MCA-20101 DISCRETE MATHEMATICAL STRUCTURES

Instruction: 4 Periods/week **Time: 3 Hours** **Credits: 4**

Internal: 25 Marks **External: 75**

Marks **Total:100Marks**

UNIT I

Introduction: Logic-Prepositional Equivalences-Truth tables -Tautologies-Predicates and Quantifiers-Sets-Operations on sets-Sequences and Summations -Growth functions - relations and their properties- binary relations and their applications -Representation of relations- Closures of relations-Equivalence relations -Partial Orderings.

UNIT II

Counting Techniques: Basics of Counting- Pigeonhole Principle- Combinations and Permutations-Generalized Permutations and Combinations

Recurrence relations: Solving Recurrence Relations-Divide and Conquer relations -Inclusion and Exclusion-Applications of Inclusion-Exclusion.

UNIT III

Graphs: Introduction to Graphs-Terminology-Relations and Directed Graphs Representations of Graphs- Isomorphism-Connectivity- Euler and Hamiltonian Paths- Shortest Path problems- Planar Graphs- Graph Coloring.

Trees: Introduction to trees- Applications of trees- Traversals-Trees and sorting Spanning Trees-Minimum Spanning Trees.

UNIT IV

Boolean Algebra and Models of Computation: Boolean Functions- Representing Boolean Functions -Logic Gates-Minimizations of Circuits-Languages and Grammars- Finite State Machines with and with no output.

Text Book:

1. Discrete Mathematics and its applications, Keneth. H. Rosen, Tata McGraw-Hill Publishing Company, New Delhi

Reference Books:

1. Discrete Mathematics for Computer Scientists & Mathematicians, Joe L.Mott, Abraham Kandel& T. P. Baker, Prentice Hall of India Ltd, New Delhi
2. Discrete mathematics, Richard Johnsonbaug, Pearson Education, New Delhi

UNIT-I

Introduction: A proposition or statement is a declarative sentence that is either true or false (but not both). For instance, the following are propositions: “Paris is in France” (true), “London is in Denmark” (false), “ 2 < 4” (true), “4 = 7 (false)”. How ever the following are not propositions: “what is your name?” (this is a question), “do your homework” (this is a command), “this sentence is false” (neither true nor false), “x is an even number” (it depends on what x represents),

“Socrates” (it is not even a sentence). The truth or falsehood of a proposition is called its truth value.

Connectives:

Connectives are used for making compound propositions. The main ones are the following (p and q represent given propositions):

Name	Represented	Meaning
Negation	¬p	“not p”
Conjunction	$p \wedge q$	“p and q”
Disjunction	$p \vee q$	“p or q(or both)”
Exclusive Or	$p \oplus q$	“either p or q, not but both”
Implication	p → q	“if p then q”
Biconditional	p ↔ q	“p if and only if q”

Truth Tables:

Logical identity

Logical identity is an operation on one logical value, typically the value of a proposition that produces a value of true if its operand is true and a value of false if its operand is false.

The truth table for the logical identity operator is as follows:

Logical Identity	
p	***p***
T	T
F	F

Logical negation

Logical negation is an operation on one logical value, typically the value of a proposition that produces a value of true if its operand is false and a value of false if its operand is true.

The truth table for **NOT p** (also written as **¬p** or **~p**) is as follows:

Logical Negation	
p	***¬p***
T	F
F	T

Logical conjunction

Logical conjunction is an operation on two logical values, typically the values of two propositions, that produces a value of *true* if both of its operands are true.

The truth table for p AND q (also written as pAq, p&q, or pq) is as follows:

Logical Conjunction		
P	***q***	***p*A*q***
T	T	T
T	F	F
F	T	F
F	F	F

In ordinary language terms, if both p and q are true, then the conjunction p A q is true. For all other assignments of logical values to p and to q, the conjunction p A q is false.

Logical disjunction

Logical disjunction is an operation on two logical values, typically the values of two propositions, that produces a value of *true* if at least one of its operands is true.

The truth table for p OR q (also written as p∨q, p||q, or p +q) isas follows:

Logical Disjunction		
p	***q***	***p∨q***
T	T	T
T	F	T
F	T	T
F	F	F

Logical implication

Logical implication and the material conditional are both associated with an operation on two logical values, typically the values of two propositions, that produces a value of *false* just in the singular case the first operand is true and the second operand is false. The truth table associated with the material conditional **if p then q** (symbolized as **p** → **q**) and the logical implication **p implies q** (symbolized as **p** ⇒ **q**) is as follows:

Logical Implication		
P	***q***	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Logical equality

Logical equality (also known as biconditional) is an operation on two logical values, typically the values of two propositions, that produces a value of *true* if both operands are false or both operands are true. The truth table for **p XNOR q** (also written as **p** ↔ **q** ,**p = q**, or **p ≡ q**) is as follows:

Logical Equality		
p	q	$p \equiv q$
T	T	T
T	F	F
F	T	F
F	F	T

Exclusive disjunction

Exclusive disjunction is an operation on two logical values, typically the values of two propositions, that produces a value of true if one, but not both, of its operands is true.

The truth table for **p XOR q** (also written as **p** $\oplus$ **q**, or **p** $\neq$ **q**) is as follows:

Exclusive Disjunction		
P	q	$p \oplus q$
T	T	F
T	F	T
F	T	T
F	F	F

Logical NAND

The logical NAND is an operation on two logical values, typically the values of two propositions, that produces a value of *false* if both of its operands are true. In other words, it produces a value of *true* if at least one of its operands is false. The truth table for **p NAND q** (also written as **p** ↑ **q** or **p | q**) is as follows:

Logical NAND		
p	***q***	***p*** ↑ ***q***
T	T	F
T	F	T
F	T	T
F	F	T

It is frequently useful to express a logical operation as a compound operation, that is, as an operation that is built up or composed from other operations. Many such compositions are possible, depending on the operations that are taken as basic or "primitive" and the operations that are taken as composite or "derivative". In the case of logical NAND, it is clearly expressible as a compound of NOT and AND. The negation of a conjunction: ¬(p A q), and the disjunction of negations: (¬p) ∨ (¬q) can be tabulated as follows:

p	*q*	*p*A*q*	¬(*p*A*q*)	¬*p*	¬*q*	*p*)∨(**q**)
T	T	T	F	F	F	F
T	F	F	T	F	T	T
F	T	F	T	T	F	T
F	F	F	T	T	T	T

Logical NOR

The logical NOR is an operation on two logical values, typically the values of two propositions, that produces a value of *true* if both of its operands are false. In other words, it produces a value of *false* if at least one of its operands is true. ↓ is also known as the Peirce arrow after its inventor, Charles Sanders Peirce, and is a Sole sufficient operator.
The truth table for **p NOR q** (also written as **p**↓**q** or **p**⊥**q**) is as follows:

Logical NOR		
p	***q***	***p*** ↓ ***q***
T	T	F
T	F	F
F	T	F
F	F	T

The negation of a disjunction ¬(*p*∨*q*), and the conjunction of negations(¬*p*)A(¬*q*) can be tabulated as follows:

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

Inspection of the tabular derivations for NAND and NOR, under each assignment of logical values to the functional arguments p and q, produces the identical patterns of functional values for $\neg(p \wedge q)$ as for $(\neg p) \vee (\neg q)$, and for $\neg(p \vee q)$ as for $(\neg p) \wedge (\neg q)$. Thus the first and second expressions in each pair are logically equivalent, and may be substituted for each other in all contexts that pertain solely to their logical values.

This equivalence is one of DeMorgan's laws.

The truth value of a compound proposition depends only on the value of its components.

Writing F for “false” and T for “true”, we can summarize the meaning of the connectives in the following way:

p	q	$\neg$p	$p \wedge q$	$p \vee q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	F	T	T	F	T	T
T	F	F	F	T	T	F	F
F	T	T	F	T	T	T	F
F	F	T	F	F	F	T	T

Note that $\vee$ represents a non-exclusive or, i.e., $p \vee q$ is true when any of p, q is true and also when both are true. On the other hand $\oplus$ represents an exclusive or, i.e., $p \oplus q$ is true only when exactly one of p and q is true.

Tautology, Contradiction, Contingency:

A proposition is said to be a tautology if its truth value is T for any assignment of truth values to its components. Example: The proposition p ∨ ¬p is a tautology.

A proposition is said to be a contradiction if its truth value is F for any assignment of truth values to its components. Example: The proposition p ∧ ¬p is a contradiction.

A proposition that is neither a tautology nor a contradiction is called a contingency.

p	¬p	p∨¬p	p∧¬p
T	F	T	F
T	F	T	F
F	T	T	F
F	T	T	F

Equivalence Implication:

We say that the statements *r* and *s* are logically equivalent if their truth tables are identical. For example the truth table of $\neg p \vee q$

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

shows that $\neg p \vee q$ is equivalent to $p \rightarrow q$. It is easily shown that the statements *r* and *s* are equivalent if and only if $r \leftrightarrow s$ is a tautology.

Normal forms:

Let A (P1, P2, P3, …, Pn) be a statement formula where P1, P2, P3, …, Pn are the atomic variables. If A has truth value T for all possible assignments of the truth values to the variables P1, P2, P3, …, Pn, then A is said to be a tautology. If A has truth value F, then A is said to be identically false or a contradiction.

Disjunctive Normal Forms

A product of the variables and their negations in a formula is called an elementary product. A sum of the variables and their negations is called an elementary sum. That is, a sum of elementary products is called a disjunctive normal form of the given formula.
Example:

$$A \tag{1}$$

$$(A \wedge B) \vee (! A \wedge C) \tag{2}$$

$$(A \wedge B \wedge ! A) \vee (C \wedge ! B) \vee (A \wedge ! C) \tag{3}$$

$$A \wedge B \tag{4}$$

$$A \vee (B \wedge C), \tag{5}$$

Conjunctive Normal Forms

A formula which is equivalent to a given formula and which consists of a product of elementary sums is called a conjunctive normal form of the given formula.

