, a reduced finite state machine is one in which each state is equivalent to itself and to no other. The partition of S in such machine has all its equivalence classes consisting of a single element.

CONSTRUCTION OF A REDUCES FINITE STATE MACHINE WHICH IS EQUIVALENT TO SOME GIVEN MACHINE

Let M be a given machine. Let the set of states S be partitioned in a set of equivalence classes [s] such that partition P = U[s]. Let φ be the function defined on the partition P such that $\varphi([s]) = s'$, where s' is an arbitrary fixed element of [s], called a representative. Clearly $s' \equiv s$ in M. Let S' in M' be defined as

$$S' = \{s' : \text{there exists } s \in S \text{ such that } \varphi([s]) = s'\}$$

and let $I' = I$ and $O' = O$, that is, both machines will have the same input and output alphabets. The functions f' and g' are defined as follows :

$$f'(s', a) = \phi([f(s', a)])$$

and

$$g'(s', a) = g(s', a),$$

where s' is both in S and S'. Therefore, the reduced machine is $M' = (I, S', O, s_i' f', g')$.

Remark: **Applying this procedure to the machines in the last example, we see that M' is equivalent reduced machine of the machine M.**

Theorem: **Let $M = M(I, S, O, s_i, f, g)$ be a finite – state machine. Then there exists an equivalent machine M' with a set of states S' such that $S' \subseteq S$ and M' is reduced.**

(Proof of this theorem is out of the scope of the course)

Definition: **Let $M = (I, S, O, s_i, f, g)$ and $M' = (I, S', O, s_i', f', g')$ be two finite state machines. Let φ be a mapping from S into S'. Then φ is called a finite state homorphism if**

$$\left.\begin{aligned}\varphi(f(s,a) &= f'(\varphi(S),a)\\ g(s,a) &= g'(\varphi(s),a)\end{aligned}\right\} \text{ forall } a \in I$$

If φ is further a one - one and onto function, then M is said to be isomorphic to M'.

Finite – state machines are used in compilers where they usually perform the task of a scanner. The machine in such a case identify variable names, operators, constants, etc. A machine which performs this scanning task is called an acceptor.

4.3. Finite – State Automata

Definition: A **finite state automaton** (.F S A) or simply an a**utomaton** M or **finite state acceptor** consists of

(1) a finite set I, called the input alphabet of input symbols

(2) a finite set S of states

(3) a subset A of S of accepting states

(4) an initial state s_0 in S

(5) a next state function f from $S \times I \rightarrow S$.

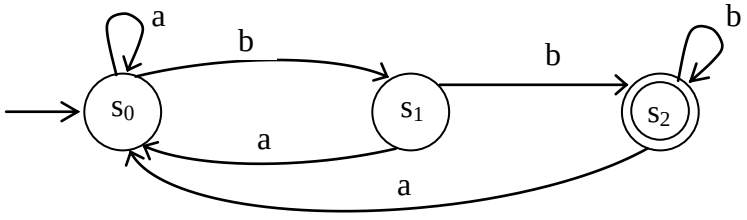
Such an automaton is denoted by M = (I, S, A, s_0, f) . Thus, finite automaton does not have an output alphabet, instead it has a set of acceptance state. The plural of automaton is **automata.**

Example : Let

I = {a, b}, S = {s_0, s_1, s_2}, A = {s_2}, $s_0 \in$ S, the initial state and f is given by the table

	f	
S \ I	a	b
s_0	S_0	s_1
s_1	s_0	s_2
s_2	s_0	s_2

The transition diagram of a finite – state automation is usually drawn with accepting states in double circles. Thus transition diagram for the example in question is



Example: Let
I = {a, b}, input symbols

S = {s_0, s_1, s_2}, internal states

$A = \{s_0, s_1\}$, yes states (accepting states)

s_0, initial state

Next state function $f : S \times I \to S$ defined by

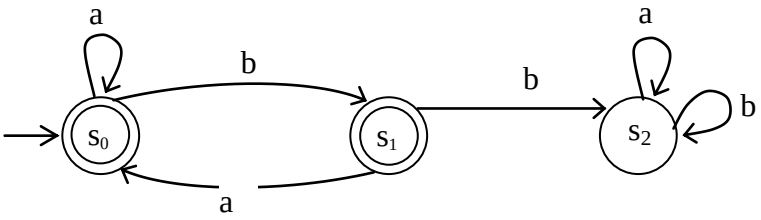
$$f(s_0, a) = s_0,\ f(s_1, a) = s_0,\ f(s_2, a) = s_2$$

$$f(s_0, b) = s_1,\ f(s_1, b) = s_2,\ f(s_2, b) = s_2$$

Then $M = (I, S, A, s_0, f)$ is a finite state automaton. Its transition table is

	f	
I / S	a	b
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_2	s_2

and the transition diagram is



If a string is input to a finite state automaton, we will end at either an accepting or a non-accepting state. The status of this final state determines whether the string is accepted by the finite state automaton.

Definition: **Let $M = (I, S, A, f, s_0)$ be a finite state automaton. Let $x_1 \ldots x_n$ be a string over I. If there exist states $s_0, s_1, \ldots\ldots, s_n$ such that**

$$f(s_{i-1}, x_i) = s_i \text{ for } i = 1, 2, \ldots, n$$

and

$$s_i \in A,$$

then we say that the string $x_1 \ldots\ldots x_n$ is accepted by A.

We call the directed path $P(s_0, \ldots, s_n)$ the path representing $x_1, \ldots, x_n$ in M. Thus M accepts $x_1 \ldots\ldots x_n$ if and only if path P ends at an accepting state.

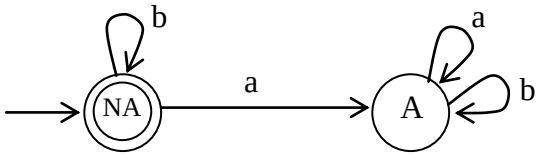
Example: **Design a finite state automation that accepts precisely those strings over {a, b} that contain no a's.**

Solution: **We want to have two states:**

A : an a was found

NA : No a's were found

The state NA is the initial state and the only accepting state.



If f is next – state function, then

f(NA, a) = A, f(NA, b) = NA

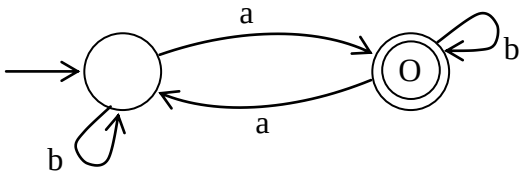
f(A, a) = A, f(A, b) = A

Example: **Design a finite – state – automaton that accepts precisely those strings over {a, b} that contains an odd number of a's.**

Solution: **There shall be two states:**

E : An even number of a's was found
O : An odd number of a's was found

The initial state is E and the accepting state is O.



If f is next – state function, then we have

f(E, a) = O

f(E, b) = E

f(O, a) = E

f(O, b) = O

Example : Let M = {I, S, A, s_0, f) be a finite state automaton with

I = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

S = {s_0, s_1, s_2}

A = {s_0}

a ∈ {0, 3, 6, 9}, b ∈ {1, 4, 7}, c ∈ {2, 5, 8}.

Next – state function f defined by

f(s_0, a) = s_0, f(s_0, b) = s_1, f(s_0, c) = s_2

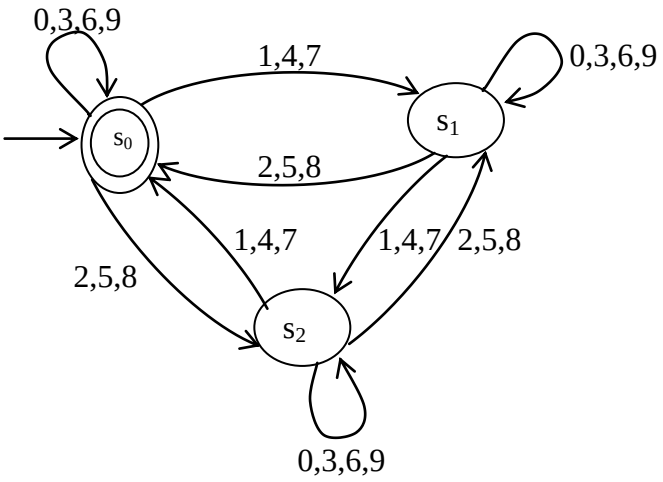
$$f(s_1, a) = s_1, \quad f(s_1, b) = s_2, \quad f(s_1, c) = s_0$$
$$f(s_2, a) = s_2, \quad f(s_2, b) = s_0, \quad f(s_2, c) = s_1$$

Draw transition table and transition diagram for this F.S.A. Does this automaton accept 258 and 142 ?

Solution: The transition table for F.S.A. is

	f		
I	a	b	c
s_0	s_0	s_1	s_2
s_1	s_1	s_2	s_0
s_2	s_2	s_0	s_1

The transition diagram for this F.S.A. is



Here $A = \{s_0\}$ is the initial stage and also is an acceptor. Further, we note that

$$f(s_0, 258) = f(f(s_0, 25), 8)$$
$$= f(f(f(s_0, 2), 5), 8)$$
$$= f(f(s_2, 5), 8)$$
$$= f(s_1, 8) = s_0 \in A$$

Thus, the string 258 determines the path

$$s_0 \xrightarrow{2} s_2 \xrightarrow{5} s_1 \xrightarrow{8} s_0 \in A$$

Hence 258 is accepted by the given Finite State Automation.

