

**CMR ENGINEERING COLLEGE**

**KANDLAKOYA (V), MEDCHAL (M), HYDERABAD**

**STEP MATERIAL ON DISCRETE MATHEMATICS**

**DEPARTMENT OF CSE(AI&ML)**

**Subject : Discrete Mathematics Year: II Year/I Sem Academic Year:2021-2022**

# INSTITUTE VISION AND MISSION

#### VISION

To be recognized as a premier institution in offering value based and futuristic quality technical education to meet the technological needs of the society

#### MISSION

* To impart value based quality technical education through innovative teachingand learning methods
* To continuously produce employable technical graduates with advanced technical skills to meet the current and future technological needs of the society
* To prepare the graduates for higher learning with emphasis on academic andindustrial research

#### DEPARTMENT VISION AND MISSION

**VISION**

To produce globally competent and industry ready graduates in Computer Science & Engineering by imparting quality education with a know-how of cutting edge technology and holistic personality

#### MISSION

* + To offer high quality education in Computer Science & Engineering in orderto build core competence for the graduates by laying solid foundation in Applied Mathematics, and program framework with a focus on concept building
  + The department promotes excellence in teaching, research, andcollaborative activities to prepare graduates for professional career or higher studies
  + Creating intellectual environment for developing logical skills and problem solving strategies, thus to develop, able and proficient computer engineer to compete in the current global scenario

# PART –A

**UNIT WISE SHORT QUESTION AND ANSWERS**

# UNIT-I

#### 1 .Define the Proposition logic with example? [2M] Ans:

A proposition is a collection of declarative statements that has either a truth value "true” or a truth value "false". A propositional consists of propositional variables and connectives. We denote the propositional variables by capital letters (A, B, C etc). The connectives connect the propositional variables.

Some examples of Propositions are given below: "Man is Mortal", it returns truth value “TRUE” as T.

"12 + 9 = 3 − 2", it returns truth value “FALSE” as F.

The following is not a Proposition

"A is less than 2".

It is because unless we give a specific value of A, we cannot say whether the statement is true orfalse.

#### Define Tautology , Contradiction and Contingency [3M]

**Ans:**

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

**truth** or a **tautology.**

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

**Contingency:** A statement formula which is neither a tautology nor a contradiction is known as a **contingency**

#### Write the negation of the following statements. [3M]

(a). Jan will take a job in industry or go to graduate school. (b). James will bicycle or run tomorrow.

(c). If the processor is fast then the printer is slow.

**Solution:** (a). Let *P* : Jan will take a job in industry.

*Q*: Jan will go to graduate school.

The given statement can be written in the symbolic as *P* ∨ *Q*. The negation of *P* ∨ *Q* is given by *¬*(*P* ∨ *Q*).

*¬*(*P* ∨ *Q*) ⇔ *¬P* 𝖠 *¬Q.*

*¬P* 𝖠 *¬Q*: Jan will not take a job in industry and he will not go to graduate school.

3

(b). Let *P* : James will bicycle.

*Q*: James will run tomorrow.

The given statement can be written in the symbolic as *P* ∨ *Q*. The negation of *P* ∨ *Q* is given by *¬*(*P* ∨ *Q*).

*¬*(*P* ∨ *Q*) ⇔ *¬P* 𝖠 *¬Q.*

*¬P* 𝖠 *¬Q*: James will not bicycle and he will not run tomorrow. (c). Let *P* : The processor is fast.

*Q*: The printer is slow.

#### 4 Write the given statement can be written in the symbolic as P → Q.

ANS:

The negation of *P → Q* is given by *¬*(*P → Q*).

*¬*(*P → Q*) ⇔ *¬*(*¬P* ∨ *Q*) ⇔ *P* 𝖠 *¬Q. P* 𝖠 *¬Q*: The processor is fast and the printer is fast.

The given statement can be written in the symbolic as *P → Q*. The negation of *P → Q* is given by *¬*(*P → Q*).

*¬*(*P → Q*) ⇔ *¬*(*¬P* ∨ *Q*) ⇔ *P* 𝖠 *¬Q. P* 𝖠 *¬Q*: The processor is fast and the printer is fast.

#### Use Demorgans laws to write the negation of each statement.

* 1. I want a car and worth a cycle.
  2. My cat stays outside or it makes a mess. (c). I‘ve fallen and I can‘t get up.

(d). You study or you don‘t get a good grade.

#### Solution:

1. I don‘t want a car or not worth a cycle.

(b).My cat not stays outside and it does not make a mess. (c).I have not fallen or I can get up.

(d).You can not study and you get a good grade.

#### Define Predicate Logic ANS

Predicate logic is an extension of Propositional logic. It adds the concept of predicates and quantifiers to better capture the meaning of statements that cannot be adequately expressed by propositional logic.

Consider the statement, “x is greater than 3″. It has two parts.

4

The first part, the variable x, is the subject of the statement.

The second part, “is greater than 3”, is the predicate.

It refers to a property that the subject of the statement can have.

The statement “ x is greater than 3″ can be denoted by p(x) where p denotes the predicate “is greater than 3” and x is the variable

#### Define Quantifier and types of quantifiers ANS:

The variable of predicates is quantified by quantifiers. There are two types of

quantifier in Predicate logic − Universal Quantifier and Existential Quantifier.

# Universal Quantifier:

Universal quantifier states that the statements within its scope are true for every value of the specific variable. It is denoted by the symbol ∀.

∀x P(x) is read as for every value of x, P(x) is true.

**Example:** "Man is mortal" can be transformed into the propositional form ∀x P(x) where P(x) is the predicate which denotes x is mortal and the universe of discourse is all men.

# Existential Quantifier:

Existential quantifier states that the statements within its scope are true for some values of the specific variable. It is denoted by the symbol ∃.∃x P(x)is read as for some values of x, P(x) is true.

**Example:** "Some people are dishonest" can be transformed into the propositional form ∃x P(x)where P(x) is the predicate which denotes x is dishonest and the universe of discourse is some people

#### Explain the types of normal forms

ANS:

1. *Disjunctive Normal* Form
2. *Conjunctive Normal* Form
3. Principal Disjunctive Normal Form
4. Principal Conjunctive Normal Form

#### Demonstrate that R is a valid inference from the premises P → Q, Q → R, and P .

Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}* | (1) | *P → Q* | Rule P |
| *{*2*}* | (2) | *P* | Rule P, |
| *{*1*,* 2*}* | (3) | *Q* | Rule T, (1), (2), and *I*13 |
| *{*4*}* | (4) | *Q → R* | Rule P |
| *{*1*,* 2*,* 4*}* | (5) | *R* | Rule T, (3), (4), and *I*13 |

#### Example: Show that R∨S follows logically from the premises C ∨D, (C ∨D) → ¬H, ¬H → (A 𝖠¬B), and (A 𝖠 ¬B) → (R ∨ S). Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}* | (1) | (*C* ∨ *D*) *→ ¬H* | Rule P |
| *{*2*}* | (2) | *¬H →* (*A* 𝖠 *¬B*) | Rule P |
| *{*1*,* 2*}* | (3) | (*C* ∨ *D*) *→* (*A* 𝖠 *¬B*) | Rule T, (1), (2), and *I*13 |
| *{*4*}* | (4) | (*A* 𝖠 *¬B*) *→* (*R* ∨ *S*) | Rule P |
| *{*1*,* 2*,* 4*}* | (5) | (*C* ∨ *D*) *→* (*R* ∨ *S*) | Rule T, (3), (4), and *I*13 |
| *{*6*}* | (6) | *C* ∨ *D* | Rule P |
| *{*1*,* 2*,* 4*,* 6*}* | (7) | *R* ∨ *S* | Rule T, (5), (6), and *I*11 |

Hence the result.

#### Show that S ∨R is tautologically implied by (P ∨Q)𝖠(P → R)𝖠(Q → S).

Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}* | (1) | *P* ∨ *Q* | Rule P |
| *{*1*}* | (2) | *¬P → Q* | Rule T, (1) *P → Q* ⇔ *¬P* ∨ *Q* |
| *{*3*}* | (3) | *Q → S* | Rule P |
| *{*1*,* 3*}* | (4) | *¬P → S* | Rule T, (2), (3), and *I*13 |
| *{*1*,* 3*}* | (5) | *¬S → P* | Rule T, (4), *P → Q* ⇔ *¬Q →*  *¬P* |
| *{*6*}* | (6) | *P → R* | Rule P |
| *{*1*,* 3*,* 6*}* | (7) | *¬S → R* | Rule T, (5), (6), and *I*13 |

*{*1*,* 3*,* 6*}* (8) *S* ∨ *R* Rule T, (7) and *P → Q* ⇔ *¬P* ∨*Q*

Hence the result.

#### Show that R 𝖠 (P ∨ Q) is a valid conclusion from the premises

*P* ∨ *Q*, *Q → R*, *P → M*, and *¬M*.

|  |  |  |  |
| --- | --- | --- | --- |
| Solution: |  |  |  |
| *{*1*}* | (1) | *P → M* | Rule P |
| *{*2*}* | (2) | *¬M* | Rule P |
| *{*1*,* 2*}* | (3) | *¬P* | Rule T, (1), (2), and *I*12 |
| *{*4*}* | (4) | *P* ∨ *Q* | Rule P |
| *{*1*,* 2*,* 4*}* | (5) | *Q* | Rule T, (3), (4), and *I*10 |
| *{*6*}* | (6) | *Q → R* | Rule P |

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*,* 2*,* 4*,* 6*}* | (7) | *R* | Rule T, (5), (6), and *I*11 |
| *{*1*,* 2*,* 4*,* 6*}* | (8) | *R* 𝖠 (*P* ∨ *Q*) | Rule T, (4), (7) and *I*9 |

Hence the result.

#### Show I12 : ¬Q, P → Q ⇒ ¬P . Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}* | (1) | *P → Q* | Rule P |
| *{*1*}* | (2) | *¬Q → ¬P* | Rule T, (1), and *P → Q* ⇔ *¬Q →*  *¬P* |
| *{*3*}* | (3) | *¬Q* | Rule P |
| *{*1*,* 3*}* | (4) | *¬P* | Rule T, (2), (3), and *I*11 |

Hence the result.

#### UNIT-II

**UNIT-III**

#### Define Mathematical Induction

**A. Mathematical Induction** is a mathematical technique which is used to prove a statement, a formula or a theorem is true for every natural number. The technique involves two steps to prove a statement, as stated below −

**Step 1(Base step)** − It proves that a statement is true for the initial value.

**Step 2(Inductive step)** − It proves that if the statement is true for the nth iteration (or number *n*), then it is also true for *(n+1 )th* iteration ( or number *n+1*).

#### Define Strong Induction

* 1. Strong Induction is another form of mathematical induction. Through this induction technique, we can prove that a propositional function, P(n) is true for all positive integers n

#### What is Recursion

* 1. The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function. Using recursive algorithm, certain problems can be solved quite easily. Examples of such problems are

[Towers of Hanoi (TOH),](http://quiz.geeksforgeeks.org/c-program-for-tower-of-hanoi/) [Inorder/Preorder/Postorder Tree Traversals,](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/) [DFS of Graph](https://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/), etc.

