LECTURE NOTES

ON

DISCRETE MATHEMATICAL STRUCTURES

Mrs. B pravallika

Assistant Professor

INFORMATION TECHNOLOGY

INSTITUTE OF AERONAUTICAL ENGINEERING

(AUTONOMOUS)

DUNDIGAL, HYDERABAD - 500 043

Mathematical Logic

Statements and notations:

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)". However 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 ∧ q	"p and q"
Disjunction	p ∨ q	"p or q (or both)"
Exclusive Or	p ⊕ q	"either p or q, but not 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

Binary operations

Truth table for all binary logical operators

Here is a truth table giving definitions of all 16 of the possible truth functions of 2 binary variables (P,Q are thus boolean variables):

P	Q		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
T	**T**		F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
T	**F**		F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
F	**T**		F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
F	**F**		F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T

where T = true and F = false.

Key:

0, false, Contradiction

1, NOR, Logical NOR

2, Converse nonimplication

3, **¬p**, Negation

4, Material nonimplication

5, **¬q**, Negation

6, XOR, Exclusive disjunction

7, NAND, Logical NAND

8, AND, Logical conjunction

9, XNOR, If and only if, Logical

biconditional 10, **q**, Projection function

11, if/then, Logical implication

12, **p**, Projection function

13, then/if, Converse implication

14, OR, Logical disjunction

15, true, Tautology

Logical operators can also be visualized using Venn diagrams.

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 p ∧ q, p & q, or p q) is as follows:

Logical Conjunction		
p	*q*	$p \land q$
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 $p \land q$ is true. For all other assignments of logical values to *p* and to *q* the conjunction $p \land q$ is false. It can also be said that if *p*, then $p \land q$ is *q*, otherwise $p \land q$ is *p*.

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) is as follows:

Logical Disjunction		
p	*q*	$p \lor q$
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 $p \rightarrow q$) and the logical implication **p implies q** (symbolized as $p \Rightarrow q$) is as follows:

Logical Implication		
p	*q*	*p → 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* ≡ *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 ⊕ q**, or **p ≠ q**) is as follows:

Exclusive Disjunction		
p	*q*	*p* ⊕ *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* ∧ *q*), and the disjunction of negations: (¬*p*) ∨ (¬*q*) can be tabulated as follows:

p	*q*	$p \wedge q$	$\neg(p \wedge q)$	¬*p*	¬*q*	$(\neg p) \vee (\neg q)$
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*) ∧ (¬*q*) can be tabulated as follows:

p	q	$p \lor q$	$\neg(p \lor q)$	$\neg p$	$\neg q$	$(\neg p) \land (\neg q)$
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 $\neg(p \land q)$ as for $(\neg p) \lor (\neg q)$, and for $\neg(p \lor q)$ as for $(\neg p) \land (\neg q)$. 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 De Morgan'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	$\neg p$	$p \land q$	$p \lor q$	$p \oplus q$	$p \rightarrow q$	$p \leftrightarrow q$
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 ∨ represents a non-exclusive or, i.e., p ∨ q is true when any of p, q is true and also when both are true. On the other hand ⊕ represents an exclusive or, i.e., p ⊕ q is true only when exactly one of p and q is true.

Well formed formulas(wff):

Not all strings can represent propositions of the predicate logic. Those which produce a proposition when their symbols are interpreted must follow the rules given below, and they are called wffs(well-formed formulas) of the first order predicate logic.

Rules for constructing Wffs

A predicate name followed by a list of variables such as P(x, y), where P ispredicate name, and x and y are variables, is called an atomic formula.

A well formed formula of predicate calculus is obtained by using the following rules.

  1. An atomic formula is a wff.

  2. If A is a wff, then 7A is also a wff.

  3. If A and B are wffs, then (A V B), (A $^\wedge$ B), (A → B) and (A D B).

  4. If A is a wff and x is a any variable, then (x)A and ($x)A are wffs.

  5. Only those formulas obtained by using (1) to (4) are wffs.

Since we will be concerned with only wffs, we shall use the term formulas for wff. We shall follow the same conventions regarding the use of parentheses as was done in the case of statement formulas.

Wffs are constructed using the following rules:

1. *True* and *False* are wffs.

2. Each propositional constant (i.e. specific proposition), and each propositional variable (i.e. a variable representing propositions) are wffs.

3. Each atomic formula (i.e. a specific predicate with variables) is a wff.

4. If *A, B,* and *C* are wffs, then so are $\neg A$, (A $\wedge$ B), (A $\vee$ B), (A → B), and (A ↔ B).

5. If *x* is a variable (representing objects of the universe of discourse), and *A* is a wff, then so are $\forall x\, A$ and $\exists x\, A$.

For example, "The capital of Virginia is Richmond." is a specific proposition. Hence it is a wff by Rule 2.

Let B be a predicate name representing "being blue" and let x be a variable. Then B(x) is an

atomic formula meaning "x is blue". Thus it is a wff by Rule 3. above. By applying Rule 5. to B(x), ∀xB(x) is a wff and so is ∃xB(x). Then by applying Rule 4. to them ∀x B(x) ∧∃x B(x) is seen to be a wff. Similarly if R is a predicate name representing "being round". Then R(x) is an atomic formula. Hence it is a wff. By applying Rule 4 to B(x) and R(x), a wff B(x) ∧R(x) is obtained.

In this manner, larger and more complex wffs can be constructed following the rules given above.

Note, however, that strings that can not be constructed by using those rules are not wffs. For example, ∀xB(x)R(x), and B(∃x) are NOT wffs, NOR are B(R(x)), and B(∃x R(x)) . More examples: To express the fact that Tom is taller than John, we can use the atomic formula *taller***(Tom, John)**, which is a wff. This wff can also be part of some compound statement such as *taller***(Tom, John)** ∧¬*taller***(John, Tom)**, which is also a wff. *If x is a variable representing people in the world, then taller(x,Tom),* ∀*x taller(x,***Tom***),* ∃*x taller(x,***Tom***),* ∃*x* ∀*y taller(x,y)* are all wffs among others. However, ∃ *x*,John) and *taller*(Tom ∧Mary,   Jim),   for   example, are **NOT** wffs.

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	F	T

shows that $\neg p \vee q$ is equivalent to $p \to q$. 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 a 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 1 in 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 the set of all human beings.

Free & Bound variables:

Let's now turn to a rather important topic: the distinction between free variable s and **bound variable**s.

Have a look at the following formula:

$$\neg(\text{THERAPIST}(x) \lor \forall x(\text{MORON}(x) \land \forall y \text{PERSON}(y)))$$

The first occurrence of x is *free*, whereas the second and third occurrences of x are *bound*, namely by the first occurrence of the quantifier ∀. The first and second occurrences of the variable y are also bound, namely by the second occurrence of the quantifier ∀.

Informally, the concept of a *bound variable* can be explained as follows: Recall that quantifications are generally of the form:

$\forall x \varphi$

or

$\exists x \varphi$

where x may be any variable. Generally, all occurences of this variable within the quantification are bound. But we have to distinguish two cases. Look at the following formula to see why:

$\exists x(\text{MAN}(x) \wedge (\forall x \text{WALKS}(x)) \wedge \text{HAPPY}(x))$

1. x may occur within another, embedded, quantification $\forall x \Psi$ or $\exists x \Psi$, such as the x in $\text{WALKS}(x)$ in our example. Then we say that it is bound by the quantifier of this embedded quantification (and so on, if there's another embedded quantification over x within Ψ).
2. Otherwise, we say that it is bound by the top-level quantifier (like all other occurences of x in our example).

Here's a full formal simultaneous definition of *free* and *bound*:

1. Any occurrence of any variable is free in any atomic formula.
2. No occurrence of any variable is bound in any atomic formula.
3. If an occurrence of any variable is free in φ or in Ψ, then that same occurrence is free in $\neg \varphi$, $(\varphi \rightarrow \Psi)$, $(\varphi \vee \Psi)$, and $(\varphi \wedge \Psi)$.
4. If an occurrence of any variable is bound in φ or in Ψ, then that same occurrence is bound in $\neg \varphi$, $(\varphi \rightarrow \Psi)$, $(\varphi \vee \Psi)$, $(\varphi \wedge \Psi)$. Moreover, that same occurrence is bound in $\forall y \varphi$ and $\exists y \varphi$ as well, for any choice of variable y.
5. In any formula of the form $\forall y \varphi$ or $\exists y \varphi$ (where y can be any variable at all in this case) the occurrence of y that immediately follows the initial quantifier symbol is bound.
6. If an occurrence of a variable x is free in φ, then that same occurrence is free in $\forall y \varphi$ and $\exists y \varphi$, for any variable y distinct from x. On the other hand, all occurrences of x that are free in φ, are bound in $\forall x \varphi$ and in $\exists x \varphi$.

If a formula contains no occurrences of free variables we call it a ***sentence***.

Rules of inference:

The two rules of inference are called rules P and T.

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

Rule T: A formula S may be introduced in a derivation if s is tautologically implied by any one or more of the preceding formulas in the derivation.

Before proceeding the actual process of derivation, some important list of implications and equivalences are given in the following tables.

Implications

I1	$P \wedge Q \Rightarrow P$	} Simplification
I2	$PQ^\wedge \Rightarrow Q$	
I3	$P \Rightarrow PVQ$	} Addition
I4	$Q \Rightarrow PVQ$	
I5	$7P \Rightarrow P \rightarrow Q$	
I6	$Q \Rightarrow P \rightarrow Q$	
I7	$7(P \rightarrow Q) \Rightarrow P$	
I8	$7(P \rightarrow Q) \Rightarrow 7Q$	
I9	$P, Q \Rightarrow P \wedge Q$	
I10	$7P, PVQ \Rightarrow Q$	(disjunctive syllogism)
I11	$P, P \rightarrow Q \Rightarrow Q$	(modus ponens)
I12	$7Q, P \rightarrow Q \Rightarrow 7P$	(modus tollens)
I13	$P \rightarrow Q, Q \rightarrow R \Rightarrow P \rightarrow R$	(hypothetical syllogism)
I14	$P V Q, P \rightarrow Q, Q \rightarrow R \Rightarrow R$	(dilemma)

Equivalences

E1	$77P \Longleftrightarrow P$	
E2	$P \wedge Q \Longleftrightarrow Q \wedge P$	} Commutative laws
E3	$P V Q \Longleftrightarrow Q V P$	
E4	$(P \wedge Q) \wedge R \Longleftrightarrow P \wedge (Q \wedge R)$	} Associative laws
E5	$(P V Q) V R \Longleftrightarrow PV (Q V R)$	
E6	$P \wedge (Q V R) \Longleftrightarrow (P \wedge Q) V (P \wedge R)$	} Distributive laws
E7	$P V (Q \wedge R) \Longleftrightarrow (P V Q) \wedge (PVR)$	
E8	$7(P \wedge Q) \Longleftrightarrow 7P V7Q$	
E9	$7(P V Q) \Longleftrightarrow 7P \wedge 7Q$	} De Morgan's laws
E10	$P V P \Longleftrightarrow P$	
E11	$P \wedge P \Longleftrightarrow P$	

E12   R V (P ∧ 7P) <=>R

E13   R ∧ (P V 7P) <=>R

E14   R V (P V 7P) <=>T

E15   R ∧ (P ∧ 7P) <=>F

E16   P → Q   <=> 7P V Q

E17   7 (P→ Q) <=> P ∧ 7Q

E18   P → Q   <=> 7Q → 7P

E19   P → (Q → R)   <=> (P ∧ Q) → R

E20   7(PD Q) <=> P D 7Q

E21   PDQ <=> (P → Q) ∧ (Q → P)

E22   (PDQ) <=> (P ∧ Q) V (7 P ∧ 7Q)

Example 1.Show that R is logically derived from P → Q, Q → R, and P

Solution.   {1}   (1)   P → Q   Rule P

              {2}   (2)   P   Rule P

              {1, 2}   (3)   Q   Rule (1), (2) and I11

              {4}   (4)   Q → R   Rule P

              {1, 2, 4}   (5)   R   Rule (3), (4) and I11.

Example 2.Show that S V R tautologically implied by (P V Q) ∧ (P → R) ∧ (Q → S).

Solution .   {1}   (1)   P V Q   Rule P

              {1}   (2)   7P → Q   T, (1), E1 and E16

              {3}   (3)   Q → S   P

              {1, 3}   (4)   7P → S   T, (2), (3), and I13

              {1, 3}   (5)   7S → P   T, (4), E13 and E1

              {6}   (6)   P → R   P

              {1, 3, 6}   (7)   7S → R   T, (5), (6), and I13

              {1, 3, 6)   (8)   S V R   T, (7), E16 and E1

Example 3. Show that 7Q, P→ Q => 7P

Solution .   {1}   (1) P → Q   Rule P

                  {1}   (2) 7P → 7Q   T, and E 18

$\{3\}$ (3) 7Q   P

$\{1, 3\}$ (4) 7P   T, (2), (3), and I11 .

Example 4 .Prove that R $\wedge$ (P V Q) is a valid conclusion from the premises PVQ ,
Q $\rightarrow$ R, P $\rightarrow$ M and 7M.

Solution . $\{1\}$ (1)   P $\rightarrow$ M   P

        $\{2\}$ (2) 7M   P

        $\{1, 2\}$ (3) 7P   T, (1), (2), and I12

        $\{4\}$ (4)   P V Q   P

        $\{1, 2 , 4\}$ (5)   Q   T, (3), (4), and I10.

        $\{6\}$ (6)   Q $\rightarrow$ R   P

        $\{1, 2, 4, 6\}$ (7)   R   T, (5), (6) and I11

        $\{1, 2, 4, 6\}$ (8)   R $\wedge$ (PVQ)   T, (4), (7), and I9.

There is a third inference rule, known as rule CP or rule of *conditional proof*.

Rule CP: If we can derives s from R and a set of premises , then we can derive R $\rightarrow$ S from the set of premises alone.

Note.   1. Rule CP follows from the equivalence E10 which states that

     (P $\wedge$ R) $\rightarrow$ S óP $\rightarrow$ (R $\rightarrow$ S).

    2. Let P denote the conjunction of the set of premises and let R be any formula The
       above equivalence states that if R is included as an additional premise and
       S is derived from P $\wedge$ R then R $\rightarrow$ S can be derived from the premises P alone.

    3. Rule CP is also called the *deduction theorem* and is generally used if the
       conclusion is of the form R $\rightarrow$ S. In such cases, R is taken as an
       additional premise and S is derived from the given premises and R.

Example 5 .Show that R $\rightarrow$ S can be derived from the premises
P $\rightarrow$ (Q $\rightarrow$ S), 7R V P , and Q.

Solution.

{1}	(1) 7R V P	P
{2}	(2) R	P, assumed premise
{1, 2}	(3) P	T, (1), (2), and I10
{4}	(4) P → (Q → S)	P
{1, 2, 4}	(5) Q → S	T, (3), (4), and I11
{6}	(6) Q	P
{1, 2, 4, 6}	(7) S	T, (5), (6), and I11
{1, 4, 6}	(8) R → S	CP.

Example 6.Show that P → S can be derived from the premises, 7P V Q, 7Q V R, and R → S .

Solution.

