DIGITAL NOTES
ON
DISCRETE MATHEMATICS

B.TECH II YEAR - I SEM
(2020-21)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

	L	T/P/D	C
II Year B.Tech IT -I Sem	**3**	**-/-/-**	**3**

(R18A506) DISCRETE MATHEMATICS

UNIT - I MATHEMATICAL LOGIC AND PREDICATES

Mathematical logic: Statements and notations, connectives, well-formed formulas, truth tables, tautology, equivalence implication; Normal forms: Disjunctive normal forms, conjunctive normal forms, principle disjunctive normal forms, principle conjunctive normal forms;

Predicate calculus: Predicative logic, statement functions, variables and quantifiers, free and bound variables, rules of inference, consistency, proof of contradiction, automatic theorem proving.

UNIT - II RELATIONS, FUNCTIONS AND LATTICES

Relations: Properties of binary relations, equivalence, compatibility and partial ordering relations, lattices, Hasse diagram; Functions: Inverse function, composition of functions, recursive functions; **Lattices:** Lattices as partially ordered sets; Definition and examples, properties of lattices, sub lattices, some special lattices.

UNIT - III ALGEBRAIC STRUCTURES AND COMBINATORICS

Algebraic structures: Algebraic systems, examples and general properties, semi groups and monoids, groups,sub groups, homomorphism, isomorphism, rings.

Combinatorics: The fundamental counting principles, permutations, disarrangements, combinations, permutations and combinations with repetitions, the binomial theorem, multinomial theorem, generalized inclusion exclusion principle.

UNIT - IV RECURRENCE RELATION

Recurrence relation: Generating functions, function of sequences calculating coefficient of generating function, recurrence relations, solving recurrence relation by substitution and generating functions, Characteristics roots solution of homogeneous recurrence relation.

UNIT - V GRAPHS AND TREES

Graphs: Basic concepts of graphs, isomorphic graphs, Euler graphs, Hamiltonian graphs, planar graphs, graph coloring, digraphs, directed acyclic graphs, weighted digraphs, region graph, chromatic numbers; **Trees:** Trees, spanning trees, minimal spanning trees.

TEXTBOOKS:

1. J. P. Tremblay, R. Manohar, Discrete Mathematical Structures with Applications to Computer Science, Tata McGraw Hill, India, 1st Edition,1997.

2. JoeL.Mott,AbrahamKandel,TheodoreP.Baker,―DiscreteMathematicsforComputerScientists and Mathematicians, Prentice Hall of India Learning Private Limited, New Delhi, India, 2nd Edition, 2010.

REFERENCE BOOKS:

1. Kenneth H. Rosen, ―Discrete Mathematics and Its Applications, Tata Mcgraw-Hill, New Delhi, India, 6th Edition,2012.

2. C. L. Liu, D. P. Mohapatra, ―Elements of Discrete Mathematics, Tata Mcgraw-Hill, India, 3rd Edition,2008.

3. Ralph P. Grimaldi, B. V. Ramana, ―Discrete and Combinatorial Mathematics - An Applied Introduction, Pearson Education, India, 5th Edition, 2011.

4. D. S. Malik, M. K. Sen, ―Discrete Mathematical Structures: Theory and Applications, Thomson Course Technology, India, 1st Edition, 2004.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

INDEX

UNIT-I

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):

- not
- conjunction
- disjunction
- conditional
- biconditional

Truth Tables:

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:

p	**¬p**
T	F

F	T

Logical conjunction

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

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

p	**q**	**p ∧ 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 ∧ q is true. For all other assignments of logical values to p and to q the conjunction p ∧ q is false.

It can also be said that if p, then p ∧ q is q, otherwise p ∧ 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:

p	**q**	**p Vq**
T	T	T
T	F	T
F	T	T
F	F	F

Logical implication

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

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:

p	**q**	**p ≡ q**
T	T	T
T	F	F
F	T	F
F	F	T

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.

- An atomic formula is a wff.
- If A is a wff, then 7A is also a wff.
- If A and B are wffs, then (A V B), (A ^ B), (A → B) and (A D B).
- If A is a wff and x is a any variable, then (x)A and ($x)A are wffs.
- 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:

- True and False are wffs.
- Each propositional constant (i.e. specific proposition), and each propositional variable (i.e. a variable representing propositions) are wffs.
- Each atomic formula (i.e. a specific predicate with variables) is a wff.
- If A, B, and C are wffs, then so areA, (A ∧ B), (A ∨B), (A->B),and (A<->B)

• If x is a variable (representing objects of the universe of discourse), and A is a wff, then so are ∀x A and x∃A .