#### What is Partial Correctness (Hoare Triple).

* 1. Program, or program segment, S is said to be partially correct with respect to the initial assertion p and the final assertion q if whenever p is true for the input values of S and S terminates, then q is true for the output values of S. The notation p{S}q indicates that the program, or program segment, S is partially correct with respect to the initial assertion p and the final assertion q.

#### What is Well Ordering

* 1. The validity of both the principle of mathematical induction and strong induction follows from a fundamental axiom of the set of integers, the well-ordering property. The well-

ordering property states that every nonempty set of nonnegative integers has a least element.

#### Define Structural Induction

* 1. It is a generalization of mathematical induction over natural numbers and can be further generalized to arbitrary Noetherian induction.

#### Define Program Correctness

A. A program is correct if it produces the correct output for every possible input. A program

has partial correctness if it produces the correct output for every input for which the program eventually halts. Therefore, a program is correct if and only if it has partial correctness and terminates.

#### Difference between Recursion and induction

A. **Induction** is when to prove that P (n) holds you need to first reduce your goal to P (0) by repeatedly applying the inductive case and then prove the resulting goal using the base

case. Similarly, **recursion** is when you first define a base case and then define the further values in terms of the previous ones.

#### Define recursive Algorithm

* 1. An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input.

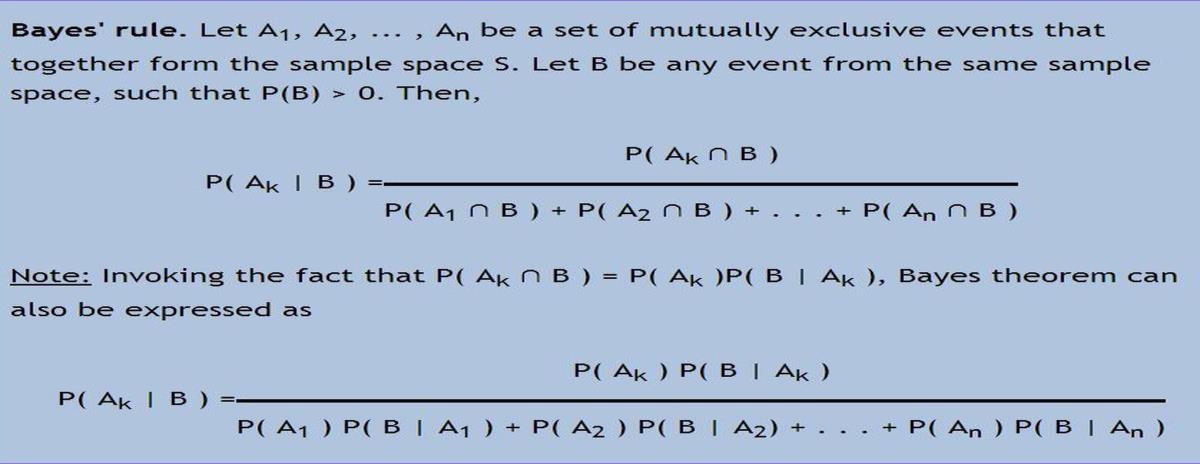
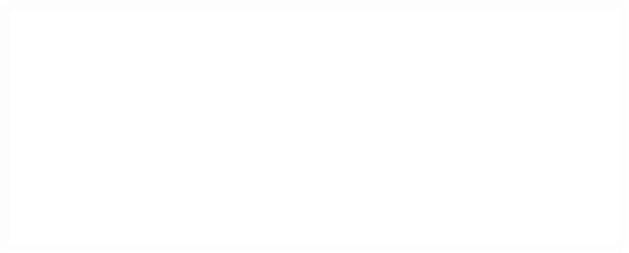
#### List the steps of Strong Induction

* 1. **Step 1(Base step)** − It proves that the initial proposition P(1) true.

**Step2(Inductive step)** − It proves that the conditional

statement [P(1) 𝖠P(2)𝖠P(3)𝖠⋯𝖠P(k)]→P(k+1)[P(1)𝖠P(2)𝖠P(3)𝖠⋯𝖠P(k)]→P(k+1) is true for positive integers k.

#### UNIT-IV



1. **Define the terms experiment, sample space.**
   1. An experiment is a procedure that yields one of a given set of possible outcomes. The sample space of the experiment is the set of possible outcomes.

An event is a subset of the sample space

#### Define probability of event

A. If S is a finite nonempty sample space of equally likely outcomes, and E is an event, that is, a subset of S, then the probability of E is p(E) = |E| / |S|

#### Define Conditional Probability

A. Let E and F be events with p(F ) > 0. The conditional probability of E given F, denoted by p(E | F ), is defined as p(E | F ) = p(E ∩ F ) / p(F ) .

#### What is a Bernoulli trial

A. A possible outcome of a Bernoulli trial is called a success or a failure. If p is the probability of a success and q is the probability of a failure, it follows that p + q = 1.

#### What is Random Variable.

A random variable is a function from the sample space of an experiment to the set of real numbers. That is, a random variable assigns a real number to each possible outcome.

#### Define Bayes Theorem

1. **What is a expected value**
   1. The expected value of a random variable is the sum over all elements in a sample space of the product of the probability of the element and the value of the random variable at this element.

#### Define Geometric Distribution

* 1. A random variable X has a geometric distribution with parameter p if p(X = k) = (1 − p)k−1p for k = 1, 2, 3,..., where p is a real number with 0 ≤ p ≤ 1.

#### Define Variance

* 1. The variance of a random variable helps us characterize how widely a random variable is distributed. In particular, it provides a measure of how widely X is distributed about its expected value.

#### What is a Probabilistic method

* 1. A technique for proving the existence of objects in a set with certain properties that proceeds by assigning probabilities to objects and showing that the probability that an object has these properties is positive.

#### List Applications of Recurrence Relations

Fibonacci Numbers The Tower of Hanoi

Codeword Enumeration Catalan numbers,

#### Define Recurrence Relation

* 1. recurrence relation: a formula expressing terms of a sequence, except for some initial terms, as a function of one or more previous terms of the sequence

#### What is Dynamic Programming

* 1. dynamic programming: an algorithmic paradigm that finds the solution to an optimization problem by recursively breaking down the problem into overlapping subproblems and combining their solutions with the help of a recurrence relation.

#### what is Divide and conquer procedure.

divide-and-conquer algorithm: an algorithm that solves a problem recursively by splitting it into a fixed number of smaller non-overlapping subproblems of the same type

#### What is Derangement

* 1. A permutation of objects such that no object is in its original place

#### Define Master Theorem

A.

#### Define Generating Function

* 1. generating function of a sequence: the formal series that has the nth term of the sequence as the coefficient of xn

#### Define linear homogeneous recurrence relation with constant coefficients

* 1. linear homogeneous recurrence relation with constant coefficients: a recurrence relation that expresses the terms of a sequence, except initial terms, as a linear combination of previous terms.

#### Define Linear nonhomogeneous recurrence relation with constant coefficients

* 1. Linear nonhomogeneous recurrence relation with constant coefficients: a recurrence relation that expresses the terms of a sequence, except for initial terms, as a linear combination of previous terms plus a function that is not identically zero that depends only on the index.

#### What is sieve of Eratosthenes

* 1. sieve of Eratosthenes: a procedure for finding the primes less than a specified positive integer

UNIT-V

#### Define Graph

* 1. A graph G = (V , E) consists of V , a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with it, called its endpoints. An edge is said to connect its endpoints.

#### Define Finite , infinite graph

* 1. A graph with an infinite vertex set or an infinite number of edges is called an infinite graph, and in comparison, a graph with a finite vertex set and a finite edge set is called a finite graph

#### Define Pseudograph

* 1. Graphs that may include loops, and possibly multiple edges connecting the same pair of vertices or a vertex to itself, are sometimes called pseudographs.

#### What is a Digraph

* 1. A directed graph (or digraph) (V , E) consists of a nonempty set of vertices V and a set of directed edges (or arcs) E. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to start at u and end at v.

#### What is a degree of a vertex

A. The degree of a vertex in an undirected graph is the number of edges incident with it, except that a

loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex v is denoted by deg(v).

#### Define Handshaking Theorem

* 1. THE HANDSHAKING THEOREM Let G = (V , E) be an undirected graph with m edges. Then 2m = ∑ v E V deg(v). (Note that this applies even if multiple edges and loops are present.)

#### What is a Complete graph, isolated and pendant vertex

* 1. A complete graph on n vertices, denoted by Kn, is a simple graph that contains exactly one edge between each pair of distinct vertices. isolated vertex: a vertex of degree zero

pendant vertex: a vertex of degree one

#### Define Bipartite Graph

A. A simple graph G is called bipartite if its vertex set V can be partitioned into two disjoint sets V1 and V2 such that every edge in the graph connects a vertex in V1 and a vertex in V2 (so that no edge

in G connects either two vertices in V1 or two vertices in V2). When this condition holds, we call the pair (V1, V2) a bipartition of the vertex set V of G

Or

A simple graph is bipartite if and only if it is possible to assign one of two different colors to each vertex of the graph so that no two adjacent vertices are assigned the same color

#### What is complete Bipartite Graph

* 1. complete bipartite graph Km,n is a graph that has its vertex set partitioned into two subsets of m and n vertices, respectively with an edge between two vertices if and only if one vertex is in the first subset and the other vertex is in the second subset.

#### Define subgraph

* 1. A subgraph of a graph G = (V , E) is a graph H = (W, F ), where W ⊆ V and F ⊆ E. A subgraph H of G is a proper subgraph of G if H ≠ G.

#### What is isomorphic graph

* 1. The simple graphs G1 = (V1, E1) and G2 = (V2, E2) are isomorphic if there exists a oneto-one and onto function f from V1 to V2 with the property that a and b are adjacent in G1 if and only if f (a)

and f (b) are adjacent in G2, for all a and b in V1. Such a function f is called an isomorphism.

#### Define Path, connected graph, Strongly connected graph

A path is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph

* An undirected graph is called connected if there is a path between every pair of distinct vertices of the graph
* A directed graph is strongly connected if there is a path from a to b and from b to a whenever a and b are vertices in the graph

#### Define terms Euler Circuit , Euler path, Hamilton path , Hamilton circuit

A. An Euler circuit in a graph G is a simple circuit containing every edge of G. An Euler path in G is a simple path containing every edge of G.

Hamilton path: a path in a graph that passes through each vertex exactly once Hamilton circuit: a circuit in a graph that passes through each vertex exactly once

#### What is traveling salesperson problem

A. The traveling salesperson problem asks for the circuit of minimum total weight in a weighted, complete, undirected graph that visits each vertex exactly once and returns to its starting point.

#### Define Planar Graph

* 1. A graph is called planar if it can be drawn in the plane without any edges crossing (where a crossing of edges is the intersection of the lines or arcs representing them at a point other than their common endpoint). Such a drawing is called a planar representation of the graph

#### Define the terms Graph coloring , chromatic number

* 1. A coloring of a simple graph is the assignment of a color to each vertex of the graph so that no two adjacent vertices are assigned the same color.