{1}	(1) 7P V Q	P
{2}	(2) P	P, assumed premise
{1, 2}	(3) Q	T, (1), (2) and I11
{4}	(4) 7Q V R	P
{1, 2, 4}	(5) R	T, (3), (4) and I11
{6}	(6) R → S	P
{1, 2, 4, 6}	(7) S	T, (5), (6) and I11
{2, 7}	(8) P → S	CP

Example 7. " If there was a ball game , then traveling was difficult. If they arrived on time, then traveling was not difficult. They arrived on time. Therefore, there was no ball game". Show that these statements constitute a valid argument.

Solution.   Let   P: There was a ball game

                   Q: Traveling was difficult.

                   R: They arrived on time.

Given premises are: P → Q, R → 7Q and R   conclusion is: 7P

{1}	(1) P → Q	P
{2}	(2) R → 7Q	P
{3}	(3) R	P
{2, 3}	(4) 7Q	T, (2), (3), and I11
{1, 2, 3}	(5) 7P	T, (2), (4) and I12

Consistency of premises:

Consistency

A set of formulas H1, H2, …, Hm is said to be consistent if their conjunction 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, at least 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.

A set of formulas H1, H2, …, Hm is inconsistent, if their conjunction implies a contradiction, that is H1^ H2^ … ^ Hm => R ^ 7R

Where R is any formula. Note that R ^ 7R is a contradiction and it is necessary and sufficient that H1, H2, …,Hm are inconsistent the formula.

Indirect method of proof

In order to show that a conclusion C follows logically from the premises H1, H2,…, Hm, we assume that C is false and consider 7C as an additional premise. If the new set of premises is inconsistent, so that they imply a contradiction, then the assumption that 7C is true does not hold simultaneously with H1^ H2^ ..… ^ Hm being true. Therefore, C is true whenever H1^ H2^ ..… ^ Hm is true. Thus, C follows logically from the premises H1, H2 ….., Hm.

Example 8 Show that 7(P ^ Q) follows from 7P^ 7Q.

Solution.

We introduce $77(P \wedge Q)$ as an additional premise and show that this additional premise leads to a contradiction.

{1}	(1)	$77(P \wedge Q)$	P assumed premise
{1}	(2)	$P \wedge Q$	T, (1) and E1
{1}	(3)	P	T, (2) and I1
{1}	(4)	$7P \wedge 7Q$	P
{4}	(5)	7P	T, (4) and I1
{1, 4}	(6)	$P \wedge 7P$	T, (3), (5) and I9

Here (6) $P \wedge 7P$ is a contradiction. Thus {1, 4} viz. $77(P \wedge Q)$ and $7P \wedge 7Q$ leads to a contradiction $P \wedge 7P$.

Example 9 Show that the following premises are inconsistent.

1. If Jack misses many classes through illness, then he fails high school.
2. If Jack fails high school, then he is uneducated.
3. If Jack reads a lot of books, then he is not uneducated.
4. Jack misses many classes through illness and reads a lot of books.

Solution.

P: Jack misses many classes.

Q: Jack fails high school.

R: Jack reads a lot of books.

S: Jack is uneducated.

The premises are $P \rightarrow Q$, $Q \rightarrow S$, $R \rightarrow 7S$ and $P \wedge R$

{1}	(1)	$P \rightarrow Q$	P
{2}	(2)	$Q \rightarrow S$	P
{1, 2}	(3)	$P \rightarrow S$	T, (1), (2) and I13
{4}	(4)	$R \rightarrow 7S$	P
{4}	(5)	$S \rightarrow 7R$	T, (4), and E18
{1, 2, 4}	(6)	$P \rightarrow 7R$	T, (3), (5) and I13
{1, 2, 4}	(7)	$7P \vee 7R$	T, (6) and E16
{1, 2, 4}	(8)	$7(P \wedge R)$	T, (7) and E8

{9}   (9)P∧ R   P
{1, 2, 4, 9) (10)   (P∧ R) ∧ 7(P∧ R)   T, (8), (9) and I9

The rules above can be summed up in the following table. The "Tautology" column shows how to interpret the notation of a given rule.

Rule of inference	Tautology	Name
$\dfrac{p}{\therefore\ p \vee q}$	$p \rightarrow (p \vee q)$	Addition
$\dfrac{p \wedge q}{\therefore\ p}$	$(p \wedge q) \rightarrow p$	Simplification
$\dfrac{\begin{array}{c}p\\ q\end{array}}{\therefore\ p \wedge q}$	$((p) \wedge (q)) \rightarrow (p \wedge q)$	Conjunction
$\dfrac{\begin{array}{c}p\\ p \rightarrow q\end{array}}{\therefore\ q}$	$((p \wedge (p \rightarrow q)) \rightarrow q$	Modus ponens
$\dfrac{\begin{array}{c}\neg q\\ p \rightarrow q\end{array}}{\therefore\ \neg p}$	$((\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$	Modus tollens
$\dfrac{\begin{array}{c}p \rightarrow q\\ q \rightarrow r\end{array}}{\therefore\ p \rightarrow r}$	$((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$	Hypothetical syllogism
$\dfrac{\begin{array}{c}p \vee q\\ \neg p\end{array}}{\therefore\ q}$	$((p \vee q) \wedge \neg p) \rightarrow q$	Disjunctive syllogism
$\dfrac{\begin{array}{c}p \vee q\\ \neg p \vee r\end{array}}{\therefore\ q \vee r}$	$((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$	Resolution

Example 1

Let us consider the following assumptions: "If it rains today, then we will not go on a canoe today. If we do not go on a canoe trip today, then we will go on a canoe trip tomorrow. Therefore (Mathematical symbol for "therefore" is $\therefore$), if it rains today, we will go on a canoe trip tomorrow. To make use of the rules of inference in the above table we let p be the proposition "If it rains today", q be " We will not go on a canoe today" and let r be "We will go on a canoe trip tomorrow". Then this argument is of the form:

$$p \rightarrow q$$
$$q \rightarrow r$$
$$\therefore \overline{p \rightarrow r}$$

Example 2

Let us consider a more complex set of assumptions: "It is not sunny today and it is colder than yesterday". "We will go swimming only if it is sunny", "If we do not go swimming, then we will have a barbecue", and "If we will have a barbecue, then we will be home by sunset" lead to the conclusion "We will be home before sunset." Proof by rules of inference: Let p be the proposition "It is sunny this today", q the proposition "It is colder than yesterday", r the proposition "We will go swimming", s the proposition "We will have a barbecue", and t the proposition "We will be home by sunset". Then the hypotheses become $\neg p \wedge q, r \rightarrow p, \neg r \rightarrow s$ and $s \rightarrow t$. Using our intuition we conjecture that the conclusion might be t. Using the Rules of Inference table we can proof the conjecture easily:

Step	Reason
1. $\neg p \wedge q$	Hypothesis
2. $\neg p$	Simplification using Step 1

3. $r \rightarrow p$ Hypothesis

4. $\neg r$ Modus tollens using Step 2 and 3

5. $\neg r \rightarrow s$ Hypothesis

6. s Modus ponens using Step 4 and 5

7. $s \rightarrow t$ Hypothesis

8. t Modus ponens using Step 6 and 7

Proof of contradiction:

The "Proof by Contradiction" is also known as reductio ad absurdum, which is probably Latin for "reduce it to something absurd".

Here's the idea:

1. Assume that a given proposition is untrue.
2. Based on that assumption reach two conclusions that contradict each other.

This is based on a classical formal logic construction known as Modus Tollens: If P implies Q and Q is false, then P is false. In this case, Q is a proposition of the form (R and not R) which is always false. P is the negation of the fact that we are trying to prove and if the negation is not true then the original proposition must have been true. If computers are not "not stupid" then they are stupid. (I hear that "stupid computer!" phrase a lot around here.)

Example:

Lets prove that there is no largest prime number (this is the idea of Euclid's original proof). Prime numbers are integers with no exact integer divisors except 1 and themselves.

1. To prove: "There is no largest prime number" by contradiction.

2. Assume: There is a largest prime number, call it p.

3. Consider the number N that is one larger than the product of all of the primes smaller than or equal to p. N=1*2*3*5*7*11...*p + 1. Is it prime?

4. N is at least as big as p+1 and so is larger than p and so, by Step 2, cannot be prime.

5. On the other hand, N has no prime factors between 1 and p because they would all leave a remainder of 1. It has no prime factors larger than p because Step 2 says that there are no primes larger than p. So N has no prime factors and therefore must itself be prime (see note below).

We have reached a contradiction (N is not prime by Step 4, and N is prime by Step 5) and therefore our original assumption that there is a largest prime must be false.

Note: The conclusion in Step 5 makes implicit use of one other important theorem: The Fundamental Theorem of Arithmetic: Every integer can be uniquely represented as the product of primes. So if N had a composite (i.e. non-prime) factor, that factor would itself have prime factors which would also be factors of N.

Automatic Theorem Proving:

Automatic Theorem Proving (ATP) deals with the development of computer programs that show that some statement (the *conjecture*) is a *logical consequence* of a set of statements (the *axioms* and *hypotheses*). ATP systems are used in a wide variety of domains. For examples, a mathematician might prove the conjecture that groups of order two are commutative, from the axioms of group theory; a management consultant might formulate axioms that describe how organizations grow and interact, and from those axioms prove that organizational death rates decrease with age; a hardware developer might validate the design of a circuit by proving a conjecture that describes a circuit's performance, given axioms that describe the circuit itself; or a frustrated teenager might formulate the jumbled faces of a Rubik's cube as a conjecture and prove, from axioms that describe legal changes to the cube's configuration, that the cube can be rearranged to the solution state. All of these are tasks that can be performed by an ATP system, given an appropriate formulation of the problem as axioms, hypotheses, and a conjecture.

The **language** in which the conjecture, hypotheses, and axioms (generically known as *formulae*) are written is a logic, often classical 1st order logic, but possibly a non-classical logic and possibly a higher order logic. These languages allow a precise formal statement of the necessary information, which can then be manipulated by an ATP system. This formality is the underlying strength of ATP: there is no ambiguity in the statement of the problem, as is often the case when using a natural language such as English. Users have to describe the problem at hand precisely and accurately, and this process in itself can lead to a clearer understanding of the problem domain. This in turn allows the user to formulate their problem appropriately for submission to an ATP system.

The **proofs** produced by ATP systems describe how and why the conjecture follows from the axioms and hypotheses, in a manner that can be understood and agreed upon by everyone, even other computer programs. The proof output may not only be a convincing argument that the conjecture is a logical consequence of the axioms and hypotheses, it often also describes a process that may be implemented to solve some problem. For example, in the Rubik's cube example mentioned above, the proof would describe the sequence of moves that need to be made in order to solve the puzzle.

ATP systems are enormously powerful computer programs, capable of solving immensely difficult problems. Because of this extreme capability, their application and operation sometimes needs to be guided by an expert in the domain of application, in order to solve problems in a reasonable amount of time. Thus ATP systems, despite the name, are often used by domain experts in an interactive way. The interaction may be at a very detailed level, where the user guides the inferences made by the system, or at a much higher level where the user determines intermediate lemmas to be proved on the way to the proof of a conjecture. There is often a synergetic relationship between ATP system users and the systems themselves:

- The system needs a precise description of the problem written in some logical form,
- the user is forced to think carefully about the problem in order to produce an appropriate formulation and hence acquires a deeper understanding of the problem,
- the system attempts to solve the problem,
- if successful the proof is a useful output,

- if unsuccessful the user can provide guidance, or try to prove some intermediate result, or examine the formulae to ensure that the problem is correctly described,
- and so the process iterates.

ATP is thus a **technology** very suited to situations where a clear thinking domain expert can interact with a powerful tool, to solve interesting and deep problems. Potential ATP users need not be concerned that they need to write an ATP system themselves; there are many ATP systems readily available for use.

UNIT- II
Relations

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)}
Definition
   The **domain** D of a binary relation S is the set of all first elements of the ordered pairs in the relation.(i.e) D(S)= {a / $ b for which **(a, b) Є S}**
   The **range** R of a binary relation S is the set of all second elements of the
   ordered **pairs in the relation.(i.e) R(S) = {b / $ a for which (a, b) Є** S}

   For example
For the relation S = {(1, 2), (3, a), (b, a) ,(b, Joe)}
   D(S) = {1, 3, b, b}   and
   R(S) = {2, a, a, Joe}
Let X and Y be any two sets. A subset of the Cartesian product X * Y defines a relation, say C. For any such relation C, we have D(C) Í X and R(C) Í Y, and the relation C is said to from X

to Y. If Y = X, then C is said to be a relation form X to X. In such case, c is called a relation in X. Thus any relation in X is a subset of X * X . The set X * X is called a *universal relation* in X, while the empty set which is also a subset of X * X is called a *void relation* in X.

For example

  Let L denote the relation "less than or equal to" and D denote the relation
  "divides" where x D y means " x divides y". Both L and D are defined on the
 set {1, 2, 3, 4}
L = {(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3), (2, 4), (3, 3), (3, 4), (4, 4)} D = {(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 4), (3, 3), (4, 4)}
L Ç D = {(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 4), (3, 3), (4, 4)} = D

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

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

Definition: A relation R in a set X is **symmetric** if for every x and y in X, whenever **x R y, then y R x.(i.e) R is symmetric in X ó (x) (y) (x Є X ^ y Є X ^ x R y ® y R x}**

For example

- The relation equality of set is symmetric.
- The relation of similarity in the set of triangles in a plane is symmetric.
- The relation of being a sister is not symmetric in the set of all people.
- However, in the set females it is symmetric.

Definition: A relation R in a set X is   **transitiv**e if, for every x, y, and z are in X, R is whenever x R y and y R z , then x R z. (i.e) transitive in X ó **(x) (y) (z) (x Є X^ y Є X^ z Є**

X ∧ **x** **R** **y**∧ **y** **R** **z**® **x** **R** **z)**

For example

- The relations $<, £, >, ³$ and $=$ are transitive in the set of real numbers
- The relations Í, Ì, Ê, É and equality are also transitive in the family of sets.
- The relation of similarity in the set of triangles in a plane is transitive.

Definition: A relation R in a set X is **irreflexive if, for every x Є X , (x, x)ÏX.**

For example

- The relation $<$ is irreflexive in the set of all real numbers.
- The relation proper inclusion is irreflexive in the set of all nonempty subsets of a universal set.
- Let $X = \{1, 2, 3\}$ and $S = \{(1, 1), (1, 2), (3, 2), (2, 3), (3, 3)\}$ is neither irreflexive nor reflexive.

Definition: A relation R in a set x is **anti symmetric** if , for every x and y in X, whenever x R y and y R x, then x = y. **Symbolically,(x) (y) (x Є X ∧ y Є X ∧ x R y ∧ y R x ®** x = y)

For example

- The relations £ , ³ and = are anti symmetric
- The relation Í is anti symmetric in set of subsets.
- The relation **"divides" is anti symmetric in set of real numbers.**
- **Consider the relation "is a son of" on the male children in a family.Evidently the relation** is not symmetric, transitive and reflexive.
- **The relation " is a divisor of " is reflexive and transitive but no**t symmetric on the set of natural numbers.
- **Consider the set H of all human beings. Let r be a relation " is married to "** R is symmetric.
- Let I be the set of integers. R on I is defined as a R b if a – b is an even number.R is an reflexive, symmetric and transitive.