The strings that cannot 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, taller∃(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

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

$\neg p \vee q$ is equivalent to $p \rightarrow q$. It is easily shown that the statements r and s are equivalent if and only if (r<->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 (1)

$(A \wedge B) \vee (! A \wedge C)$ (2)

$(A \wedge B \wedge ! A) \vee (C \wedge ! B) \vee (A \wedge ! C)$ (3)

$A \wedge B$ (4)

$A \vee (B \wedge C)$, (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 (1)

$(A \vee B) \wedge (! A \vee C)$ (2)

$A \vee B$ (3)

$A \wedge (B \vee C)$, (4)

PDNF:

It stands for Principal Disjunctive Normal Form. It refers to the Sum of Products, i.e., SOP. For eg. : If P, Q, R are the variables then (P . Q' . R) + (P' . Q . R) + (P . Q . R') is an example of an expression in PDNF. Here '+' i.e. sum is the main operator.

PCNF:

It stands for Principal Conjunctive Normal Form. It refers to the Product of Sums, i.e., POS. For eg. : If P, Q, R are the variables then (P + Q'+ R).(P'+ Q + R).(P + Q + R') is an example of an expression in PCNF. Here '.' i.e. product is the main operator.

Properties of PCNF and PDNF:

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

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.

Qantifiers:

In predicate logic, predicates are used alongside quantifiers to express the extent to which a predicate is true over a range of elements. Using quantifiers to create such propositions is called quantification.

There are two types of quantification-

1. Universal Quantification- Mathematical statements sometimes assert that a property is true for all the values of a variable in a particular domain, called the domain of discourse. Such a statement is expressed using universal quantification.

The universal quantification of P(x) for a particular domain is the proposition that asserts that P(x) is true for all values of x in this domain. The domain is very important here since it decides the possible values of x. The meaning of the universal quantification of P(x) changes when the domain is changed. The domain must be specified when a universal quantification is used, as without it, it has no meaning.

Formally,

The universal quantification of P(x) is the statement

"P(x) for all values of x in the domain"

The notation ∀P(x) denotes the universal quantification of P(x).

Here ∀is called the universal quantifier.

∀P(x) is read as "for all x P(x)".

Example 1: Let P(x) be the statement "x + 2 > x". What is the truth value of the statement ∀ xP(x)?

Solution: As x+2 is greater than x for any real number, so P(x) \equiv T for all x or ∀xP(x) \equiv T.

2. Existential Quantification- Some mathematical statements assert that there is an element with a certain property. Such statements are expressed by existential quantification. Existential quantification can be used to form a proposition that is true if and only if P(x) is true for at least one value of x in the domain.

Formally,

The existential quantification of P(x) is the statement

"There exists an element x in the domain such that P(x)"

The notation ∃P(x) denotes the existential quantification of P(x).

Here ∃ s called the existential quantifier.

∃P(x) is read as "There is atleast one such x such that P(x)".

Example : Let P(x) be the statement "x > 5″. What is the truth value of the statement ∃xP(x) ?

Solution: P(x) is true for all real numbers greater than 5 and false for all real numbers less than 5. So ∃ xP(x) \equiv T.

Free & Bound variables:

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

Have a look at the following formula:

$\neg(\text{THERAPIST}(x) \vee \forall x(\text{MORON}(x) \wedge \forall y\text{PERSON}(y)))$

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 occurrences of this variable within the quantification are bound.

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.

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∧Q =>P	} Simplification
I2	PQ∧ =>Q	
I3	P=>PVQ	} Addition
I4	Q =>PVQ	
I5	7P => P→ Q	
I6	Q => P→ Q	
I7	7(P→Q) =>P	
I8	7(P → Q) >Q	
I9	P, Q => P ∧ Q	
I10	7P, PVQ => Q	(disjunctive syllogism)
I11	P, P→ Q => Q	(modus ponens)
I12	7Q, P → Q => 7P	(modus tollens)
I13	P → Q, Q → R => P → R	(hypothetical syllogism)
I14	P V Q, P → Q, Q → R => R	(dilemma)