Example:

$$A \tag{1}$$

$$(A \vee B) \wedge (! A \vee C) \tag{2}$$

$$A \vee B \tag{3}$$

$$A \wedge (B \vee C), \tag{4}$$

Predicates

Predicative logic:

A predicate or propositional function is a statement containing variables. For instance “x + 2 = 7”, “X is American”, “x < y”, “p is a prime number” are predicates. The truth value of the predicate depends on the value assigned to its variables. For instance if we replace x with 1in the predicate “x + 2 = 7” we obtain “1 + 2 = 7”,which is false, but if we replace it with 5 we get “5
+ 2 = 7”, which is true. We represent a predicate by a letter followed by the variables enclosed between parenthesis: P (x), Q (x, y), etc. An example for P (x) is a value of x for which P (x) is true. A counterexample is a value of x for which P (x) is false. So, 5 is an example for “x + 2 = 7”, while 1 is a counterexample. Each variable in a predicate is assumed to belong to a universe

(or domain) of discourse, for instance in the predicate “n is an odd integer” ’n’ represents an integer, so the universe of discourse of n is the set of all integers. In “X is American” we may assume that X is a human being, so in this case the universe of discourse is these to fall human beings.

Consistency of premises:

Consistency

A set of formulas H1,H2,…,Hm is said to be consistent if the irconjunction has the truth value T for some assignment of the truth values to be atomic appearing in H1, H2, …, Hm.

Inconsistency

If for every assignment of the truth values to the atomic variables, atleast one of the formulas H1, H2, ... Hm is false, so that their conjunction is identically false, then the formulas
H1,H2, …, Hm are called inconsistent.

Types of Relations

1. Reflexive Relation: A relation R on set A is said to be a reflexive if **(a, a)** ∈ **R** for every **a** ∈ A.

Example: If A = {1, 2, 3, 4} then R = {(1, 1) (2, 2), (1, 3), (2, 4), (3, 3), (3, 4), (4, 4)}. Is a relation reflexive?

Solution: The relation is reflexive as for every a ∈ A. (a, a) ∈ R, i.e. (1, 1), (2, 2), (3, 3), (4, 4) ∈ R.

2. Irreflexive Relation: A relation R on set A is said to be **irreflexive if (a, a)** ∉ **R** for every **a** ∈ **A**.

Example: Let A = {1, 2, 3} and R = {(1, 2), (2, 2), (3, 1), (1, 3)}. Is the relation R reflexive or irreflexive?

Solution: The relation R is not reflexive as for every a ∈ A, (a, a) ∉ R, i.e., (1, 1) and (3, 3) ∉ R. The relation R is not irreflexive as (a, a) ∉ R, for some a ∈ A, i.e., (2, 2) ∈ R.

3. Symmetric Relation: A relation R on set A is said to be symmetric iff (a, b) ∈ R ⇔ (b, a) ∈ R.

Example: Let A = {1, 2, 3} and R = {(1, 1), (2, 2), (1, 2), (2, 1), (2, 3), (3, 2)}. Is a relation R symmetric or not?

Solution: The relation is symmetric as for every (a, b) ∈ R, we have (b, a) ∈ R, i.e., (1, 2), (2, 1), (2, 3), (3, 2) ∈ R but not reflexive because (3, 3) ∉ R.

Example of Symmetric Relation:

1. Relation ⊥r is symmetric since a line a is ⊥r to b, then b is ⊥r to a.
2. Also, Parallel is symmetric, since if a line a is ∥ to b then b is also ∥ to a.

Antisymmetric Relation: A relation R on a set A is antisymmetric iff (a, b) ∈ R and (b, a) ∈ R then a = b.

Example1: Let A = {1, 2, 3} and R = {(1, 1), (2, 2)}. Is the relation R antisymmetric?

Solution: The relation R is antisymmetric as a = b when (a, b) and (b, a) both belong to R.

Example2: Let A = {4, 5, 6} and R = {(4, 4), (4, 5), (5, 4), (5, 6), (4, 6)}. Is the relation R antisymmetric?

Solution: The relation R is not antisymmetric as 4 ≠ 5 but (4, 5) and (5, 4) both belong to R.

5. Asymmetric Relation: A relation R on a set A is called an Asymmetric Relation if for every (a, b) ∈ R implies that (b, a) does not belong to R.

6. Transitive Relations: A Relation R on set A is said to be transitive iff (a, b) ∈ R and (b, c) ∈ R – (a, c) ∈ R.

Example1: Let A = {1, 2, 3} and R = {(1, 2), (2, 1), (1, 1), (2, 2)}. Is the relation transitive?

Solution: The relation R is transitive as for every (a, b) (b, c) belong to R, we have (a, c) $\in$ R i.e, (1, 2) (2, 1) $\in$ R $\Rightarrow$ (1, 1) $\in$ R.

7. Identity Relation: Identity relation I on set A is reflexive, transitive and symmetric. So identity relation I is an Equivalence Relation.

Example: A= {1, 2, 3} = {(1, 1), (2, 2), (3, 3)}

8. Void Relation: It is given by R: A $\rightarrow$B such that R = $\emptyset$ ($\subseteq$ A x B) is a null relation. Void Relation R = $\emptyset$ is symmetric and transitive but not reflexive.

9. Universal Relation: A relation R: A $\rightarrow$B such that R = A x B ($\subseteq$ A x B) is a universal relation. Universal Relation from A $\rightarrow$B is reflexive, symmetric and transitive. So this is an equivalence relation.

Closure Properties of Relations

Consider a given set A, and the collection of all relations on A. Let P be a property of such relations, such as being symmetric or being transitive. A relation with property P will be called a P-relation. The P-closure of an arbitrary relation R on A, indicated P (R), is a P-relation such that

1. R $\subseteq$ P (R) $\subseteq$ S

(1) Reflexive and Symmetric Closures: The next theorem tells us how to obtain the reflexive and symmetric closures of a relation easily.

Theorem: Let R be a relation on a set A. Then:

- R $\cup$ Δ_A is the reflexive closure of R
- R $\cup$ R^{-1} is the symmetric closure of R.

Example1:

1. Let A = {k, l, m}. Let R is a relation on A defined by
2. R = {(k, k), (k, l), (l, m), (m, k)}.

Find the reflexive closure of R.

Solution: R $\cup$ Δ is the smallest relation having reflexive property, Hence,

R_F = R $\cup$ Δ = {(k, k), (k, l), (l, l), (l, m), (m, m), (m, k)}.

Example2: Consider the relation R on A = {4, 5, 6, 7} defined by

1. R = {(4, 5), (5, 5), (5, 6), (6, 7), (7, 4), (7, 7)}

Find the symmetric closure of R.

Solution: The smallest relation containing R having the symmetric property is $R \cup R^{-1}$,i.e.

$R_S = R \cup R^{-1} = \{(4, 5), (5, 4), (5, 5), (5, 6), (6, 5), (6, 7), (7, 6), (7, 4), (4, 7), (7, 7)\}$.

(2) Transitive Closures: Consider a relation R on a set A. The transitive closure R of a relation R of a relation R is the smallest transitive relation containing R.

Recall that $R^2 = R \circ R$ and $R^n = R^{n-1} \circ R$. We define

$$R^* = \bigcup_{i=1}^{\infty} R^i$$

The following Theorem applies:

Theorem1: R^* is the transitive closure of R

Suppose A is a finite set with n elements.

$R^* = R \cup R^2 \cup ..... \cup R^n$

Theorem 2: Let R be a relation on a set A with n elements. Then

Transitive (R) = $R \cup R^2 \cup ..... \cup R^n$

Example1: Consider the relation R = {(1, 2), (2, 3), (3, 3)} on A = {1, 2, 3}. Then $R^2 = R \circ R$ = {(1, 3), (2, 3), (3, 3)} and $R^3 = R^2 \circ R$ = {(1, 3), (2, 3), (3, 3)} Accordingly, Transitive (R) = {(1, 2), (2, 3), (3, 3), (1, 3)}

Example2: Let A = {4, 6, 8, 10} and R = {(4, 4), (4, 10), (6, 6), (6, 8), (8, 10)} is a relation on set A. Determine transitive closure of R.

Solution: The matrix of relation R is shown in fig:

$$M_R = \begin{array}{c} \\ 4 \\ 6 \\ 8 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc} 4 & 6 & 8 & 10 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Now, find the powers of M_R as in fig:

$$M_{R^2} = \begin{array}{c} \\ 4 \\ 6 \\ 8 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc} 4 & 6 & 8 & 10 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad M_{R^3} = \begin{array}{c} \\ 4 \\ 6 \\ 8 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc} 4 & 6 & 8 & 10 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$M_{R^4} = \begin{array}{c} \\ 4 \\ 6 \\ 8 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc} 4 & 6 & 8 & 10 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Hence, the transitive closure of M_R is M_{R^*} as shown in Fig (where M_{R^*} is the ORing of a power of M_R).

$$M_{R^*} = M_R \vee M_{R^2} \vee M_{R^3} \vee M_{R^4}; \qquad M_{R^*} = \begin{array}{c} \\ 4 \\ 6 \\ 8 \\ 10 \end{array} \begin{array}{c} \begin{array}{cccc} 4 & 6 & 8 & 10 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Thus, R^* = {(4, 4), (4, 10), (6, 8), (6, 6), (6, 10), (8, 10)}.

Partial Ordering Relations:

Definition

A binary relation R in a set P is called ***partial order relation*** or *partial or dering* in P iff R is reflexive, anti symmetric, and transitive.

A partial order relation is denoted by the symbol£.,If £ is a partial ordering on P, then the ordered pair (P, £) is called a ***partially ordered set*** or a ***poset***.

- **Let R be these to f real numbers. The relation "less than or equal to" or**
  o, is a partial ordering on R.
- Let X be a set and r(X) be its power set. The relation subset,ÍonX is partial ordering.
- **Let Sn be these to f divisors of n. The relation D means "divides" on Sn,** is partial ordering on Sn .

RELATIONS

Introduction : The elements of a set may be related to one another. For example, in the set of natural numbers there is the 'less than' relation between the elements. The elements of one set may also be related to the elements another set.

Binary Relation

A binary relation between two sets A and B is a rule R which decides, for any elements, whether a is in relation R to b. If so, we write a R b. If a is not in relation R to b, then we shall write a /R b.