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 Є A, 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   have   to   show   that   relation   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, then 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 R b and b R c => a R c.

Hence R is an equivalence relation.

Problem 2: Let x = {1, 2, 3, … 7} and R = {(x, y) / x − y is divisible by 3} Show that R is an equivalence relation.

Solution: For any a Є X, a- a is divisible by
3, Hence a R a, R is reflexive
For any a, b Є X, 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 by 3
and b–c is divisible by 3. 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 the set of all integers. Let m be a fixed integer. Two integers a and
b are said to be congruent modulo m if and only if m divides a-b, in which case we write a °

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

Solution :

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

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

- a R b and b R c => a ºb (mod m) and bº c (mod m)
    o m divides a – b and m divides b-c
    o a – b = km and b – **c = lm for some k ,l Є z**
    o (a – b) + (b – c) = km + lm
    o a – c = (k +l) m
    o aº c (mod m)
    o a R c
    o R is transitive

Hence the congruence relation is an equivalence relation.

Equivalence Classes:

Let R be an equivalence relation on a set A. For any **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 the set of integer and R be the relation **called "congruence modulo 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 to Y and S be a relation from Y to Z. Then the relation R

o S is given by R o S = {(x, z) / x Î X Ù z Î Z Ù y Î Y such that (x, y) Î R Ù (y, z) Î S)} is called the composite relation of R and S.

The operation of obtaining R o 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 R o S = {(1, 5), (3, 2), (2, 5)} and S o R = {(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 R as R2; R o R o R as R3 and so 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) Î R} 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+ = R U R2 U R3…in X. is called the *transitive closure* of R in X**

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

Thus, R+ = R U R2 U R3 U R4 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.

Partial Ordering Relations:

Definition

A binary relation R in a set P is called ***partial order relation*** or *partial ordering* 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 the set of real numbers. The relation "less than or equal to " or**
  - , is a partial ordering on R.
- Let X be a set and r(X) be its power set. The relation subset, Í on X is partial ordering.
- **Let Sn be the set of divisors of n. The relation D means "divides" on Sn ,**is partial ordering on Sn .

In a partially ordered set (P, £), an element y Î P is said to cover an element x Î P if x <y and if there does not exist any element z Î P such that x £ z and z £ y; that is, y covers x Û (x < y Ù (x £ z £ y Þ x = z Ú z = y))

A partial order relation £ on a set P can be represented by means of a diagram known as a Hasse diagram or partial order set diagram of (P, £). In such a diagram, each element is represented by a small circle or a dot. The circle for x Î P is drawn below the circle for y Î P if x < y, and a line is drawn between x and y if y covers x.

If x < y but y does not cover x, then x and y are not connected directly by a single line. However, they are connected through one or more elements of P.

Hasse Diagram:

A Hasse diagram is a digraph for a poset which does not have loops and arcs implied by the transitivity.

Example 10: For the relation {< a, a >, < a, b >, < a, c >, < b, b >, < b, c >, < c, c >} on set {a, b,c}, the Hasse diagram has the arcs {< a, b >, < b, c >} as shown below.
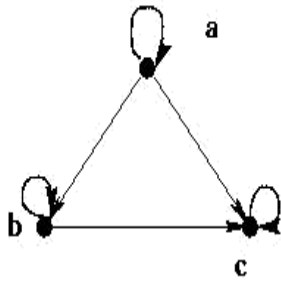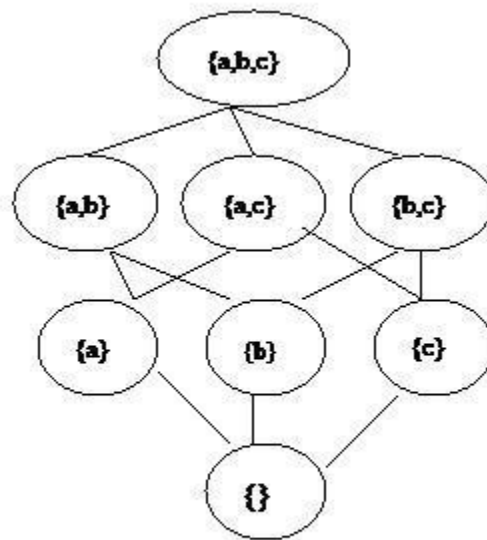
Digraph for Partial Order
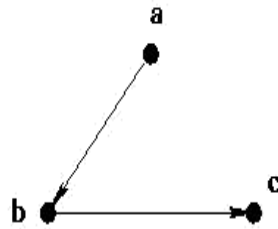
Hasse Diagram

Ex: Let A be a given finite set and r(A) its power set. Let Í be the subset relation on the elements of r(A). Draw Hasse diagram of (r(A), Í) for A = {a, b, c}

Functions

Introduction

A function is a special type of relation. It may be considered as a relation in which each element of the domain belongs to only one ordered pair in the relation. Thus a function from A **to B is a subset of A X B having the property that for each a ЄA, there is one and only one b Є B** such that (a, b) Î G.

Definition

Let A and B be any two sets. A relation f from A to B is called a function **if for every a Є A there is a unique b Є B such that (a, b) Є f .**

Note that the definition of function requires that a relation must satisfy two additional conditions in order to qualify as a function.

The first condition is that every a Є A must be related to some b Є B, (i.e) the domain of f must be A and not merely subset of A. The second requirement of uniqueness can be expressed **as (a, b) Є f ^ (b, c) Є f => b = c**
Intuitively, a function from a set A to a set B is a rule which assigns to every element of A, a unique element of B. **If a ЄA, then the unique element of B assigned to a under f is denoted by f** (a).The usual notation for a function f from A to B is f: A® B defi**ned by a ® f (a) where a Є A,** f(a) is called the image of **a** under f and **a** is called pre image of f(a).

- Let X = Y = **R** and f(x) = x2 + 2. Df = R and Rf Í R.

- Let X be the set of all statements in logic and let Y = {True,

 False}. A mapping f: X®Y is a function.

- A program written in high level language is mapped into a machine language by a compiler. Similarly, the output from a compiler is a function of its input.

- Let X = Y = **R** and f(x) = x2 is a function from X ® Y,and g(x2) = x is not a function from X ® Y.

 A mapping f: A ® B is called **one-to-one** (injective or 1 −1) if distinct elements of A are mapped into distinct elements of B. (i.e) f is one-to-one if
 a1 = a2 => f (a1) = f(a2) or equivalently f(a1) ¹ f(a2) => a1 ¹ a2
For example, f: N ® N given by f(x) = x is 1-1 where N is the set of a natural numbers.
 A mapping f: A® B is called **onto (surjective) if for every b Є B there is an a Є A such** that f (a) = B. i.e. if every element of B has a pre-image in A. Otherwise it is called **into**.

For example, f: Z®Z given by f(x) = x + 1 is an onto mapping.
A mapping is both 1-1 and onto is called bijective

.

For example f: R®R given by f(x) = X + 1 is bijective.

Definition: A mapping f: R® b is called a **constant mapping** if, for all aÎA, f (a) =
     b, a fixed element.
For example f: Z®Z given by f(x) = 0, for all x ÎZ is a constant mapping.

Definition

     A mapping f: A®A is called the **identity mapping** of A if f (a) = a, for all aÎA. Usually it
 is denoted by IA or simply I.

Composition of functions:

    If f: A®B and g: B®C are two functions, then the composition of functions f and g, denoted
by g o f, is the function is given by g o f : A®C and is given by
     **g o f = {(a, c) / a Є A ∧ c Є C ∧ $bÎ B ': f(a)= b ∧ g(b) =
c}** and (g of)(a) = ((f(a))

Example 1: Consider the sets A = {1, 2, 3},B={a, b} and C = {x,
   y}. Let f: A® B be defined by f (1) = a ; f(2) = b and f(3)=b and
   Let g: B® C be defined by g(a) = x and g(b) = y
    (i.e) f = {(1, a), (2, b), (3, b)} and g = {(a, x), (b,
y)}. Then g o f: A®C is defined by
        (g of) (1) = g (f(1)) = g(a) =   x
        (g o f) (2) = g (f(2)) = g(b) =   y
        (g o f) (3) = g (f(3)) = g(b) = y
    i.e., g o f = {(1, x), (2, y),(3, y)}

If f: A® A and g: A®A, where A= {1, 2, 3}, are given by
     f = {(1, 2), (2, 3), (3, 1)} and g = {(1, 3), (2, 2), (3, 1)} Then
g of = {(1, 2), (2, 1), (3, 3)}, fog= {(1, 1), (2, 3), (3, 2)}
     f of = {(1, 3), (2, 1), (3, 2)} and gog= {(1, 1), (2, 2), (3, 3)}

Example 2: Let f(x) = x+2, g(x) = x – 2 and h(x) = 3x for x Î R, where R is the set of real
 numbers.
            Then f o f = {(x, x+4)/xÎ R} f
                o g = {(x, x)/ x Î X} g
                o f = {(x, x)/ xÎ X}
                g o g = {(x, x-4)/x Î X}

$$h \; o \; g = \{(x,3x-6)/ \; x \; Î \; X\} \; h$$
$$o \; f = \{(x, \; 3x+6)/ \; x \; Î \; X\}$$

Inverse functions:

Let f: A® B be a one-to-one and onto mapping. Then, its inverse, denoted by f -1 is given by f -1 = {(b, a) / (a, b) Î f} Clearly f-1: B® A is one-to-one and onto.

Also   we   observe   that   f   o   f   -1   =   IB   and   f   -1o   f   =   IA.
If f -1   exists then f is called **invertible**.

*For example:L*et f: R ®R be defined by f(x) = x + 2
Then f -1: R® R is defined by f -1(x) = x - 2

Theorem: Let f: X ®Y and g: Y ® Z be two one to one and onto functions. Then gof is also one to one and onto function.

Proof

Let f:X ® Y g : Y ® Z be two one to one and onto functions. Let x1, x2 Î X

- g o f (x1) = g o f(x2),
- g (f(x1)) = g(f(x2)),
- g(x1) = g(x2) since [f is 1-1]

x1 = x2 since [g is 1-1}
        so that gof is 1-1.

   By the definition of composition, gof : X ® Z is a function.
   We have to prove that every element of z Î Z an image element for some x Î X
    under gof.
  Since g is onto $ y ÎY ': g(y) = z   and   f is onto from X to Y,
                $ x ÎX   ': f(x) = y.
    Now, gof (x) = g (f (x))
                 = g(y)   [since f(x) = y]
                 = z [since g(y) = z]
                  which shows that gof is onto.

Theorem   (g o f) -1 = f -1 o g -1

      (i.e) the inverse of a composite function can be expressed in terms of the composition of the inverses in the reverse order.

Proof.

      f: A ® B is one to one and onto.

      g: B ® C is one to one and onto.

      gof: A ® C is also one to one and onto. Þ

      (gof) -1: C ® A is one to one and onto.

      Let a Î A, then there exists an element b Î b such that f (a) = b Þ a = f-1 (b). Now b Î B Þ there exists an element c Î C such that g (b) = c Þ b = g -1(c). Then (gof)(a) = g[f(a)] = g(b) = c Þ a = (gof) -1(c). **…….(1)** (f -1 o g-1) (c) = f -1(g -1 (c)) = f -1(b) = a Þ a = (f -1 o g -**1)(c)** **….(2)** Combining (1) and (2), we have

      (gof) -1 = f -1 o g -1

Theorem: If f: A ® B is an invertible mapping , then

      f o f -1 = I B and f-1 o f = IA

Proof: f is invertible, then f -1 is defined by f(a) = b ó f-1(b) =

      a where a Î A and bÎ B .

      Now we have to prove that f of -1 = IB

      . Let bÎ B and f -1(b) = a, a Î A

      then fof-1(b) = f(f-1(b))

               = f(a) = b

      therefore f o f -1 (b) = b " b Î B => f o f -1 = IB

      Now f -1 o f(a) = f -1 (f(a)) = f -1 (b) = a

      therefore f -1 o f(a) = a " a Î A => f -1 o f = IA.

      Hence the theorem.

Recursive Functions:

The term "recursive function" is often used informally to describe any function that is defined with recursion. There are several formal counterparts to this informal definition, many of which only differ in trivial respects.

Kleene (1952) defines a "partial recursive function" of nonnegative integers to be any function f that is defined by a noncontradictory system of equations whose left and right sides are composed from (1) function symbols (for example, f, g, h, etc.), (2) variables for nonnegative integers (for example, x, y, z, etc.), (3) the constant 0, and (4) the successor function $S(x) = x + 1$.

For example,

$$f(x, 0) = 0 \tag{1}$$

$$f(x, S(y)) = g(f(x, y), x) \tag{2}$$

$$g(x, 0) = x \tag{3}$$

$$g(x, S(y)) = S(g(x, y)) \tag{4}$$

defines $f(x, y)$ to be the function $x \cdot y$ that computes the product of x and y.

Note that the equations might not uniquely determine the value of f for every possible input, and in that sense the definition is "partial." If the system of equations determines the value of f for every input, then the definition is said to be "total." When the term "recursive function" is used alone, it is usually implicit that "total recursive function" is intended. Note that some authors use the term "general recursive function to mean partial recursive function, although others use it to mean "total recursive function."

The set of functions that can be defined recursively in this manner is known to be equivalent to the set of functions computed by Turing machines and by the lambda calculus.

Lattice and its Properties:

Introduction:

A lattice is partially ordered set (L, £) in which every pair of elements a, b ÎL has a greatest lower bound and a least upper bound.
The glb of a subset, {a, b} Í L will be denoted by a * b and the lub by a Å b.
.
Usually, for any pair a, b Î L, GLB {a, b} = a * b, is called the **meet** or **product** and LUB{a, b} = a Å b, is called the **join** or **sum** of a and b.

Example1 Consider a non-**empty set S and let P(S) be its power set. The relation Í "contained in" is a partial ordering on P(S). For any two subsets A, BÎ P(S)** GLB {A, B} and LUB {A, B} are evidently A Ç B and A È B respectively.

Example2 Let I+ be the set of positive integers, and D denote the relation of **"division"** in I+ such that for any a, b Î I+ , a D b iff a divides b. Then (I+, D) is a lattice in which the join of a and b is given by the least common multiple(LCM) of a and b, that is, a Å b = LCM of a and b, and the meet of a and b, that is , a * b is the greatest common divisor (GCD) of a and b.

A lattice can be conveniently represented by a diagram.
For example, let Sn be the set of all divisors of n, where n is a positive integer. Let D denote the

relation "division" such that for any a, b Î Sn, a D b iff a divides b.

Then (Sn, D) is a lattice with a * b = gcd(a, b) and a Å b = lcm(a, b).

Take n=6. Then S6 = {1, 2, 3, 6}. It can be represented by a diagram in Fig(1).

Take n=8. Then S8 = {1, 2, 4, 8}

Two lattices can have the same diagram. For example if S = {1, 2, 3} then (p(s), Í) and (S6,D) have the same diagram viz. fig(1), but the nodes are differently labeled . We observe that for any partial ordering relation £ on a set S the converse relation ³ is also partial ordering relation on S. If (S, £) is a lattice With meet a * b and join a Å b , then (S, ³) is the also a lattice with meet a Å b and join a * b i.e., the GLB and LUB get interchanged . Thus we have the principle of duality of lattice as follows.