On the other hand,

$$f(s_0, 142) = f(f(s_0, 14), 2)$$
$$= f(f(f(s_0, 1), 4), 2)$$
$$= f(f(s_1, 4), 2)$$
$$= f(s_2, 2)$$
$$= s_1 \notin A$$

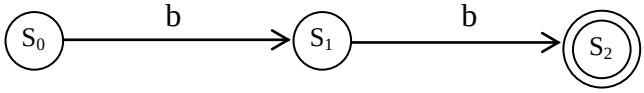
Thus, the string 142 determines the path

$$\mathbf{s_0 \xrightarrow{1} s_1 \xrightarrow{4} s_2 \xrightarrow{2} s_1 \notin A .}$$

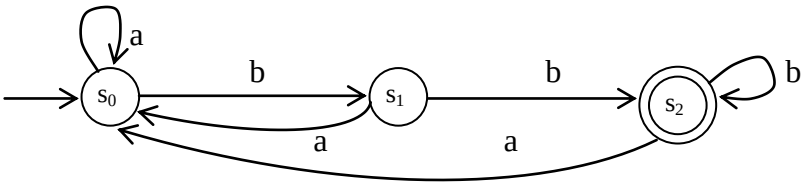
Hence 142 is not accepted by the given Finite State Automaton.

Example: Construct F S A which will accept precisely those strings from I = {a, b} which end in two b's.

Solution: As per our requirement b b should be accepted by M but $\in$ or b should not be accepted. Thus we need three states: s_0 (the initial state) , s_1 and s_2 as shown below:



The state s_2 should be the accepting state. Further $f(s_0, a)$ should not be equal to s_1, because then a b may be the last letters. $f(s_2, a)$ should not be equal to s_2, otherwise ba would be last letters. However, $f(s_2, b)$ may be equal to s_2 because in that case we have last two letters as b's. Thus the automaton is as shown below :

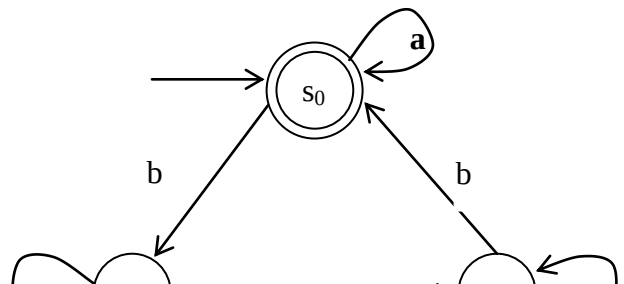


Example: Let I = {a,b}. Construct an automaton M such that L(M) consists of those strings where the number of b's is divisible by 3.

Solution: **We take s_0 as the initial state. If we define the next state function f by**

$$\mathbf{f(s_0, b) = s_1}$$
$$\mathbf{f(s_1, b) = s_2}$$
$$\mathbf{f(s_2, b) = s_0}$$

and take s_0 as the accepting state, then M shall be



a **a**

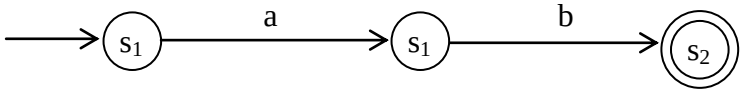
b

Example: **Let I = {a, b}. Construct an automaton M such that L(M) consists of those words which begin with a and end with b.**

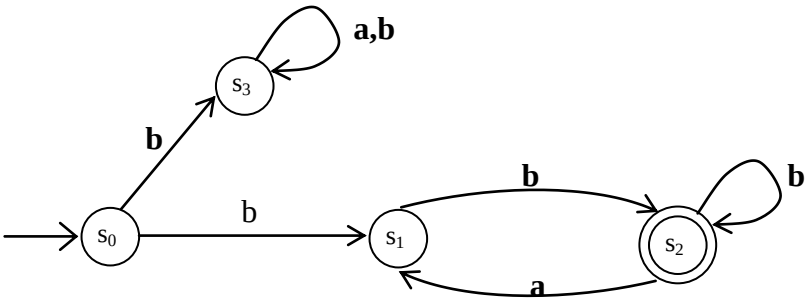
Solution: **Let s_0 be the initial state. If we define f as**

$f(s_0, a) = s_1$, $f(s_1,b) = s_2$ (accepting state)

Then, we have three states as



Now we cannot take $f(s_0, b) = s_0$ or $f(s_0, b) = s_1$, because then b will be starting letter. So, we have to take $f(s_0, b) = s_3$. We cannot take $f(s_3,a) = s_0$, s_1, s_2 because in that case the string would end in a. We cannot have $f(s_2, a) = s_2$, because then a will be last letter. Thus The automaton M will be



In this automaton, any string shall begin with a and end in b.

4.4. Non – Deterministic Finite State Automaton

Definition: A non – deterministic finite – state automaton **is a 5 – tuple M = (I, S, A, s_0, f) consisting of**
(1) A finite set I of input symbols
(2) A finite set S of states
(3) A subset A of S of accepting states
(4) An initial state function $s_0 \in S$
(5) A next state function f from $S \times I$ into P(S)

Thus, in a non – deterministic finite state automaton, the next state function leads us to a set of states, whereas in a finite state automaton, the next state function takes us to a uniquely defined state.

Example: Find the transition diagram for N D F S A

$$M = (I, S, A, s_0, f),$$

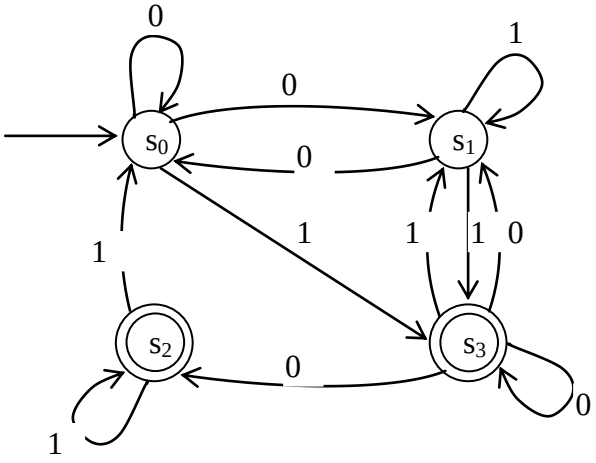
where

$$I = \{0, 1\}, S = \{s_0, s_1, s_2, s_3\}, A = \{s_2, s_3\}$$

and the next state function f is given by

	f	
S \ I	0	1
s_0	$\{s_0, s_1\}$	$\{s_3\}$
s_1	$\{s_0\}$	$\{s_1, s_3\}$
s_2	ϕ	$\{s_0, s_2\}$
s_3	$\{s_1, s_2, s_3\}$	$\{s_1\}$

Solution: **Here the initial state is s_0 and the accepting states are s_2 and s_3. The transitional diagram of this N D F S A is**



Definition: Let M = (I, S, A, s_0, f) be a non – deterministic finite state automaton. The null string is **accepted** by M if and only if $s_0 \in A$. **If $w = a_1 a_2 \ldots a_n$ is a non – null string over I and there exists states $s_0, s_1, \ldots, s_n$ such that**

(1) s_0 is the initial state

(2) $s_i = f(s_{i-1}, a_i)$

(3) $s_n \in A$,

then we say that w is accepted by M.

We denote by AC(M), the set of strings accepted by M and say that M accept AC(M).

Definition: Two non – deterministic finite state automata M and M′ are said to be **equivalent** if

$$AC(M) = AC(M') .$$

Example: Let

$$M = (I, S, A, s_0, f)$$

be a N D F S A with

$$I = \{0, 1\}, S = \{s_0, s_1, s_2, s_3, s_4\}, A = \{s_2, s_4\},$$

so as the initial state and the next state function defined by the transition table given below:

	f	
I / S	0	1
s_0	$\{s_0, s_3\}$	$\{s_0, s_1\}$
s_1	φ	$\{s_2\}$
s_2	$\{s_2\}$	$\{s_2\}$
s_3	$\{s_4\}$	ϕ
s_4	$\{s_4\}$	$\{s_4\}$

Determine whether M accept the words (i) w = 010 and (ii) w = 01001.

Solution: (i) The word w = 010 determines the path $s_0 \xrightarrow{0} \{s_0, s_3\} \xrightarrow{1} f(s_0, 1) \cup f(s_3, 1) = \{s_0, s_1\}$

$\cup \varphi = \{s_0, s_1\} \xrightarrow{0} f(s_0, 0) \cup f(s_1, 0) = \{s_0, s_3\} \cup \varphi = \{s_0, s_3\}$

But $A \cap \{s_0, s_3\} = \{s_2, s_4\} \cap \{s_0, s_3\} = \phi$. Hence the word w = 010 is not acceptable to the given non – deterministic finite state automaton.

(ii) We have seen above that

$$s_0 \xrightarrow{0} \{s_0, s_3\} \xrightarrow{1} \{s_0, s_1\} \xrightarrow{0} \{s_0, s_3\}$$

Therefore the word w = 01001 determines the path

$$s_0 \xrightarrow{0} \{s_0, s_3\} \xrightarrow{1} \{s_0, s_1\} \xrightarrow{0} \{s_0, s_3\} \xrightarrow{0} f(s_0, 0) \cup f(s_3, 0)$$

$$= \{s_0, s_3\} \cup \{s_n\}$$

$$= \{s_0, s_3, s_4\} \xrightarrow{1} f(s_0, 1) \cup f(s_3, 1\} \cup f(s_4, 1\}$$

$$= \{s_0, s_1\} \cup \varphi \cup \{s_4\}$$

$$= \{s_0, s_1, s_4\}$$

so that

$$A \cap \{s_0, s_1, s_4\} = \{s_2, s_4\} \cap \{s_0, s_1, s_4\} = \{s_4\} \neq \varphi.$$

Hence the string 01001 is acceptable to the given N D F S A.

4.5 The Equivalence of D F S A and N D F S A

We have seen that in the definition of finite state automaton, the next state function is from $S \times I$ into S, whereas in the definition of N D F S A, the next state function is from $S \times I$ into P(S). Thus, **every D F S A is an N D F S A,** that is, the class of languages accepted by N D F S A includes the languages accepted by D F S A. However, these are the only languages accepted by N D F S A. In other words, **for every N D F S A, we can construct an equivalent D F S A.** In this direction, we have the following :

Theorem: Let L be a set accepted by a non – deterministic finite automaton. Then there exists a deterministic finite automaton that accepts L.