We can also consider a R b as the ordered pair (a, b) in which case we can define a binary relation from A to B as a subset of A X B. This subset is denoted by the relation R.

In general, any set of ordered pairs defines a binary relation.

For example, the relation of father to his child is F = {(a, b) / a is the father of b} In this relation F, the first member is the name of the father and the second is the name of the child.
The definition of relation permits any set of ordered pairs to define a relation.

For example, the set S given by S = {(1, 2), (3, a), (b, a) ,(b, Joe)}

Properties of Binary Relations:

Definition: A binary relation R in a set X is reflexive if, for every x Є X, x R x, That is (x, x) Є R, or R is reflexive in X ó (x) (x Є X ® x R x).

For example

1. The relation £ is reflexive in the set of real numbers..
2. The set inclusion is reflexive in the family of all subsets of a universal set..
3. The relation equality of set is also reflexive..
4. The relation is parallel in the set lines in a plane..
5. The relation of similarity in the set of triangles in a plane is reflexive..

UNIT-II
Counting Techniques

Basics of Counting

- **Counting**: Fundamental concept in combinatorics used to determine the number of ways to arrange or select objects.
- **Basic Principles**:
  - **Addition Principle**: If there are n1n_1n1 ways to do one thing and n2n_2n2 ways to do another, and these two things cannot both happen at the same time, there are n1+n2n_1 + n_2n1+n2 ways to do either.
  - **Multiplication Principle**: If there are n1n_1n1 ways to do one thing and n2n_2n2 ways to do another, there are n1×n2n_1 \times n_2n1×n2 ways to do both.

Basic Counting Principles

Sum Rule Principle: Assume some event E can occur in m ways and a second event F can occur in n ways, and suppose both events cannot occur simultaneously. Then E or F can occur in m + n ways.

In general, if there are n events and no two events occurs in same time then the event can occur in n_1+n_2..... n ways.

Example: If 8 male processor and 5 female processor teaching DMS then the student can choose professor in 8+5=13 ways.

Product Rule Principle: Suppose there is an event E which can occur in m ways and, independent of this event, there is a second event F which can occur in n ways. Then combinations of E and F can occur in m n ways.

The Pigeonhole Principle

If n pigeonholes are occupied by n+1 or more pigeons, then at least one pigeonhole is occupied by greater than one pigeon. Generalized pigeonhole principle is: - If n pigeonholes are occupied by kn+1 or more pigeons, where k is a positive integer, then at least one pigeonhole is occupied by k+1 or more pigeons.

Example1: Find the minimum number of students in a class to be sure that three of them are born in the same month.

Solution: Here n = 12 months are the Pigeonholes
And k + 1 = 3
K = 2

Example2: Show that at least two people must have their birthday in the same month if 13 people are assembled in a room.

Solution: We assigned each person the month of the year on which he was born. Since there are 12 months in a year.

So, according to the pigeonhole principle, there must be at least two people assigned to the same month.

Permutation and Combinations:

Permutation:

Any arrangement of a set of n objects in a given order is called Permutation of Object. Any arrangement of any r ≤ n of these objects in a given order is called an r-permutation or a permutation of n object taken r at a time.

It is denoted by P (n, r)

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

Theorem: Prove that the number of permutations of n things taken all at a time is n!.

Proof: We know that

$$n_{P_n} = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n!$$

Example: 4 x n_{p3}=n+1_{P3}

Solution: 4 x $\frac{n!}{(n-3)!} = \frac{(n+1)!}{(n+1-3)!}$

$$\frac{4 \times n!}{(n-3)!} = \frac{(n+1) \times n!}{(n-2)(n-3)!}$$

4 (n-2) = (n+1)

4n - 8 = n+1

3n = 9

n = 3.

Permutation with Restrictions:

The number of permutations of n different objects taken r at a time in which p particular objects do not occur is

$$\mathbf{n - p}\ \mathbf{P_r}$$

The number of permutations of n different objects taken r at a time in which p particular objects are present is

$$\mathbf{n - p}\ \mathbf{P_{r-p}}\ \mathbf{x}\ \mathbf{r_{P_p}}$$

Example: How many 6-digit numbers can be formed by using the digits 0, 1, 2, 3, 4, 5, 6, 7, 8 if every number is to start with '30' with no digit repeated?

Solution: All the numbers begin with '30.'So, we have to choose 4-digits from the remaining 7-digits.

∴ Total number of numbers that begins with '30' is

$7_{P4} = \frac{7!}{(7-4)!} = \frac{7\ X\ 6\ X\ 5\ X\ 4\ X\ 3!}{3!}$ =840.

Combination:

A Combination is a selection of some or all, objects from a set of given objects, where the order of the objects does not matter. The number of combinations of n objects, taken r at a time represented by n_{Cr} or C (n, r).

$$n_{C_r} = \frac{n!}{r!\,(n-r)!}$$

Proof: The number of permutations of n different things, taken r at a time is given by

$$n_{P_r} = \frac{n!}{(n-r)!}$$

As there is no matter about the order of arrangement of the objects, therefore, to every combination of r things, there are r! arrangements i.e.,

$$n_{P_r} = r!\ n_{C_r} \quad \text{or} \quad n_{C_r} = \frac{n_{P_r}}{r!} = \frac{n!}{(n-r)!r!}, n \geq r$$

Thus,

$$n_{C_r} = \frac{n!}{r!(n-r)!}$$

Example: A farmer purchased 3 cows, 2 pigs, and 4 hens from a man who has 6 cows, 5 pigs, and 8 hens. Find the number m of choices that the farmer has.

The farmer can choose the cows in C (6, 3) ways, the pigs in C (5, 2) ways, and the hens in C (8, 4) ways. Thus the number m of choices follows:

$$m = \binom{6}{3}\binom{5}{4}\binom{8}{4} = \frac{6.5.4}{3.2.1} \times \frac{5.4}{4} \times \frac{8.7.6.5}{4.3.2.1} = 20 \times 10 \times 70 = 14000$$

Recurrence relations:

A recurrence relation is an equation that recursively defines a sequence where the next term is a function of the previous terms (Expressing F_n as some combination of F_i with $i<n$).

Example – Fibonacci series – $F_n=F_{n-1}+F_{n-2}$, Tower of Hanoi – $F_n=2F_{n-1}+1$

Linear Recurrence Relations:

A linear recurrence equation of degree k or order k is a recurrence equation which is in the format $x_n=A_1x_{n-1}+A_2x_{n-1}+A_3x_{n-1}+\ldots A_kx_{n-k}$ (A_n is a constant and $A_k\neq0$) on a sequence of numbers as a first-degree polynomial.

Non-Homogeneous Recurrence Relation and Particular Solutions:

A recurrence relation is called non-homogeneous if it is in the form

$F_n=AF_{n-1}+BF_{n-2}+f(n)$ where $f(n)\neq0$

Its associated homogeneous recurrence relation is $F_n=AF_{n-1}+BF_{n-2}$

The solution (a_n) of a non-homogeneous recurrence relation has two parts.

First part is the solution (a_h) of the associated homogeneous recurrence relation and the second part is the particular solution (a_t).

$a_n=a_h+a_t$

Equivalence Relation:

Definition: A relation R in a set A is called an **equivalence** relation if

- a R a for every i.e. R is reflexive
- **a R b => b R a for every a, b Є A i.e. R is symmetric**
- a R b and b R c=>a R c for every **a, b, c ЄA, i.e. R is transitive.**

For example

- The relation equality of numbers on set of real numbers.
- The relation being parallel on a set of lines in a plane.

Problem1: Let us consider the Set T of triangles in a plane. Let us define a relation **R in T as R= {(a, b) / (a,b Є T and a is similar to b}**
We hae to show that R is an equivalence relation

Solution:

- A triangle a is similar to itself. a R a
- If the triangle a is similar to the triangle b, the n triangle b is similar to the triangle a then a R b => b R a
- If a is similar to b and b is similar to c, then a is similar to c (i.e) a Rb and b R c=>a R c.

Hence R is an equivalence relation.

Problem2: Let x={1,2,3,…7}and R={(x, y)/x **–** y is divisible by3} Show that R is an equivalence relation.

Solution: For any a Є X, a-ais divisible by
3, Hence a R a, R is reflexive
For any a,bЄX, if a–b is divisible by 3,then b **–**a is also divisible by 3, R is symmetric.
For any a, b, c Є, if a R b and b R c, then a – b is divisible by3 and b**–**cisdivisibleby3. So that(a**–**b)+(b**–** c) is also divisible by 3, hence a **–** c is also divisible by 3. Thus R is transitive.
Hence R is equivalence.

Problem3: Let Z be these to fall integers. Let m be a fixed integer. Two integers a and
Are said to be congruent modulo m if and only if m divides a-b, in which case we write aº

b(modm). This relation is called the relation of congruence modulo m and we can show that is an equivalence relation.

Solution:

- a-a=0and m divides a **–**a(i.e)**a Ra,(a, a)Є R, R is** reflexive .
- a R b=m divides a-b

m dividesb-
a b º a (mod
m) b R a
that is R is symmetric.

- aR b and bR c=> aºb (mod m)and bº c(mod m)
    - m divides a**–** band m divides b-c
    - a**–** b = km and b **–c= lm for some k ,l Єz**
    - (a**–** b)+(b **–** c)=km +lm
    - a**–** c=(k+l)m
    - aºc(mod m)
    - aR c
    - R is transitive

Hence the congruence relation is an equivalence relation.

Equivalence Classes:

Let R be an equivalence relation on a set A. For an**y a ЄA, the equivalence class** generated by a **is the set of all elements b Є A such a R b and is denoted [a].** It is also called the R **– equivalence class and denoted by a Є A.** i.e., **[a] = {b Є A / b R a}**

Let Z be these to f integer and R be the relation **called“ congruencemodulo 3”** defined by R = {(x, y)/ xÎ Z Ù yÎZ Ù (x-y) is divisible by 3}

Then the equivalence classes are

[0]= {…-6,-3, 0,3, 6,**…}**
[1]= {…,-5,-**2, 1,4, 7,…}**
[2]= {…,-4,-**1, 2,5, 8,…}**

Composition of binary relations:

Definition: Let R be a relation from X toY and S be a relation from YtoZ. Then the relation R

o S is given by RoS={(x,z)/xÎXÙz ÎZÙ yÎY such that(x, y)ÎRÙ (y, z) ÎS)}is called the composite relation of R and S.