Any statement about lattices involving the operations ^ and V and the relations £ and ³ remains true if ^, V, ³ and £ are replaced by V, ^, £ and ³ respectively.

The operation ^ and V are called duals of each other as are the relations £ and ³.. Also, the lattice (L, £) and (L, ³) are called the duals of each other.

Properties of lattices:

Let (L, £) be a lattice with the binary operations * and Å then for any a, b, c Î L,

- a * a = a   a Ă a = a   (Idempotent)
- a * b = b * a   ,   a Å b = b Å a   (Commutative)
- (a * b) * c = a * (b * c) , (a Å) Å c = a Å (b Å c)

                                          o (Associative)

- a * (a Å b) = a   ,   a Å (a * b) = a   (absorption)

For any a ÎL, a £ a, a £ LUB {a, b} => a £ a * (a Å b). On the other hand, GLB {a, a Å b} £ a i.e., (a Å b) Å a, hence a * (a Å b) = a

Theorem 1

Let (L, £) be a lattice with the binary operations * and Å denote the operations of meet and join respectively For any a, b Î L,

      a £ b ó a * b = a ó a Å b = b

Proof

Suppose that a £ b. we know that a £ a, a £ GLB {a, b}, i.e., a £ a * b.

But from the definition of a * b, we get a * b £ a.

Hence a £ b => a * b = a **…………………………**   (1)

Now we assume that a * b = a; but is possible only if a £ b,

that is a * b = a => a £ b ………………………   (2)

From (1) and (2), we get a £ b ó a * b = a.

Suppose a * b = a.

then b Å (a * b) = b Å a = **a Å b** …………………….   (3)

but b Å (a * b) = b (by iv)……………………..   (4)

Hence a Å b = b, from (3) => (4)

Suppose aÅ b = b, i.e., LUB {a, b} = b, this is possible only if a£ b, thus(3) => (1)

(1) => (2) => (3) => (1). Hence these are equivalent.

Let us assume a * b = a.

Now (a * b) Å b = a Å b

We know that by absorption law , (a * b) Å b = b

so that a Å b = b, therefore a * b = a Þ a Å b = b   (5)

similarly, we can prove a Å b = b Þ   a * b = a   (6)

From (5) and (6), we get

a * b = a Û a Å b = b

Hence the theorem.

Theorem2 For any a, b, c Î L, where (L, £) is a lattice. b

       £ c => { a * b £ a * c and

           { a Å b £ a Å c

Proof   Suppose b £ c. we have proved that b £ a ó b * c = b…….. (1)

   Now consider

   (a * b) * (a * c) = (a * a) * (b * c)   (by Idempotent)

                = a * (b * c)

                = a * b   (by (1))

 Thus (a * b) * (a * c) = a * b which => (a * b) £ (a * c)

 Similarly (a Å b) Å (a Å c) = (a Å a) Å (b Å c)

                   = a Å (b Å c)

                 = a Å c

which => (a Å b) £ (a Å c)

note:These properties are known as **isotonicity.**

UNIT-III
Algebraic structures

Algebraic systems:

An algebraic system, loosely speaking, is a set, together with some operations on the set. Before formally defining what an algebraic system is, let us recall that a n -ary operation (or operator) on a set A is a function whose domain is A_n and whose range is a subset of A . Here, n is a non-negative integer. When $n=0$, the operation is usually called a nullary operation, or a constant, since one element of A is singled out to be the (sole) value of this operation. A finitary operation on A is just an n -ary operation for some non-negative integer n .

Definition. An *algebraic system* is an ordered pair (A,O) , where A is a set, called the underlying set of the algebraic system, and O is a set, called the operator set, of finitary operations on A .

We usually write A , instead of (A,O) , for brevity.

A prototypical example of an algebraic system is a group, which consists of the underlying set G , and a set O consisting of three operators: a constant e called the multiplicative identity, a unary operator called the multiplicative inverse, and a binary operator called the multiplication.

For a more comprehensive listing of examples, please see this entry.

Remarks.

- An algebraic system is also called algebra for short. Some authors require that A be non-empty.

  Note that A is automatically non-empty if O contains constants. A *finite algebra* is an algebra whose underlying set is finite.

- By definition, all operators in an algebraic system are finitary. If we allow O to contain infinitary operations, we have an *infinitary algebraic system*. Other generalizations are possible. For example, if the operations are allowed to be multivalued, the algebra is said to be a *multialgebra*. If the operations are not everywhere defined, we get a *partial algebra*. Finally, if more than one underlying set is involved, then the algebra is said to be *many-sorted*.

The study of algebraic systems is called the theory of universal algebra. The first important thing in studying algebraic system is to compare systems that are of the same ``type''. Two algebras are said to have the same *type* if there is a one-to-one correspondence between their operator sets such that an n -ary operator in one algebra is mapped to an n-ary operator in the other algebra.

Examples:

Some recurring universes: **N**=natural numbers; **Z**=integers; **Q**=rational numbers; **R**=real numbers; **C**=complex numbers.

N is a pointed unary system, and under addition and multiplication, is both the standard interpretation of Peano arithmetic and a commutative semiring.

Boolean algebras are at once semigroups, lattices, and rings. They would even be abelian groups if the identity and inverse elements were identical instead of complements.

Group-like structures

- Nonzero **N** under addition (+) is a magma.
- **N** under addition is a magma with an identity.
- **Z under subtraction (−) is a quasigroup.**
- Nonzero **Q** under division (÷) is a quasigroup.
- Every group is a loop, because $a * x = b$ if and only if $x = a^{-1} * b$, and $y * a = b$ if and only if $y = b * a^{-1}$.
- 2x2 matrices(of non-zero determinant) with matrix multiplication form a group.
- **Z** under addition (+) is an abelian group.
- Nonzero **Q** under multiplication (×) is an abelian group.
- Every cyclic group G is abelian, because if x, y are in G, then $xy = a^m a^n = a^{m+n} = a^{n+m} = a^n a^m = yx$. In particular, **Z** is an abelian group under addition, as is the integers modulo n **Z**/n**Z**.
- A monoid is a category with a single object, in which case the composition of morphisms and the identity morphism interpret monoid multiplication and identity element, respectively.
- The Boolean algebra **2** is a boundary algebra.

General Properties:

Property of Closure

If we take two *real numbers* and multiply them together, we get another real number. (The real numbers are all the rational numbers and all the irrational numbers.) Because this is always true, we say that the real numbers are "closed under the operation of multiplication": there is no way to escape the set. When you combine any two elements of the set, the result is also included in theset.

Real numbers are also closed under addition and subtraction. They are not closed under the square root operation, because the square root of -1 is not a real number.

Inverse

The inverse of something is that thing turned inside out or upside down. The inverse of an operation undoes the operation: division undoes multiplication.

A number's *additive inverse* is another number that you can add to the original number to get the additive identity. For example, the additive inverse of 67 is -67, because 67 + -67 = 0, the additive identity.

Similarly, if the product of two numbers is the *multiplicative identity,* the numbers are *multiplicative inverses.* Since 6 * 1/6 = 1 (the multiplicative identity), the multiplicative inverse of 6 is 1/6.

Zero does not have a multiplicative inverse, since no matter what you multiply it by, the answer is always 0, not 1.

Equality

The equals sign in an equation is like a scale: both sides, left and right, must be the same in order for the scale to stay in balance and the equation to be true.

The *addition property of equality* says that if a = b, then a + c = b + c: if you add the same number to (or subtract the same number from) both sides of an equation, the equation continues to be true.

The *multiplication property of equality* says that if a = b, then a * c = b * c: if you multiply (or divide) by the same number on both sides of an equation, the equation continues to be true.

The *reflexive property of equality* just says that a = a: anything is congruent to itself: the equals sign is like a mirror, and the image it "reflects" is the same as the original.

The *symmetric property of equality* says that if a = b, then b = a.

The *transitive property of equality* says that if a = b and b = c, then a = c.

Semi groups and monoids:

In the previous section, we have seen several algebraic system with binary operations. Here we consider an algebraic system consisting of a set and an associative binary operation on the set and then the algebraic system which possess an associative property with an identity element. These algebraic systems are called semigroups and monoids.

Semi group

Let S be a nonempty set and let * be a binary operation on S. The algebraic system (S, *) is called a semi-group if * is associative

if a * (b*c) = (a * b) * c for all a, b, c $\hat{I}$ S.

Example The N of natural numbers is a semi-group under the operation of usual addition of numbers.

Monoids

Let M be a nonempty set with a binary operation * defined on it. Then (M, *) is called a monoid if

- * is associative

(i.e) a * (b * c) = (a * b) * c for all a, b, c $\hat{I}$ M and

there exists an element e in M such that

a * e = e * a = a for all a $\hat{I}$ M
e is called the identity element in (M,*).

It is easy to prove that the identity element is unique. From the definition it follows that (M,*) is a semigroup with identity.

Example1 Let S be a nonempty set and r(S) be its power set. The algebras (r(S),U) and (r(S), Ç) are monoids with the identities f and S respectively.

Example2 Let N be the set of natural numbers, then (N,+), (N, X) are monoids with the identities 0 and 1 respectively.

Groups Sub Groups:

Recalling that an algebraic system (S, *) is a semigroup if the binary operation * is associative. If there exists an identity element e $\hat{I}$ S, then (S,*) is monoid. A further condition is imposed on the elements of the monoid, i.e., the existence of an inverse for each element of S then the algebraic system is called a group.

Definition

Let G be a nonempty set, with a binary operation * defined on it. Then the algebraic system (G,*) is called a **group** if

- * is associative i.e. a * (b * c) = (a * b) * c for all a, b, c,Î G.

- there exists an element e in G such that a * e = e * a = a for all a Î G

- for each a Î G there is an element denoted by a-1 in G such that

  a * a-1 = a-1 * a = e, a-1 is called the inverse of a.

From the definition it follows that (G,*) is a monoid in which each element has an inverse   w.r.t. * in G.

A group (G,*) in which * is commutative is called an abelian group or a commutative group. If * is not commutative then (G,*) is called a non-abelian group or non-commutative group.

The order of a group (G,*) is the number of elements of G, when G is finite and is denoted by o(G) or |G|

Examples 1. (Z5, +5) is an abelian group of order 5.
  2. G = {1, -1, i, -i} is an abelian group with the binary operation x is defined
    as 1 x 1 = 1, -1 x -1 = 1, i x i = -1 , -i x -i = **1, …**

Homomorphism of semigroups and monoids

Semigroup homomorphism.

Let (S, *) and (T, D) be any two semigroups. A mapping g: S ® T such that any two elements a, b Î S , g(a * b) = g(a) D g(b) is called a semigroup homomorphism.

Monoid homomorphism

Let (M, *,eM) and (T, D,eT) be any two monoids. A mapping g: M® T such that any two elements a, b Î M ,
  g(a * b) = g(a) D g(b) and
  g(eM) = eT
  is called a monoid homomorphism.

Theorem 1 Let (s, *) , (T, D) and (V, Å) be semigroups. A mapping g: S ® T and
  h: T ® V be semigroup homomorphisms. Then (hog): S ® V is a
  semigroup homomorphism from (S,*) to(V,Å).

Proof. Let a, b Î S. Then

$$(h \circ g)(a * b) = h(g(a * b))$$
$$= h(g(a) \, D \, g(b))$$
$$= h(g(a)) \, Å \, h(g(b))$$
$$= (h \circ g)(a) \, Å \, (h \circ g)(b)$$

Theorem 2 Let (s,*) be a given semigroup. There exists a homomorphism g: S ® SS, where (SS, o) is a semigroup of function from S to S under the operation of composition.

Proof   For any element a Î S, let g(a) = fa where f aÎ SS and f a is defined by

   f a(b) = a * b   for any a, bÎ S
   g(a * b) = f a*b

   Now   f a*b(c)   = (a * b) * c = a*(b * c)
   where   = f a(f b(c)) = (f a o f b) (c).
   Therefore, g(a * b) = f a*b = f a o f b = g(a) o g(b), this shows that g: S ® SS is a homomorphism.

Theorem 3 For any commutative monoid (M, *),the set of idempotent elements of M forms a submonoid.

Proof.   Let S be the set of idempotent elements of M.
         Since the identity element e Î M is idempotent, e Î S.
         Let a, b Î S, so that a* a = a and b * b = b
         Now (a * b) * (a * b) = (a * b) * (b * a)   [(M, *) is a commutative monoid]
                       = a * (b * b) * a
                       = a * b * a
                       = a * a * b
                       = a * b

Hence a * b Î S and (S, *) is a submonoid.

Isomorphism:

In abstract algebra, an **isomorphism** is a bijective map f such that both f and its inverse f^{-1} are homomorphisms, i.e., *structure-preserving* mappings. In the more general setting of category theory, an **isomorphism** is a morphism f: $X \rightarrow Y$ in a category for which there exists an "inverse" f^{-1}: $Y \rightarrow X$, with the property that both $f^{-1}f = \mathrm{id}_X$ and $f f^{-1} = \mathrm{id}_Y$.

Informally, an isomorphism is a kind of mapping between objects, which shows a relationship between two properties or operations. If there exists an isomorphism between two structures, we

call the two structures **isomorphic**. In a certain sense, isomorphic structures are **structurally identical**, if you choose to ignore finer-grained differences that may arise from how they are defined.

Purpose:

Isomorphisms are studied in mathematics in order to extend insights from one phenomenon to others: if two objects are isomorphic, then any property which is preserved by an isomorphism and which is true of one of the objects, is also true of the other. If an isomorphism can be found from a relatively unknown part of mathematics into some well studied division of mathematics, where many theorems are already proved, and many methods are already available to find answers, then the function can be used to map whole problems out of unfamiliar territory over to "solid ground" where the problem is easier to understand and work with.

Elementary Combinatorics

Basis of counting:

If X is a set, let us use |X| to denote the number of elements in X.

Two Basic Counting Principles

Two elementary principles act as "building blocks" for all counting problems. The first principle says that the whole is the sum of its parts; it is at once immediate and elementary.

<u>Sum Rule</u>: <u>The principle of disjunctive counting</u> :

If a set X is the union of disjoint nonempty subsets S1, ..., Sn, then | X | = | S1 | + | S2 | + ... + | Sn |.

We emphasize that the subsets S1, S2, ..., Sn must have no elements in common. Moreover, since X = S1 U S2 U ......U Sn, each element of X is in exactly one of the subsets Si. In other words, S1, S2, ..., Sn is a partition of X.

If the subse**ts S1, S2, ..., Sn were allowed to overlap, then a more profound**

principle will be needed--the principle of inclusion and exclusion.

Frequently, instead of asking for the number of elements in a set *perse*, some

problems ask for how many ways a certain event can happen.

The difference is largely in semantics, for if A is an event, we can let X be the set of ways

that A can happen and count the number of elements in X. Nevertheless, let us state the sum rule

for counting events.

If E1, ..., En are mutually exclusive events, and E1 can happen e1 ways, E2 happen e2

ways,.... ,En can happen en ways, E1 or E2 or .... or En can happen e1 + e2 + ... + en ways.

Again we emphasize that mutually exclusive events E1 and E2 mean that E1 or E2 can

happen but both cannot happen simultaneously.