Proof: Let $M = (I, S, A, s_0, f)$ be an N D F S A accepting L. Define a D F S A,

$$M' = (I, S', A', s_0', f')$$

as follows:

The states of M′ are all the subsets of the set of all states of M , that is, $S' = 2^S$. Also $s_0' = \{s_0\}$ and A′ is the set of all states in S′ containing a final state of M, that is, $A' = \{s \in S' : s \cap A \neq \phi\}$.

Further, for $s \in S'$ and $a \in I$, let

$$f'(s, a) = \bigcup_{\sigma \in s} f(\sigma, a)$$

To prove that M′ accept the same language as M, it is sufficient to show that for any string $x \in I^*$ (the set of strings formed by I),

$$f'^*(s_0', x) = f^*(s_0, x) \qquad (1)$$

We shall prove (1) by using induction on the length of the input string x.

If $x = \wedge$, then

$$f'^*(s_0, x) = f'^*(s_0', \wedge)$$

$$= s_0' \text{ (by definition of } f'^*)$$

$$= \{s_0\} \text{ by the definition of } s_0'$$

$$= f^*(s_0, \wedge) \text{ (by the definition of } f^*)$$

$$= f^*(s_0, x)$$

Thus (1) holds for $| x | = 0$ (i.e. for $x = \wedge$).

The induction hypothesis is that x is a string satisfying

$$f'^*(s_0', x) = f^*(s_0, x)$$

and we want to show that

$$f'^*(s_0', x\ a) = f^*(s_0, x\ a) \text{ for } a \in I.$$

To show it, we have

$$f'^*(s_0', x\ a) = f'\ (f'^*\ (s_0', x), a) \text{ (by the definition of } f'^*)$$

$$= f'(f^*(s_0, x), a) \text{ (by induction hypothesis)}$$

$$= \bigcup_{\sigma \in f^*(s_0, x)} f(\sigma, a) \text{ (by the definition of } f')$$

$$= f^*(s_0, x\ a) \text{ (by the definition of } f^*)$$

We know that a string x is accepted by M′ if $f'^*(s_0', x) \in A'$ that is, if $f^*\ (s_0, x) \in A'$ and using the definition of A′, it follows that this is true if and only if

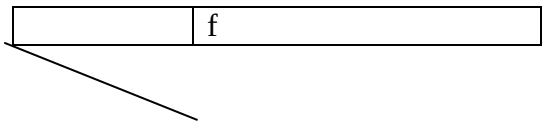
$$f^*(s_0, x) \cap A \neq \varphi,$$

that is, if $f^*(s_0, x) \in A$, that is, if x is accepted by M. Thus x is accepted by M′ if and only if x is accepted by M. This completes the proof of the theorem.

Example: Construct deterministic finite state automaton equivalent to the following non – deterministic finite state automaton :

$$M = (\{0, 1\}, \{s_0, s_1\}, s_0, \{s_1\}, f) ,$$

where f is given by the table



I S	0	1
s_0	$\{s_0, s_1\}$	$\{s_1\}$
s_1	φ	$\{s_0, s_1\}$

Solution: Let

$$M' = \{\{0, 1\}, \{\varphi, \{s_0\}, \{s_1\}, \{s_0, s_1\}, s_0' = \{s_0\}, A', f'\}$$

be the D F S A, where

$$A' = \{s \in \{\varphi, (s_0, \{s_1\}, \{s_0, s_1\} : s \cap \{s_1\} \neq \varphi$$

and $= \{s_1\}$ and $\{s_0, s_1\}$ **(Accepting states)**

$$f'(s, a) = \bigcup_{\sigma \in s} f(\sigma, a) \text{ for } s \in \{\varphi, \{s_0\}, \{s_1\}, \{s_0, s_1\}\}$$

We have

$\{s_0\}$ as the initial state

The finite set of states is $\{\varphi, \{s_0\}, \{s_1\}, \{s_0, s_1\}\}$

The finite set of inputs is {0, 1}

The accepting states are $[s_1]$ and $[s_0, s_1]$.

Now

$$f'(\varphi, 0) = \phi \text{ and } f'(\varphi, 1) = \varphi$$

$$f'([s_0], 0) = f(s_0, 0) = [s_0, s_1]$$

$$f'([s_0], 1) = f(s_0, 1) = [s_1]$$

$$f'([s_1], 0) = f(s_1, 0) = \phi$$

$$f'([s_1], 1) = f(s_1, 1) = [s_0, s_1]$$

$$f'([s_0, s_1], 0) = f(s_0, 0) \cup f\{s_1, 0\}$$

$$= \{s_0, s_1\} \cup \{s_1\}$$

$$= [s_0, s_1]$$

$$f'(\{s_0, s_1\}, 1) = f(s_0, 1) \cup f(s_1, 1)$$

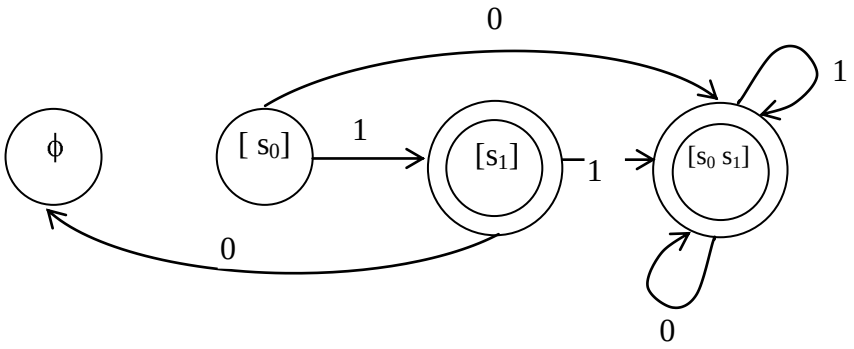
$$= \{s_1\} \cup \{s_0, s_1\}$$

$$= [s_0, s_1]$$

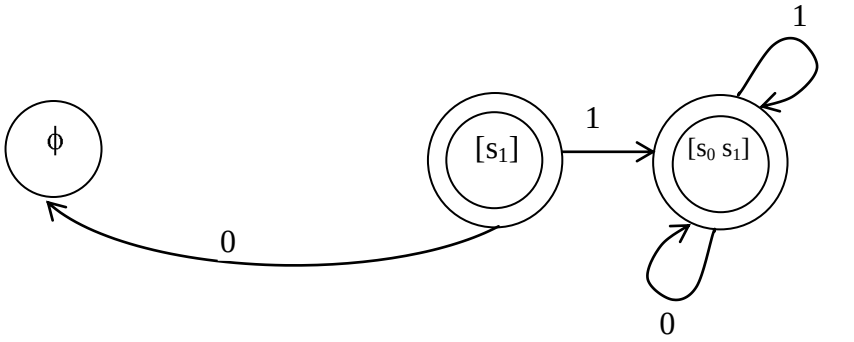
Hence the next state function and the transition diagram for D F S A are as given below :

	f ′

I / S	0	1
φ	φ	φ
$[s_0]$	$[s_0, s_1]$	$[s_1]$
$[s_1]$	ϕ	$[s_0, s_1]$
$[s_0, s_1]$	$[s_0, s_1]$	$[s_0, s_1]$



It may be mentioned here that **a state which is never entered may be deleted from the transition diagram**. In view of this, the above transition diagram becomes



Thus, we note that **if N D F S A has n states, then D F S A will have 2^n states.**

Example: Draw transition diagram of the N D F S A

$$M' = (\{a, b\}, \{s_0, s_1, s_2\}, \{s_0\}, s_0, f),$$

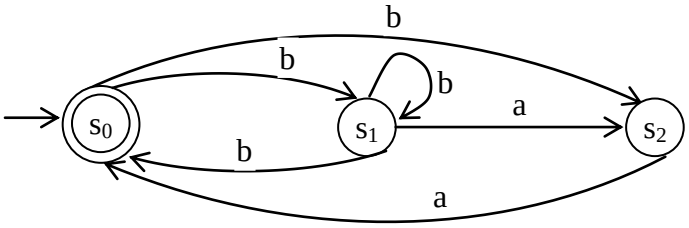
where f is given by

	f	
I / S	a	b
s_0	φ	$\{s_1,s_2\}$
s_1	$\{s_2\}$	$\{s_0,s_1\}$
s_2	$\{s_0\}$	ϕ

Also find equivalent D F S A.

Solution: Here

Initial stage is s_0
Set of Accepting state is $\{s_0\}$
Finite set of states is $\{s_0, s_1, s_2\}$
Finite set of inputs is {a, b}
Hence the transition diagram is



Let

$M' = (\{0, 1\}, \{\varphi, \{s_0\}, \{s_1\}, \{s_2\}, \{s_0, s_1\},$

$\{s_0, s_2\}, \{s_1, s_2\}\ \{s_0, s_1, s_2\}\}\ s_0', A', f')$

be the equivalent D F S A, where

$$s_0' = [s_0]$$

and set of accepting states is

$A' = \{s \in \{\phi, \{s_0\},...., \{s_0, s_1, s_2\} : s \cap \{s_0\} \neq \varphi\}$

$= \{s_0\}, \{s_0, s_1\}, \{s_0, s_2\}, \{s_0, s_1, s_2\}$

Further

$f'(\varphi , a) = \varphi , f'(\varphi , b) = \varphi$

$f'([s_0] , a) = f(s_0 , a) = \varphi \ , \quad f'([s_0] ,b) = f(s_0 , b) = [s_1\ s_2]$

$f'([s_1], a) = f(s_1, a) = [s_2]$, $f'([s_1], b) = f(s_1, b) = [s_0\ s_1]$

$f'([s_2], a) = f(s_2, a) = [s_0]$, $f'([s_2], b) = f(s_2, b) = \varphi$

$f'([s_0\ s_1], a) = f(s_0, a) \cup f(s_1, a) = [s_2]$, $f'([s_0\ s_1], b) = [s_0\ s_1\ s_2]$