The operation of obtaining Ro S is called the **composition of relations**.

Example: Let R={(1,2),(3,4),(2,2)} and
S={(4,2),(2, 5),(3, 1),(1,3)}
Then RoS ={(1,5), (3,2), (2, 5)}and S oR ={(4, 2),(3,2),(1, 4)}
It is to be noted that R o S ≠ S o R.
Also Ro(S o T)=(R o S)o T=R o S o T

Note: We write R o RasR2; Ro R oR as R3andso on.

Definition

Let R be a relation from X to Y,a relation R from Y to X is called the **converse** of R, **where the ordered pairs of Ř are obtained by interchanging the numbers in each of the ordered** pairs of R. This means for x Î X and y Î **Y, that x R y ó y Ř x.**
Then the relation Ř is given by R={(x, y)/(y, x)ÎR} is called the converse
Of R Example:
Let R={(1,2),(3,4),(2,2)}
Then Ř={(2,1),(4,3),(2, 2)}

Note: If R is an equivalence rel**ation, then Ř is also an equivalence**

relation. Definition: Let X be any finite set and R be a relation in X.

The relation
R+=RUR2UR3…in X. is called the *transitive closure* of Rin X

Example: Let R={(a, b),(b,c),(c, a)}.
NowR2 =R oR ={(a, c),(b, a), (c, b)}
R3=R2o R={(a,a),(b,b), (c, c)}
R4=R3 oR ={(a, b),(b, c),(c,a)}=R
R5=R3o R2 =R2 and so on.

Thus, R+ = RUR2 UR3 UR4 U…
=R U R2 U R3.
={(a,b),(b,c),(c,a),(a,c),(b,a),(c,b),(a,b),(b,b),(c,c)}

We see that R+ is a transitive relation containing R. In fact, it is the smallest transitive relation containing R.

Inclusion-Exclusion Principle

Let A, B be any two finite sets. Then $n(A \cup B) = n(A) + n(B) - n(A \cap B)$

Here "include" n (A) and n (B) and we "exclude" $n(A \cap B)$

Example 1:

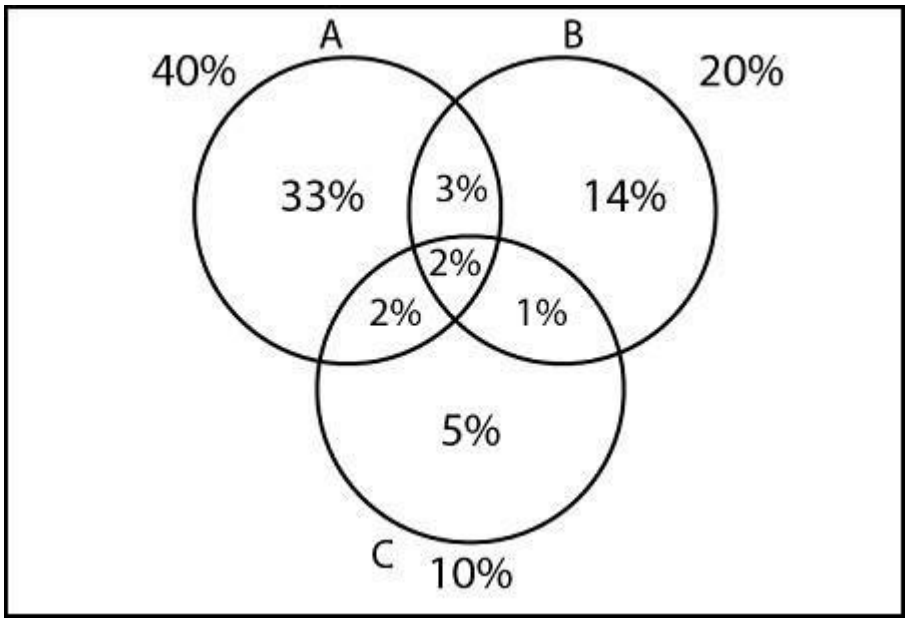
Suppose A, B, C are finite sets. Then $A \cup B \cup C$ is finite and $n(A \cup B \cup C) = n(A) + n(B) + n(C) - n(A \cap B) - n(A \cap C) - n(B \cap C) + n(A \cap B \cap C)$

Example 2:

In a town of 10000 families it was found that 40% of families buy newspaper A, 20% family buy newspaper B, 10% family buy newspaper C, 5% family buy newspaper A and B, 3% family buy newspaper B and C and 4% family buy newspaper A and C. If 2% family buy all the newspaper. Find the number of families which buy

1. Number of families which buy all three newspapers.
2. Number of families which buy newspaper A only
3. Number of families which buy newspaper B only
4. Number of families which buy newspaper C only
5. Number of families which buy None of A, B, C
6. Number of families which buy exactly only one newspaper
7. Number of families which buy newspaper A and B only
8. Number of families which buy newspaper B and C only
9. Number of families which buy newspaper C and A only
10. Number of families which buy at least two newspapers
11. Number of families which buy at most two newspapers
12. Number of families which buy exactly two newspapers

Solution:



1. Number of families which buy all three newspapers:

    1. n (A U B U C) = n(A) + n(B) + n(C) - n(A ∩ B) - n(A ∩ C) - n(B ∩ C) + n(A ∩ B ∩ C)2. n (A U B U C) = 40 + 20 + 10 - 5 - 3 - 4 + 2 = 60%

2. Number of families which buy newspaper A only

    1. = 40 - 7 = 33%

3. Number of families which buy newspaper B only

    1. = 20 - 6 = 14%

4. Number of families which buy newspaper C only

    1. = 10 - 5 = 5%

5. Number of families which buy None of A, B, and C

n (A UB UC)^c = 100 - n (A U B U C)
n (A UB UC)^c = 100 - [40 + 20 + 10 - 5- 3- 4 + 2]
n (A UB UC)^c = 100 - 60 = 40 %

6. Number of families which buy exactly only one newspaper

1. = 33 + 14 + 5 = 52%

7. Number of families which buy newspaper A and B only

    1. = 3%

8. Number of families which buy newspaper B and C only

    1. = 1%

9. Number of families which buy newspaper C and A only

    1. = 2%

10. Number of families which buy at least two newspapers

    1. = 8%

11. Number of families which buy at most two newspapers

    1. = 98%

12. Number of families which buy exactly two newspapers

    1. = 6%

UNIT-III

Graphs and Trees

Representation of Graphs:

There are two different sequential representations of a

graph.Theyare • Adjacency Matrix representation

- Path Matrix representation

Adjacency Matrix Representation

Suppose G is a simple directed graph with m nodes, and suppose the nodes of G have been ordered and are called v1, v2, . . . , vm. Then the adjacency matrix A= (aij) of the graph G is the m x m matrix defined as follows:

1 if vi is adjacent to Vj, that is,if there is an edge(Vi,Vj) aij=0 otherwise

Suppose G is an undirected graph. Then the adjacency matrix A of G will be a symmetric matrix, i.e., one in which aij = aji; for every i and j.

Drawbacks

It may be difficult to insert and delete nodes in G.

If the number of edges is 0(m) or 0(mlog2m), then the matrix A will be sparse, hence a great deal of space will be wasted.

Path Matrix Represenation

Let G be a simple directed graph with m nodes, v1,v2,...,vm. The path matrix of G is the m-square matrix P = (pij) defined as follows:

1ifthereisapathfromVitoVj

Pij=0 otherwise

Graphs and Multigraphs

A graph G consists of two things:

weighted or Labeled Graph

1. Aset V of elements called nodes (or points or vertices)

2. Aset E of edges such that each edge e in E is identified with a unique

(unordered) pair [u, v] of nodes in V, denoted by e = [u, v]

Sometimes we indicate the parts of a graph by writing G=(V,E).

Suppose e = [u, v]. Then the nodes u and v are called the endpoints of e, and u and v are said to be adjacent nodes or neighbors. The degree of a node u, written deg(u), is the number of edges containing u. If deg(u)=0—that is, if u does not be long to any edge—the n u is called an isolated node.

Path and Cycle

A path P of length n from a node u to a node v is defined as a sequence of n +1 nodes. P =(v0, v1, v2, . . . , vn)such that u =v0; vi-1 is adjacent to vi for i = 1,2, . . ., n and v n =v.

Types of Path

1. Simple Path
2. Cycle Path

(i) Simple Path

Simple path is a path in which first and last vertex are different(V0≠Vn)

(ii) Cycle Path

A cycle path is a path in which the first and last vertex are the same (V0=Vn). It is also called a closed path.

Connected Graph

A graph G is said to be connected if there is a path between any two of its nodes.

Complete Graph

A graph GGG is said to be complete if every node uuu in GGG is adjacent to every other node vvv in GGG.

Tree

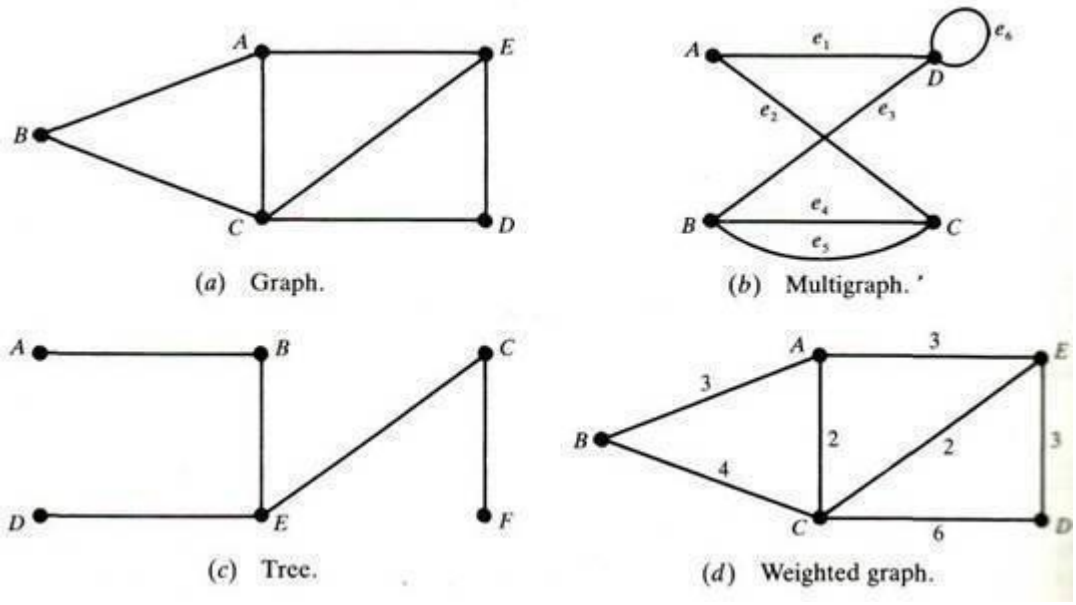
A connected graph T without any cycles is called a tree graph or free tree or, simply, a tree.