The sum rule can also be formulated in terms of choices: If an object can be selected from

a reservoir in e1 ways and an object can be selected from a separate reservoir in e2 ways and an

object can be selected from a separate reservoir in e2 ways, then the selection of one

object from either one reservoir or the other can be made
 in e1 + e2 ways.

Product Rule: The principle of sequencing counting

If S1, ….., Sn are nonempty sets, then the number of elements in the Cartesian product
S1 x S2 x ….. x Sn is the product $\prod_{i=1}^{n}$ |S i |. That is,
$$| S1 \text{ x } S2 \text{ x } \ldots \ldots \text{ x } Sn | = \prod_{i=1}^{n} | S i |.$$

Observe that there are 5 branches in the first stage corresponding to the 5 elements of S1 and to each of these branches there are 3 branches in the second stage corresponding to the 3 elements of S2 giving a total of 15 branches altogether. Moreover, the Cartesian product S1 x S2 can be partitioned as (a1 x S2) U (a2 x S2) U (a3 x S2) U (a4 x S2) U (a5 x S2), where (ai x S2) = {(ai, b1), (ai i, b2), (ai, b3)}. Thus, for example, (a3 x S2) corresponds to the third branch in the first stage followed by each of the 3 branches in the second stage.

More generally, if a1,….., an are the n distinct elements of S1 and b1,….,bm are the m
distinct elements of S2, then S1 x S2 = U$_{i=1}^{n}$ (ai x S2).
For if x is an arbitrary element of S1 x S2 , then x = (a, b) where a Î S1 and b Î S2.
Thus, a = ai for some i and b = bj for some j. Thus, x = (ai, bj) Î(ai x S2) and
therefore x Î Un$_{i=1}$(ai x S2).
Conversely, if x Î U$_{i=1}^{n}$(ai x S2), then x Î (ai x S2) for some i and thus x = (ai, bj) where bj
is some element of S2. Therefore, x Î S1 x S2.

Next observe that (ai x S2) and (aj x S2) are disjoint if **i ≠ j since if**
 x Î (ai x S2) ∩ (aj x S2) then x = (ai, bk) for some k and x = (aj, b1) for some l.
But then (ai, bk) = (aj, bl) implies that ai = aj and bk = bl. But since i **≠ j , ai ≠ a j.**
Thus, we conclude that S1 x S2 is the disjoint union of the sets (ai x S2). Furthermore |ai x S2| = |S2| since there is obviously a one-to-one correspondence between the sets ai x S2 and
S2, namely, (ai, bj) → bj.

Then **by the sum rule $|S1 \times S2| = \sum_{i=1}^{n} | ai \times S2|$**

$$\textbf{7.(n summands)} \ |S2| + |S2| + \ldots + |S2|$$

8.n $|S2|$

9.nm.

Therefore, we have proven the product rule for two sets. The general rule follows by mathematical induction.

We can reformulate the product rule in terms of events. If events E1, E2 , …., En can happen e1, e2,…., and en ways, respectively, then the sequence of events E1 first, followed by **E2,…., followed by En can happen e1e2 …en ways.**

In terms of choices, the product rule is stated thus: If a first object can be chosen e1 ways, **a second e2 ways , …, and an nth object can be made in e1e2….en ways.**

Combinations & Permutations:

Definition.

A combination of n objects taken r at a time (called an r-combination of n objects) is an **unordered selection** of r of the objects.

A permutation of n objects taken r at a time (also called an r-permutation of n objects) is an **ordered selection** or **arrangement** of r of the objects.

Note that we are simply defining the terms r-combinations and r-permutations here and have not mentioned anything about the properties of the n objects.

For example, these definitions say nothing about whether or not a given element may appear more than once in the list of n objects.

In other words, it may be that the n objects do not constitute a set in the normal usage of the word.

SOLVED PROBLEMS

Example1. Suppose that the 5 objects from which selections are to be made are: a, a, a, b, c. then the 3-combinations of these 5 objects are : aaa, aab, aac, abc. The permutations are:

aaa, aab, aba, baa, aac, aca, caa,

abc, acb, bac, bca, cab, cba.

Neither do these definitions say anything about any rules governing the selection of the r-objects: on one extreme, objects could be chosen where all repetition is forbidden, or on the other extreme, each object may be chosen up to t times, or then again may be some rule of selection between these extremes; for instance, the rule that would allow a given object to be repeated up to a certain specified number of times.

We will use expressions like {3 . a , 2. b ,5.c} to indicate either

(1) that we have 3 + **2 + 5 =10 objects including 3a's , 2b's and 5c's, or (2) that we have 3** objects a, b, c, where selections are constrained by the conditions that a can be selected at most three times, b can be selected at most twice, and c can be chosen up to five times.

The numbers 3, 2 and 5 in this example will be called repetition numbers.

Example 2   The 3-combinations of {3. a, 2. b, 5. c} are:

aaa, aab, aac, abb, abc,
ccc, ccb, cca, cbb.

Example 3.   The 3-combinations of {3 . a, 2. b, 2. c , 1. d} are:

aaa, aab, aac, aad, bba, bbc, bbd,
cca, ccb, ccd, abc, abd, acd, bcd.

In order to include the case where there is no limit on the number of times an object can be repeated in a selection (except that imposed by the size of the selection) we use the **symbol ∞ as a repetition number to mean that an object can occur an infinite number of times.**

Example 4. The 3-**combinations of {∞. a, 2.b, ∞.c} are the same as in Example 2 even though** a and c can be repeated an infinite number of times. This is because, in 3-combinations, 3 is the limit on the number of objects to be chosen.

If we are considerin**g selections where each object has ∞ as its repetition number then** we designate such selections as selections with unlimited repetitions. In particular, a selection of r objects in this case will be called r-combinations with unlimited repetitions and any ordered arrangement of these r objects will be an r-permutation with unlimited repetitions.

Example5 The combinations of a ,b, c, d with unlimited repetitions are the 3-combinations **of {∞ . a , ∞. b, ∞. c, ∞. d}. These are 20 such 3**-combinations, namely:

aaa, aab, aac, aad,
bbb, bba, bbc, bbd,
ccc, cca, ccb, ccd,
ddd, dda, ddb, ddc,
abc, abd, acd, bcd.

Moreover, there are 43 = 64 of 3-permutations with unlimited repetitions since the first position can be filled 4 ways (with a, b, c, or d), the second position can be filled 4 ways, and likewise for the third position.

The 2-**permutations of {∞. a, ∞. b, ∞. c, ∞. d} do not present such a formi**dable list and so we tabulate them in the following table.

2-combinations with Unlimited Repetitions	2-permutations with Unlimited Repetitions
aa	aa
ab	ab, ba
ac	ac, ca
ad	ad, da
bb	bb
bc	bc, cb
bd	bd, db
cc	cc
cd	cd, dc
dd	dd
10	16

Of course, these are not the only constraints that can be placed on selections; the possibilities are endless. We list some more examples just for concreteness. We might, for example, consider selections o**f {∞.a, ∞. b, ∞. c} where b can be chosen only even** number of times. Thus, 5-combinations with these repetition numbers and this constraint would be those 5-combinations with unlimited repetitions and where b is chosen 0, 2, or 4 times.

Example6 The 3-**combinations of {∞ .a, ∞ .b, 1 .c,1 .d} where b can be chosen only an even**
number of times are the 3-combinations of a, b, c, d where a can be chosen up 3 times, b can be
chosen 0 or 2 times, and c and d can be chosen at most once. The 3-cimbinations subject to these
constraints are:

        aaa, aac, aad, bbc, bbd, acd.

      As another example, we might be interested in, selections of {∞.a, 3.b, 1.c} where a can
be chosen a prime number of times. Thus, the 8-combinations subject to these constraints would
be all those 8-combinations where a can be chosen 2, 3, 5, or 7 times, b can chosen up to 3 times,
and c can be chosen at most once.

      There are, as we have said, an infinite variety of constraints one could place on
selections. You can just let your imagination go free in conjuring up different constraints on
the selection, would constitute an r-combination according to our definition. Moreover, any
arrangement of these r objects would constitute an r-permutation.

      While there may be an infinite variety of constraints, we are primarily interested in two
major types: one we have already described—combinations and permutations with unlimited
repetitions, the other we now describe.

      If the repetition numbers are all 1, then selections of r objects are called r-combinations
without repetitions and arrangements of the r objects are r-permutations without repetitions.
We remind you that r-combinations without repetitions are just subsets of the n elements
containing exactly r elements. Moreover, we shall often drop the repetition number 1 when
considering r-combinations without repetitions. For example, when considering r-combinations
of {a, b, c, d} we will mean that each repetition number is 1 unless otherwise designated, and,
of course, we mean that in a given selection an element need not be chosen at all, but, if it is
chosen, then in this selection this element cannot be chosen again.

Example7.   Suppose selections are to be made from the four objects a, b, c, d.

2-combinations without Repetitions	2-Permutations without Repetitions
ab	ab, ba

ac	ac, ca
ad	ad, da
bc	bc, cb
bd	bd, db
cd	cd, dc

6	12

There are six 2-combinations without repetitions and to each there are two 2-permutations giving a total of twelve 2-permutations without repetitions.

Note that total number of 2-combinations with unlimited repetitions in Example 5 included six 2-combinations without repetitions of Example.7 and as well 4 other 2-combinations where repetitions actually occur. Likewise, the sixteen 2-permutations with unlimited repetitions included the twelve 2-permutations without repetitions.

3-combinations without Repetitions	3-Permutations without Repetitions
abc	abc, acb, bac, bca, cab, cba
abd	abd, adb, bad, bda, dab, dba
acd	acd, adc, cad, cda, dac, dca
bcd	bcd, bdc, cbd, cdb, dbc, dcb

4	24

Note that to each of the 3-combinations without repetitions there are 6 possible 3-permutations without repetitions. Momentarily, we will show that this observation can be generalized.

Combinations And Permutations With Repetitions:

General formulas for enumerating combinations and permutations will now be presented. At this time, we will only list formulas for combinations and permutations without repetitions or with unlimited repetitions. We will wait until later to use generating functions to give general techniques for enumerating combinations where other rules govern the selections.

Let P (n, r) denote the number of r-permutations of n elements without repetitions.

Theorem 5.3.1.(Enumerating r-permutations without repetitions).

$$P(n, r) = n(n\text{-}1)\ldots\ldots (n - r + 1) = n! / (n\text{-}r)!$$

Proof. Since there are n distinct objects, the first position of an r-permutation may be filled in n ways. This done, the second position can be filled in n-1 ways since no repetitions are allowed and there are n – 1 objects left to choose from. The third can be filled in n-2 ways. By applying the product rule, we conduct that

$$P (n, r) = n(n\text{-}1)(n\text{-}2)\ldots\ldots (n - r + 1).$$

From the definition of factorials, it follows that

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

When r = n, this formula becomes

$$P (n, n) = n! / 0! = n!$$

When we explicit reference to r is not made, we assume that all the objects are to be arranged; thus we talk about the permutations of n objects we mean the case r=n.

Corollary 1.   There are n! permutations of n distinct objects.

Example 1.   There are 3! = 6 permutations of {a, b, c}.

There are 4! = 24 permutations of (a, b, c, d). The number of 2-permutations {a, b, c, d, e} is P(5, 2) = 5! / (5 - 2)! = 5 x 4 = 20. The number of 5-letter words using the letters a, b, c, d, and e at most once is P (5, 5) = 120.

Example 2 There are P (10, 4) = 5,040 4-digit numbers that contain no repeated digits since **each such number is just an arrangement of four of the digits 0, 1, 2, 3 , …., 9 (leading zeroes are** allowed). There are P (26, 3) P(10, 4) license plates formed by 3 distinct letters followed by 4 distinct digits.

Example3. In how many ways can 7 women and 3 men be arranged in a row if the 3 men must always stand next to each other?

There are 3! ways of arranging the 3 men. Since the 3 men always stand next to each **other, we treat them as a single entity, which we denote by X. Then if W1, W2, ….., W7** represents the women, we next are interested in the number of ways of arranging {X, W1, W2, **W3,……., W7}. There are 8! permutations these 8 objects. Hence there are (3!)** (8!) permutations altogether. (of course, if there has to be a prescribed order of an arrangement on the 3 men then there are only 8! total permutations).

Example4. In how many ways can the letters of the English alphabet be arranged so that there are exactly 5 letters between the letters a and b?

There are P (24, 5) ways to arrange the 5 letters between a and b, 2 ways to place a and b, and then 20! ways to arrange any 7-letter word treated as one unit along with the remaining 19 letters. The total is P (24, 5) (20!) (2).

permutations for the objects are being arranged in a line. If instead of arranging objects in a line, we arrange them in a circle, then the number of permutations decreases.

Example 5. In how many ways can 5 children arrange themselves in a ring?

Solution. Here, the 5 children are not assigned to particular places but are only arranged relative to one another. Thus, the arrangements (see Figure 2-3) are considered the same if the children are in the same order clockwise. Hence, the position of child C1 is immaterial and it is only the position of the 4 other children relative to C1 that counts. Therefore, keeping C1 fixed in position, there are 4! arrangements of the remaining children.

Binomial Coefficients: In mathematics, the **binomial coefficient** $\binom{n}{k}$ is the coefficient of the x^k term in the polynomial expansion of the binomial power $(1 + x)^n$.

In combinatorics, $\binom{n}{k}$ is interpreted as the number of k-element subsets (the k-combinations) of an n-element set, that is the number of ways that k things can be "chosen" from a set of n things. Hence, $\binom{n}{k}$ is often read as "n choose k" and is called the **choose function** of n and k. The notation $\binom{n}{k}$ was introduced by Andreas von Ettingshausen in 182, although the numbers were already known centuries before that (see Pascal's triangle). Alternative notations include C(n, k), $_nC_k$, C_k^n, C_k^n, C_n^k, in all of which the C stands for combinations or *choices*.

For natural numbers (taken to include 0) n and k, the binomial coefficient $\binom{n}{k}$ can be defined as the coefficient of the monomial X^k in the expansion of $(1 + X)^n$. The same coefficient also occurs (if $k \leq n$) in the binomial formula

$$(x + y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

(valid for any elements x, y of a commutative ring), which explains the name "binomial coefficient".

Another occurrence of this number is in combinatorics, where it gives the number of ways, disregarding order, that a k objects can be chosen from among n objects; more formally, the number of k-element subsets (or k-combinations) of an n-element set. This number can be seen to be equal to the one of the first definition, independently of any of the formulas below to compute it: if in each of the n factors of the power $(1 + X)^n$ one temporarily labels the term X with an index i (running from 1 to n), then each subset of k indices gives after expansion a contribution X^k, and the coefficient of that monomial in the result will be the number of such subsets. This shows in particular that $\binom{n}{k}$ is a natural number for any natural numbers n and k. There are many other combinatorial interpretations of binomial coefficients (counting problems for which the answer is given by a binomial coefficient expression), for instance the number of words formed of n bits (digits 0 or 1) whose sum is k, but most of these are easily seen to be equivalent to counting k-combinations.

Several methods exist to compute the value of $\binom{n}{k}$ without actually expanding a binomial power or counting k-combinations.