$f'([s_0\ s_2], a) = f(s_0, a) \cup f(s_2, a) = [s_0]$,

$f'([s_0\ s_2], b) = f(s_0, b) \cup f(s_2, b) = [s_1\ s_2]$

$f'([s_1\ s_2], a) = f(s_1, a) \cup f(s_2, a) = [s_0\ s_2]$,

$f'([s_1\ s_2], b) = f(s_1, b) \cup f(s_2, b) = [s_0\ s_1]$

$f'([s_0\ s_1\ s_2], a) = f(s_0, a) \cup f(s_1, a) \cup f(s_2, a) = [s_0\ s_2]$,

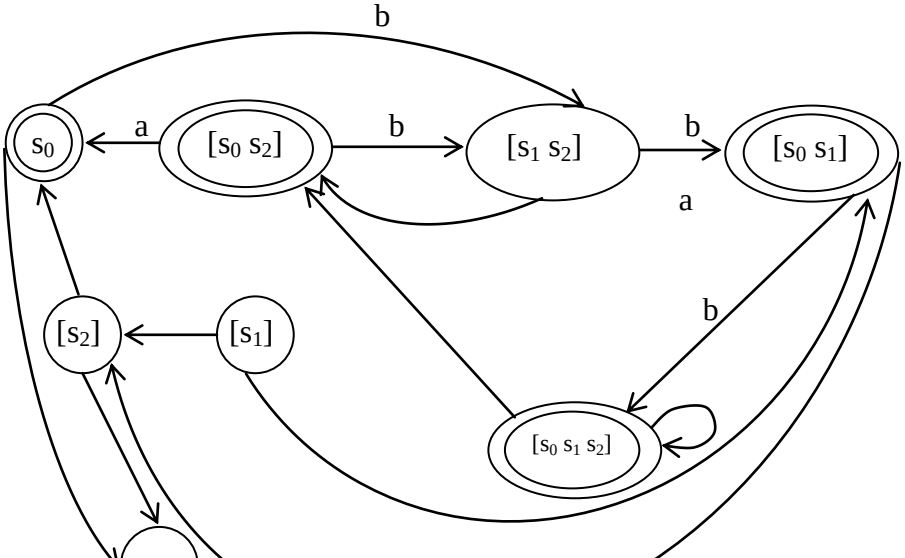
$f'([s_0\ s_1\ s_2], b) = f(s_0, b) \cup f(s_1, b) \cup f(s_2, b)$

$= [s_0\ s_1\ s_2]$

Thus, the transition table of D F S A is

	F	
I / S	a	b
φ	ϕ	φ
$[s_0]$	φ	$[s_1\ s_2]$
$[s_1]$	$[s_2]$	$[s_0\ s_1]$
$[s_2]$	$[s_0]$	φ
$[s_0\ s_1]$	$[s_2]$	$[s_0\ s_1\ s_2]$
$[s_0\ s_2]$	$[s_0]$	$[s_1\ s_2]$
$[s_1\ s_2]$	$[s_0, s_2]$	$[s_0, s_1]$
$[s_0\ s_1\ s_2]$	$[s_0\ s_2]$	$[s_0\ s_1\ s_2]$

The transition diagram of deterministic finite state automaton is therefore as shown in the diagram below:



a a

b

b

a

b

a

a b

Since the state [s_1] is never entered, it may be removed .

4.6 Moore Machine and Mealy Machine

We have seen that in case of finite automaton, the output is limited to a binary signal "accept" or "don′t accept". However, some models in which the output is chosen from some other alphabet have also been considered. There are two different approaches.

(1) If the output function depends only on the present state and is independent of the current input, the model is called a **Moore Machine.**

(2) If the output function is a function of transition, i.e. a function of present state and the present input, the model is called a **Mealy Machine.**

MOORE MACHINE

A **Moore machine** is a six – tuple

$$(I, S, O, s_0, f, g),$$

where

(1) I is a finite set of input symbols

(2) S is a finite set of internal states

(3) O is a finite set of output symbols

(4) s_0 is the initial state

(5) f is the transition (next – state) function from $S \times I$ into S

(6) g is the output function mapping S into O.

The output in response to input $a_1\ a_2 \ldots\ldots\ldots a_n$, $n \geq 0$ is $g(s_0)\ g(s_1) \ldots\ldots g(s_n)$, where $s_0, s_1, \ldots\ldots\ldots s_n$ is the sequence of states such that

$$f(s_{i-1}, a_i) = s_i \quad , 1 \leq i \leq n.$$

Moore Machine gives output $g(s_0)$ in response to input $\in$ (empty string).

Obviously, D F S A is a special case of a Moore Machine, **where the output alphabet is {0, 1} and the state s is "accepting" if and only if g(s) = 1.**

Example: Let

$$M = (I, S, O, s_0, f, g)$$

be a Moore Machine, where

$$I = \{0, 1\}, S = \{s_0, s_1, s_2, s_3\}$$

$$O = \{0, 1\}, s_0 \text{ is initial state ,}$$

f is transition function such that

$$f(s_0, 0) = s_3, \quad f(s_0, 1) = s_1$$

$$f(s_1, 0) = s_1, \quad f(s_1, 1) = s_2$$

$$f(s_2, 0) = s_2, \quad f(s_2, 1) = s_3$$

$$f(s_3, 0) = s_3, \quad f(s_3, 1) = s_0 \ ,$$

and g is the output function such that

$$g(s_0) = 0, \quad g(s_1) = 1, \quad g(s_2) = 0, \quad g(s_3) = 0$$

Determine the transition table for M and the output string for the input string 0111.

Solution: The transition table for this Moore machine is

	f		G
S \ I	0	1	
s_0	s_3	s_1	0
s_1	s_1	s_2	1
s_2	s_2	s_3	0
s_3			0
		s_3	
		s_0	

The input string is 0111. We note that
For empty string $\in$, the output is $g(s_0) = 0$

$$f(s_0, 0) = s_3 \text{ and } g(s_3) = 0$$

$$f(s_3, 1) = s_0 \text{ and } g(s_0) = 0$$

$$f(s_0, 1) = s_1 \text{ and } g(s_1) = 1$$

$f(s_1, 1) = s_2$ and $g(s_2) = 0$

Thus the output string is

0 0 0 1 0.

MEALY MACHINE

A **Mealy Machine** M is a six – tuple

$$(I, S, O, s_0, f, g),$$

where

(1) I is a finite set of input symbols

(2) S is a finite set of internal states

(3) O is a finite set of output symbols

(4) s_0 is the initial state

(5) f is the transition (next – state) function from $S \times I$ into S

(6) g is the output function mapping $S \times I$ into O.

The output given by M in response to input $a_1\ a_2 \ldots a_n$ is $g(s_0, a_1)\ g(s_1, a_2)\ g(s_2, a_3) \ldots\ldots g(s_{n-1}, a_n)$, where $s_0, s_1, \ldots .s_n$ is the sequence of states such that $g(s_{i-1}, a_i) = s_i$, $1 \leq i \leq n$.

Note that the output sequence in case of Mealy Machine has length n, whereas the length of output sequence in case of Moore Machine is n + 1. Further, Mealy Machine gives output $\in$ for the input string $\in$.

4.7 Equivalence of Moore and Mealy Machines

We know that the output string length in case of Mealy machine is one less than the output string length in case of Moore machine.

Neglecting the response of a Moore machine to input $\in$, we say that Moore Machine M and Mealy machine M′ are **equivalent** if for all input string v

$$k\ \Delta_{M'}(v) = \Delta_M(v),$$

where $\Delta_{M'}(v)$ and $\Delta_M(v)$ are output produced by M′ and M on input v and k is output of M for its initial state.

Theorem: Let $M_1 = (I, S, O, s_0, f, g)$ be a Moore machine. Then there is a Mealy machine $M_2 = (I, S, O, s_0, f, g')$ which is equivalent to M_1.

Proof: Define $g' : S \times I \rightarrow O$ by

$$g'(s, a) = g(f(s, a)\ ,\ \text{for all } s \in S \text{ and } a \in I.$$

Then M_1 and M_2 enter the same sequence of states on the same input and with each transition M_2 emits the output that M_1 associates with the state entered.

Example: Let the transition table of a Moore machine $M_1 = (\{0, 1\}, \{s_0, s_1, s_2, s_3\}, \{0, 1\}, s_0, f, g)$ be as given below:

	f		g
I	0	1	
→ s_0	s_3	s_1	0
s_1	s_1	s_2	1
s_2	s_2	s_3	0
s_3			0
		s_3	
		s_0	

Construct a Mealy machine M_2 equivalent to M.

Solution: Let

$$M_2 = (\{0, 1\}, \{s_0, s_1, s_2, s_3\}, \{0, 1\}, f, g', s_0)$$

be the equivalent Mealy machine, where

$$g'(s, a) = g(f(s, a)) , s \in S, a \in I.$$

Thus

$$g'(s_0, 0) = g(f(s_0, 0) = g(s_3) = 0$$

$$g'(s_0, 1) = g(f(s_0, 1) = g(s_1) = 1$$

$$g'(s_1, 0) = g(f(s_1, 0) = g(s_1) = 1$$

$$g'(s_1, 1) = g(f(s_1, 1) = g(s_2) = 0$$

$$g'(s_2, 0) = g(f(s_2, 0) = g(s_2) = 0$$

$$g'(s_2, 1) = g(f(s_2, 1) = g(s_3) = 0$$

$$g'(s_3, 0) = g(f(s_3, 0) = g(s_3) = 0$$

$$g'(s_3, 1) = g(f(s_3, 1) = g(s_0) = 0$$

Thus the transition table for Mealy machine is

	f		g′	
I	0	1	0	1
→ s_0	s_3	s_1	0	1
s_1	s_1	s_2	1	0
s_2	s_2	s_3	0	0
s_3			0	0
		s_3		
		s_0		

Theorem: Let $M_1 = (I, S, O, s_0, f, g)$ be a Mealy machine. Then there is a Moore machine $M_2 = (I, S', O, s_0', f', g')$ which is equivalent to M_1.