Labeled or Weighted Graph

If the weight is assigned to each edge of the graph then it is called as Weighted or Labeled graph.

The definition of a graph may be generalized by permitting the following:

- ***Multiple edges***: Distinct edges e and e' are called multiple edges if the y connect the same endpoints, that is, if e = [u, v] and e' = [u, v].
- ***Loops*:** An edge e is called a loop if it has identical end points, that is, if e=[u, u].
- ***Finite Graph***: A multigraph M is said to be finite if it has a finite number of node s and a finite number of edges.



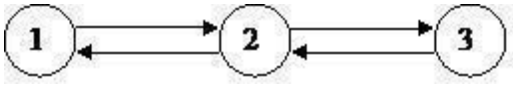
Directed Graphs

A directed graph G, also called a digraph or graph is the same as a multigraph except that each edge e in G is assigned a direction, or in other words, each edge e is identified with an ordered pair (u, v) of nodes in G.

Outdegree and Indegree

Indegree: The indegree of a vertex is the number of edges for which v is head

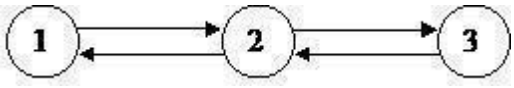
Example



Indegreeof1=1
Indegreepf2=2

Outdegree: The out degree of a node or vertex is the number of edges for which v is tail.

Example



Outdegreeof 1 =1
Outdegreeof 2 =2

Simple Directed Graph

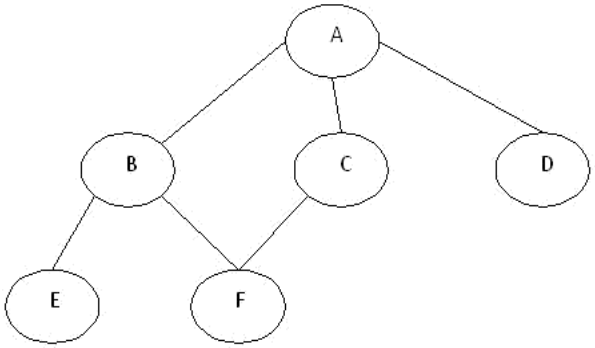
AdirectedgraphGissaidtobesimpleifGhasnoparalleledges.AsimplegraphG may have loops, but it cannot have more than one loop at a given node.

Graph Traversal

The breadth first search (BFS) and the depth first search (DFS) are the two algorithms usedfor traversing and searching a node in a graph. They can also be used to find out whether a node is reachable from a given node or not.

Depth First Search (DFS)

The aim of DFS algorithm is to traverse the graph in such a way that it tries to go far from the root node. Stack is used in the implementation of the depth first search. Let's see how depth first search works with respect to the following graph:



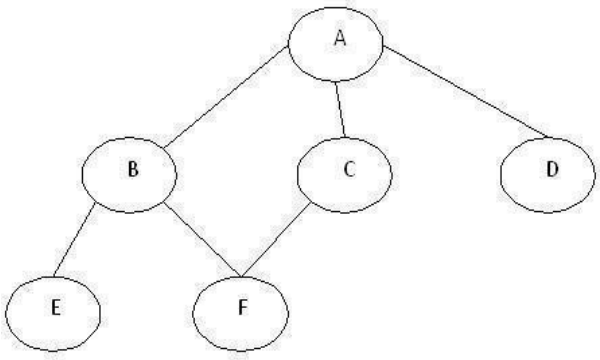
As stated before, in DFS, nodes are visited by going through the depth of the tree from the starting node. If we do the depth first traversal of the above graph and print the visited node, it will be “A B E F C D”. DFS visits the root node and then its children nodes until it reaches the end node, i.e. E and F nodes, then moves up to the parent nodes.

Algorithmic Steps

1. **Step1**: Push the root node in the Stack.
2. **Step2**: Loop until stack is empty.
3. **Step3**: Peek the node of the stack.
4. **Step4**: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
5. **Step5**: If the node does not have any unvisited child nodes, pop the node from the stack.

Breadth First Search(BFS)

This is a very different approach for traversing the graph nodes. The aim of BFS algorithm is to traverse the graph as close as possible to the root node. Queue is used in the implementation of the breadth first search. Let’s see how BFS traversal works with respect to the following graph:



If we do the breadth first traversal of the above graph and print the visited node as the output, it will print the following output. “ABC DEF”. The BFS visits the nodes level by level, so it will start with level 0 which is the root node, and then it moves to the next levels which are B, C and D, then the last levels which are E and F.

Algorithmic Steps

1. **Step1**: Push the root node in the Queue.
2. **Step2**: Loop until the queue is empty.
3. **Step3**: Remove the node from the Queue.
4. **Step4**: If the removed node has unvisited child nodes, mark the mas visited and insert the unvisited children in the queue.

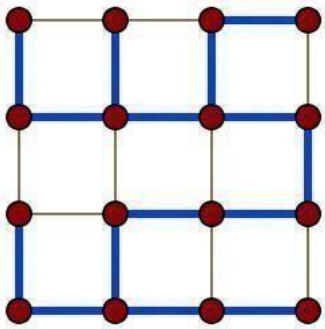
   Based upon the above steps, the following Java code shows the implementation of the BFS algorithm:

Spanning Trees:

In the mathematical field of graph theory, a **spanning tree** *T* of a connected, undirected graph *G* is a tree composed of all the vertices and some (or perhaps all) of the edges of *G*. Informally, a spanning tree of *G* is a selection of edges of *G* that form a tree *spanning* every vertex. That is, every vertex lies in the tree, but no cycles (or loops) are formed. On the other hand, every bridge of *G* must belong to *T*.

A spanning tree of a connected graph *G* can also be defined as a maximal set of edges of *G* that contains no cycle, or as a minimal set of edges that connect all vertices.

Example



A spanning tree (blue heavy edges) of a grid graph.

Spanning forests

A **spanning forest** is a type of subgraph that generalises the concept of a spanning tree. However, there are two definitions in common use. One is that a spanning forest is a subgraph that consists of a spanning tree in each connected component of a graph. (Equivalently, it is a maximal cycle-free subgraph.) This definition is common in computer science and optimisation. It is also the definition used when discussing minimum spanning forests, the generalization to disconnected graphs of minimum spanning trees. Another definition, common in graph theory, is that a spanning forest is any subgraph that is both a forest (contains no cycles) and spanning (includes every vertex).

Counting spanning trees

The number *t(G)* of spanning trees of a connected graph is an important invariant. In some cases, it is easy to calculate *t(G)*directly. It is also widely used in data structures in different computer

languages. For example, if *G* is itself a tree, then *t(G)=1*, while if *G* is the cycle graph C_n with *n* vertices, then *t(G)=n*. For any graph *G*, the number *t(G)* can be calculated using Kirchhoff's matrix-tree theorem (follow the link for an explicit example using the theorem).

Planar Graphs:

In graph theory, a **planar graph** is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints.

A planar graph already drawn in the plane without edge intersections is called a **plane graph** or **planar embedding of the graph**. A plane graph can be defined as a planar graph with a mapping from every node to a point in 2D space, and from every edge to a plane curve, such that the extreme points of each curve are the points mapped from its end nodes, and all curves are disjoint except on their extreme points. Plane graphs can be encoded by combinatorial maps.

It is easily seen that a graph that can be drawn on the plane can be drawn on the sphere as well, and vice versa.

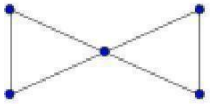
The equivalence class of topologically equivalent drawings on the sphere is called a **planar map**. Although a plane graph has an **external** or **unbounded** face, none of the faces of a planar map have a particular status.

Applications

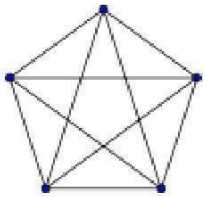
- Telecommunications–e.g. spanning trees
- Vehicle routing–e.g. planning routes on roads without under passes
- VLSI – e.g. laying out circuits on computer chip.
- ThepuzzlegamePlanarityrequirestheplayerto"untangle"aplanargraphsothatnone of its edges intersect.

Example graphs

Planar **Nonplanar**

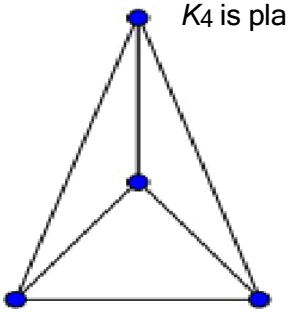


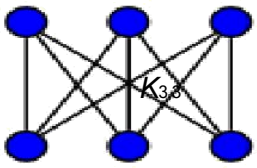
Butterfly graph



K_5

The complete graph K_4 is planar





Graph Theory and Applications:

Graphs are among the most ubiquitous models of both natural and human-made structures. They can be used to model many types of relations and process dynamics in physical, biological and social systems.

Many problems of practical interest can be represented by graphs.

In computer science, graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc.

One practical example: The link structure of a website could be represented by a directed graph. The vertices are the web pages available at the website and a directed edge from page *A* to page *B* exists if and only if *A* contains a link to *B*.

A similar approach can be taken to problems in travel, biology, computer chip design, and many other fields. The development of algorithms to handle graphs is therefore of major interest in computer science.