Binomial Multinomial theorems:

Binomial theorem:

In elementary algebra, the **binomial theorem** describes the algebraic expansion of powers of a binomial. According to the theorem, it is possible to expand the power $(x + y)^n$ into a sum involving terms of the form $ax^b y^c$, where the coefficient of each term is a positive integer, and the sum of the exponents of x and y in each term is n. For example,

$$(x + y)^4 = x^4 + 4x^3 y + 6x^2 y^2 + 4xy^3 + y^4.$$

The coefficients appearing in the binomial expansion are known as binomial coefficients. They are the same as the entries of Pascal's triangle, and can be determined by a simple formula involving factorials. These numbers also arise in combinatorics, where the coefficient of $x^{n-k} y^k$ is equal to the number of different combinations of k elements that can be chosen from an n-element set.

According to the theorem, it is possible to expand any power of $x + y$ into a sum of the form

$$(x + y)^n = \binom{n}{0} x^n y^0 + \binom{n}{1} x^{n-1} y^1 + \binom{n}{2} x^{n-2} y^2 + \binom{n}{3} x^{n-3} y^3 + \cdots$$
$$\cdots + \binom{n}{n-1} x^1 y^{n-1} + \binom{n}{n} x^0 y^n,$$

where $\binom{n}{k}$ denotes the corresponding binomial coefficient. Using summation notation, the formula above can be written

$$(x + y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k.$$

This formula is sometimes referred to as the **Binomial Formula** or the **Binomial Identity**.

A variant of the binomial formula is obtained by substituting 1 for x and x for y, so that it involves only a single variable. In this form, the formula reads

$$(1 + x)^n = \binom{n}{0} x^0 + \binom{n}{1} x^1 + \binom{n}{2} x^2 + \cdots + \binom{n}{n-1} x^{n-1} + \binom{n}{n} x^n,$$

or equivalently

$$(1 + x)^n = \sum_{k=0}^{n} \binom{n}{k} x^k.$$

EXAMPLE

Simplify (x+v(x2-1)) + (x- v(x2-1))6

Solution: let vx2-1 = a, so we have:
    (x=a)6 + (x-a)6

= [x6+6C1x5.a+6C2.x4.a2 + 6C3x3a3 + 6C4x2a4 + 6C5xa5 +6C6a6]

  [x6-6C1x5a+6C2.x4.a2 − 6C3x3a3 + 6C4x2a4 − 6C5xa5 +6C6a6]

☐2[x6+6C2x4a2+6C4x2a4+6C6a6]

☐2[x6+15x4(x2-1)+15x2(x2-1)2+(x2-1)3]

☐2[x6+15x6-15x4+15x6+15x2-30x4+x6-1-3x4+3x3]

☐2[32x6-48x4+18x2-1]

Multinomial theorem:

In mathematics, the **multinomial theorem** says how to write a power of a sum in terms of powers of the terms in that sum. It is the generalization of the binomial theorem to polynomials.

For any positive integer m and any nonnegative integer n, the multinomial formula tells us how a polynomial expands when raised to an arbitrary power:

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

The summation is taken over all sequences of nonnegative integer indices k_1 through k_m such the sum of all k_i is n. That is, for each term in the expansion the exponents must add up to n. Also, as with the binomial theorem, quantities of the form x^0 that appear are taken to equal 1 (even when x equals zero). Alternatively, this can be written concisely using multiindices as

$$(x_1 + \cdots + x_m)^n = \sum_{\alpha} \binom{n}{\alpha} x^{\alpha}$$

where $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_m)$ and x $= x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_m^{\alpha_m}$.
Example

$$(a + b + c)^3 = a^3 + b^3 + c^3 + 3a^2 b + 3a^2 c + 3b^2 a + 3b^2 c + 3c^2 a + 3c^2 b + 6abc.$$

We could have calculated each coefficient by first expanding
$(a + b + c)^2 = a^2 + b^2 + c^2 + 2ab + 2bc + 2ac$, then self-multiplying it again to get $(a + b + c)^3$ (and then if we were raising it to higher powers, we'd multiply it by itself even some more).

However this process is slow, and can be avoided by using the multinomial theorem. The multinomial theorem "solves" this process by giving us the closed form for any coefficient we

might want. It is possible to "read off" the multinomial coefficients from the terms by using the multinomial coefficient formula. For example:

$a^2 b^0 c^1$ has the coefficient $\binom{3}{2,0,1} = \dfrac{3!}{2! \cdot 0! \cdot 1!} = \dfrac{6}{2 \cdot 1 \cdot 1} = 3$

$a^1 b^1 c^1$ has the coefficient $\binom{3}{1,1,1} = \dfrac{3!}{1! \cdot 1! \cdot 1!} = \dfrac{6}{1 \cdot 1 \cdot 1} = 6$.

We could have also had a 'd' variable, or even more variables—hence the *multinomial* theorem.

The principles of Inclusion – Exclusion:

Let $|A|$ denote the cardinality of set A, then it follows immediately that

$$|A \cup B| = |A| + |B| - |A \cap B|, \tag{1}$$

where $\cup$ denotes union, and $\cap$ denotes intersection. The more general statement

$$\left| \bigcup_{i=1}^{N} E_i \right| \leq \sum_{i=1}^{N} |E_i|, \tag{2}$$

also holds, and is known as Boole's inequality.

This formula can be generalized in the following beautiful manner. Let $\mathcal{A} = \{A_i\}_{i=1}^{p}$ be a p-system of S consisting of sets A_1, ..., A_p, then

$$|A_1 \cup A_2 \cup \ldots \cup A_p| = \sum_{1 \leq i \leq p} |A_i| - \sum_{1 \leq i_1 < i_2 \leq p} |A_{i_1} \cap A_{i_2}| + \sum_{1 \leq i_1 < i_2 < i_3 \leq p} |A_{i_1} \cap A_{i_2} \cap A_{i_3}| - \ldots + (-1)^{p-1} |A_1 \cap A_2 \cap \cdots \cap A_p|$$

where the sums are taken over k-subsets of $\mathcal{A}$. This formula holds for infinite sets S as well as finite sets.

The principle of inclusion-exclusion was used by Nicholas Bernoulli to solve the recontres problem of finding the number of derangements.

For example, for the three subsets $A_1 = \{2, 3, 7, 9, 10\}$, $A_2 = \{1, 2, 3, 9\}$, and $A_3 = \{2, 4, 9, 10\}$ of $S = \{1, 2, \ldots, 10\}$, the following table summarizes the terms appearing the sum.

#	term	set	length
1	A_1	{2, 3, 7, 9, 10}	5
	A_2	{1, 2, 3, 9}	4
	A_3	{2, 4, 9, 10}	4
2	$A_1 \cap A_2$	{2, 3, 9}	3
	$A_1 \cap A_3$	{2, 9, 10}	3
	$A_2 \cap A_3$	{2, 9}	2
3	$A_1 \cap A_2 \cap A_3$	{2, 9}	2

$|A_1 \cup A_2 \cup A_3|$ is therefore equal to, (5 + 4 + 4) - (3 + 3 + 2) + 2 = 7, corresponding to the seven elements $A_1 \cup A_2 \cup A_3 = \{1, 2, 3, 4, 7, 9, 10\}$.

Pigeon hole principles and its application:

The statement of the *Pigeonhole Principle*:

If m pigeons are put into m pigeonholes, there is an empty hole *iff* there's a hole with more than one pigeon.

If n > m pigeons are put into m pigeonholes, there's a hole with more than one pigeon.

Example:

Consider a chess board with two of the diagonally opposite corners removed. Is it possible to cover the board with pieces of domino whose size is exactly two board squares?

Solution

No, it's not possible. Two diagonally opposite squares on a chess board are of the same color. Therefore, when these are removed, the number of squares of one color exceeds by 2 the number of squares of another color. However, every piece of domino covers exactly two squares and these are of different colors. Every placement of domino pieces establishes a 1-1 correspondence between the set of white squares and the set of black squares. If the two sets have different number of elements, then, by the Pigeonhole Principle, no 1-1 correspondence between the two sets is possible.

Generalizations of the pigeonhole principle

A generalized version of this principle states that, if n discrete objects are to be allocated to m containers, then at least one container must hold no fewer than $\lceil n/m \rceil$ objects, where $\lceil x \rceil$ is the ceiling function, denoting the smallest integer larger than or equal to x. Similarly, at least one container must hold no more than $\lfloor n/m \rfloor$ objects, where $\lfloor x \rfloor$ is the floor function, denoting the largest integer smaller than or equal to x.

A probabilistic generalization of the pigeonhole principle states that if n pigeons are randomly put into m pigeonholes with uniform probability $1/m$, then at least one pigeonhole will hold more than one pigeon with probability

$$1 - \frac{(m)_n}{m^n},$$

where $(m)_n$ is the falling factorial $m(m-1)(m-2)...(m-n+1)$. For $n = 0$ and for $n = 1$ (and $m > 0$), that probability is zero; in other words, if there is just one pigeon, there cannot be a conflict. For $n > m$ (more pigeons than pigeonholes) it is one, in which case it coincides with the ordinary pigeonhole principle. But even if the number of pigeons does not exceed the number of pigeonholes ($n \leq m$), due to the random nature of the assignment of pigeons to pigeonholes there is often a substantial chance that clashes will occur. For example, if 2 pigeons are randomly assigned to 4 pigeonholes, there is a 25% chance that at least one pigeonhole will hold more than one pigeon; for 5 pigeons and 10 holes, that probability is 69.76%; and for 10 pigeons and 20 holes it is about 93.45%. If the number of holes stays fixed, there is always a greater probability of a pair when you add more pigeons. This problem is treated at much greater length at birthday paradox.

A further probabilistic generalisation is that when a real-valued random variable X has a finite mean $E(X)$, then the probability is nonzero that X is greater than or equal to $E(X)$, and similarly the probability is nonzero that X is less than or equal to $E(X)$. To see that this implies the standard pigeonhole principle, take any fixed arrangement of n pigeons into m holes and let X be the number of pigeons in a hole chosen uniformly at random. The mean of X is n/m, so if there are more pigeons than holes the mean is greater than one. Therefore, X is sometimes at least 2.

Applications:

The pigeonhole principle arises in computer science. For example, collisions are inevitable in a hash table because the number of possible keys exceeds the number of indices in the array. No hashing algorithm, no matter how clever, can avoid these collisions. This principle also proves that any general-purpose lossless compression algorithm that makes at least one input file smaller will make some other input file larger. (Otherwise, two files would be compressed to the same smaller file and restoring them would be ambiguous.)

A notable problem in mathematical analysis is, for a fixed irrational number a, to show that the set {[na]: n is an integer} of fractional parts is dense in [0, 1]. After a moment's thought, one

small positive number and a is some arbitrary irrational number. But if one takes M such that $1/M < e$, by the pigeonhole principle there must be n_1, n_2 $\{1, 2, ..., M + 1\}$ such that n_1a and n_2a are in the same integer subdivision of size $1/M$ (there are only M such subdivisions between consecutive integers). In particular, we can find n_1, n_2 such that n_1a is in $(p + k/M, p + (k + 1)/M)$, and n_2a is in $(q + k/M, q + (k + 1)/M)$, for some p, q integers and k in $\{0, 1, ..., M − 1\}$. **We** can then easily verify that $(n_2 − n_1)a$ is in $(q − p − 1/M, q − p + 1/M)$. This implies that $[na] < 1/M < e$, where $n = n_2 − n_1$ or $n = n_1 − n_2$. This shows that 0 is a limit point of $\{[na]\}$. We can then use this fact to prove the case for p in $(0, 1]$: find n such that $[na] < 1/M < e$; then if p $(0, 1/M]$, we are done. Otherwise in $(/ , (+ 1)/]$, and by setting $= \sup\{ : [ ] < / $.

$\in$

$|[(k + 1)na] − p| < 1/M < e.$

UNIT-IV

Recurrence Relation

Generating Functions:

In mathematics, a **generating function** is a formal power series in one indeterminate, whose coefficients encode information about a sequence of numbers a_n that is indexed by the natural numbers. Generating functions were first introduced by Abraham de Moivre in 1730, in order to solve the general linear recurrence problem. One can generalize to formal power series in more than one indeterminate, to encode information about arrays of numbers indexed by several natural numbers.

Generating functions are not functions in the formal sense of a mapping from a domain to a codomain; the name is merely traditional, and they are sometimes more correctly called **generating series**.

Ordinary generating function

The *ordinary generating function* of a sequence a_n is

$$G(a_n; x) = \sum_{n=0}^{\infty} a_n x^n.$$

When the term *generating function* is used without qualification, it is usually taken to mean an ordinary generating function.

If a_n is the probability mass function of a discrete random variable, then its ordinary generating function is called a probability-generating function.

The ordinary generating function can be generalized to arrays with multiple indices. For example, the ordinary generating function of a two-dimensional array $a_{m,\,n}$ (where n and m are natural numbers) is

$$G(a_{m,n}; x, y) = \sum_{m,n=0}^{\infty} a_{m,n} x^m y^n.$$

Example:

$$G(n^2; x) = \sum_{n=0}^{\infty} n^2 x^n = \frac{x(x+1)}{(1-x)^3}$$

Exponential generating function

The *exponential generating function* of a sequence a_n is

$$EG(a_n; x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}.$$

Example:

$$EG(n^2; x) = \sum_{n=0}^{\infty} \frac{n^2 x^n}{n!} = x(x+1)e^x$$

Function of Sequences:

Generating functions giving the first few powers of the nonnegative integers are given in the following table.

n^p	$f(x)$	series
1	$\frac{x}{1-x}$	$x + x^2 + x^3 + \dots$
n	$\frac{x}{(1-x)^2}$	$x + 2x^2 + 3x^3 + 4x^4 + \dots$
n^2	$\frac{x(x+1)}{(1-x)^3}$	$x + 4x^2 + 9x^3 + 16x^4 + \dots$
n^3	$\frac{x(x^2+4x+1)}{(1-x)^4}$	$x + 8x^2 + 27x^3 + \dots$
n^4	$\frac{x(x+1)(x^2+10x+1)}{(1-x)^5}$	$x + 16x^2 + 81x^3 + \dots$

There are many beautiful generating functions for special functions in number theory. A few particularly nice examples are

$$f(x) = \frac{1}{(x)_\infty} \tag{2}$$

$$= \sum_{n=0}^{\infty} P(n) x^n \tag{3}$$

$$= 1 + x + 2x^2 + 3x^3 + \dots \tag{4}$$

for the partition function P, where $(q)_\infty$ is a q-Pochhammer symbol, and

$$f(x) \quad = \quad \frac{x}{1 - x - x^2} \tag{5}$$

$$= \quad \sum_{n=0}^{\infty} F_n x^n \tag{6}$$

$$= \quad x + x^2 + 2x^3 + 3x^4 + \dots \tag{7}$$

for the Fibonacci numbers F_n.

Generating functions are very useful in combinatorial enumeration problems. For example, the subset sum problem, which asks the number of ways $c_{m,s}$ to select out of M given integers such that their sum equals s, can be solved using generating functions.

Calculating Coefficient of generating function:

By using the following polynomial expansions, we can calculate the coefficient of a generating function.