The chromatic number of a graph is the least number of colors needed for a coloring of this graph.

The chromatic number of a graph G is denoted by χ (G). (Here χ is the Greek letter chi.)

#### Define Tree, forest ,rooted tree

* 1. tree: a connected undirected graph with no simple circuits forest: an undirected graph with no simple circuits

rooted tree: a directed graph with a specified vertex, called the root, such that there is a unique path to every other vertex from this root

#### What is a decision tree and game tree.

* 1. Decision tree: a rooted tree where each vertex represents a possible outcome of a decision and the leaves represent the possible solutions of a problem

game tree: a rooted tree where vertices represents the possible positions of a game as it progresses and edges represent legal moves between these positions

#### Define Huffman coding

* 1. Huffman coding: a procedure for constructing an optimal binary code for a set of symbols, given the frequencies of these symbols

#### Write about two algorithms which gives minimum spanning tree

A.

* Prim’s algorithm: a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges with minimal weight among all edges incident to a vertex already in the tree so that no edge produces a simple circuit when it is added
* Kruskal’s algorithm: a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges of least weight that are not already in the tree such that no edge produces a simple circuit when it is added

#### UNIT-V

56

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PART –B**  **UNIT WISE ESSAY QUESTION AND ANSWERS UNIT-I**  **1. Define Logical connective and Construct the truth tables? ANS :**  **Defination:** The words or phrases or symbols which are used to make a proposition by two or more propositions are called **logical connectives** or **simply connectives.**  There are five basic connectives called negation, conjunction, disjunction, conditional and biconditional. which areOR () , AND () , Negation/ NOT (¬), Conditional or Implication / if-then (→), Bi conditional or If and only if ().  **Negation (¬)** − The negation of a proposition A (written as ¬A) is false when A is true and is true when A is false.  The truth table is as follows –  **A ¬A**  True False  False True  **AND (**  **): −** The AND operation of two propositions A and B (written as A  B) is true if both the propositional variable A and B is true. The truth table is as follows − | | | | |
|  | **A** | **B** | **A****B** | 57 |
| True | True | False |
| True | False | False |
| False | True | False |
| False | False | True |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**OR** () − The OR operation of two propositions A and B (written as A  B) is true if at least any of the propositional variable A or B is true.The truth table is as follows −

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A**  **B** |
| T | T | T |
| T | F | T |
| F | T | T |
| F | F | F |

**Implication / if-then (→)** − An implication A→B is False if A is true and B is false. The rest cases are true.

The truth table is follows

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A → B** |
| True | True | True |
| True | False | False |
| False | True | True |
| False | False | True |

**If and only if** (): A  B is bi-conditional logical connective which is true when p and q are both false or both are true. The truth table is as follows −

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **A** **B** |
| True | True | True |
| True | False | False |
| False | True | False |

58

|  |  |  |
| --- | --- | --- |
| False | False | True |

#### Write a Logica equivalence formulae

**ANS:**

Equivalence Formulas:

* 1. Idempotent laws:
     1. P ∨ P ⇔ P (b) P 𝖠 P ⇔ P
  2. Associative laws:

(a) (P ∨ Q) ∨ R ⇔ P ∨ (Q ∨ R) (b) (P 𝖠 Q) 𝖠 R ⇔ P 𝖠 (Q 𝖠 R)

* 1. Commutative laws:
     1. P ∨ Q ⇔ Q ∨ P (b) P 𝖠 Q ⇔ Q 𝖠 P
  2. Distributive laws:

P ∨ (Q 𝖠 R) ⇔ (P ∨ Q) 𝖠 (P ∨ R) P 𝖠 (Q ∨ R) ⇔ (P 𝖠 Q) ∨ (P 𝖠 R)

* 1. Identity laws:
     1. (i) P ∨ F ⇔ P (ii) P ∨ T ⇔ T
     2. (i) P 𝖠 T ⇔ P (ii) P 𝖠 F ⇔ F
  2. Component laws:

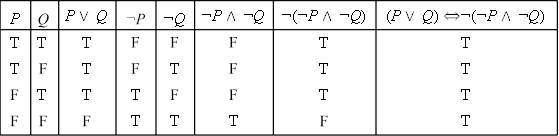
|  |  |  |
| --- | --- | --- |
| (a) (i) P ∨ ¬P ⇔ T | (ii) P 𝖠 ¬P ⇔ F | . |
| (b) (i) ¬¬P ⇔ P  7. Absorption laws: | (ii) ¬T ⇔ F , ¬F ⇔ T |  |

(a) P ∨ (P 𝖠 Q) ⇔ P (b) P 𝖠 (P ∨ Q) ⇔ P

8. Demorgan‘s laws:

(a) ¬(P ∨ Q) ⇔ ¬P 𝖠 ¬Q (b) ¬(P 𝖠 Q) ⇔ ¬P ∨ ¬Q

**3. (a) Prove P** ∨ **Q** ⇔ **¬(¬P** 𝖠 **¬Q).**

Solution

59

As *P* ∨ *Q ¬*(*¬P* 𝖠 *¬Q*) is a tautology, then *P* ∨ *Q* ⇔ *¬*(*¬P* 𝖠 *¬Q*).

#### Prove (P → Q) ⇔ (¬P ∨ Q).

Solution:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *P* | *Q* | *P → Q* | *¬P* | *¬P* ∨ *Q* | (*P → Q*) (*¬P* ∨*Q*) |
| T | T | T | F | T | T |
| T | F | F | F | F | T |
| F | T | T | T | T | T |
| F | F | T | T | T | T |

As (*P → Q*) (*¬P* ∨ *Q*) is a tautology then (*P → Q*) ⇔ (*¬P* ∨ *Q*).

#### 4. (a). Prove that P → (Q → R) ⇔ P → (¬Q ∨ R) ⇔ (P 𝖠 Q) → R.

Solution: *P →* (*Q → R*) ⇔ *P →* (*¬Q* ∨ *R*) [∵ *Q → R* ⇔ *¬Q* ∨ *R*]

- *¬P* ∨ (*¬Q* ∨ *R*) [∵ *P → Q* ⇔ *¬P* ∨ *Q*]

* (*¬P* ∨ *¬Q*) ∨ *R* [by Associative laws]
* *¬*(*P* 𝖠 *Q*) ∨ *R* [by De Morgan‘s laws]

- (*P* 𝖠 *Q*) *→ R*[∵ *P → Q* ⇔ *¬P* ∨ *Q*]*.*

#### Prove that (P → Q) 𝖠 (R → Q) ⇔ (P ∨ R) → Q.

Solution: (*P → Q*) 𝖠 (*R → Q*) ⇔(*¬P* ∨ *Q*) 𝖠 (*¬R* ∨ *Q*)

-(*¬P* 𝖠 *¬R*) ∨ *Q* ⇔

*¬*(*P* ∨ *R*) ∨ *Q* ⇔ *P* ∨

*R → Q*

#### ( c ). Prove that P → (Q → P ) ⇔ ¬P → (P → Q).

Solution: *P→* (*Q → P* ) ⇔ *¬P* ∨ (*Q → P* )

- *¬P* ∨ (*¬Q* ∨ *P* )

-(*¬P* ∨ *P* ) ∨ *¬Q*

* *T* ∨ *¬Q*
* *T*

and

*¬P →* (*P → Q*) ⇔ *¬*(*¬P* ) ∨ (*P → Q*)

- *P* ∨ (*¬P* ∨ *Q*) ⇔ 60

(*P* ∨ *¬P* ) ∨ *Q* ⇔ *T*

∨*Q*

- *T*

So, *P →* (*Q → P* ) ⇔ *¬P →* (*P → Q*).

**5. Prove that (¬P** 𝖠 **(¬Q** 𝖠 **R))** ∨ **(Q** 𝖠 **R)** ∨ **(P** 𝖠 **R)** ⇔ **R.**

Solution: (*¬P* 𝖠 (*¬Q* 𝖠 *R*)) ∨ (*Q* 𝖠 *R*) ∨ (*P* 𝖠 *R*)

* + - * ((*¬P* 𝖠 *¬Q*) 𝖠 *R*) ∨ ((*Q* ∨ *P* ) 𝖠 *R*) [Associative and Distributive laws]
      * (*¬*(*P* ∨ *Q*) 𝖠 *R*) ∨ ((*Q* ∨ *P* ) 𝖠 *R*) [De Morgan‘s laws]
      * (*¬*(*P* ∨ *Q*) ∨ (*P* ∨ *Q*)) 𝖠 *R* [Distributive laws]

- *T*𝖠 *R* [∵ *¬P* ∨ *P* ⇔ *T* ]

* + - * *R*

**6. Show ((P** ∨ **Q)** 𝖠 **¬(¬P** 𝖠 **(¬Q** ∨ **¬R)))** ∨ **(¬P** 𝖠 **¬Q)** ∨ **(¬P** 𝖠 **¬R) is tautology.**

Solution: By De Morgan‘s laws, we have

*¬P* 𝖠 *¬Q* ⇔ *¬*(*P* ∨ *Q*)

*¬P* ∨ *¬R* ⇔ *¬*(*P* 𝖠 *R*)

Therefore Also

(*¬P* 𝖠 *¬Q*) ∨(*¬P* 𝖠 *¬R*) ⇔ *¬*(*P* ∨ *Q*) ∨ *¬*(*P* 𝖠 *R*)

⇔ *¬*((*P* ∨ *Q*) 𝖠 (*P* ∨ *R*))

*¬*(*¬P* 𝖠 (*¬Q* ∨ *¬R*)) ⇔ *¬*(*¬P* 𝖠 *¬*(*Q* 𝖠 *R*))

- *P* ∨ (*Q* 𝖠 *R*)

- (*P* ∨ *Q*) 𝖠 (*P* ∨ *R*)

Hence ((*P* ∨ *Q*) 𝖠 *¬*(*¬P* 𝖠 (*¬Q* ∨ *¬R*))) ⇔ (*P* ∨ *Q*) 𝖠 (*P* ∨ *Q*) 𝖠 (*P* ∨ *R*)

⇔(*P* ∨ *Q*) 𝖠 (*P* ∨ *R*) Thus ((*P* ∨ *Q*) 𝖠 *¬*(*¬P* 𝖠 (*¬Q* ∨ *¬R*))) ∨(*¬P* 𝖠 *¬Q*) ∨(*¬P* 𝖠 *¬R*)

- [(*P* ∨ *Q*) 𝖠 (*P* ∨ *R*)] ∨ *¬*[(*P* ∨ *Q*) 𝖠 (*P* ∨ *R*)]

- *T*

Hence the given formula is a tautology.