There, the transformation of graphs is often formalized and represented by graph rewrite systems.

They are either directly used or properties of the rewrite systems (e.g. confluence) are studied. Complementary to graph transformation systems focussing on rule- based in-memory manipulation of graphs are graph databases geared towards transaction-safe, persistent storing and querying of graph-structured data.

Graph-theoretic methods, in various forms, have proven particularly useful in linguistics, since natural language often lends itself well to discrete structure.

Traditionally, syntax and compositional semantics follow tree-based structures, whose expressive power lies in the Principle of Compositionality, modeled in a hierarchical graph.

Within lexical semantics, especially as applied to computers, modeling word meaning is easier when a given word is understood in terms of related words; semantic networks are therefore important in computational linguistics.

Still other methods in phonology (e.g. Optimality Theory, which uses lattice graphs) and morphology (e.g. finite-state morphology, using finite-state transducers) are common in the analysis of language as a graph.

Indeed, the usefulness of this area of mathematics to linguistics has borne organizations such as Text Graphs, as well as various 'Net' projects, such as WordNet, Verb Net, and others.

Graph theory is also used to study molecules in chemistry and physics. In condensed matter physics, the three dimensional structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms.

For example, Franzblau's shortest-path (SP) rings. In chemistry a graph makes a natural model for a molecule, where vertices represent atoms and edges bonds.

This approach is especially used in computer processing of molecular structures, ranging from chemical editors to database searching.

In statistical physics, graphs can represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems.

Graph theory is also widely used in sociology as a way, for example, to measure actors' prestige or to explore diffusion mechanisms, notably through the use of social network analysis software.

Like wise, graph theory is useful in biology and conservation efforts where a vertex can represent regions where certain species exist (or habitats) and the edges represent migration

paths, or movement between the regions. This information is important when looking at breeding patterns or tracking the spread of disease, parasites or how changes to the movement can affect other species.

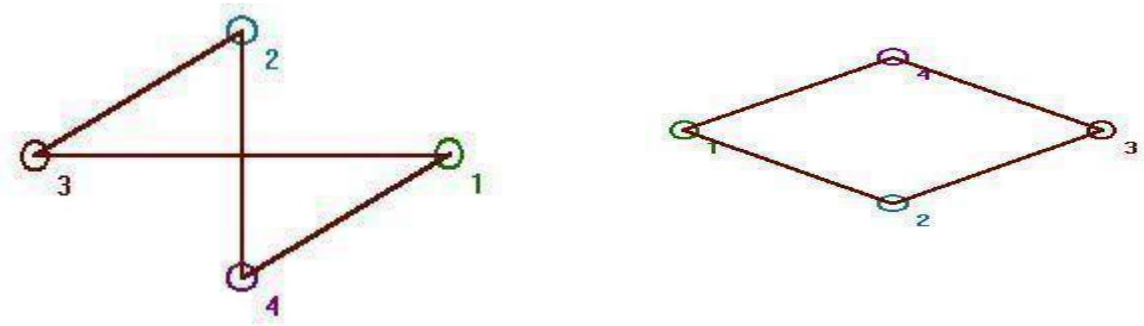
In mathematics, graphs are useful in geometry and certain parts of topology, e.g. Knot Theory. Algebraic graph theory has close links with group theory.

A graph structure can be extended by assigning a weight to each edge of the graph. Graphs with weights, or weighted graphs, are used to represent structures in which pairwise connections have some numerical values. For example if a graph represents a road network, the weights could represent the length of each road.

Basic Concepts Isomorphism:

Let G1and G1betwo graphs and let f be a function from the vertex set ofG1to the vertex set of G2. Suppose that f is one-to-one and onto & f(v) is adjacent to f(w) in G2 if and only if v is adjacent to w in G1.

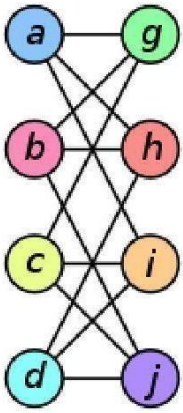
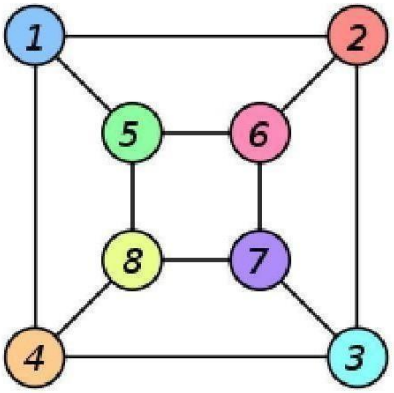
Then we say that the function f is an isomorphism and that the two graphs G1 and G2 are isomorphic. So two graphs G1 and G2 are isomorphic if there is a one-to-one correspondence between vertices of G1 and those of G2 with the property that if two vertices of G1 are adjacent then so are their images in G2. If two graphs are isomorphic then as far as we are concerned they are the same graph though the location of the vertices may be different. To show you how the program can be used to explore isomorphism draw the graph in figure 4 with the program (first get the null graph on four vertices and then use the right mouse to add edges).



Save this graph as Graph1(you need to click Graph then Save). Now getthecircuitgraphwith4 vertices. It looks like figure 5, and we shall call it C(4).

Example:

The two graphs shown below are isomorphic, despite their different looking drawings.

GraphG	GraphH	Anisomorphism betweenGandH
		$f(a)=1$ $f(b)=6$ $f(c)=8$ $f(d)=3$ $f(g)=5$ $f(h)=2$ $f(i)=4$ $f(j)=7$

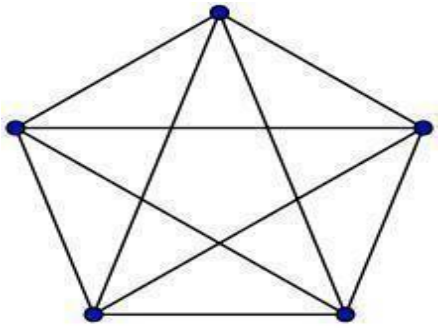
Subgraphs:

A **subgraph** of a graph *G* is a graph whose vertex set is a subset of that of *G*, and whose adjacency relation is a subset of that of *G* restricted to this subset. In the other direction, a **super graph** of a graph *G* is a graph of which *G* is a subgraph. We say a graph *G* **contains** another graph *H* if some subgraph of *G* is *H* or is isomorphic to *H*.

A subgraph *H* is a **spanning subgraph**, or **factor**, of a graph *G* if it has the same vertex set as *G*. We say *H* spans *G*.

A subgraph *H* of a graph *G* is said to be **induced** if, for any pair of vertices *x* and *y* of H, *x y* is an edge of *H* if and only if *x y* is an edge of G. In other words, *H* is an induced subgraph of *G* if it has all the edges that appear in *G* over the same vertex set. If the vertex set of *H* is the subset *S*of *V(G)*, then *H* can be written as *G*[*S*] and is said to be **induced by *S***.

A graph that does *not* contain *H* as an induced subgraph is said to be ***H*-free**.
A **universal graph** in a class ***K*** of graphs is a simple graph in which every element in ***K*** can be embedded as a subgraph.



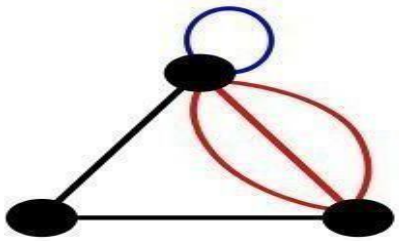
K_5, a complete graph. If a subgraph looks like this, the vertices in that subgraph form aclique of size 5.

Multigraphs:

In mathematics, a **multigraph** or **pseudograph** is a graph which is permitted to have multiple edges, (also called "parallel edges"), that is, edges that have the same end nodes. Thus two vertices may be connected by more than one edge. Formally, a multigraph G is an ordered pair $G:=(V, E)$ with

- V a set of *vertices* or *nodes*,
- E a multi set of unordered pairs of vertices, called *edges* or *lines*.

Multigraphs might be used to model the possible flight connections offered by an airline. In this case the multigraph would be a directed graph with pairs of directed parallel edges connecting cities to show that it is possible to fly both *to* and *from* these locations.



A multigraph with multiple edges (red) and a loop (blue). Not all authors allow multigraphs to have loops.

Euler circuits:

In graph theory, an **Eulerian trail** is a trail in a graph which visits every edge exactly once. Similarly, an **Eulerian circuit** is an Eulerian trail which starts and ends on the same vertex. They were first discussed by Leonhard Euler while solving the famous Seven Bridges of Königsberg problem in 1736. Mathematically the problem can be stated like this:

Given the graph on the right, is it possible to construct a path (or a cycle, i.e. a path starting and ending on the same vertex) which visits each edge exactly once?

Euler proved that a necessary condition for the existence of Eulerian circuits is that all vertices in the graph have an even degree, and stated without proof that connected graphs with all vertices of even degree have an Eulerian circuit. The first complete proof of this latter claim was published in 1873 by Carl Hierholzer.

The term **Eulerian graph** has two common meanings in graph theory. One meaning is a graph with an Eulerian circuit, and the other is a graph with every vertex of even degree. These definitions coincide for connected graphs.

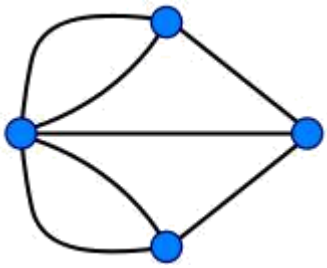
For the existence of Eulerian trails it is necessary that no more than two vertices have an odd degree; this means the Königsberg graph is *not* Eulerian. If there are no vertices of odd degree, all Eulerian trails are circuits. If there are exactly two vertices of odd degree, all Eulerian trails start at one of them and end at the other. Sometimes a graph that has an Eulerian trail but not an Eulerian circuit is called **semi-Eulerian**.

An **Eulerian trail**, **Eulerian trail** or **Euler walk** in an undirected graph is a path that uses each edge exactly once. If such a path exists, the graph is called **traversable** or **semi-eulerian**.