Equivalences

E1	77P <=>P	
E2	P ∧ Q <=> Q ∧P	} Commutative laws
E3	P V Q <=> Q V P	
E4	(P ∧ Q) ∧ R <=> P ∧ (Q ∧ R)	} Associative laws
E5	(P V Q) V R <=> PV (Q V R)	
E6	P ∧ (Q V R) <=> (P ∧ Q) V (P ∧ R)	} Distributive laws
E7	PV (Q ∧ R) <=> (P V Q) ∧ (PVR)	
E8	7(P ∧ Q) <=> 7P V7Q	
E9	7(P V Q) <=>7P ∧ 7Q	} De Morgan's laws

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 .Prove that R ∧ (P V Q) is a valid conclusion from the premises PVQ , Q → R, P → M and 7M.

Solution .

{1}	(1)	P → 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 → R	P
{1, 2, 4, 6}	(7)	R	T, (5), (6) and I11
{1, 2, 4, 6}	(8)	R ∧ (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 → S from the set of premises alone.

- 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 ∧ R then R → S can be derived from the premises P alone.
- Rule CP is also called the deduction theorem and is generally used if the conclusion is of the form R → S. In such cases, R is taken as an additional premise and S is derived from the given premises and R.

Example .Show that R → S can be derived from the premises P → (Q → 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 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

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

Solution.

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

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

Example Show that the following premises are inconsistent.

- If Jack misses many classes through illness, then he fails high school.
- If Jack fails high school, then he is uneducated.
- If Jack reads a lot of books, then he is not uneducated.
- 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→ Q, Q → S, R→ 7S and P^ R

{1}	(1)	P→Q	P
{2}	(2)	Q→ S	P
{1, 2}	(3)	P → S	T, (1), (2) and I13
{4}	(4)	R→ 7S	P
{4}	(5)	S → 7R	T, (4), and E18
{1, 2, 4}	(6)	P→7R	T, (3), (5) and I13
{1, 2, 4}	(7)	7PV7R	T, (6) and E16
{1, 2, 4}	(8)	7(P^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**

$$\begin{array}{l} p \wedge q \\ \hline \therefore p \end{array} \qquad (p \wedge q) \rightarrow p$$

Simplification

$$\begin{array}{l} p \\ \hline \therefore p \vee q \end{array} \qquad p \rightarrow (p \vee q)$$

Addition

$$\begin{array}{l} p \\ q \\ \hline \therefore p \wedge q \end{array} \qquad ((p) \wedge (q)) \rightarrow (p \wedge q)$$

Conjunction

$$\begin{array}{l} p \\ p \rightarrow q \\ \hline \therefore q \end{array} \qquad ((p \wedge (p \rightarrow q)) \rightarrow q$$

Modus ponens

$$\begin{array}{l} \neg q \\ p \rightarrow q \\ \hline \therefore \neg p \end{array} \qquad ((\neg q \wedge (p \rightarrow q)) \rightarrow \neg p$$

Modus tollens

$$\begin{array}{l} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array} \qquad ((p \rightarrow q) \wedge (q \rightarrow r)) \rightarrow (p \rightarrow r)$$

Hypothetical syllogism

$$\begin{array}{l} p \vee q \\ \neg p \\ \hline \therefore q \end{array} \qquad ((p \vee q) \wedge \neg p) \rightarrow q$$

Disjunctive syllogism

$$\begin{array}{l} p \vee q \\ \neg p \vee r \\ \hline \therefore q \vee r \end{array} \qquad ((p \vee q) \wedge (\neg p \vee r)) \rightarrow (q \vee r)$$

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), 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:

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

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:

- Assume that a given proposition is untrue.
- 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.

- To prove: "There is no largest prime number" by contradiction.
- Assume: There is a largest prime number, call it p.
- 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?
- N is at least as big as p+1 and so is larger than p and so, by Step 2, cannot be prime.
- 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

Introduction

The elements of a set may be related to one another. For example, the set of natural numbers. 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 aRb.

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.