#### Show that (P 𝖠 Q) → (P ∨ Q) is a tautology.

Solution: (*P* 𝖠 *Q*) *→* (*P* ∨ *Q*) ⇔ *¬*(*P* 𝖠 *Q*) ∨ (*P* ∨ *Q*) [∵ *P → Q* ⇔ *¬P* ∨ *Q*]

* (*¬P* ∨ *¬Q*) ∨ (*P* ∨ *Q*) [by De Morgan‘s laws]
  + (*¬P* ∨ *P* ) ∨ (*¬Q* ∨ *Q*) [by Associative laws and commutative laws]
  + (*T* ∨ *T* )[by negation laws]
  + *T*

Hence, the result.

#### Write the negation of the following statements. (a). If it is raining, then the game is canceled.

**(b). If he studies then he will pass the examination.**

#### Are (p → q) → r and p → (q → r) logically equivalent? Justify your answer by using the rules of logic to simply both expressions and also by using truth tables.

Solution: (*p → q*) *→ r* and *p →* (*q → r*) are not logically equivalent because

and

Method I: Consider

(*p → q*) *→ r* ⇔ (*¬p* ∨ *q*) *→ r*

- *¬*(*¬p* ∨ *q*) ∨ *r* ⇔

(*p* 𝖠 *¬q*) ∨ *r*

- (*p* 𝖠 *r*) ∨ (*¬q* 𝖠 *r*)

*p →* (*q → r*) ⇔ *p →* (*¬q* ∨ *r*)

- *¬p* ∨ (*¬q* ∨ *r*) ⇔

*¬p* ∨ *¬q* ∨ *r.*

Method II: (Truth Table Method)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *p* | *q* | *r* | *p → q* | (*p → q*) *→ r* | | *q → r* | *p →* (*q → r*) |
| T | T | T | T |  | T | T | T |
| T | T | F | T |  | F | F | F |
| T | F | T | F |  | T | T | T |
| T | F | F | F |  | T | T | T |
| F | T | T | T |  | T | T | T |
| F | T | F | T |  | F | F | T |
| F | F | T | T |  | T | T | T |
| F | F | F | T |  | F | T | T |

#### What is duality? Write the dual of the following formulas:

1. **(P** ∨ **Q)** 𝖠 **R**
2. **(P** 𝖠 **Q)** ∨ **T**

**(iii). (P** 𝖠 **Q)** ∨ **(P** ∨ **¬(Q** 𝖠 **¬S))**

**ANS**: Two formulas A and A∗ are said to be duals of each other if either one can be obtained from the other by replacing 𝖠 by ∨ and ∨ by 𝖠.

The connectives ∨ and 𝖠 are called duals of each other.

If the formula A contains the special variable T or F , then A∗, its dual is obtained by replacing T by F and F by T in addition to the above mentioned interchanges.

Solution: The duals of the formulas may be written as (i). (*P* 𝖠 *Q*) ∨ *R*

(ii). (*P* ∨ *Q*) 𝖠 *F*

(iii). (*P* ∨ *Q*) 𝖠 (*P* 𝖠 *¬*(*Q* ∨ *¬S*))

Result 1: The negation of the formula is equivalent to its dual in which every variable is replaced by its negation.

#### Prove that

**(a). ¬(P** 𝖠 **Q) → (¬P** ∨ **(¬P** ∨ **Q))** ⇔ **(¬P** ∨ **Q)**

**(b). (P** ∨ **Q)** 𝖠 **(¬P** 𝖠 **(¬P** 𝖠 **Q))** ⇔ **(¬P** 𝖠 **Q)**

Solution: (a).*¬*(*P* 𝖠 *Q*) *→* (*¬P* ∨ (*¬P* ∨ *Q*)) ⇔ (*P* 𝖠 *Q*) ∨ (*¬P* ∨ (*¬P* ∨ *Q*)) [∵ *P → Q* ⇔ *¬P* ∨ *Q*]

- (*P* 𝖠 *Q*) ∨ (*¬P* ∨ *Q*)

- (*P* 𝖠 *Q*) ∨ *¬P* ∨ *Q*

- ((*P*𝖠 *Q*) ∨ *¬P* )) ∨ *Q*

- ((*P* ∨ *¬P* ) 𝖠 (*Q* ∨ *¬P* )) ∨ *Q*

- (*T* 𝖠 (*Q* ∨ *¬P* )) ∨ *Q*

- (*Q* ∨ *¬P* ) ∨ *Q*

* *Q* ∨ *¬P*
* *¬P* ∨ *Q*

(b). From (a) Writing the dual

(*P* 𝖠 *Q*) ∨(*¬P* ∨(*¬P* ∨ *Q*)) ⇔ *¬P* ∨ *Q* 63

(*P* ∨ *Q*) 𝖠 (*¬P* 𝖠 (*¬P* 𝖠 *Q*)) ⇔(*¬P* 𝖠 *Q*)

#### Write Tautological Implications with example

A statement formula *A* is said to *tautologically imply* a statement *B* if and only if *A → B*

is a tautology.

In this case we write *A* ⇒ *B*, which is read as ‘*A* implies *B*‘.

Note: ⇒ is not a connective, *A* ⇒*B* is not a statement formula.

*A* ⇒ *B* states that *A → B* is tautology.

Clearly *A* ⇒ *B* guarantees that *B* has a truth value *T* whenever *A* has the truth value *T* .

One can determine whether *A* ⇒ *B* by constructing the truth tables of *A* and *B* in the same manner as was done in the determination of *A* ⇔ *B*.

**Example:** Prove that (*P → Q*) ⇒ (*¬Q → ¬P* ). Solution:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *P* | *Q* | *¬P* | *¬Q* | *P → Q* | *¬Q → ¬P* | (*P → Q*) *→* (*¬Q → ¬P* ) |
| T | T | F | F | T | T | T |
| T | F | F | T | F | F | T |
| F | T | T | F | T | T | T |
| F | F | T | T | T | T | T |

Since all the entries in the last column are true, (*P → Q*) *→* (*¬Q → ¬P* ) is a tautology. Hence (*P → Q*) ⇒ (*¬Q → ¬P* ).

In order to show any of the given implications, it is suﬃcient to show that an assignment of the truth

value *T* to the antecedent of the corresponding conditional leads to the truth value *T* for the consequent. This procedure guarantees that the conditional becomes tautology, thereby proving the implication.

#### (a). Prove that ¬Q 𝖠 (P → Q) ⇒ ¬P .

Solution: Assume that the antecedent *¬Q* 𝖠 (*P → Q*) has the truth value *T* , then both *¬Q* and *P → Q* have the truth value *T* , which means that *Q* has the truth value *F* , *P → Q* has the truth value *T* . Hence *P* must have the truth value *F* .

Therefore the consequent *¬P* must have the truth value *T*.

*¬Q* 𝖠 (*P → Q*) ⇒ *¬P* .

Another method to show *A* ⇒ *B* is to assume that the consequent *B* has the truth value *F* and then show that this assumption leads to *A* having the truth value *F* . Then *A → B* must have the truth value *T* .

#### 12 (b). Show that ¬(P → Q) ⇒ P . 64

Solution: Assume that *P* has the truth value *F* . When *P* has *F* , *P → Q* has *T* , then *¬*(*P → Q*) has *F*

. Hence *¬*(*P → Q*) *→ P* has *T* .

#### Write the NAND, NOR connectives which have useful applications in the design of computers.

ANS:

**NAND:** The word NAND is a combination of ‘NOT‘ and ‘AND‘ where ‘NOT‘ stands for negation and ‘AND‘ for the conjunction. It is denoted by the symbol *↑*.

If *P* and *Q* are two formulas then

*P ↑ Q* ⇔ *¬*(*P* 𝖠 *Q*) The connective *↑* has the following equivalence:

*P ↑ P* ⇔ *¬*(*P* 𝖠 *P* ) ⇔ *¬P* ∨ *¬P* ⇔ *¬P* .

(*P ↑ Q*) *↑* (*P ↑ Q*) ⇔ *¬*(*P ↑ Q*) ⇔ *¬*(*¬*(*P* 𝖠 *Q*)) ⇔ *P* 𝖠 *Q*. (*P ↑ P* ) *↑* (*Q ↑ Q*) ⇔ *¬P ↑ ¬Q* ⇔ *¬*(*¬P* 𝖠 *¬Q*) ⇔ *P* ∨

*Q*.

NAND is Commutative: Let *P* and *Q* be any two statement formulas.

(*P ↑ Q*) ⇔ *¬*(*P* 𝖠 *Q*)

- *¬*(*Q* 𝖠 *P* ) ⇔

(*Q ↑ P* )

∴ NAND is commutative.

NAND is not Associative: Let *P* , *Q* and *R* be any three statement formulas. Consider *↑* (*Q ↑ R*) ⇔ *¬*(*P* 𝖠 (*Q ↑ R*)) ⇔ *¬*(*P* 𝖠 (*¬*(*Q* 𝖠 *R*)))

⇔ *¬P* ∨(*Q* 𝖠 *R*)) (*P ↑ Q*) *↑ R* ⇔ *¬*(*P* 𝖠 *Q*) *↑ R*

- *¬*(*¬*(*P* 𝖠 *Q*) 𝖠 *R*) ⇔

(*P* 𝖠 *Q*) ∨ *¬R*

Therefore the connective *↑* is not associative.

**NOR:** The word NOR is a combination of ‘NOT‘ and ‘OR‘ where ‘NOT‘ stands for negation and

‗OR‘ for the disjunction. It is denoted by the symbol *↓*.

If *P* and *Q* are two formulas then

*P ↓ Q* ⇔ *¬*(*P* ∨ *Q*) The connective *↓* has the following equivalence:

*P ↓ P* ⇔ *¬*(*P* ∨ *P* ) ⇔ *¬P* 𝖠 *¬P* ⇔ *¬P* .

(*P ↓ Q*) *↓* (*P ↓ Q*) ⇔ *¬*(*P ↓ Q*) ⇔ *¬*(*¬*(*P* ∨ *Q*)) ⇔ *P* ∨ *Q*. (*P ↓ P* ) *↓* (*Q ↓ Q*) ⇔ *¬P ↓ ¬Q* ⇔ *¬*(*¬P* ∨ *¬Q*) ⇔ *P* 𝖠 *Q*.

NOR is Commutative: Let *P* and *Q* be any two statement formulas.

(*P ↓ Q*) ⇔ *¬*(*P* ∨ *Q*) 65

- *¬*(*Q* ∨ *P* ) ⇔

(*Q ↓ P* )

∴ NOR is commutative.

NOR is not Associative: Let *P* , *Q* and *R* be any three statement formulas. Consider

*P↓* (*Q ↓ R*) ⇔ *¬*(*P* ∨ (*Q ↓ R*))

- *¬*(*P* ∨ (*¬*(*Q* ∨ *R*)))

⇔ *¬P* 𝖠 (*Q* ∨ *R*) (*P ↓ Q*) *↓ R* ⇔ *¬*(*P* ∨ *Q*) *↓ R*

- *¬*(*¬*(*P* ∨ *Q*) ∨ *R*) ⇔

(*P* ∨ *Q*) 𝖠 *¬R*

Therefore the connective *↓* is not associative.

Evidently, *P ↑ Q* and *P ↓ Q* are duals of each other.