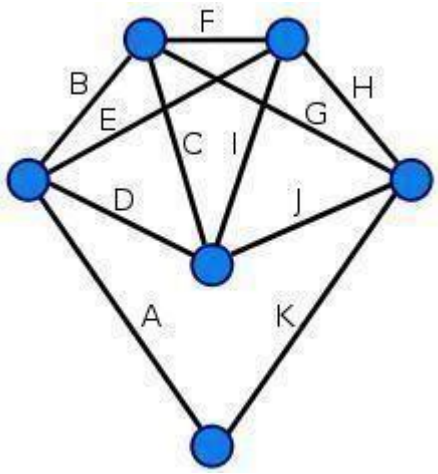
An **Eulerian cycle**, **Eulerian circuit** or **Euler tour** in an undirected graph is a cycle that uses each edge exactly once. If such a cycle exists, the graph is called **unicursal**. While such graphs are Eulerian graphs, not every Eulerian graph possesses an Eulerian cycle.

For directed graphspathhastobereplacedwithdirectedpathandcyclewithdirectedcycle.

The definition and properties of Eulerian trails, cycles and graphs are valid for multigraphs as well.



This graph is not Eulerian, therefore, a solution does not exist.



Every vertex of this graph has an even degree, therefore this is an Eulerian graph. Following the edges in alphabetical order gives an Eulerian circuit/cycle.

Hamiltonian graphs:

In the mathematical field of graph theory, a **Hamiltonian path** (or **traceable path**) is a path in an undirected graph which visits each vertex exactly once. A **Hamiltonian cycle** (or **Hamiltonian circuit**) is a cycle in an undirected graph which visits each vertex exactly once and also returns to the starting vertex. Determining whether such paths and cycles exist in graphs is the Hamiltonian path problem which is NP-complete.

Hamiltonian paths and cycles are named after William Rowan Hamilton who invented the Icosian game, now also known as *Hamilton's puzzle*, which involves finding a Hamiltonian cycle in the edge graph of the dodecahedron. Hamilton solved this problem using the Icosian Calculus, an algebraic structure based on roots of unity with many similarities to the quaternions (also invented by Hamilton). This solution does not generalize to arbitrary graphs.

A *Hamiltonian path* or *traceable path* is a path that visits each vertex exactly once. A graph that contains a Hamiltonian path is called a **traceable graph**. A graph is **Hamilton-connected** if for every pair of vertices there is a Hamiltonian path between the two vertices.

A *Hamiltonian cycle*, *Hamiltonian circuit*, *vertex tour* or *graph cycle* is a cycle that visits each vertex exactly once (except the vertex which is both the start and end, and so is visited twice). A graph that contains a Hamiltonian cycle is called a **Hamiltonian graph**.

Similar notions may be defined for *directed graphs*, where each edge(arc) of a path or cycle can only be traced in a single direction (i.e., the vertices are connected with arrows and the edges traced "tail-to-head").

A**Hamiltoniandecomposition**isanedgedecompositionofagraphintoHamiltoniancircuits.

Examples

- a complete graph with more than two vertices is Hamiltonian
- every cycle graph is Hamiltonian
- every tournament has an odd number of Hamiltonian paths
- every platonic solid, considered as a graph, is Hamiltonian

Chromatic Numbers:

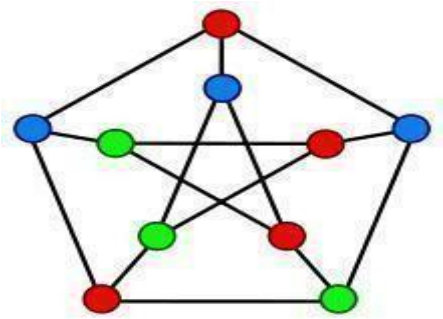
In graph theory, **graph coloring** is a special case of graph labeling; it is an assignment of labels traditionally called "colors" to elements of a graph subject to certain constraints. In its simplest form, it is a way of coloring the vertices of a graph such that no two adjacent vertices share the same color; this is called a **vertex coloring**. Similarly, an **edge coloring** assigns a color to each edge so that no two adjacent edges share the same color, and a **face coloring** of a planar graph assigns a color to each face or region so that no two faces that share a boundary have the same color.

Vertex coloring is the starting point of the subject, and other coloring problems can be transformed into a vertex version. For example, an edge coloring of a graph is just a vertex coloring of its line graph, and a face coloring of a planar graph is just a vertex coloring of its planar dual. However, non-vertex coloring problems are often stated and studied *as is*. That is partly for perspective, and partly because some problems are best studied in non-vertex form, as for instance is edge coloring.

The convention of using colors originates from coloring the countries of a map, where each face is literally colored. This was generalized to coloring the faces of a graph embedded in the plane. By planar duality it became coloring the vertices, and in this form it generalizes to all graphs. In mathematical and computer representations it is typical to use the first few positive or nonnegative integers as the "colors". In general one can use any finite set as the "color set". The nature of the coloring problem depends on the number of colors but not on what they are.

Graph coloring enjoys many practical applications as well as theoretical challenges. Beside the classical types of problems, different limitations can also be set on the graph, or on the way a

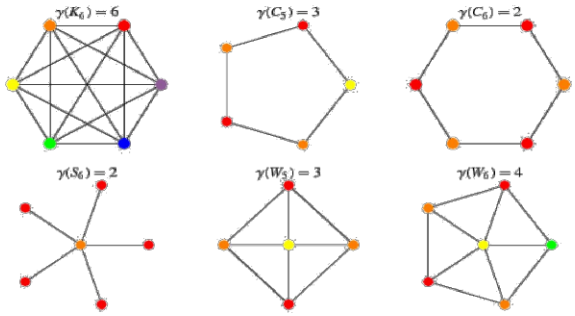
color is assigned, or even on the color itself. It has even reached popularity with the general public in the form of the popular number puzzle Sudoku. Graph coloring is still a very active field of research.



A proper vertex coloring of the Petersen graph with 3 colors, the minimum number possible.

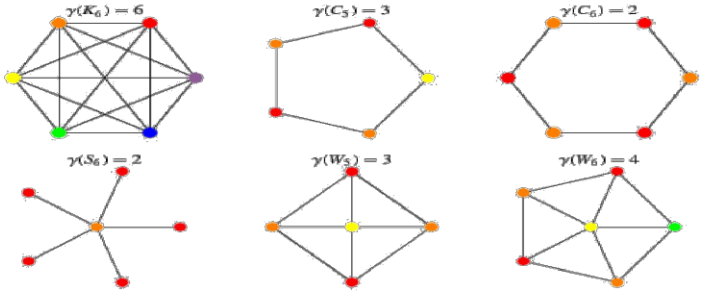
Vertex coloring

	$\gamma(G)$
	n
	$\begin{cases} 3 & \text{for n odd} \\ 2 & \text{for n even} \end{cases}$
$S_n \quad n>1$	
$W_n \quad n>2$	$\begin{cases} 3 & \text{for n odd} \\ 4 & \text{for n even} \end{cases}$



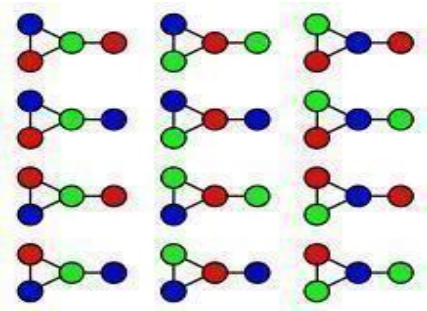
When used without any qualification, a **coloring** of a graph is almost always a *proper vertex coloring*, namely a labelling of the graph's vertices with colors such that no two vertices sharing the same edge have the same color. Since a vertex with a loop could never be properly colored, it is understood that graphs in this context are loopless.

	$\gamma(G)$
	n
	$\begin{cases} 3 & \text{for n odd} \\ 2 & \text{for n even} \end{cases}$
$S_n \quad n>1$	
$W_n \quad n>2$	$\begin{cases} 3 & \text{for n odd} \\ 4 & \text{for n even} \end{cases}$



The terminology of using *colors* for vertex labels goes back to map coloring. Labels like *red* and *blue* are only used when the number of colors is small, and normally it is understood that the labels are drawn from the integers {1,2,3,...}.

A coloring using at most *k* colors is called a (proper) ***k*-coloring**. The smallest number of colors needed to color a graph *G* is called its **chromatic number**, $\chi(G)$. A graph that can be assigned a (proper) *k*-coloring is ***k*-colorable**, and it is ***k*-chromatic** if its chromatic number is exactly *k*. A subset of vertices assigned to the same color is called a *color class*, every such class forms an independent set. Thus, a *k*-coloring is the same as a partition of the vertex set into *k* independent sets, and the terms *k-partite* and *k-colorable* have the same meaning.



UNIT-IV
Boolean Algebra and Models of Computation

Boolean Functions

- **Definition**: Boolean functions are mathematical expressions that involve Boolean variables and operators (AND, OR, NOT) and return Boolean values (TRUE or FALSE).
- **Variables**: Represented by letters (e.g., A, B, C) and take values 0 (false) or 1 (true).
- **Operators**:
  - **AND (·)**: A · B = 1 if both A and B are 1.
  - **OR (+)**: A + B = 1 if either A or B is 1.
  - **NOT (¬ or ')**: ¬A or A' = 1 if A is 0.

Representing Boolean Functions

- **Truth Tables**: List all possible values of input variables and the corresponding output.
- **Algebraic Expressions**: Use AND, OR, and NOT operators to express the function.
- **Canonical Forms**:
  - **Sum of Products (SOP)**: OR of AND terms (e.g., A·B + ¬A·C).
  - **Product of Sums (POS)**: AND of OR terms (e.g., (A+B)·(¬A+C)).

Logic Gates

- **Basic Gates**:
  - **AND Gate**: Output is 1 if all inputs are 1.
  - **OR Gate**: Output is 1 if at least one input is 1.
  - **NOT Gate**: Output is the complement of the input.
- **Derived Gates**:
  - **NAND Gate**: Output is the complement of the AND gate (¬(A · B)).
  - **NOR Gate**: Output is the complement of the OR gate (¬(A + B)).
  - **XOR Gate**: Output is 1 if inputs are different (A ⊕ B).
  - **XNOR Gate**: Output is 1 if inputs are the same (¬(A ⊕ B)).