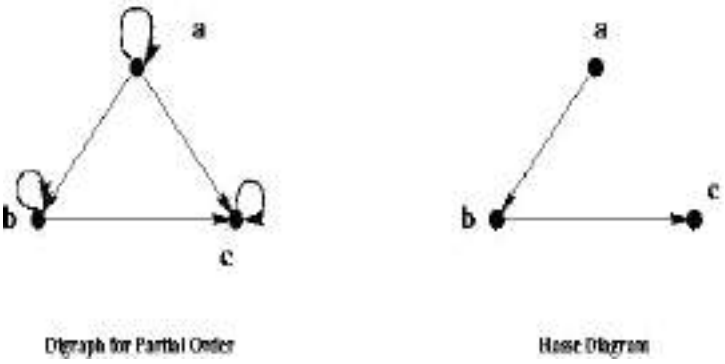
For example

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}

Hasse Diagram:

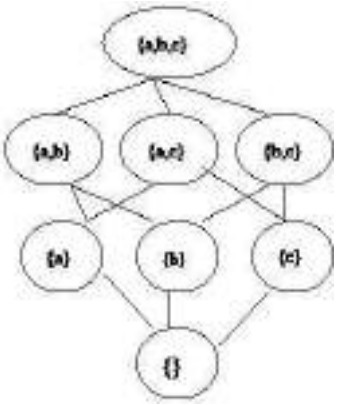
A Hasse diagram is a digraph for a poset which does not have loops and arcs implied by the transitivity.

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 defined 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**.

example: Let 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 $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$

(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.

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 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,

$f(x, 0) = 0$ (1)

$f(x, S(y)) = g(f(x, y), x)$ (2)

$g(x, 0) = x$ (3)

$g(x, S(y)) = S(g(x, y))$ (4)

defines $f(x, y)$ to be the function $x y$ 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

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 and y * a = b if and only if y = b * a .
- 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
- 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 the set. 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

i.e. if a * (b*c) = (a * b) * c for all a, b, c **Î** 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 Î M and there exists an element e in M such that a * e = e * a = a for all a Î 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:

An algebraic system (S, *) is a semigroup if the binary operation * is associative. If there exists an identity element e Î 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 o g)(a * b) = h(g(a* b))

= h(g(a) D g(b))

= h(g(a)) Å h(g(b))

= (h o g)(a) Å (h o 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.

= [x6+6C1x5.a+6C2.x4.a2 + 6C3x3a3 + 6C4x2a4 + 6C5xa5 +6C6a6] [x6-6C1x5a+6C2.x4.a2 – 6C3x3a3 + 6C4x2a4 – 6C5xa5 +6C6a6]

x6+6C2x4a2+6C4x2a4+6C6a6] x6+15x4(x2-1)+15x2(x2-1)2+(x2-1)3]

x6+15x6-15x4+15x6+15x2-30x4+x6-1-3x4+3x3] 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 km 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 that appear are taken to equal 1 (even when x equals zero). Alternatively, this can be written as

$$(x_1 + \cdots + x_m)^n = \sum_{|\alpha|=n} \binom{n}{\alpha} x^\alpha$$ **where α = (α1,α2,…,α**m) and x

Example

$(a + \mathbf{b} + \mathbf{c})^3 = a^3 + b^3 + c^3 + 3a^2b + 3a^2c + 3b^2c + 3b^2a + 3c^2a + 3c^2a + 6abc$

However calculation in above 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:

$$\binom{3}{2,0,1} = \frac{3!}{2! \cdot 0! \cdot 1!} = \frac{6}{2 \cdot 1 \cdot 1} = 3$$

$$\binom{3}{1,1,1} = \frac{3!}{1! \cdot 1! \cdot 1!} = \frac{6}{1 \cdot 1 \cdot 1} = 6$$

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

The principles of Inclusion – Exclusion:

In combinatorics (combinatorial mathematics), the **inclusion–exclusion principle** is a counting technique which generalizes the familiar method of obtaining the number of elements in the union of two finite sets; symbolically expressed as

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

where A and B are two finite sets and |S| indicates the cardinality of a set S (which may be considered as the number of elements of the set, if the set is finite). The formula expresses the fact that the sum of the sizes of the two sets may be too large since some elements may be counted twice. The double-counted elements are those in the intersection of the two sets and the count is corrected by subtracting the size of the intersection.

The principle is more clearly seen in the case of three sets, which for the sets A, B and C is given by

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

Ex: How many integers in {1,...,100} are not divisible by 2, 3 or 5?