Since

#### Write and give examples of Inverse, Converse, and Contra-positive ANS

Implication / if-then (→)(→) is also called a conditional statement. It has two parts −

* Hypothesis, p
* Conclusion, q

As mentioned earlier, it is denoted as p→qp→q.

Example of Conditional Statement − “If you do your homework, you will not be punished.”

Here, "you do your homework" is the hypothesis, p, and "you will not be punished" is the conclusion, q.

**Inverse** − An inverse of the conditional statement is the negation of both the hypothesis and the conclusion.

If the statement is “If p, then q”, the inverse will be “If not p, then not q”. Thus the inverse of p→qp→q is ¬p→¬q¬p→¬q.

Example − The inverse of “If you do your homework, you will not be punished” is “If you do not do your homework, you will be punished.”

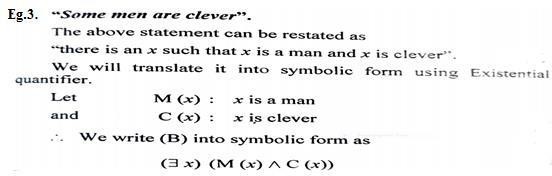
**Converse** − The converse of the conditional statement is computed by interchanging the hypothesis and the conclusion.

If the statement is “If p, then q”, the converse will be “If q, then p”. The converse of p→qp→q is q→pq→p.

Example − The converse of "If you do your homework, you will not be punished" is "If you will not be punished, you do your homework”.

**Contra-positive** − The contra-positive of the conditional is computed by interchanging the hypothesis and the conclusion of the inverse statement.

If the statement is “If p, then q”, the contra-positive will be “If not q, then not p”.

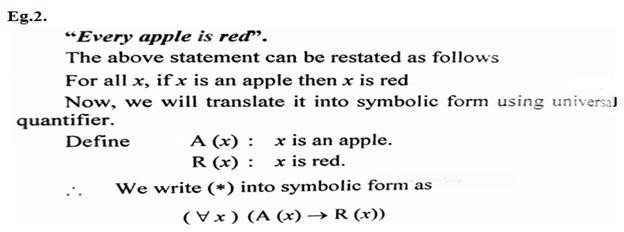
The contra-positive of p→qp→q is ¬q→¬p¬q→¬p.

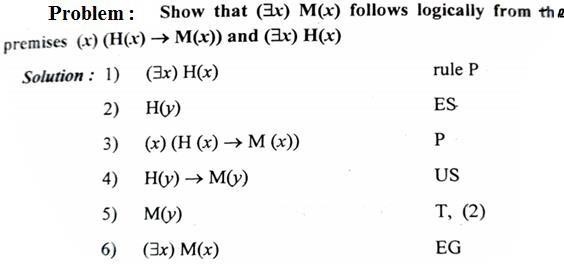
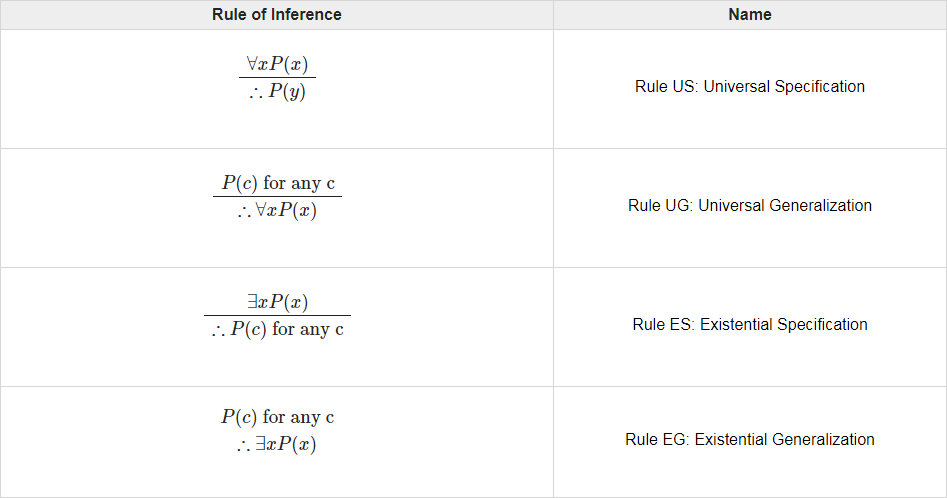
Example − The Contra-positive of " If you do your homework, you will not be punished” is "If you are punished, you did not do your homework”.

#### Explain nested quantifiers with exampleNested Quantifiers ANS:

If we use a quantifier that appears within the scope of another

quantifier, it is called nested quantifier.



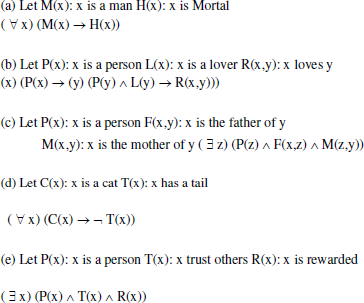


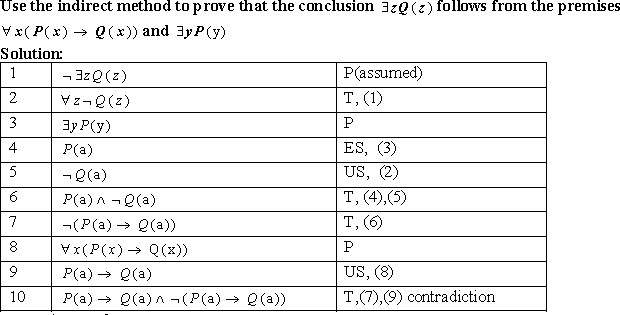
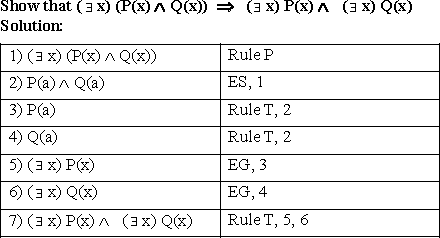
#### Symbolize the following statements: (a).All men are mortal

**(b).All the world loves alover**

#### ( c ). X is the father of mother of Y (d)No cats has a tail

**(e).Some people who trust others are rewarded**





#### Explain two Methods to obtain PDNF of a given formula

1. **By Truth table:**
   1. Construct a truth table of the given formula.
   2. For every truth value *T* in the truth table of the given formula, select the minterm which also has the value *T* for the same combination of the truth values of *P* and *Q*.
   3. The disjunction of these minterms will then be equivalent to the given formula.

Example: Obtain the PDNF of *P → Q*. Solution: From the truth table of *P → Q*

|  |  |  |  |
| --- | --- | --- | --- |
| *P* | *Q* | *P → Q* | Minterm |
| T T F  F | T F T  F | T F T  T | *P* 𝖠 *Q P* 𝖠 *¬Q*  *¬P* 𝖠 *Q*  *¬P* 𝖠*¬Q* |

The PDNF of *P → Q* is (*P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *¬Q*).

∴ *P → Q* ⇔ (*P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *¬Q*).

Example: Obtain the PDNF for (*P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *R*) ∨ (*Q* 𝖠 *R*). Solution:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *P* | *Q* | *R* | Minterm | *P* 𝖠 *Q* | *¬P* 𝖠 *R* | *Q* 𝖠 *R* | (*P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *R*) ∨ (*Q* 𝖠 *R*) |
| T | T | T | *P* 𝖠 *Q* 𝖠 *R* | T | F | T | T |
| T | T | F | *P* 𝖠 *Q* 𝖠 *¬R* | T | F | F | T |
| T | F | T | *P* 𝖠 *¬Q* 𝖠 *R* | F | F | F | F |
| T | F | F | *P* 𝖠 *¬Q* 𝖠 *¬R* | F | F | F | F |
| F | T | T | *¬P* 𝖠 *Q* 𝖠 *R* | F | T | T | T |
| F | T | F | *¬P* 𝖠 *Q* 𝖠 *¬R* | F | F | F | F |
| F | F | T | *¬P* 𝖠 *¬Q* 𝖠 *R* | F | T | F | T |
| F | F | F | *¬P* 𝖠 *¬Q* 𝖠 *¬R* | F | F | F | F |

The PDNF of (*P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *R*) ∨ (*Q* 𝖠 *R*) is

(*P* 𝖠 *Q* 𝖠 *R*) ∨ (*P* 𝖠 *Q* 𝖠 *¬R*) ∨ (*¬P* 𝖠 *Q* 𝖠 *R*) ∨ (*¬P* 𝖠 *¬Q* 𝖠 *R*).

#### Without constructing the truth table:

In order to obtain the principal disjunctive normal form of a given formula is con- structed as follows:

1. First replace *→*, by their equivalent formula containing only 𝖠, ∨and *¬*.
2. Next, negations are applied to the variables by De Morgan‘s laws followed by the application of distributive laws.
3. Any elementarily product which is a contradiction is dropped. Minterms are ob-tained in the disjunctions by introducing the missing factors. Identical minterms appearing in the disjunctions are deleted.

#### Obtain the principal disjunctive normal form of

**(a) ¬P**∨ **Q; (b) (P** 𝖠 **Q)** ∨ **(¬P** 𝖠 **R)** ∨ **(Q** 𝖠 **R).**

Solution:

*(a) ¬P* ∨ *Q* ⇔ (*¬P* 𝖠 *T* ) ∨ (*Q* 𝖠 *T* ) [∵ *A* 𝖠 *T* ⇔ *A*]

- (*¬P* 𝖠 (*Q* ∨ *¬Q*)) ∨ (*Q* 𝖠 (*P* ∨ *¬P* )) [∵ *P* ∨ *¬P* ⇔ *T* ] 71

- (*¬P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *¬Q*) ∨ (*Q* 𝖠 *P* ) ∨ (*Q* 𝖠 *¬P* )

[∵ *P* 𝖠 (*Q* ∨ *R*) ⇔ (*P* 𝖠 *Q*) ∨ (*P* 𝖠 *R*)

⇔(*¬P* 𝖠 *Q*) ∨(*¬P* 𝖠 *¬Q*) ∨(*P* 𝖠 *Q*) [∵ *P* ∨ *P* ⇔ *P* ]

**(b) (P** 𝖠 **Q)** ∨ **(¬P** 𝖠 **R)** ∨ **(Q** 𝖠 **R)**

- (*P* 𝖠 *Q* 𝖠 *T* ) ∨ (*¬P* 𝖠 *R* 𝖠 *T* ) ∨ (*Q* 𝖠 *R* 𝖠 *T* )

- (*P* 𝖠 *Q* 𝖠 (*R* ∨ *¬R*)) ∨ (*¬P* 𝖠 *R* 𝖠 (*Q* ∨ *¬Q*)) ∨ (*Q* 𝖠 *R* 𝖠 (*P* ∨ *¬P* ))

- (*P* 𝖠 *Q* 𝖠 *R*) ∨ (*P* 𝖠 *Q* 𝖠 *¬R*) ∨ (*¬P* 𝖠 *R* 𝖠 *Q*)(*¬P* 𝖠 *R* 𝖠 *¬Q*)

∨ (*Q* 𝖠 *R* 𝖠 *P* ) ∨ (*Q* 𝖠 *R* 𝖠 *¬P* )

- (*P* 𝖠 *Q* 𝖠 *R*) ∨ (*P* 𝖠 *Q* 𝖠 *¬R*) ∨ (*¬P* 𝖠 *Q* 𝖠 *R*) ∨ (*¬P* 𝖠 *¬Q* 𝖠 *R*)

*P* ∨ (*P* 𝖠 *Q*) ⇔ *P*

*P* ∨ (*¬P* 𝖠 *Q*) ⇔ *P* ∨ *Q*

Solution: We write the principal disjunctive normal form of each formula and com-pare these normal forms.