Proof: Let b_0 be arbitrary member of finite set O of output symbols. Set

$$M_2 = (I, S \times O, O, [s_0, b_0], f', g')$$

Thus the states of M_2 consists of pairs [q, b], where $q \in S$, $b \in O$.
Define f′ by

$$f'([q, b], a) = [f(q, a), g(q, a)]$$

and g′ by

$$g'([q, b]) = b.$$

The component b in a state [q, b] is the output made by M_1 on some transition into state q. Only the first component of M_2's states determine the moves made by the machine M_2. Induction on n shows that M_1 enters states $q_0, q_1, \ldots, q_n$ on input $a_1\ a_2 \ldots\ldots a_n$ and emits outputs $b_1, b_2 \ldots, b_n$, then M_2 enters states $[q_0, b_0]$, $[q_1, b_1], \ldots\ldots, [q_n, b_n]$ and emits outputs $b_0, b_1, b_2 \ldots.., b_n$.

Example: Let M_1 be a Mealy machine whose transition table is

	f		g	
I / S	0	1	0	1
s_0	s_3	s_1	0	1
s_1	s_1	s_2	1	0
s_2	s_2	s_3	0	0
s_3			0	0
		s_3		

	s_0	

Find equivalent Moore Machine M_2.

Solution: The states M_2 are

$[s_0, 0], [s_1, 0], [s_1, 0], [s_1, 1], [s_2, 0], [s_2, 1], [s_3, 0], [s_3, 1]$.

We select $b_0 = 0$ making $[s_0, 0]$ as start state for M_2.

The transitions and outputs of M_2 are as follows:

$f'([s_0, 0], 0) = [f(s_0, 0), g(s_0, 0)] = [s_3, 0]; g'([s_0, 0]) = 0$

$f'([s_0, 0], 1) = [f(s_0, 1), g(s_0, 1)] = [s_1, 1]; g'([s_0, 0]) = 0$

$f'([s_0, 1], 0) = [f(s_0, 0), g(s_0, 0)] = [s_3, 0]; g'([s_0, 1]) = 1$

$f'([s_0, 1], 1) = [f(s_0, 1), g(s_0, 1)] = [s_1, 1]; g'([s_0, 1]) = 1$

$f'([s_1, 0], 0) = [f(s_1, 0), g(s_1, 0)] = [s_1, 0]; g'([s_1, 0]) = 0$

$f'([s_1, 0], 1) = [f(s_1, 1), g(s_1, 1)] = [s_2, 0]; g'([s_1, 0]) = 0$

$f'([s_1, 1], 0) = [f(s_1, 0), g(s_1, 0)] = [s_1, 0]; g'([s_1, 1]) = 1$

$f'([s_1, 1], 1) = [f(s_1, 1), g(s_1, 1)] = [s_2, 0]; g'([s_1, 1]) = 1$

$f'([s_2, 0], 0) = [f(s_2, 0), g(s_2, 0)] = [s_2, 0]; g'([s_2, 0]) = 0$

$f'([s_2, 0], 1) = [f(s_2, 1), g(s_2, 1)] = [s_3, 0]; g'([s_2, 0]) = 0$

$f'([s_2, 1], 0) = [f(s_2, 0), g(s_2, 0)] = [s_2, 0]; g'([s_2, 1]) = 1$

$f'([s_2, 1], 1) = [f(s_2, 1), g(s_2, 1)] = [s_3, 0]; g'([s_2, 1]) = 1$

$f'([s_3, 0], 0) = [f(s_3, 0), g(s_3, 0)] = [s_3, 0]; g'([s_3, 0]) = 0$

$f'([s_3, 0], 1) = [f(s_3, 1), g(s_3, 1)] = [s_0, 0]; g'([s_3, 0]) = 0$

$f'([s_3, 1], 0) = [f(s_3, 0), g(s_3, 0)] = [s_3, 0]; g'([s_3, 1]) = 1$

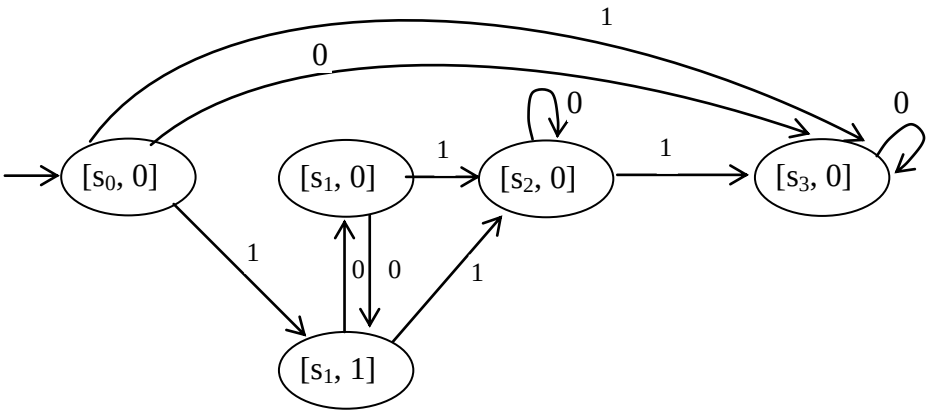
$f'([s_3, 1], 1) = [f(s_3, 1), g(s_3, 1)] = [s_0, 0]; g'([s_3, 1]) = 1$

Thus the transition table and transition diagram of Moore machine M_2 which is equivalent to given Mealy machine M_1 are :

	f'		g'
I	0	1	
$\rightarrow [s_0, 0]$		**[**	0
***** $[s_0, 1]$		**s_3, 0]**	1 *****
$[s_1, 0]$		**[s_1, 1]**	0

$[s_1, 1]$	$[s_3, 0]$	$[s_1, 1]$	1
$[s_2, 0]$	$[s_1, 0]$	$[s_2, 0]$	0
* $[s_2, 1]$	$[s_1, 0]$	$[s_2, 0]$	1 *
$[s_3, 0]$	$[s_2, 0]$	$[s_3, 0]$	0
* $[s_3, 1]$	$[s_2, 0]$	$[s_3, 0]$	1 *
	$[s_3, 0]$	$[s_0, 0]$	
	$[s_3, 0]$	$[s_0, 0]$	

and



The states $[s_0, 1]$, $[s_2, 1]$, $[s_3, 1]$ can never be entered and so have been removed from the diagram.

Leaving aside the outputs corresponding to the removed states which have been marked by * in the transition table, the outputs are 0, 0, 1, 0, 0 .

Unit-5

Languages and Grammars

Formal languages are used to model nature languages and to communicate with computers. Before giving definition to formal language, we define some elementary notions.

5.1 Basic Concepts

Definition: Let A be a non – empty set of symbols. Then a finite sequence of the elements of A is called a **word** or **string** w on the set A. For example,

$$w = a\ b\ b\ a\ a\ a\ b$$

is a string on A = {a, b}.

The set A is called **alphabet** and its elements are called **letters.** The empty sequence of letters is also considered as a string and is denoted by $\in$, λ or 1. This is called **empty word.**

The set of all words on the set A is denoted by A*.

The length of the string (word) w is the number of elements in the string and is denoted by l(w) or |w|. The length of $\in$ is 0.

For example, thus the length of the word w cited above is 7.

Definition: Let u and v be two strings on alphabet A. The **concatenation** of u and v is the word obtained by writing down the letters of u followed by the letters of v. It is denoted by uv.

For example, if u = a b c a b and v = c c a b b a, then concatenation of u and v is

$$u\ v = a\ b\ c\ a\ b\ c\ c\ a\ b\ b\ a = a^4\ b^4\ c^3$$

We observe that

$$l(u\ v) = l(u) + l(v).$$

Also, we note that for any words u, v, w, we have

$$(u\ v)\ w = u(v\ w)$$

Thus, the **concatenation operation** on an alphabet A is **associative,** but not **commutative** because $u\ v \neq v\ u$.

Definition: Let $u = x_1\ x_2 \ldots\ldots\ldots x_n$ be a word on an alphabet A. Then any sequence $v = x_i\ x_{i+1} \ldots\ldots\ldots x_j$ is called a **subword** of u.

The subword which begins with the first letter of u is called an **initial segment** of u.

For example, x_1, x_2, x_3 is an initial segment of u.

Let F denote the set of all non – empty words from an alphabet A with the operation of concatenation. We know that F is a semi – group, called **Free semigroup** over A or the free semigroup generated by A.

Further, since $\in$ is an identity element for the operation of concatenation, A* becomes a monoid and is called **Free monoid** over A.

5.2. Language, Regular Expressions and Language Defined Regular Expressions

Definition: Let A be a finite set of symbols. A (formal) **language** L over A is a subset of A*, the set of all string over A.

For example, let A = {a, b}. Then the set L of all strings over A containing an odd number of a's is a language over A.

Similarly, $\{a, ab, ab^2, \ldots\ldots\}$ is a language over A. This consists of all words beginning with a and followed by zero or more b's.

Let L_1 and L_2 be languages over an alphabet A. Then the concatenation of L_1 and L_2, denoted by L_1L_2 , is the language defined by

$$L_1\ L_2 = \{u\ v : u \in L_1, v \in L_2\}$$

Thus $L_1\ L_2$ is the set of all words formed by the concatenation of a word from L_1 with a word from L_2. For example, let

$$L_1 = \{\ a, b^3\},\ L_2 = \{a^3, a\ b^2, b\}$$

Then

$$L_1\ L_2 = \{a^4, a^2\ b^2, a\ b, b^3\ a^3, b^3\ a\ b^2, b^4\}$$

is a language.

Since concatenation of words is associative, it follows that concatenation of languages is associative.

Definition: The **Power of a language L** are defined as

$$L^0 = \{ \in \}, L^1 = L, L^2 = LL, \ldots\ldots, L^{m+1} = L^m L, m > 1.$$

Definition: The unary operation L* of a language L, called the **Kleene closure of L** is defined as the infinite union

$$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots\ldots = \bigcup_{k=1}^{\infty} L^k$$

If we leave apart $L^0 = \{ \in \}$, then we write

$$L^+ = L^1 \cup L^2 \cup \ldots\ldots\ldots = \bigcup_{k=1}^{\infty} L^k .$$

Definition: The regular expressions over an alphabet A and the sets they denote are defined recursively as follows:

(1) The empty string $\in$ is a regular expression and denotes the set $\{\in\}$.