Polynomial Expansions:

1) $\dfrac{1 - x^{m+1}}{1 - x} = 1 + x + \dots + x^m +$

2) $\dfrac{1}{1 - x} = 1 + x + x^2 + \dots$

3) $(1 + x)^n = 1 + C(n,1)x + C(n,2)x^2 + \dots + C(n,r)x^r + \dots + C(n,n)x^n$

4) $(1 - x^m)^n = 1 + C(n,1)x^m + C(n,2)x^{2m} + \dots + (-1)^k C(n,k)x^{km} + \dots + (-1)^n C(n,n)x^{nm}$

5) $\dfrac{1}{(1-x)^n} = 1 + C(1+n-1,1)x + C(2+n-1,2)x^2 + \dots + C(r+n-1,r)x^r + \dots$

6) If h(x)=f(x)g(x), where f(x) $= a_0 + a_1 x + a_2 x^2 + \dots$ and g(x) $= b_0 + b_1 x + b_2 x^2 + \dots$, then

h(x) $= a_0 b_0 + (a_1 b_0 + a_0 b_1)x + (a_2 b_0 + a_1 b_1 + a_0 b_2)x^2 + \dots + (a_r b_0 + a_{r-1} b_1 + a_{r-2} b_2 + \dots + a_0 b_r)x^r + \dots$

Example

Find the coefficient of x^{16} in $(x^2 + x^3 + x^4 + \ldots)^5$

$$x^{16} \text{ in } x^{10}(1-x)^{-5} [i.e, \text{ the } x^6 \text{ term in} (1-x)^{-5} \text{ is}$$

$$\text{multiplied by } x^{10} \text{ to become the } x^{16} \text{ term in } x^{10}(1-x)^{-5}]$$

To simplify the expression, we extract x^2 from each polynomial factor and the apply identity (2).

$$(x^2 + x^3 + x^4 + \ldots)^5 = [x^2(1 + x + x^2 + \ldots)]^5$$

$$= x^{10}(1 + x + x^2 + \ldots)^5$$

$$= x^{10} \frac{1}{(1-x)^5}$$

Thus the coefficient of x^{16} in $(x^2 + x^3 + x^4 + \ldots)^5$ is the coefficient of x^{16} in $x^{10}(1-x)^{-5}$ [i.e., the x^6 term in $(1-x)^{-5}$ is multiplied by to become the x^{16} term in $x^{10}(1-x)^{-5}$]

$$\frac{1}{(1-x)^n} = 1 + C(1+n-1,1)x + C(2+n-1,2)x^2 + \ldots + C(r+n-1,r)x^r + \ldots$$

From expansion (5) we see that the coefficient of x^6 in $(1-x)^{-5}$ is $C(6+5-1,6)$

More generally, the coefficient of x^r in

$$x^r \text{ in } x^{10}(1-x)^{-5} \text{ equals the coefficient of } x^{r-10} \text{ in}$$

$$(1-x)^{-5}, \text{namely}, C((r-10)+5-1,(r-10)).$$

Recurrence relations:

Introduction :A recurrence relation is a formula that relates for any integer n ≥ 1, the n-th term of a sequence A = {ar}∞r=0 to one or more of the terms a0,a1,….,an-1. Example. If Sn denotes the sum of the first n positive integers, then

10. Sn = n + Sn-1. Similarly if d is a real number, then the *n*th term of an arithmetic progression with common difference d satisfies the relation

11. an = an -1 + d. Likewise if pn denotes the nth term of a geometric progression with common ratio r, then

☐ pn = r pn – 1. We list other examples as:

an – 3an-1 + 2an-2 = 0.

an – 3 an-1+ 2 an-2 = n2 + 1.

an – (n - 1) an-1 - (n - 1) an-2 = 0.

an – 9 an-1+ 26 an-2 – 24 an-3 = 5n.

an – 3(an-1)2 + 2 an-2 = n.

an = a0 an-1+ a1 an-2+ … + an-1a0.

a2n + (an-1)2 = -1.

Definition. Suppose n and k are nonnegative integers. A recurrence relation of the form   c0(n)an + c1(n)an-1 + …. + ck(n)an-k = f(n) for n ≥ k, where c0(n), c1(n),…., ck(n), and f(n) are functions of n is said to be a **linear recurrence relation**. If c0(n) and ck(n) are not identically zero, then it is said to be a **linear recurrence relation** *degree* **k**. If c0(n), c1(n),…., ck(n) are constants, then the recurrence relation is known as a **linear relation with constant coefficients**. If f(n) is identically zero, then the recurrence relation is said to be **homogeneous**; otherwise, it is **inhomogeneous.**

   Thus, all the examples above are linear recurrence relations except (8), (9), and (10); the relation (8), for instance, is not linear because of the squared term.

The relations in (3), (4) , (5), and (7) are linear with constant coefficients.

Relations (1), (2), and (3) have degree 1; (4), (5), and (6) have degree 2; (7) has degree 3. Relations (3) , (4), and (6) are homogeneous.

        There are no general techniques that will enable one to solve all recurrence relations. There are, nevertheless, techniques that will enable us to solve linear recurrence relations with constant coefficients.

SOLVING RECURRENCE RELATIONS BY SUSTITUTION AND GENERATING FUNCTIONS

     We shall consider four methods of solving recurrence relations in this and the next two sections:

    5. Substitution (also called iteration),

    6. Generating functions,

    7. Characteristics roots, and

    8. Undetermined coefficients.

      In the substitution method the recurrence relation for an is used repeatedly to solve for a general expression for an in terms of n. We desire that this expression involve no other terms of the sequence except those given by boundary conditions.

       The mechanics of this method are best described in terms of examples. We used this method in Example5.3.4. Let us also illustrate the method in the following examples.

Example

Solve the recurrence relation an = a n-1 + f(n) for n ³1 by substitution

a1= a0 + f(1)
a2 = a1 + f(2) = a0 + f(1) + f(2))

a3 = a2 + f(3)= a0 + f(1) + f(2) + f(3)

.

.

.

an = a0 + f(1) + f(2) +….+
f(n) n
= a0 + ∑ f(k)
K = 1

Thus, an is just the sum of the f(k) „s plus a0.

More generally, if c is a constant then we can solve an = c a n-1 + f(n) for n ³1 in the same way:

a1 = c a0 + f(1)

a2 = c a1 + f(2) = c (c a0 + f(1)) + f(2)
  = c2 a0 + c f(1) + f(2)

a3= c a2 + f(3) = c(c 2 a0 + c f(1) + f(2)) +
f(3) =c3 a0 + c2 f(1) + c f(2) + f(3)
.
.
.

an = c a n-1 + f(n) = c(c n-1 a0 + c n-2 f(1) +. . . + c n-2 + f(n-1)) +
  f(n) =c n a0 + c n-1 f(1) + c n-2 f(2) +. . .+ c f(n-1) + f(n)
Or
an = c n a0 + ∑c n-k f(k)

Solution of Linear Inhomogeneous Recurrence Relations:

The equation $a_n + c_1 a_{n-1} + c_2 a_{n-2} = f(n)$, where c_1 and c_2 are constant, and $f(n)$ is not identically 0, is called a second-order linear **inhomogeneous** recurrence relation (or difference equation) with constant coefficients. The homogeneous case, which we"ve looked at already, occurs when

$f(n) \equiv 0$. The inhomogeneous case occurs more frequently. The homogeneous case is so important largely because it gives us the key to solving the inhomogeneous equation. If you"ve studied linear differential equations with constant coefficients, you"ll see the parallel. We will call the

difference obtained by setting the right-hand side equal to 0, the "associated homogeneous equation." We know how to solve this. Say that V is a solution. Now suppose that (n) is any particular solution of the inhomogeneous equation. (That is, it solves the equation, but does not necessarily match the initial data.) Then $U=V+(n)$ is a solution to the inhomogeneous equation, which you can see simply by substituting U into the equation. On the other hand, every solution U of the inhomogeneous equation is of the form $U=V+n$) where V is a solution of the homogeneous equation, and $g(n)$ is a particular solution of the inhomogeneous equation. The proof of this is straightforward. If we have two solutions to the inhomogeneous equation, say $U1$ and $U2$, then their difference $U1-U2=V$ is a solution to the homogeneous equation, which you can check by substitution. But then $U1=V+U2$, and we can set $U2=($ $)$, since by assumption, $U2$ is a particular solution. This leads to the following theorem: **the general solution to the inhomogeneous equation is the general solution to the associated homogeneous equation, plus any particular solution to the inhomogeneous equation.** This gives the following procedure for solving the inhomogeneous equation:

4. Solve the associated homogeneous equation by the method we"ve learned. This will involve variable (or undetermined) coefficients.

5. Guess a particular solution to the inhomogeneous equation. It is because of the guess that I"ve called this a procedure, not an algorithm. For simple right-hand sides , we can say how to compute a particular solution, and in these cases, the procedure merits the name "algorithm."

6. The general solution to the inhomogeneous equation is the sum of the answers from the two steps above.

7. Use the initial data to solve for the undetermined coefficients from step 1.

To solve the equation $an- 6\ an1 + 8\ an2 = 3$. Let"s suppose that we are also given the initial data $a0 = 3$, $a1 = 3$. The associated homogeneous equation is $an - 6an-1 + 8an-2 = 0$, so the characteristic equation is $r2 - 6r+ 8 = 0$, which has roots $r1 = 2$ and $r2 = 4$. Thus, the general solution to the associated homogeneous equation is $12n+ 24$. When the right-hand side is a polynomial, as in this case, there will always be a particular solution that is a polynomial. Usually, a polynomial of the same degree will work, so we"ll guess in this case that there is a constant C that solves the homogeneous equation. If that is so, then $an = an -1 = an -2 = C$, and substituting into the equation gives $C - C + C = 3$, and we find that $C = 1$. Now, the general solution to the inhomogeneous equations is $12 + 24\ n\ 1$. Reassuringly, this is the answer given in the back of the book. Our initial data lead to the equations $c + c2 + 1 = 3$ and $2c1 + 4c2 + 1 = 3$, whose solution is $1 \in 3$, $2 = c -1$. Finally, the solution to the inhomogeneous equation, with the initial condition given, is $an = 3 \cdot 2\ n - 4n + 1$. Sometimes, a polynomial of the same degree as the right-hand side doesn"t work. This happens when the characteristic equation has 1 as a root. If our equation had been $an- 6\ an-1 + 5\ an-2 = 3$, when we guessed that the particular solution was a constant C, we"d have arrived at the equation $C - C + C = 3$, or $0 = 3$. The way to deal with this is to increase the degree of the polynomial. Instead of assuming that the solution is constant, we"ll assume that it"s linear. In fact, we"ll guess that it is of the form

$g_n = nC$. Then we have $nC - 6n - 1 \ C + 5n - 2 \ C = 3$, which simplifies to $C - 10C = 3$ so that $C = -34$. Thus, $g_n = -3 n4$. This won"t be enough if 1 is a root of multiplicity 2, that is, if $r - 1$ 2 is a factor of the characteristic polynomial. Then there is a particular solution of the form $g_n = Cn2$. For second-order equations, you never have to go past this. If the right-hand side is a polynomial of degree greater than 0, then the process works juts the same, except that you start with a polynomial of the same degree, increase the degree by 1, if necessary, and then once more, if need be. For example, if the right-hand side were $f_n = 2 \ n - 1$, we would start by guessing a particular solution $g_n \ C1n \ C2$. If it turned out that 1 was a characteristic root, we would amend our guess to $g_n \ C1 \ iC2 \ C3n$ If 1 is a double root, this will fail also, but g_n $= C1nC2 \ C3 \ C4$ will work in this case.

Another case where there is a simple way of guessing a particular solution is when the right-hand side is an exponential, say $f_n \ Cn$. In that case, we guess that a particular solution is just a constant multiple of f, say $(n \ )kCn$ Again, we gave trouble when 1 is a characteristic root. We then guess that $g_n = knCn$, which will fail only if 1 is a double root. In that case we must use $g_n \ kn2Cn$, which is as far as we ever have to go in the second-order case. These same ideas extend to higher-order recurrence relations, but we usually solve them numerically, rather than exactly. A third-order linear difference equation with constant coefficients leads to a cubic characteristic polynomial. There is a formula for the roots of a cubic, but it"s very complicated.

For fourth-degree polynomials, there"s also a formula, but it"s even worse. For fifth and higher degrees, no such formula exists. Even for the third-order case, the exact solution of a simple-looking inhomogeneous linear recurrence relation with constant coefficients can take pages to write down. The coefficients will be complicated expressions involving square roots and cube roots. For most, if not all, purposes, a simpler answer with numerical coefficients is better, even though they must in the nature of things, be approximate.

The procedure I"ve suggested may strike you as silly. After all, we"ve already solved the characteristic equation, so we know whether 1 is a characteristic root, and what it"s multiplicity is. Why not start with a polynomial of the correct degree? This is all well and good, while you"re taking the course, and remember the procedure in detail. However, if you have to use this procedure some years from now, you probably won"t remember all the details. Then the method I"ve suggested will be valuable. Alternatively, you can start with a general polynomial of the maximum possible degree This leads to a lot of extra work if you"re solving by hand, but it"s the approach I prefer for computer solution.

UNIT- V

Graph Theory

Representation of Graphs:

There are two different sequential representations of a graph. They are

- 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 $A = (a_{ij})$ 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 $a_{ij} = a_{ji}$; for every i and j.

Drawbacks

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

13. If the number of edges is 0(m) or 0(m log2 m), 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 $P = (p_{ij})$ defined as follows:

1 if there is a path from Vi to Vj

Pij =0 otherwise

Graphs and Multigraphs

A graph G consists of two things:
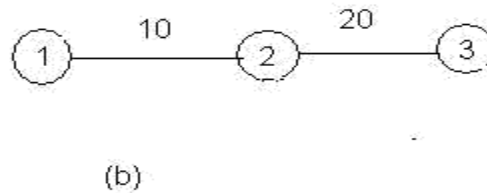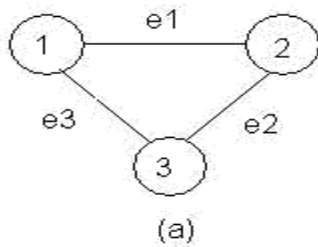
weighted or Labeled Graph

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

2. A set 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 belong to any edge—then 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 vn = 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 $\neq$ Vn)

(ii) Cycle Path

Cycle path is a path in which first and last vertex are same (V0 = Vn).It is also called as 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 G is said to be complete if every node u in G is adjacent to every other node v in G.

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 they 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 endpoints, that is, if e = [u, u].
- *Finite Graph*:A multigraph M is said to be finite if it has a finite number of nodes and a finite number of edges.
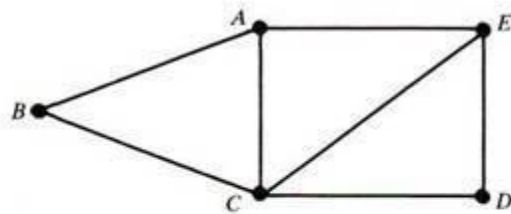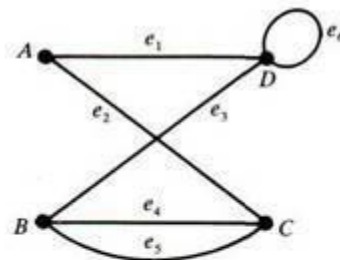
(a) Graph.   (b) Multigraph.
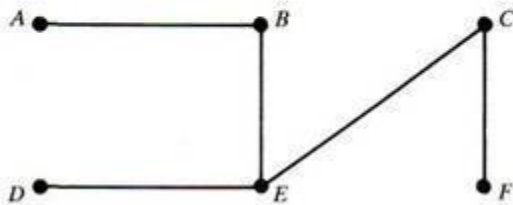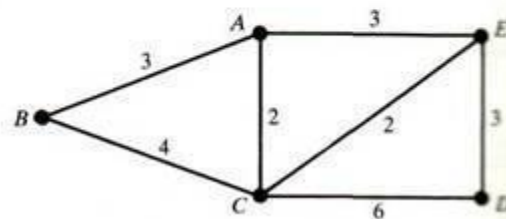
(c) Tree.   (d) Weighted graph.