Let S = {1,...,100} and P_1 the property that an integer is divisible by 2, P_2 the property that an integer is divisible by 3 and P_3 the property that an integer is divisible by 5. Letting A_i be the subset of S whose elements have property P_i we have by elementary counting: $|A_1| = 50$, $|A_2| = 33$, and $|A_3| = 20$. There are 16 of these integers divisible by 6, 10 divisible by 10 and 6 divisible by 15. Finally, there are just 3 integers divisible by 30, so the number of integers not divisible by any of 2, 3 or 5 is given by:

$$100 - (50 + 33 + 20) + (16 + 10 + 6) - 3 = 26.$$

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

$$\mathrm{EG}(a_n; x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}.$$ The exponential generating function of a sequence a_n is

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

Function of Sequences:

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

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

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

$$f(x) = \frac{1}{(x)_\infty} = \sum_{n=0}^{\infty} P(n)x^n = 1+x+2x^2+3x^3+\ldots$$

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

$$f(x) = \frac{x}{1-x-x^2} = \sum_{n=0}^{\infty} F_n x^n = x+x^2+2x^3+3x^4+\ldots$$

for the Fibonacci numbers F_n.

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

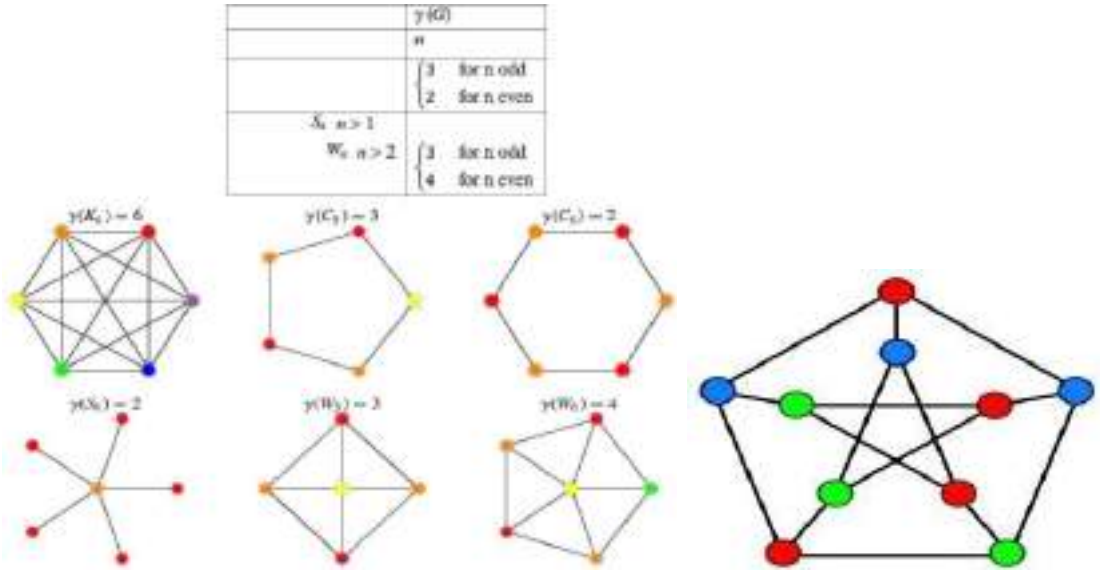
Calculating Coefficient of generating function:

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

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.

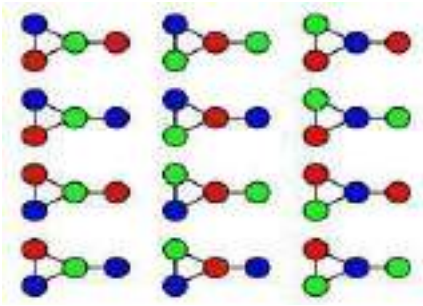


A proper vertex coloring of the Petersen graph with 3 colors, the minimum number possible.

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.

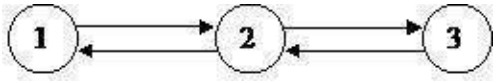
Directed Graphs

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.

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

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

Indegree of 1=1

Indegree 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.