(2) ϕ or ( ) is a regular expression and denote the empty set.

(3) Each letter a in A is a regular expression and denotes the set {a}.

(4) If r is a regular expression denoting the language R, then (r*) is a regular expression on and denotes the set R*.

(5) If r and s are regular expression denoting the language R and S , then $(r \vee s)$ or (r + s) is a regular expression and denotes the set $R \cup S$.

(6) If r and s are regular expressions denoting the languages (sets) R and S, then (r s) is a regular expression and denotes the set R S.

Thus, a regular expression r is a special kind of a string (word) which uses the letters of A and the five symbols

$$(\quad) , \quad * , \quad \bullet , \quad \vee , \quad \in (\text{or } \wedge)$$

For example,

(i) the regular expression (0 + 1)* denotes all the strings of 0's and 1's.

(ii) the regular expressions (1 + 10)* denotes all the strings of 0's and 1's and beginning with 1 and **not** having two consecutive 0's.
(iii) The regular expression (0 + 1)* 00 (0 + 1)* denotes all the strings of 0's and 1's with at least two consecutive 0's.

(iv) (0 + 1)* 0 1 1 denotes all strings of 0's and 1's ending in 0 1 1

(v) 0* 1* 2* denotes all the strings with any number of 0's followed by any number of 1's followed by any number of 2's.

Definition: The **language L(r) over A** defined by a regular expression r over A is as follows:

(1) $L(\in) = \{\in\}$

(2) $L(( )) = \varphi$, the empty set

(3) L(a) = {a}, where a is a letter in A.

(4) $L(r^*) = (L(r))^*$, the Kleene closure of L(r).

(5) $L(r_1 + r_2) = L(r_1) \cup L(r_2)$, the union of languages

(6) $L(r_1 r_2) = L(r_1) L(r_2)$, the concatenation of the languages.

Definition: Let L be a language over A. Then L is said to be a **regular language** over A if there exists a regular expression r over A such that L = L(r).

Example: Let A = {a, b}. If

(i) r = a*, then L(r) consists of all powers of a
(ii) r = a a*, then L(r) consists of all positive powers of a, that is all words in a excluding the empty word.
(iii) r = a + b*, then L(r) consists of a or any word in b, that is

$$L(r) = \{ a, \in, b , b^2 \ldots..\}$$

(iv) r = (a + b)*, then L(r) consists of all strings of a and b, i.e. all words (strings) over A.
(v) r = (a + b)* a a, then L(r) denoted all strings of a and b ending in a a, i.e. L(s) consists of the concatenation of any word in A with a a (or a^2).

Example: Let $L = \{a^m b^n : m, n > 0\}$ be a language over A = {a, b}. Find a regular expression r such that

$$L = L(r)$$

Solution: The given language L consists of strings beginning with one or more a's followed by one or more b's. Hence

$$R = a\, a^*\, b\, b^*$$

5.3 Language Determined by a Finite – State Automaton

Let M be a finite state automaton with input set A. Then M defines a language over A, denoted by L(M), as follows:

Let $u = a_1\, a_2 \ldots\ldots\ldots a_n$ be a string on A. Then u determines a sequence of states

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots\ldots\ldots \rightarrow s_n$$

where s_0 is the initial state and

$$f(s_{i-1}, a_i) = s_i \text{ for } i \geq 1.$$

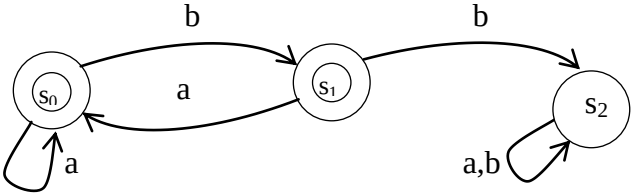
In other words, u determines the path

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2, \ldots\ldots\ldots\ldots\ldots, \xrightarrow{a_n} s_n$$

A finite state machine M is said to **accept** (recognize) the word u if the final state s_n belong to an accepting state in A (subset of internal states S).

The **language L(M) of the finite state automaton M is the collection of all words from the input set A which are accepted by M.**

Example: Determine the language L(M) of the finite state automaton whose transition diagram is given below



Solution: Let M = (I, S, A, s_0, f) be the finite – state automaton. Then, we note that s_0 is the initial state, S = {s_0, s_1, s_2} and

$$f(s_0, a) = a,\ f(s, b) = s_1$$

$$f(s_1, a) = s_0,\ f(s_1, b) = s_2$$

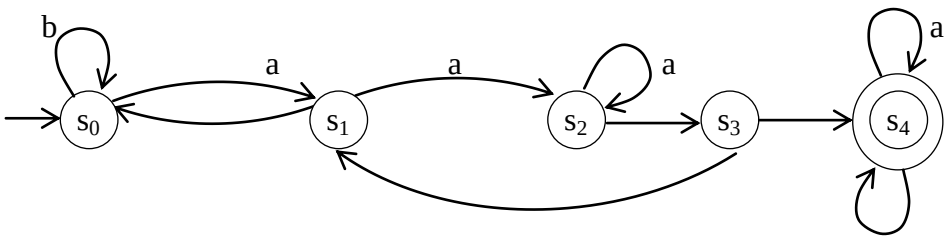
$$f(s_2, a) = s_2,\ f(s_2, b) = s_2.$$

Also A = {s_0, s_1}and I = {a, b}.
We note that

(i) We can never leave s_2
(ii) The state s_2 is the only rejecting (non – accepting) state
(iii) a string in which there appear two successive b's is not accepted by M.

Thus L(M) consists of all strings (words) from I = {a, b} which do not have two successive b's.

Example: Find the language accepted by the automaton M shown in the transition diagram below:



b b

b

a b

Solution: Let M = (I, S, A, s_0, f) be the F S A. Then, we have

I = {a, b}, S = {s_0, s_1, s_2, s_3, s_4}, s_0 is the initial state,

A = {s_4}

and f is given by

$f(s_0, a) = s_1$, $f(s_0, b) = s_0$, $f(s_1, a) = s_2$, $f(s_2, a) = s_2$, $f(s_2, b) = b$,

$f(s_3, b) = s_4$, $f(s_3, a) = s_1$, $f(s_4, b) = s_4$, $f(s_4, a) = s_4$.

We note that

$s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_3 \xrightarrow{b} s_4$ (accepting state)

$s_0 \xrightarrow{b} s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_3 \xrightarrow{b} s_4$ (accepting state)

$s_0 \xrightarrow{b} s_0 \xrightarrow{a} s_1 \xrightarrow{a} s_2 \xrightarrow{b} s_3 \xrightarrow{b} s_4 \xrightarrow{a} s_4 \xrightarrow{b} s_4$ (accepting state)

Hence, L(M) consists of all words which contain a a b b as a subword.

5.4. Grammars

Definition: A **phrase – structure Grammar** or simply a **Grammar** G consists of

(1) A finite set N of **non – terminal symbols (or variables)**
(2) A finite set T of **terminal symbols,** where $N \cap T = \phi$
(3) A finite subset P of $[(N \cup T)^* - T^*] \times (N \cup T)^*$, called **the set of productions.** Thus a production is an ordered pair (A, B), written as $A \to B$, where $A \in [(N \cup T)^* - T^*]$ must include at least one non – terminal symbol whereas $B \in (N \cup T)^*$ can consist of any combination of non – terminals and terminal symbols.

(4) A **starting symbol** $\sigma \in N$.

A grammar G is denoted by G(N, T, P, σ).

Terminals will be denoted by lower case letters a, b, c,… whereas non – terminals will be denoted by A, B, C…….

Example: Let

$$N = \{\sigma, A\}$$

$$T = \{a, b\}$$

$$P = \{\sigma \to b\,\sigma, \sigma \to b\,A, A \to a\,A, A \to b\}\ ,$$

where σ is the starting symbol.

Then $G = (N, T, P, \sigma)$ is a grammar.

Since $\sigma \to b\,\sigma, \sigma \to b\,A$

and

$$A \to a\,A, A \to b\ ,$$

we can also write the productions as

$$\sigma \to (b\,\sigma, b\,A), A \to (a\,A, b)\ .$$

Definition: Let $G = (N, T, P, \sigma)$ be a grammar and let $\alpha \to \beta$ be a production. If $x\,\alpha\,y \in (N \cup T)^*$, then $x\,\beta\,y$ is said to be **directly derivable** from $x\,\alpha\,y$ and we write

$$x\,\alpha\,y \Rightarrow x\,\beta\,y.$$

Further, if $\alpha_i \in (N \cup T)^*$ for i = 1, 2, ……, n, and α_{i+1} is directly derivable from α_i for i = 1, 2, ……, n-1, we say that α_n **is derivable from** α_1 and write

$$\alpha_1 \Rightarrow \alpha_n\ .$$

We call

$$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots\ldots\ldots \Rightarrow \alpha_n\ ,$$

the **derivation of** α_n (from α_1).

By convention, any element of $(N \cup T)^*$ is derivable from itself.

Definition: The language generated by a grammar G, written L(G), consists of all strings over T derivable from the start symbol σ. Thus

$$\mathbf{L(G) = \{\, v \in T^* : \sigma \Rightarrow \ldots\ldots\ldots \Rightarrow v\}}$$

Definition: A **sentential form** is any derivative of the unique non – terminal symbol S.

The language L(G) generated by the grammar G is the set of all sentential forms whose symbols are terminals.

Example: Let

$$G = \{N, T, \sigma, P\}$$

be a grammar, where

$N = \{\sigma\}$, $T = \{a, b\}$, σ is starting symbol, and the production P are

$$P = \{\sigma \to a\ ,\ \sigma \to \sigma\ a\ ,\ \sigma \to b \text{ and } \sigma \to b\ \sigma\}.$$

Obtain sentential form and find the language generated by G.