(*a*) *P* ∨ (*P* 𝖠 *Q*) ⇔(*P* 𝖠 *T* ) ∨ (*P* 𝖠 *Q*) [∵ *P* 𝖠 *Q* ⇔ *P* ]

-(*P* 𝖠 (*Q* ∨ *¬Q*)) ∨ (*P* 𝖠 *Q*) [∵ *P* ∨ *¬P* ⇔ *T* ]

- ((*P* 𝖠 *Q*) ∨ (*P* 𝖠 *¬Q*)) ∨ (*P* 𝖠 *Q*) [by distributive laws]

-(*P* 𝖠 *Q*) ∨ (*P* 𝖠 *¬Q*) [∵ *P* ∨ *P* ⇔ *P* ] which is the required PDNF.

Now, ⇔ *P* 𝖠 *T*

- *P* 𝖠 (*Q* ∨ *¬Q*)

⇔(*P* 𝖠 *Q*) ∨(*P* 𝖠 *¬Q*) which is the required PDNF.

Hence, *P* ∨ (*P* 𝖠 *Q*) ⇔ *P* .

(*b*) *P* ∨ (*¬P* 𝖠 *Q*) ⇔ (*P* 𝖠 *T* ) ∨ (*¬P* 𝖠 *Q*)

- (*P* 𝖠 (*Q* ∨ *¬Q*)) ∨ (*¬P* 𝖠 *Q*)

⇔(*P* 𝖠 *Q*) ∨(*P* 𝖠 *¬Q*) ∨(*¬P* 𝖠 *Q*) which is the required PDNF.

Now,

*P* ∨ *Q* ⇔ (*P* 𝖠 *T* ) ∨ (*Q* 𝖠 *T* )

- (*P* 𝖠 (*Q* ∨ *¬Q*)) ∨ (*Q* 𝖠 (*P* ∨ *¬P* ))

- (*P* 𝖠 *Q*) ∨ (*P* 𝖠 *¬Q*) ∨ (*Q* 𝖠 *P* ) ∨ (*Q* 𝖠 *¬P* )

⇔(*P* 𝖠 *Q*) ∨(*P* 𝖠 *¬Q*) ∨(*¬P* 𝖠 *Q*) 72

which is the required PDNF.

Hence, *P* ∨ (*¬P* 𝖠 *Q*) ⇔ *P* ∨ *Q*.

#### 19 . Obtain the principal disjunctive normal form of

*P →* ((*P → Q*) 𝖠 *¬*(*¬Q* ∨ *¬P* )).

Solution: Using *P → Q* ⇔ *¬P* ∨ *Q* and De Morgan‘s law, we obtain

*→* ((*P → Q*) 𝖠 *¬*(*¬Q* ∨ *¬P* )) ⇔ *¬P*

∨ ((*¬P* ∨ *Q*) 𝖠 (*Q* 𝖠 *P* ))

- *¬P* ∨ ((*¬P* 𝖠 *Q* 𝖠 *P* ) ∨ (*Q* 𝖠 *Q* 𝖠 *P* )) ⇔

*¬P* ∨ *F* ∨ (*P* 𝖠 *Q*)

- *¬P* ∨ (*P* 𝖠 *Q*)

- (*¬P* 𝖠 *T* ) ∨ (*P* 𝖠 *Q*)

- (*¬P* 𝖠 (*Q* ∨ *¬Q*)) ∨ (*P* 𝖠 *Q*)

⇔(*¬P* 𝖠 *Q*) ∨(*¬P* 𝖠 *¬Q*) ∨(*P* 𝖠 *Q*) Hence (*P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *Q*) ∨ (*¬P* 𝖠 *¬Q*) is the required DNF.

#### Explain Principal Conjunctive Normal Form with example

The dual of a minterm is called a Maxterm. For a given number of variables, the *maxterm* consists of disjunctions in which each variable or its negation, but not both, appears only once. Each of the maxterm has the truth value *F* for exactly one com-bination of the truth values of the variables. Now we define the principal conjunctive normal form.

For a given formula, an equivalent formula consisting of conjunctions of the max-terms only is known as its *principle conjunctive normal form*. This normal form is also called the *product-of-sums canonical form.*The method for obtaining the PCNF for a given formula is similar to the one described previously for PDNF.

Example: Obtain the principal conjunctive normal form of the formula (¬P→R)𝖠(Q↔P) Solution:

(*¬P → R*) 𝖠 (*Q ↔ P* )

- [*¬*(*¬P* ) ∨ *R*] 𝖠 [(*Q → P* ) 𝖠 (*P → Q*)]

- (*P* ∨ *R*) 𝖠 [(*¬Q* ∨ *P* ) 𝖠 (*¬P* ∨ *Q*)] 73

- (*P* ∨ *R* ∨ *F* ) 𝖠 [(*¬Q* ∨ *P* ∨ *F* ) 𝖠 (*¬P* ∨ *Q* ∨ *F* )]

- [(*P* ∨ *R*) ∨ (*Q* 𝖠 *¬Q*)] 𝖠 [*¬Q* ∨ *P* ) ∨ (*R* 𝖠 *¬R*)] 𝖠 [(*¬P* ∨ *Q*) ∨ (*R* 𝖠 *¬R*)]

- (*P* ∨ *R* ∨ *Q*) 𝖠 (*P* ∨ *R* ∨ *¬Q*) 𝖠 (*P* ∨ *¬Q* ∨ *R*) 𝖠 (*P* ∨ *¬Q* ∨ *¬R*)

𝖠 (*¬P* ∨ *Q* ∨ *R*) 𝖠 (*¬P* ∨ *Q* ∨ *¬R*)

⇔(*P* ∨ *Q* ∨ *R*) 𝖠 (*P* ∨ *¬Q* ∨ *R*) 𝖠 (*P* ∨ *¬Q* ∨ *¬R*) 𝖠 (*¬P* ∨ *Q* ∨ *R*) 𝖠 (*¬P* ∨ *Q* ∨ *¬R*) which is required principal conjunctive normal form.

#### Find the PDNF form PCNF of S : P ∨ (¬P → (Q ∨ (¬Q → R))).

Solution:

which is the PCNF. ⇔ *P* ∨ (*¬P →* (*Q* ∨ (*¬Q → R*)))

- *P* ∨ (*¬*(*¬P* ) ∨ (*Q* ∨ (*¬*(*¬Q*) ∨ *R*))

- *P* ∨ (*P* ∨ *Q* ∨ (*Q* ∨ *R*)))

- *P* ∨ (*P* ∨ *Q* ∨ *R*)

- *P* ∨ *Q* ∨ *R*

Now PCNF of *¬S* is the conjunction of remaining maxterms, so

PCNF of *¬S* : (*P* ∨ *Q* ∨ *¬R*) 𝖠 (*P* ∨ *¬Q* ∨ *R*) 𝖠 (*P* ∨ *¬Q* ∨ *¬R*) 𝖠 (*¬P* ∨ *Q* ∨ *R*)

𝖠(*¬P* ∨ *Q* ∨ *¬R*) 𝖠 (*¬P* ∨ *¬Q* ∨ *R*) 𝖠(*¬P* ∨ *¬Q* ∨ *¬R*) Hence the PDNF of *S* is

*¬*(PCNF of *¬S*) : (*¬P* 𝖠 *¬Q* 𝖠 *R*) ∨ (*¬P* 𝖠 *Q* 𝖠 *¬R*) ∨ (*¬P* 𝖠 *Q* 𝖠 *R*) ∨ (*P* 𝖠 *¬Q* 𝖠 *¬R*)

∨ ( *P* 𝖠 *¬Q* 𝖠 *R*) ∨ (*P* 𝖠 *Q* 𝖠 *¬R*) ∨ (*P* 𝖠 *Q* 𝖠 *R*)

#### Determine whether the conclusion C follows logically from the premises

*H*1 and *H*2.

* 1. *H*1 : *P → Q H*2 : *P C* : *Q*
  2. *H*1 : *P → Q H*2 : *¬P C* : *Q*

(*c*) *H*1 : *P → Q H*2 : *¬*(*P* 𝖠 *Q*) *C* : *¬P*

(d) *H*1 : *¬P H*2 : *P Q C* : *¬*(*P* 𝖠 *Q*)

*(e) H*1 : *P → Q H*2 : *Q C* : *P*

Solution: We first construct the appropriate truth table, as shown in table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *P* | *Q* | *P → Q* | *¬P* | *¬*(*P* 𝖠 *Q*) | *P* | *Q* |
| T | T | T | F | F | T | |
| T | F | F | F | T | F | |
| F | T | T | T | T | F | |
| F | F | T | T | T | T | |

#### Define Rules of Inference with example

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}*  *{*2*}* | (1)  (2) | *¬R* ∨ *P* Rule P  *R* Rule P (assumed premise) | |
| *{*1*,* 2*}* | (3) | *P* | Rule T, (1), (2), and *I*10 |
| *{*4*}* | (4) | *P →* (*Q → S*) | Rule P |
| *{*1*,* 2*,* 4*}* | (5) | *Q → S* | Rule T, (3), (4), and *I*11 |
| *{*6*}* | (6) | *Q* | Rule P |
| *{*1*,* 2*,* 4*,* 6*}* | (7) | *S* | Rule T, (5), (6), and *I*11 |

The following are two important rules of inferences.

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 one or more of the preceding formulas in the derivation.

Implication Formulas

*I*1 : *P* 𝖠*Q* ⇒ *P* (simplification)

*I*2 : *P* 𝖠 *Q* ⇒ *Q I*3 : *P* ⇒ *P* ∨ *Q I*4 : *Q* ⇒ *P* ∨ *Q*

*I*5 : *¬P* ⇒ *P → Q I*6 : *Q* ⇒ *P → Q I*7 : *¬*(*P → Q*) ⇒*P*

*I*8 : *¬*(*P → Q*) ⇒ *¬Q*

*I*9 : *P, Q* ⇒ *P* 𝖠 *Q*

*I* :

10 *¬P, P* ∨ *Q* ⇒ *Q* (disjunctive syllogism)

*I*

11 : *P, P → Q*⇒ *Q* (modus ponens)

*I*

12 : *¬Q, P → Q* ⇒ *¬P* (modus tollens)

*I*

13 : *P → Q, Q → R* ⇒ *P→ R* (hypothetical syllogism)

*I*

14 : *P* ∨ *Q, P → R, Q → R* ⇒ *R* (dilemma)

#### Example: Show that R → S can be derived from the premises P → (Q → S), ¬R ∨ P , and Q.

Solution: Instead of deriving *R → S*, we shall include *R* as an additional premise and show *S*

first.