Minimization of Circuits

- **Purpose**: Simplify Boolean expressions to reduce the number of gates and inputs, optimizing the circuit design.
- **Methods**:
  - **Algebraic Manipulation**: Apply Boolean algebra rules to simplify expressions.
  - **Karnaugh Maps (K-Maps)**: Visual method to simplify expressions by grouping ones in a grid.
  - **Quine-McCluskey Method**: Tabular method for minimization, useful for more complex functions.

Languages and Grammars

- **Languages**: Sets of strings over an alphabet that follow specific rules.
- **Grammars**: Define the syntax rules for generating strings in a language.
    - **Types** (Chomsky Hierarchy):
        - **Type 0**: Unrestricted grammars.
        - **Type 1**: Context-sensitive grammars.
        - **Type 2**: Context-free grammars (CFGs), used for programming languages.
        - **Type 3**: Regular grammars, used for simple pattern matching.
- **Components of Grammar**:
    - **Terminals**: Basic symbols from which strings are formed.
    - **Non-terminals**: Variables representing sets of strings.
    - **Production Rules**: Rules for replacing non-terminals with terminals or other non-terminals.
    - **Start Symbol**: Non-terminal symbol from which derivations begin.

Finite State Machines (FSMs)

- **Definition**: Abstract computational models with a finite number of states.
- **Components**:
    - **States**: Distinct configurations of the machine.
    - **Alphabet**: Set of symbols that FSM processes.
    - **Transition Function**: Rules for moving from one state to another based on input symbols.
    - **Start State**: The state where computation begins.
    - **Accept States**: States that signify successful completion.
- **Types**:
    - **Finite State Machines with Output**:
        - **Mealy Machine**: Outputs depend on states and inputs.
        - **Moore Machine**: Outputs depend only on states.
    - **Finite State Machines with No Output**:
        - Accepts or rejects input strings based on reaching an accept state.

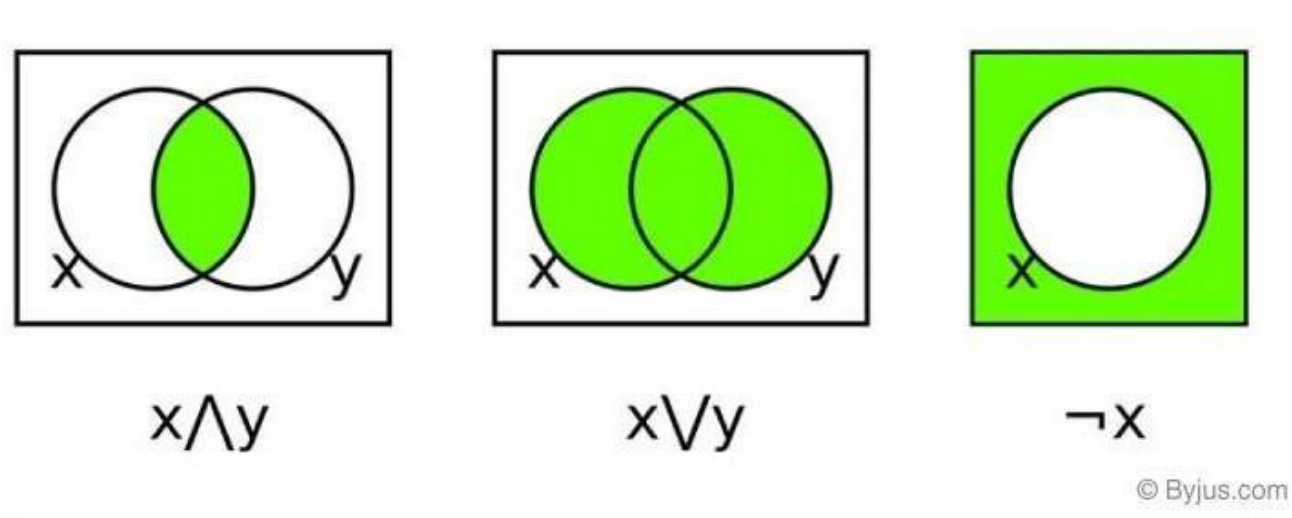
Boolean algebra is the category of algebra in which the variable's values are the truth values, **true and false,** ordinarily denoted 1 and 0 respectively. It is used to analyze and simplify digital circuits or digital gates. It is also called **Binary Algebra** or **logical Algebra**. It has been fundamental in the development of digital electronics and is provided for in all modern programming languages. It is also used in set theory and statistics.

The important operations performed in Boolean algebra are – **conjunction (∧), disjunction (∨) and negation (¬)**. Hence, this algebra is far way different from elementary algebra where the values of variables are numerical and arithmetic operations like addition, subtraction is been performed on them.

Boolean Algebra Operations

The basic operations of Boolean algebra are as follows:

- Conjunction or AND operation
- Disjunction or OR operation
- Negation or Not operation



Below is the table defining the symbols for all three basic operations.

Operator	Symbol	Precedence
NOT	‘ (or) ¬	Highest
AND	. (or) ∧	Middle
OR	+ (or) ∨	Lowest

Suppose A and B are two Boolean variables, then we can define the three operations as;

- A conjunction B or A AND B, satisfies A ∧ B = True, if A = B = True or else A ∧ B = False.
- A disjunction B or A OR B, satisfies A ∨ B = False, if A = B = False, else A ∨ B = True.
- Negation A or ¬A satisfies ¬A = False, if A = True and ¬A = True if A = False

Boolean Expression

A logical statement that results in a Boolean value, either be True or False, is a Boolean expression. Sometimes, synonyms are used to express the statement such as ‘Yes’ for ‘True’ and ‘No’ for ‘False’. Also, 1 and 0 are used for digital circuits for True and False, respectively.

Boolean expressions are the statements that use logical operators, i.e., AND, OR, XOR and NOT. Thus, if we write X AND Y = True, then it is a Boolean expression.

Boolean Algebra Terminologies

Now, let us discuss the important terminologies covered in Boolean algebra.

Boolean Algebra: Boolean algebra is the branch of algebra that deals with logical operations and binary variables.

Boolean Variables: A Boolean variable is defined as a variable or a symbol defined as a variable or a symbol, generally an alphabet that represents the logical quantities such as 0 or 1.

Boolean Function: A Boolean function consists of binary variables, logical operators, constants such as 0 and 1, equal to the operator, and the parenthesis symbols.

Literal: A literal may be a variable or a complement of a variable.

Complement: The complement is defined as the inverse of a variable, which is represented by a bar over the variable.

Truth Table: The truth table is a table that gives all the possible values of logical variables and the combination of the variables. It is possible to convert the Boolean equation into a truth table. The number of rows in the truth table should be equal to 2^n, where "n" is the number of variables in the equation. For example, if a Boolean equation consists of 3 variables, then the number of rows in the truth table is 8. (i.e.,) $2^3 = 8$.

Boolean Algebra Truth Table

Now, if we express the above operations in a truth table, we get;

A	B	A ∧ B	A ∨ B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

A	¬A
True	False
False	True

Boolean Algebra Rules

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- The complement of a variable is represented by an overbar.

  $Thus, complement\ of\ variable\ B\ is\ represented\ as\ \overline{B}.\ Thus\ if\ B$=0 $then\ \overline{B}$ =1 $and\ B$ =1 $then\ \overline{B}$ =0.

- OR-ing of the variables is represented by a plus (+) sign between them. For example, the OR-ing of A, B, and C is represented as A + B + C.
- Logical AND-ing of the two or more variables is represented by writing a dot between them, such as A.B.C. Sometimes, the dot may be omitted like ABC.

Related Links	
Truth Table	Tautology
Conjunction	Mathematical Logic

Laws of Boolean Algebra

There are six types of Boolean algebra laws. They are:

- Commutative law
- Associative law
- Distributive law
- AND law
- OR law
- Inversion law

Those six laws are explained in detail here.

Commutative Law

Any binary operation which satisfies the following expression is referred to as a commutative operation. Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

- A. B = B. A
- $A + B = B + A$

Associative Law

It states that the order in which the logic operations are performed is irrelevant as their effect is the same.

- (A. B). C = A . (B . C)
- (A + B) + C = A + (B + C)

Distributive Law

Distributive law states the following conditions:

- A. (B + C) = (A. B) + (A. C)
- A + (B. C) = (A + B) . (A + C)

AND Law

These laws use the AND operation. Therefore they are called AND laws.

- A .0 = 0
- A . 1 = A
- A. A = A
- $A.\bar{A}$ =0

OR Law

These laws use the OR operation. Therefore they are called OR laws.

- A + 0 = A
- A + 1 = 1
- A + A = A
- $A+\bar{A}$ =1

Inversion Law

In Boolean algebra, the inversion law states that double inversion of variable results in the original variable itself.

- $\bar{\bar{A}}=A$

Boolean Algebra Theorems

The two important theorems which are extremely used in Boolean algebra are De Morgan's First law and De Morgan's second law. These two theorems are used to change the Boolean expression. This theorem basically helps to reduce the given Boolean expression in the simplified form. These two De Morgan's laws are used to change the expression from one form to another form. Now, let us discuss these two theorems in detail.

De Morgan's First Law:

De Morgan's First Law states that (A.B)' = A'+B'.

The first law states that the complement of the product of the variables is equal to the sum of their individual complements of a variable.

The truth table that shows the verification of De Morgan's First law is given as follows:

A	B	A'	B'	(A.B)'	A'+B'
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

The last two columns show that (A.B)' = A'+B'.

Hence, De Morgan's First Law is proved.

De Morgan's Second Law:

De Morgan's Second law states that (A+B)' = A'. B'.

The second law states that the complement of the sum of variables is equal to the product of their individual complements of a variable.

The following truth table shows the proof for De Morgan's second law.

A	B	A'	B'	(A+B)'	A'. B'
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

The last two columns show that (A+B)' = A'. B'.

Hence, De Morgan's second law is proved.

The other theorems in Boolean algebra are complementary theorem, duality theorem, transposition theorem, redundancy theorem and so on. All these theorems are used to simplify the given Boolean expression. The reduced Boolean expression should be equivalent to the given Boolean expression.