Solution: We note that

$$\begin{aligned}\sigma &\Rightarrow \sigma\ a\\ &\Rightarrow \sigma\ a\ a\\ &\Rightarrow b\ \sigma\ a\ a\\ &\Rightarrow b\ b\ \sigma\ a\ a\\ &\Rightarrow b\ b\ b\ a\ a\end{aligned}$$

Thus $b^3\ a^2 = b\ b\ b\ a\ a$ is a sentential form. Hence the language generated by G is

$$L(G) = \{b^n\ a^m : n \geq 0, m \geq 0\}.$$

Definition: Let $G = (N, T, P, \sigma)$ be a grammar and let λ be the null string. If every production is of the form

$$\alpha\ A\ \beta \to \alpha\ \delta\ \beta,$$

where $\alpha, \beta \in (N \cup T)^*$, $A \in N$, $\delta \in (N \cup T)^* - \{\lambda\}$. Then G is called a **context – sensitive (or type – 1) grammar.**

Definition: A grammar $G = (N, T, P, \sigma)$ is said to be **context – free** (or **type – 2**) grammar if the productions are of the form.

$$A \to \delta,$$

where $A \in N$, $\delta \in (N \cup T)^*$.

Thus, in this case, we can replace A by δ regardless of A where A appears.

Definition: A grammar $G = (N, T, P, \sigma)$ is said to be **regular** (or **type – 3**) grammar if every production is of the form

$$A \to a \text{ or } A \to a\,B \text{ or } A \to \lambda\,,$$

where $A, B \in N$, $a \in T$.

Thus, in this case, we replace a non – terminal symbol by a terminal symbol, by a terminal symbol followed by a non – terminal symbol, or by the null string.

We further note that a regular grammar is context free grammar and that a context free grammar with no productions of the form $A \to \lambda$ is a context – sensitive grammar.

Example: Name the type of the grammar G defined by T = {a, b, c}, N = {σ, A, B, C, D, E}, starting symbol σ and productions

$$\sigma \to a\,A\,B,\ \sigma \to a\,B,\ A \to a\,A\,c,\ A \to a\,c,\ B \to D\,c,\ D \to b,$$
$$C\,D \to C\,E,\ C\,E \to D\,E,\ D\,E \to D\,C,\ C\,c \to D\,c\,c.$$

Also find its language.

Solution: The production $C\,E \to D\,E$ says that we can replace C by D if C is followed by E. The production $C\,c \to D\,c\,c$ says that we can replace C by D C if C is followed by c.

Thus the grammar is context – sensitive.

We can derive D C from C D since

$$C\,D \Rightarrow C\,E \Rightarrow D\,E \Rightarrow D\,C.$$

We note that

$$\sigma \Rightarrow a\,A\,B \Rightarrow a\,a\,A\,c\,B \Rightarrow a\,a\,a\,c\,c\,B \Rightarrow a\,a\,a\,C\,C\,D\,c$$

$$\Rightarrow a\,a\,a\,C\,D\,C\,c \Rightarrow a\,a\,a\,D\,C\,C\,c \Rightarrow a\,a\,a\,D\,C\,D\,c\,c$$

$$\Rightarrow a\,a\,a\,D\,D\,C\,c\,c \Rightarrow a\,a\,a\,D\,D\,D\,c\,c\,c \Rightarrow a\,a\,a\,b\,b\,b\,c\,c\,c$$

Thus $a^3\,b^3\,c^3$ is in L(G). Proceeding in this way, we can show that

$$L(G) = \{a^n\,b^n\,c^n : n \in \mathbf{N}\}.$$

Example: Determine, whether the given grammar is context – sensitive, context free, regular or none of these:

$$G = (N, T, \sigma, P),$$

Where N = {σ, A}, T = {a, b}, starting symbol is σ and the productions are

$\sigma \to b\,\sigma, \sigma \to \sigma\, A, A \to a\,\sigma, A \to b\, A, A \to a, \sigma \to b.$

Solution: We note that

(i) $A \to \delta$, $A \in N$, $\delta \in (N \cup T)^*$

Hence the grammar is context – free grammar.

(ii) $A \to a$ or $A \to a\, B$, $A \in N$, $B \in N$, $a \in T$.

Hence the grammar is regular.

(iii) The grammar is also context sensitive because

$$\alpha\, A\, \beta \to \alpha\, \delta\, \beta,$$

where $\alpha, \beta \in (N \cup T)^*$, $A \in N$, $\delta \in (N \cup T)^* - \{\lambda\}$.

Example : Find a context-free grammar G which generates the language

$$L = \{\, a^n\, b^n : \; n > 0\} \,.$$

Solution : Here

$$T = \{a, b\} \,.$$

If we consider the productions

$$\sigma \to ab\,, \quad \sigma \to a\sigma\, b\,,$$

then we note that

$$\sigma \Rightarrow a\,\sigma\, b \Rightarrow a\, ab\, b$$

$$\sigma \Rightarrow a\,\sigma\, b \;\; a\, a\,\sigma\, b\, b \Rightarrow a\, a\, a\, b\, b\, b$$

$$\sigma \Rightarrow a\,\sigma\, b \Rightarrow a\, a\,\sigma\, b\, b \Rightarrow a\, a\, a\,\sigma\, b\, b\, b \Rightarrow a\, a\, a\, a\, b\, b\, b\, b$$

…….

In general

$$L(G) = \{a^n\, b^n\,, \; n > 0\} \,.$$

Hence the grammar with production

$$\sigma \to ab, \;\; \sigma \to a\,\sigma\, b$$

generates L(G).

5.5 Derivation Trees of Context – Free Grammars

Let G be a context free grammar. An ordered rooted tree which represents any derivation of a word in L(G) is called a **Derivation Tree** or **parse tree.**

Example: Consider the language

$$L = \{a^n b^n : n > 0\}$$

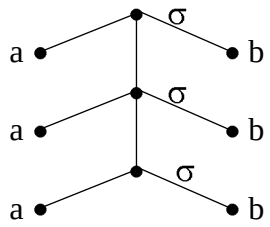
We have seen that context free grammar which generates L(G) is

$$N = \{\sigma\},\ T = \{a, b\},\ P = \{\sigma \to a\ \sigma\ b,\ \sigma \to a\ b\}$$

The word w = a a a b b b is derived as

$$\sigma \Rightarrow a\ \sigma\ b \Rightarrow a\ a\ \sigma\ b\ b \Rightarrow a\ a\ a\ b\ b\ b$$

The following figure will therefore be its derivation tree:



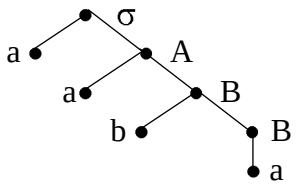
Example: Find the derivation tree for the word a a b a in L(G) where G has the productions

$$\sigma \to a\ A,\ A \to a\ B,\ B \to b\ B,\ B \to a.$$

Solution: The word a a b a is derived as

$$\sigma \Rightarrow a\ A \Rightarrow a(a\ B) \Rightarrow a\ a\ (b\ B) \Rightarrow a\ a\ b\ a$$

and therefore the derivation tree of a a b a is



Definition: A language is said to be **context – sensitive** if there is a context – sensitive grammar G with L = L(G).
Definition: A language is said to be **context free** if there is a context – free grammar G with L = L(G).

Definition: A language is said to be **regular** if there is a regular grammar G with L = L(G).

Example: Is the language

$$L = \{a^n b^n, n = 1, 2, \ldots..\}$$

over {a, b} context free?

Solution: Let G be grammar defined by

N = {σ}, T = {a, b}, σ is staring symbol and production as

$$\sigma \to a\,\sigma\,b\,,\ \sigma \to a\,b$$

Then derivation of σ are

$$\sigma \Rightarrow a\,\sigma\,b$$
$$\Rightarrow a\,a\,\sigma\,b\,b$$

$$\Rightarrow a^{n-1}\,\sigma\,b^{n-1}$$
$$\Rightarrow a^{n-1}\,a\,b\,b^{n-1} = a^n\,b^n.$$

Thus the grammar G generates the language L(G). Also the grammar G is context free. Hence the language L = [a^n b^n, n = 1, 2, …..] is context free language.

5.6 Similarity of Regular Grammar and Finite State Automata

We now show that **regular grammar and finite state automata are essentially the same**. After that we would be able to say that

"**A language is a regular set (**or just **regular) if it is accepted by some finite automaton."**

Theorem: Let M be a finite – state automaton given as a transition diagram. Let σ be the initial state. Let T be the set of input symbols and let N be the set of states. Let P be the set of productions

$$s \to x\,s'$$

if there is an edge labeled x from the state s to the state s′, and

$$s \to \lambda$$

if s is an accepting state. Let

$$G = (N, T, P, \sigma)$$

be the regular grammar. Then the set of strings accepted by M is equal to L(G).

Proof: Let AC(M) denote the set of strings by M. We first show that AC(M) $\subseteq$ L(G). So, let $\alpha \in$ AC(M). If α is the null string, then σ is an accepting state. In this case G contains the production.

$$\sigma \to \lambda$$

The derivation

$$\sigma \Rightarrow \lambda \qquad \text{(i)}$$

shows that $\alpha \in L(G)$.

Now let $\alpha \in$ AC(M) and let α is not a null string. Then

$$\alpha = a_1\, a_2 \ldots\ldots\ldots a_n,\ a_i \in T.$$

Since α is accepted by M, there is a path

$$(\sigma, s_1, s_2, \ldots\ldots\ldots, s_n)\ ,$$

where s_n is an accepting state with edges successively labeled $a_1, \ldots\ldots, a_n$. It follows that G contains the productions

$$\sigma \to a_1\, s_1$$
$$s_1 \to a_2\, s_2$$
$$\text{----------}$$
$$\text{----------}$$
$$s_{n-1} \to a_n\, s_n$$

Since s_n is an accepting state, G also contains the production

$$s_n \to \lambda.$$

The derivation

$$\begin{aligned}
\sigma &\Rightarrow a_1\, s_1 \\
&\Rightarrow a_1\, a_2\, s_2 \\
&\Rightarrow a_1\, a_2\, a_3\, s_3 \\
&\text{----------} \\
&\text{----------} \\
&\Rightarrow a_1\, a_2 \ldots\ldots a_n\, s_n \\
&\Rightarrow a_1\, a_2 \ldots\ldots a_n \qquad (\because s_n \to \lambda) \qquad \text{(ii)}
\end{aligned}$$

shows that $\alpha = a_1\, a_2 \ldots\ldots\ldots a_n \in L(G)$.