#### Show that P → S can be derived from the premises ¬P ∨ Q, ¬Q ∨ R, and R → S.

Solution: We include *P* as an additional premise and derive *S*.

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}*  *{*2*}* | (1)  (2) | *¬P* ∨ *Q*  *P* | Rule P  Rule P (assumed premise) |
| *{*1*,* 2*}* | (3) | *Q* | Rule T, (1), (2), and *I*10 |
| *{*4*}* | (4) | *¬Q* ∨ *R* | Rule P |
| *{*1*,* 2*,* 4*}* | (5) | *R* | Rule T, (3), (4), and *I*10 |
| *{*6*}* | (6) | *R → S* | Rule P |
| *{*1*,* 2*,* 4*,* 6*}* | (7) | *S* | Rule T, (5), (6), and *I*11 |
| *{*1*,* 2*,* 4*,* 6*}* | (8) | *P → S* | Rule CP |

#### 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 us indicate the statements as follows:

*P* : There was a ball game. *Q*: Traveling was diﬃcult. *R*: They arrived on time.

Hence, the given premises are *P → Q*, *R → ¬Q*, and *R*. The conclusion is *¬P* .

|  |  |  |
| --- | --- | --- |
| *{*1*}* | (1) *R → ¬Q* | Rule P |
| *{*2*}* | (2) *R* | Rule P |
| *{*1*,* 2*}* | (3) *¬Q* | Rule T, (1), (2), and *I*11 |
| *{*4*}* | (4) *P → Q* | Rule P |
| *{*4*}* | (5) *¬Q → ¬P* | Rule T, (4), and *P → Q* ⇔ *¬Q →*  *¬P* |
| *{*1*,* 2*,* 4*}* | (6) *¬P* | Rule T, (3), (5), and *I*11 |

#### By using the method of derivation, show that following statements con-stitute a valid argument: ‖If A works hard, then either B or C will enjoy. If B enjoys, then A will not work hard. If D enjoys, then C will not. Therefore, if A works hard, D will not enjoy.

**Solution**: Let us indicate statements as follows:

Given premises are *P →* (*Q*∨*R*), *Q → ¬P* , and *S → ¬R*. The conclusion is *P → ¬S*. We include *P* as an additional premise and derive *¬S*.

*{*1*}* (1) *P* Rule P (additional premise)

|  |  |  |  |
| --- | --- | --- | --- |
| *{*2*}*  *{*1*,* 2*}* | (2) *P →* (*Q* ∨ *R*)  (3) *Q* ∨ *R* | | Rule P  Rule T, (1), (2), and *I*11 |
| *{*1*,* 2*}* | (4) | *¬Q → R* | Rule T, (3) and *P → Q* ⇔ *P* ∨ *Q* |
| *{*1*,* 2*}* | (5) | *¬R → Q* | Rule T, (4), and *P → Q* ⇔ *¬Q →*  *¬P* |
| *{*6*}* | (6) | *Q → ¬P* | Rule P |
| *{*1*,* 2*,* 6*}* | (7) | *¬R → ¬P* | Rule T, (5), (6), and *I*13 |
| *{*1*,* 2*,* 6*}* | (8) | *P → R* | Rule T, (7) and *P → Q* ⇔ *¬Q → ¬P* |
| *{*9*}* | (9) | *S → ¬R* | Rule P |
| *{*9*}* | (10) | *R → ¬S* | Rule T, (9) and *P → Q* ⇔ *¬Q → ¬P* |
| *{*1*,* 2*,* 6*,* 9*}* | (11) | *P → ¬S* | Rule T, (8), (10) and *I*13 |
| *{*1*,* 2*,* 6*,* 9*}* | (12) | *¬S* | Rule T, (1), (11) and *I*11 |

#### Determine the validity of the following arguments using propositional logic:

**‖Smoking is healthy. If smoking is healthy, then cigarettes are prescribed by physi- cians. Therefore, cigarettes are prescribed by physicians‖.**

**Solution:** Let us indicate the statements as follows:

*P* : Smoking is healthy.

*Q*: Cigarettes are prescribed by physicians.

Hence, the given premises are *P* , *P → Q*. The conclusion is *Q*.

*{*1*}* (1) *P→ Q* Rule P

*{*2*}* (2) *P* Rule P

*{*1*,* 2*}* (3) *Q* Rule T, (1), (2), and *I*11 78

Hence, the given statements constitute a valid argument.

#### Explain Indirect Method of Proof with example

The method of using the rule of conditional proof and the notion of an inconsistent set of premises is called the *indirect method of proof* or *proof by contradiction*.

In order to show that a conclusion *C* follows logically from the premises *H*1*, H*2*, · · · , Hm*, we assume that *C* is false and consider *¬C* as an additional premise. If the new set of premises is inconsistent, so that they imply a contradiction. Therefore, the assump-tion that *¬C* is

true does not hold.

Hence, *C* is true whenever *H*1*, H*2*, · · · , Hm* are true. Thus, *C* follows logically from the premises *H*1*, H*2*, · · · , Hm*.

Example: Show that *¬*(*P* 𝖠 *Q*) follows from *¬P* 𝖠 *¬Q*.

Solution: We introduce *¬¬*(*P* 𝖠*Q*) as additional premise and show that this additional premise leads to a contradiction.

|  |  |  |
| --- | --- | --- |
| *{*1*}*  *{*1*}* | (1) *¬¬*(*P* 𝖠 *Q*)  (2) *P* 𝖠 *Q* | Rule P (assumed)  Rule T, (1), and *¬¬P* ⇔  *P* |
| *{*1*}* | (3) *P* | Rule T, (2), and *I*1 |
| *{*4*}* | (4) *¬P* 𝖠 *¬Q* | Rule P |
| *{*4*}* | (5) *¬P* | Rule T, (4), and *I*1 |
| *{*1*,* 4*}* | (6) *P* 𝖠 *¬P* | Rule T, (3), (5), and *I*9 |

Hence, our assumption is wrong.

Thus, *¬*(*P* 𝖠 *Q*) follows from *¬P* 𝖠 *¬Q*.

#### Using the indirect method of proof, show that

**P → Q, Q → R, ¬(P** 𝖠 **R), P** ∨ **R** ⇒ **R.**

Solution: We include *¬R* as an additional premise. Then we show that this leads to a contradiction.

79

|  |  |  |
| --- | --- | --- |
| *{*1*}* | (1) *P → Q* | Rule P |
| *{*2*}* | (2) *Q → R* | Rule P |
| *{*1*,* 2*}* | (3) *P → R* | Rule T, (1), (2), and *I*13 |
| *{*4*}* | (4) *¬R* | Rule P (assumed) |
| *{*1*,* 2*,* 4*}* | (5) *¬P* | Rule T, (4), and *I*12 |
| *{*6*}* | (6) *P* ∨ *R* | Rule P |
| *{*1*,* 2*,* 4*,* 6*}* | (7) *R* | Rule T, (5), (6) and *I*10 |
| *{*1*,* 2*,* 4*,* 6*}* | (8) *R* 𝖠*¬R* | Rule T, (4), (7), and *I*9 |

Hence, our assumption is wrong.

#### Show that the following set of premises are inconsistent, using proof by contradiction P → (Q ∨ R), Q → ¬P, S → ¬R, P ⇒ P → ¬S.

Solution: We include *¬*(*P → ¬S*) as an additional premise. Then we show that this leads to a contradiction.

∴ *¬*(*P → ¬S*) ⇔ *¬*(*¬P* ∨ *¬S*) ⇔ *P* 𝖠 *S*.

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}* | (1) | *P →* (*Q* ∨ *R*) | Rule P |
| *{*2*}* | (2) | *P* | Rule P |
| *{*1*,* 2*}* | (3) | *Q* ∨ *R* | Rule T, (1), (2), and Modus Ponens |
| *{*4*}* | (4) | *P* 𝖠 *S* | Rule P (assumed) |
| *{*1*,* 2*,* 4*}* | (5) | *S* | Rule T, (4), and *P* 𝖠 *Q* ⇒*P* |
| *{*6*}* | (6) | *S → ¬R* | Rule P |
| *{*1*,* 2*,* 4*,* 6*}* | (7) | *¬R* | Rule T, (5), (6) and Modus Ponens |
| *{*1*,* 2*,* 4*,* 6*}* | (8) | *Q* | Rule T, (3), (7), and *P* 𝖠 *Q, ¬Q* ⇒ *P* |
| *{*9*}* | (9) | *Q → ¬P* | Rule P |
| *{*1*,* 2*,* 4*,* 6*}* | (10) | *¬P* | Rule T, (8), (9), and *P* 𝖠 *Q, ¬Q* ⇒*P* |
| *{*1*,* 2*,* 4*,* 6*}* | (11) | *P* 𝖠 *¬P* | Rule T, (2), (10), and *P, Q* ⇒ *P* 𝖠*Q* |
| *{*1*,* 2*,* 4*,* 6*}* | (12) | *F* | Rule T, (11), and *P* 𝖠 *¬P* ⇔*F* |

Hence, it is proved that the given premises are inconsistent.

80

#### Example: Show that (x)(P (x) ∨ Q(x)) ⇒ (x)P (x) ∨ (∃x)Q(x). Solution: We shall use the indirect method of proof by assuming

**¬((x)P (x)**∨**(**∃**x)Q(x)) as an additional premise.**

|  |  |  |  |
| --- | --- | --- | --- |
| *{*1*}*  *{*1*}* | (1) *¬*((*x*)*P* (*x*) ∨ (∃*x*)*Q*(*x*))  (2) *¬*(*x*)*P* (*x*) 𝖠 *¬*(∃*x*)*Q*(*x*) | | Rule P (assumed)  Rule T, (1) *¬*(*P* ∨ *Q*) ⇔ *¬P* 𝖠 *¬Q* |
| *{*1*}* | (3) | *¬*(*x*)*P* (*x*) | Rule T, (2), and *I*1 |
| *{*1*}* | (4) | (∃*x*)*¬P* (*x*) | Rule T, (3), and *¬*(*x*)*A*(*x*) ⇔ (∃*x*)*¬A*(*x*) |
| *{*1*}* | (5) | *¬*(∃*x*)*Q*(*x*) | Rule T, (2), and *I*2 |
| *{*1*}* | (6) | (*x*)*¬Q*(*x*) | Rule T, (5), and *¬*(∃*x*)*A*(*x*) ⇔  (*x*)*¬A*(*x*) |
| *{*1*}* | (7) | *¬P* (*y*) | Rule ES, (5), (6) and *I*12 |
| *{*1*}* | (8) | *¬Q*(*y*) | Rule US, (6) |
| *{*1*}* | (9) | *¬P* (*y*) 𝖠 *¬Q*(*y*) | Rule T, (7), (8)and *I*9 |
| *{*1*}* | (10) | *¬*(*P* (*y*) ∨ *Q*(*y*)) | Rule T, (9), and *¬*(*P* ∨ *Q*) ⇔ *¬P* 𝖠 *¬Q* |
| *{*11*}* | (11) | (*x*)(*P* (*x*) ∨ *Q*(*x*)) | Rule P |
| *{*11*}* | (12) | (*P* (*y*) ∨ *Q*(*y*)) | Rule US |

*{*1*,* 11*}* (13) *¬*(*P* (*y*) ∨ *Q*(*y*)) 𝖠 (*P* (*y*) ∨*Q*(*y*)) Rule T, (10), (11), and *I*9

*{*1*,* 11*}* (14) *F* Rule T, and (13)

which is a contradiction.Hence, the statement is valid.