Directed Acyclic Graph (DAG)

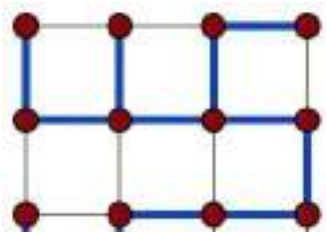
A directed acyclic graph (DAG) is a finite directed graph with no directed cycles. That is, it consists of finitely many vertices and edges (also called *arcs*), with each edge directed from one vertex to another, such that there is no way to start at any vertex *v* and follow a consistently-directed sequence of edges that eventually loops back to *v* again. Equivalently, a DAG is a directed graph that has a topological ordering, a sequence of the vertices such that every edge is directed from earlier to later in the sequence. Every directed acyclic graph has

- G has no simple cycles and has $n - 1$ edges.

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.



Given an undirected and connected graph G=(V,E), a spanning tree of the graph G is a tree that spans G(that is, it includes every vertex of G) and is a subgraph of G (every edge in the tree belongs to G)

Minimum Spanning Tree

The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

There are two famous algorithms for finding the Minimum Spanning Tree:

Kruskal's Algorithm

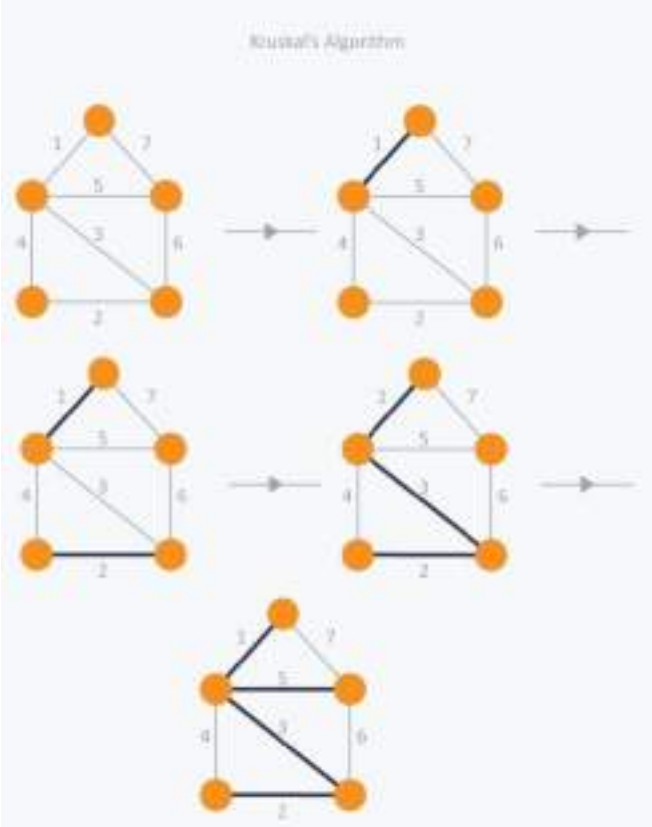
Kruskal's Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

In Kruskal's algorithm, at each iteration we will select the edge with the lowest weight. So, we will start with the lowest weighted edge first i.e., the edges with weight 1. After that we will select the second lowest weighted edge i.e., edge with weight 2. Notice these two edges

are totally disjoint. Now, the next edge will be the third lowest weighted edge i.e., edge with weight 3, which connects the two disjoint pieces of the graph. Now, we are not allowed to pick the edge with weight 4, that will create a cycle and we can't have any cycles. So we will select the fifth lowest weighted edge i.e., edge with weight 5. Now the other two edges will create cycles so we will ignore them. In the end, we end up with a minimum spanning tree with total cost 11 (= 1 + 2 + 3 + 5).



Prim's Algorithm

Prim's Algorithm also use Greedy approach to find the minimum spanning tree. In Prim's Algorithm we grow the spanning tree from a starting position. Unlike an **edge** in Kruskal's, we add **vertex** to the growing spanning tree in Prim's.

Algorithm Steps:

- Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.
- Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert the vertices, that are connected to growing spanning tree, into the Priority Queue.
- Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

In Prim's Algorithm, we will start with an arbitrary node (it doesn't matter which one) and mark it. In each iteration we will mark a new vertex that is adjacent to the one that we have already marked. As a greedy algorithm, Prim's algorithm will select the cheapest edge and mark the vertex. So we will simply choose the edge with weight 1. In the next iteration we have three options, edges with weight 2, 3 and 4. So, we will select the edge with weight 2 and mark the vertex. Now again we have three options, edges with weight 3, 4 and 5. But we can't choose edge with weight 3 as it is creating a cycle. So we will select the edge with weight 4 and we end up with the minimum spanning tree of total cost 7 (= 1 + 2 +4).