It remains to show that $L(G) \subseteq$ Ac(M). Suppose that $\alpha \in L(G)$. If α is the null string, then α must result from the derivation

$$\sigma \Rightarrow \lambda$$

Thus the production $\sigma \to \lambda$ is in the grammar. Hence σ is an accepting state in M and so $\alpha \in$ Ac(M).

Now let $\alpha \in$ L(G) be a non – null string. Then

$$\alpha = a_1\, a_2 \ldots\ldots\ldots a_n\ ,\ \ a_i \in T.$$

So there is a derivation of the form (ii). If in the transition diagram, we begin at σ and trace the path

$$(\sigma, s_1, s_2, \ldots\ldots, s_n)\ ,$$

we can generate the string α. The last production used in (ii) is $s_n \to \lambda$. Thus the last state reached is an accepting state. Therefore, α is accepted by M, that is, L(G) $\subseteq$ A c(M). Hence

$$L(G) = A\ c(M)\ .$$

Thus, Given a finite state automaton M, we can construct a regular grammar G such that the set of strings accepted by M is equal to L(G).

Example: Let G(T, N, P, σ) be a regular grammar, where

$$T = \{a, b\},\ N = \{\sigma, A\},\ \sigma \text{ is starting symbol and}$$

$$P = \{\sigma \to b\,\sigma\ ,\ \sigma \to a\ A\ ,\ A \to b\ A\ ,\ A \to b\}.$$

Does there exist finite state automaton corresponding to G?

Solution: Let the inputs symbol be the terminal symbols and the states be the non – terminal symbols, where σ is the initial state.

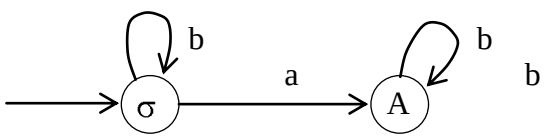
For each production of the form

$$s \to x\ s',$$

draw an edge from state s to state s′ and label it x. Thus the productions

$$\sigma \to b\,\sigma\ ,\ \sigma \to a\ A\ ,\ A \to b\ A$$

yield the graph



The last production A $\to$ b is equivalent to two productions

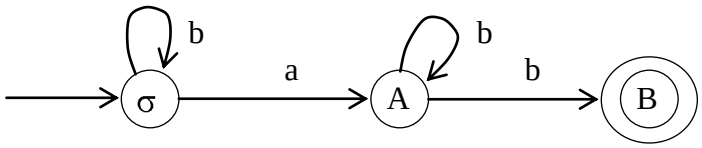
$$A \to bB \text{ and } B \to \lambda \ ,$$

where B is an additional and Mor – terminal symbol.

The productions

$$\sigma \to b\,\sigma \ , \ \sigma \to a\,A \ , \ A \to bA \ , \ A \to bB$$

gives us the graph



,

and the production

$$B \to \lambda$$

indicates that B is an accepting state.

We note that

(i) Vertex A has no outgoing edge labeled as a
(ii) Vertex B has no outgoing edge
(iii) A has two outgoing edges labeled as b .

Thus, the above graph is not finite – state automation but a **non – deterministic finite state automation (I, S, A, σ, f),** where I = {a, b}, S = {σ, A, B}, A = {B}, initial state σ and next state function f is defined by

	F	
S \ I	a	b
σ	{A}	{σ}
A	ϕ	{A, B}
B	φ	ϕ

We further notice that

(i) the string b b a b b is in L(G) since

$$b \Rightarrow b\,\sigma$$
$$\Rightarrow b\,b\,\sigma$$
$$\Rightarrow b\,b\,a\,A$$

$$\Rightarrow b\ b\ a\ b\ A$$

$$\Rightarrow b\ b\ a\ b\ b\ B$$

$$\Rightarrow b\ b\ a\ b\ b$$

Also the string b b a b b is accepted by the non-deterministic finite state automation obtained above since the path

$$\sigma \xrightarrow{b} \sigma \xrightarrow{b} \sigma \xrightarrow{a} A \xrightarrow{b} A \xrightarrow{b} B$$

which ends at state B(Accepting state) represents the string b b a b b

Theorem: Let G(T, N, P, σ) be a regular grammar and let I = T , S = N $\cup$ {F}, where F $\notin$ N $\cup$ T, σ as initial state, A = {F} $\cup$ {s : s $\rightarrow \lambda \in$ P} and f be defined by

$$F(s, x) = \{s' : s \rightarrow x\ s' \in P\} \cup \{F : s \rightarrow x\ F \in P\}.$$

Then the non – deterministic finite state automaton M = (I, S, A, σ, f) accept the strings L(G).

(The proof is same as the proof for finite state automation).

5.7 Kleene Theorem and Pumping Lemma

We know that a non – deterministic finite state automaton can be converted into an equivalent finite state automaton.

Thus it follows that

Theorem (Kleene) : A language L is regular if and only if there exists a finite – state automaton that accept strings in L.

Theorem (Pumping Lemma) : Let M be an automaton over A such that

(i) M has k states $s_0, s_1 \ldots\ldots, s_k$

(ii) M accepts a word v from A where | v | > k.

Then

$$w = x\ y\ z,$$

where, for every m, $v_m = x\ y^m z$ is accepted by M.

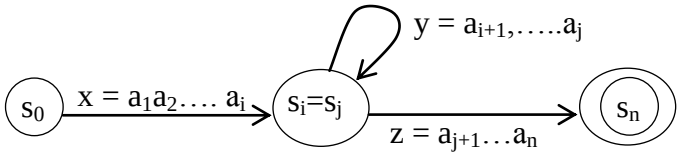
Proof: Let $s_0, s_1, \ldots\ldots, s_k$ be the states of automaton M over A and let M accepts a word $v = a_1\ a_2 \ldots a_n$ over A such that n > k. Let the sequence of states determined by the word v be

$$P = (s_0, s_1, \ldots\ldots, s_n) .$$

Since n > k, two of the states in P must be equal. Suppose $s_i = s_j$, i < j. Letting

$$x = a_1\, a_2 \ldots .a_i,\ y = a_{i+1}\, a_{i+2} \ldots\ldots a_j\ ,\ z = a_{j+1}\, a_{j+2} \ldots . a_n$$

We see that x y ends in $s_i = s_j$ and so x y^2, x y^3,…., xy^m (for all m) also end in s_i. Thus for every m, v_m = x y^m z ends in s_n, which is an accepting state



Example: Show that language L = {a^m b^m : m is positive} is not regular.

Solution: Suppose on the contrary that L is regular. Then, by Kleene Theorem, there exists a finite state automaton M which accept L = {a^m b^m : m is positive}. Suppose M has k states. Let v = a^k b^k be a word. Then length of v is greater than k, the number of states in M. Therefore, by Pumping Lemma,

$$v = x\, y\, z\ ,\ y \neq \lambda.$$

and x y^2 z is also accepted by M.

If y consists of only a's or only b's, then v_2 = x y^2 z will not have same number of a's or b's. If y consists of both a's and b's, then v_2 will have a's following b's. In either case v_2 does not belong to L which is a contradiction. Thus L is not regular.

5.8 Ambiguous Grammar

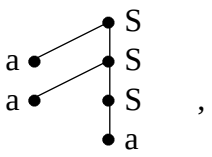
Definition: A context – free grammar G is called a **ambiguous grammar** if there is at least one string in L(G) which has more than one derivation trees.

Example: Show that grammar G with productions

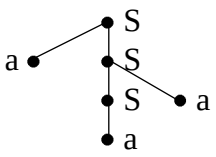
$$S \rightarrow a\ S\ ,\ S \rightarrow S\ a\ ,\ S \rightarrow a$$

is ambiguous.

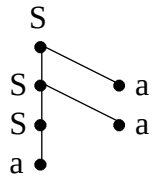
Solution: We note that the string a a a can be generated by four derivation trees



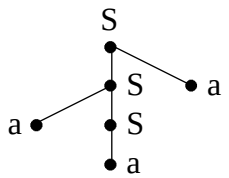
,



$(S \Rightarrow a\ S \Rightarrow a\ a\ S \Rightarrow a\ a\ a)$ $(S \Rightarrow a\ S \Rightarrow a\ S\ a \Rightarrow a\ a\ a)$



and



$(S \Rightarrow S\ a \Rightarrow S\ S\ a \Rightarrow a\ a\ a)$ $(S \Rightarrow S\ a \Rightarrow a\ S\ a \Rightarrow a\ a\ a)$

Hence G is ambiguous.

Example: Show that the grammar

$$G = (\{S\}, \{a, +\}, S, P)$$

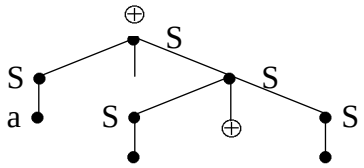
with production

$$P = (S \rightarrow S + S, S \rightarrow a)$$

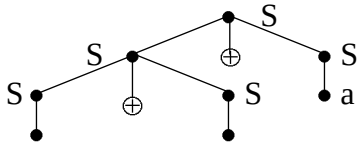
is ambiguous.

Solution: We note that word a + a + a can be generated in two ways :

(i) $S \Rightarrow S + S$
$\Rightarrow S + S + S$
$\Rightarrow a + S + S \Rightarrow a + a + S$
$\Rightarrow a + a + a$



(ii) $S \Rightarrow S + S$
$\Rightarrow S + S + S$
$\Rightarrow S + S + a$
$\Rightarrow a + S + a$
$\Rightarrow a + a + a$



Thus, the word a + a + a has two derivation tree. Hence G is ambiguous.