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
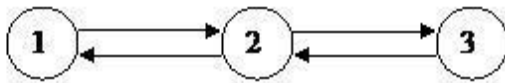
Indegree of 1 = 1
Indegree pf 2 = 2

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

Example
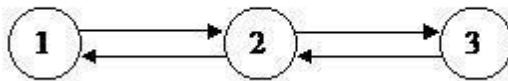
Outdegree of 1 =1
Outdegree of 2 =2

Simple Directed Graph

A directed graph G is said to be simple if G has no parallel edges. A simple graph G 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 used for 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:
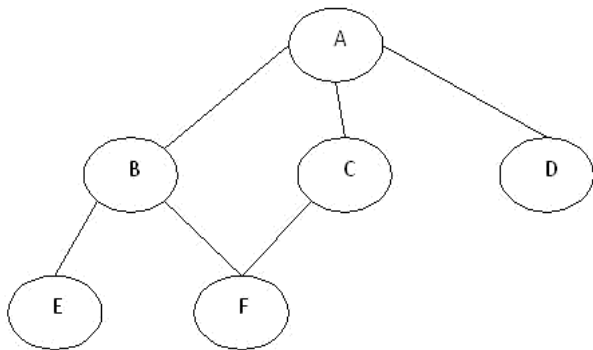
As stated before, in DFS, nodes are visited by going through the depth of the tree from the starting node. If we do the depth first traversal of the above graph and print the visited node, it will be "A B E F C D". DFS visits the root node and then its children nodes until it reaches the end node, i.e. E and F nodes, then moves up to the parent nodes.

Algorithmic Steps

6. **Step 1**: Push the root node in the Stack.
7. **Step 2**: Loop until stack is empty.
8. **Step 3**: Peek the node of the stack.
9. **Step 4**: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
10. **Step 5**: If the node does not have any unvisited child nodes, pop the node from the stack.

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

```java
public void dfs()
{
  //DFS uses Stack data structure
  Stack s=new Stack();
  s.push(this.rootNode);
  rootNode.visited=true;
  printNode(rootNode);
  while(!s.isEmpty())
  {
        Node n=(Node)s.peek();
        Node child=getUnvisitedChildNode(n);
        if(child!=null)
        {
              child.visited=true;
              printNode(child);
              s.push(child);
```

```
            }
            else
            {
                    s.pop();
            }
        }
    //Clear visited property of
    nodes clearNodes();
    }
```

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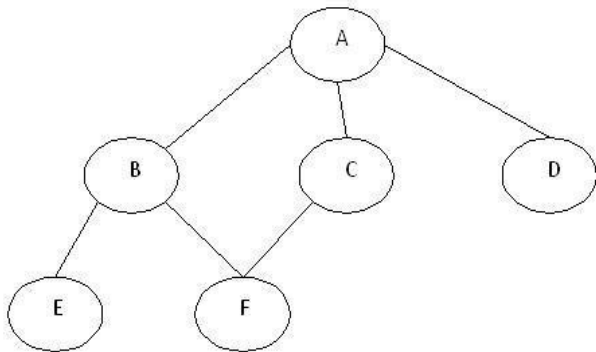
If we do the breadth first traversal of the above graph and print the visited node as the output, it will print the following output. "A B C D E F". The BFS visits the nodes level by level, so it will start with level 0 which is the root node, and then it moves to the next levels which are B, C and D,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

Algorithmic Steps

1. **Step 1**: Push the root node in the Queue.
2. **Step 2**: Loop until the queue is empty.
3. **Step 3**: Remove the node from the Queue.
4. **Step 4**: If the removed node has unvisited child nodes, mark them as 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:

```java
public void bfs()
{
  //BFS uses Queue data structure

  Queue q=new LinkedList();
  q.add(this.rootNode);
  printNode(this.rootNode);
  rootNode.visited=true;
  while(!q.isEmpty())
  {
          Node n=(Node)q.remove();
          Node child=null;
          while((child=getUnvisitedChildNode(n))!=null)
          {
                  child.visited=true;
                  printNode(child);
                  q.add(child);
          }
  }
  //Clear visited property of
  nodes clearNodes();
}
```
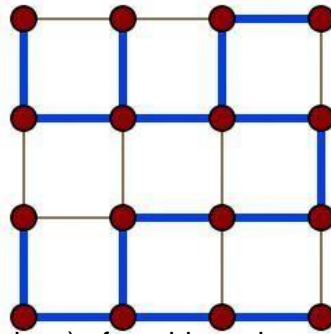
Spanning Trees:

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).

Cayley's formula is a formula for the number of spanning trees in the complete graph K_n with n vertices. The formula states that $t(K_n) = n^{n-2}$. Another way of stating Cayley's formula is   that there are exactly n^{n-2} labelled trees with n vertices. Cayley's formula can be proved using Kirchhoff's matrix-tree theorem or via the Prüfer code.

If G is the complete bipartite graph $K_{p,q}$, then $t(G) = p^{q-1}q^{p-1}$, while if G is the n-dimensional

$$t(G) = 2^{2^n-n-1}\prod_{k=2}^{n} k^{\binom{n}{k}}$$

hypercube graph Q_n, then   . These formulae are also consequences of the matrix-tree theorem.

If G is a multigraph and e is an edge of G, then the number $t(G)$ of spanning trees of G satisfies the *deletion-contraction recurrence $t(G)=t(G-e)+t(G/e)$*, where $G-e$ is the multigraph obtained by deleting e and G/e is the contraction of G by e, where multiple edges arising from

this contraction are not deleted.

Uniform spanning trees

A spanning tree chosen randomly from among all the spanning trees with equal probability is called a uniform spanning tree (UST). This model has been extensively researched in probability and mathematical physics.

Algorithms

The classic spanning tree algorithm, depth-first search (DFS), is due to Robert Tarjan. Another important algorithm is based on breadth-first search (BFS).

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 underpasses
- VLSI – e.g. laying out circuits on computer chip.
- The puzzle game Planarity requires the player to "untangle" a planar graph so that none of its edges intersect.
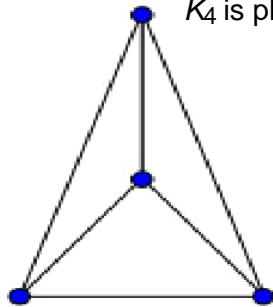
Example graphs

Planar   **Nonplanar**

Butterfly graph

K_5
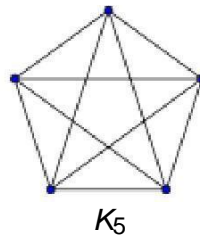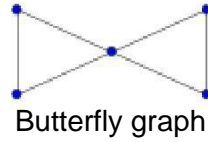
The complete graph
K_4 is planar

$K_{3,3}$
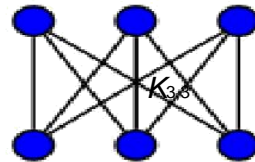
Graph Theory and Applications:

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 TextGraphs, as well as various 'Net' projects, such as WordNet, VerbNet, 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.Likewise, 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 G1 and G1 be two graphs and let f be a function from the vertex set of G1 to 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).
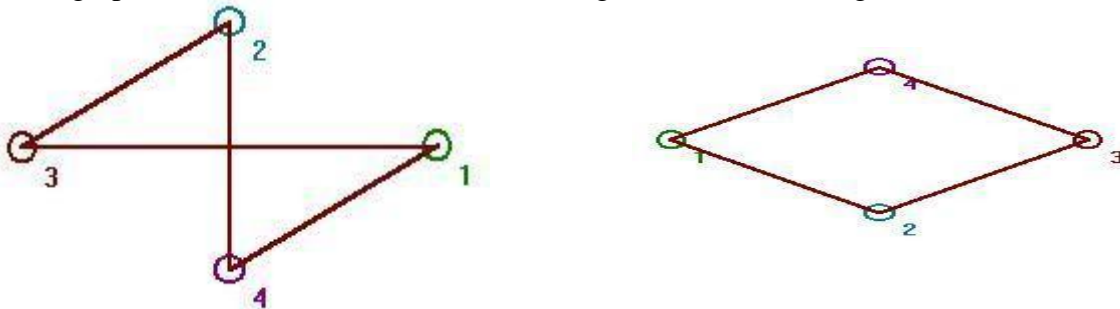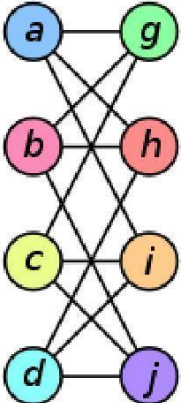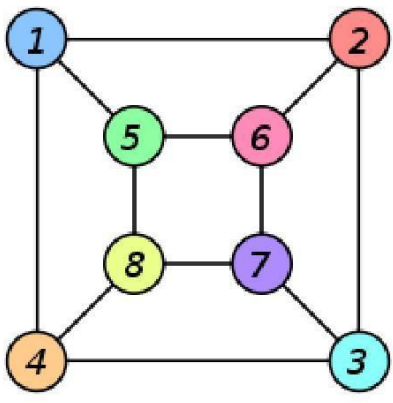

Save this graph as Graph 1 (you need to click Graph then Save). Now get the circuit graph with 4 vertices. It looks like figure 5, and we shall call it C(4).

Example:

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

| **Graph G**	**Graph H**	**An isomorphism between G and H**
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

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 **supergraph** 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, *xy* is an edge of *H* if and only if *xy* 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.
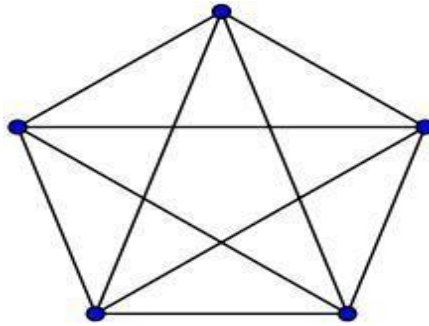
K5, a complete graph. If a subgraph looks like this, the vertices in that subgraph form a clique of size 5.

Multi graphs:

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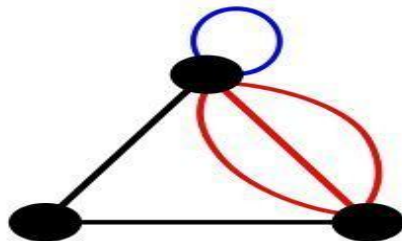
A multigraph with multiple edges (red) and a loop (blue). Not all authors allow multigraphs to have loops.

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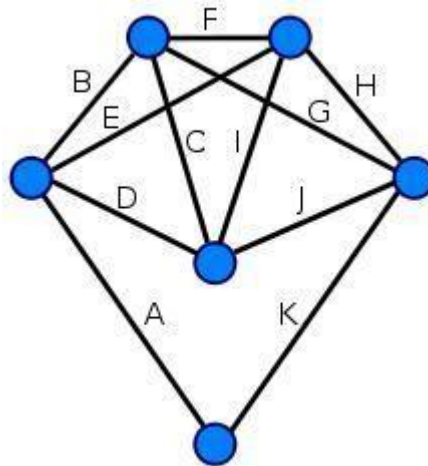
An **Eulerian cycle**, **Eulerian circuit** or **Euler tour** in an undirected graph is a cycle that uses each edge exactly once. If such a cycle exists, the graph is called **unicursal**. While such graphs are Eulerian graphs, not every Eulerian graph possesses an Eulerian cycle.

For directed graphs path has to be replaced with directed path and cycle with directed cycle.

The definition and properties of Eulerian trails, cycles and graphs are valid for multigraphs as well.
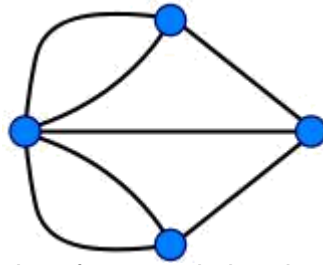
This graph is not Eulerian, therefore, a solution does not exist.

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 **Hamiltonian decomposition** is an edge decomposition of a graph into Hamiltonian circuits.

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
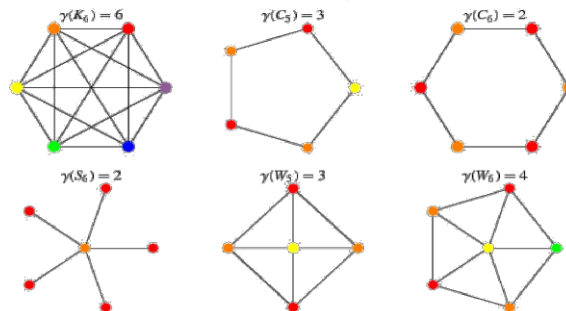
color is assigned, or even on the color itself. It has even reached popularity with the general public in the form of the popular number puzzle Sudoku. Graph coloring is still a very active field of research.
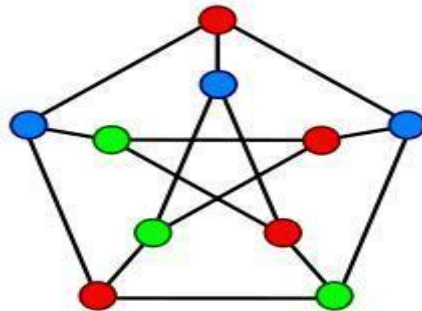
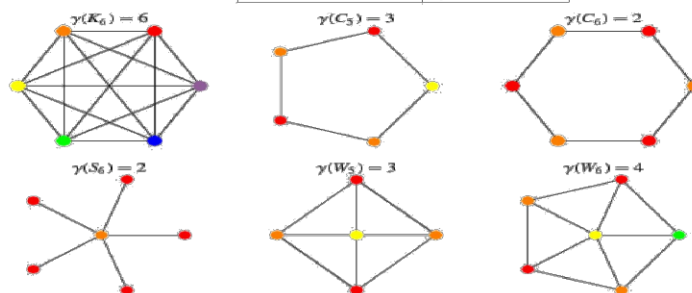A proper vertex coloring of the Petersen graph with 3 colors, the minimum number possible.

Vertex coloring

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.

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.
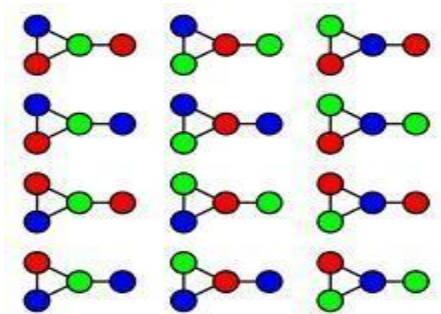
This graph can be 3-colored in 12 different ways.
The following table gives the chromatic number for familiar classes of graphs.

graph G	
complete graph K_n	
cycle graph C_n, $n > 1$	
star graph ,	2
wheel graph ,	
,	2
wheel graph ,	