81

82

#### UNIT-II

* 1. **Define a set and types of sets**

ANSWER

**Set definition** : A set is collection of well defined objects

Each of the objects in the set is called a member of an element of the set. Books, cities, numbers, animals, flowers, etc.

Elements of a set are usually denoted by lower-case letters. While sets are denoted by capital letters of English larguage. The symbol ∈ indicates the membership in a set.

If ―*a* is an element of the set *A*‖, then we write *a* ∈ *A.*

The symbol ∈ is read ―is a member of ‖ or ―is an element of ‖.

The symbol  is used to indicate that an object is not in the given set. The symbol  is read ―is not a member of ‖ or ―is not an element of ‖.If *x* is not an element of the set *A* then we write *x*  *A*.

***Subset:***

A set *A* is a subset of the set *B* if and only if every element of *A* is also an element of *B*. We also say that *A* is contained in *B*, and use the notation *A*  *B*.

***Proper Subset:***

A set *A* is called proper subset of the set *B*. If (*i*) *A* is subset of *B* and (*ii*) *B* is not a subset *A* i.e., *A* is said to be a proper subset of *B* if every element of *A* belongs to the set *B*, but there is atleast one element of *B*, which is not in *A*. If *A* is a proper subset of *B*, then we denote it by *A*  *B*.

**Super set:** If *A* is subset of *B*, then *B* is called a superset of *A*.

**Null set:** The set with no elements is called an empty set or null set. A Null set is designated by the symbol  . The null set is a subset of every set, i.e., If *A* is any set then   *A*.

**Universal set:**

In many discussions all the sets are considered to be subsets of one particular set. This set is called the universal set for that discussion. The Universal set is often designated by the script letter  . Universal set in

not unique and it may change from one discussion to another.

**Power set:**

The set of all subsets of a set *A* is called the power set of *A*.

The power set of *A* is denoted by *P* (*A*). If *A* has *n* elements in it, then *P* (*A*) has 2*n* elements:

**Disjoint sets:**

Two sets are said to be disjoint if they have no element in common.

**Union of two sets:**

The union of two sets *A* and *B* is the set whose elements are all of the elements in *A* or in *B* or in both. The union of sets *A* and *B* denoted by *A*  *B* is read as ―*A* union *B*‖.

**Intersection of two sets:**

The intersection of two sets *A* and *B* is the set whose elements are all of the elements common to both *A* and *B*. The intersection of the sets of ―*A*‖ and ―*B*‖ is denoted by *A B* and is read as ―*A* intersection *B*‖

**Difference of sets:** 83

If *A* and *B* are subsets of the universal set *U*, then the relative complement of *B* in *A*is the set of all elements in

*A* which are not in *A*. It is denoted by *A* – *B* thus: *A* – *B* = {*x* | *x* ∈ *A* and *x**B*}

**Complement of a set:**

If *U* is a universal set containing the set *A*, then *U* – *A* is called the complement of *A*. It is denoted by *A*1 . Thus

*A*1 = {*x*: *x**A*}

* 1. **Explain Inclusion-Exclusion Principle with example Inclusion-Exclusion Principle:**

The inclusion–exclusion principle is a counting technique which generalizes the familiar method of obtaining the number of elements in the unionof two finite sets; symbolically expressed as

|A 𝖴 B| = |A| + |B| − |A ∩ B|.

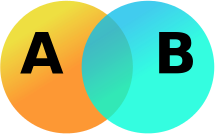


Fig.Venn diagram showing the union of sets A and B

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 𝖴 B𝖴 BC| = |A| + |B|+ |C| − |A ∩ B|− |C ∩ B| − |A ∩ C|+|A ∩B∩C|.

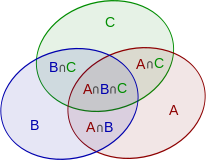
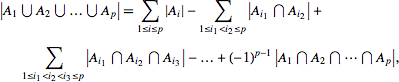


Fig.Inclusion–exclusion illustrated by a Venn diagram for three sets

This formula can be verified by counting how many times each region in the Venn diagram figure is included in the right-hand side of the formula. In this case, when removing the contributions of over-counted elements, the number of elements in the mutual intersection of the three sets has been subtracted too often, so must be added back in to get the correct total.

In general, Let A1, · · · , Ap be finite subsets of a set U. Then,

84

* 1. **a). How many natural numbers n ≤ 1000 are not divisible by any of**

**2, 3? Ans: Let A2 = {n** ∈ **N | n ≤ 1000, 2|n} and A3 = {n** ∈ **N | n ≤**

**1000, 3|n}.**

Then, |A2 𝖴 A3| = |A2| + |A3| − |A2 ∩ A3| = 500 + 333 − 166 = 667.

So, the required answer is 1000 − 667 = 333.

**b). How many integers between 1 and 10000 are divisible by none of 2, 3, 5, 7?**

Ans: For i ∈ {2, 3, 5, 7}, let Ai = {n ∈ N | n ≤ 10000, i|n}.

Therefore, the required answer is 10000 − |A2 𝖴 A3 𝖴 A5 𝖴 A7| = 2285.

* 1. **Define Relation,Binary relation, domain,range with example?**

**ANS** 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.

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, then we write a R b. If a is not in relation R to b, then 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)} 85

* 1. **Define a relation between two sets A = {5, 6, 7} and B = {x, y}.**

Solution: If A = {5, 6, 7} and B = {x, y}, then the subset R = {(5, x), (5, y), (6, x), (6, y)} is a relation from A to B.

**A = {2, 3, 4} and B = {3, 4, 5, 6, 7}. Define a relation from A to B by (a, b)** ∈ **R if a divides b.**

Solution: We obtain R = {(2, 4), (2, 6), (3, 3), (3, 6), (4, 4)}.

Domain of R = {2, 3, 4} and range of R = {3, 4, 6}.

* 1. **Write Relational Properties ANS**

A relation R on a set X is said to be

* + - Reflexive relation if xRx or (x, x) ∈ R, ∀x ∈ X
    - Symmetric relation if xRy then yRx, ∀x, y ∈ X
    - Transitive relation if xRy and yRz then xRz, ∀x, y, z ∈ X
    - Irreflexive relation if x

x or (x, x) R, ∀x ∈ X

* Antisymmetric relation if for every x and y in X, whenever xRy and yRx, then x = y.

**Examples**: (i). If R1 = {(1, 1), (1, 2), (2, 2), (2, 3), (3, 3)} be a relation on A = {1, 2, 3}, then R1 is a reflexive relation, since for every x ∈ A, (x, x) ∈ R1.

(ii). If R2 = {(1, 1), (1, 2), (2, 3), (3, 3)} be a relation on A = {1, 2, 3}, then R2 is not a reflexive relation, since for every 2 ∈ A, (2, 2) R2.

(iii). If R3 = {(1, 1), (1, 2), (1, 3), (2, 2), (2, 1), (3, 1)} be a relation on A = {1, 2, 3},

then R3 is a symmetric relation.

(iv). If R4 = {(1, 2), (2, 2), (2, 3)} on A = {1, 2, 3} is an antisymmetric.

**Example**: Given S = {1, 2, ..., 10} and a relation R on S, where R = {(x, y)| x + y = 10}.

What are the properties of the relation R?

Solution: Given that S = {1, 2, ..., 10}

= {(x, y)| x + y = 10}



= {(1, 9), (9, 1), (2, 8), (8, 2), (3, 7), (7, 3), (4, 6), (6, 4), (5, 5)}.

1. For any x ∈ S and (x, x) R. Here, 1 ∈ S but (1, 1) R.

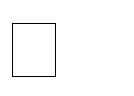
the relation R is not reflexive. It is also not irreflexive, since (5, 5) ∈ R (ii). (1, 9) ∈ R ⇒ (9, 1) ∈ R (2, 8) ∈ R

⇒ (8, 2) ∈ R…..

the relation is symmetric, but it is not antisymmetric.

(iii). (1, 9) ∈ R and (9, 1) ∈ R

⇒ (1, 1)



R

The relation R is not transitive. Hence, R is symmetric.

* 1. **Convert the Relation in Matrix format with example**

**ANS** Relation Matrix: A relation R from a finite set X to a finite set Y can be repre-sented by a matrix is called the relation matrix of R.

Let X = {x1, x2, ..., xm} and Y = {y1, y2, ..., yn} be finite sets containing m and n elements, respectively, and R be the relation from A to B. Then R can be represented by an m × n matrix

MR = [rij ], which is defined as follows:

1, if (xi , y j )  *R*

*rij* = 0, if (xi , y j )  *R*

**Example.** Let A = {1, 2, 3, 4} and B = {b1, b2, b3}. Consider the relation R = {(1, b2), (1, b3), (3, b2), (4, b1), (4, b3)}. Determine the matrix of the relation.

Solution: A = {1, 2, 3, 4}, B = {b1, b2, b3}.

Relation R = {(1, b2), (1, b3), (3, b2), (4, b1), (4, b3)}.

Matrix of the relation R is written as

 0 1 1

 0 0 0 

That is *MR* =  

 0 0 1 

 1 0 1



**Example**: Let *A* = *{*1*,* 2*,* 3*,* 4*}*. Find the relation *R* on *A* determined by the matrix

1 0



*M* = 0 0

1 0



1 0

*R* 1 0 

0

0





1 1

0 1

Solution: The relation *R* = *{*(1*,* 1)*,* (1*,* 3)*,*

(2*,* 3)*,* (3*,* 1)*,* (4*,* 1)*,* (4*,* 2)*,* (4*,* 4)*}*.

#### What is Digraph. Explain with example

**ANS Graph of a Relation:** A relation can also be represented pictorially by drawing its graph.

Let R be a relation in a set X = {x1, x2, ..., xm}. The elements of X are represented by points or circles called nodes. These nodes are called vertices.

If (xi, xj ) ∈ R, then we connect the nodes xi and xj by means of an arc and put an arrow on the arc in the direction from xi to xj . This is called an edge.

If all the nodes corresponding to the ordered pairs in R are connected by arcs with proper arrows, then we get a graph of the relation R.

Note: (i). If xiRxj and xj Rxi, then we draw two arcs between xi and xj with arrows pointing in both directions.

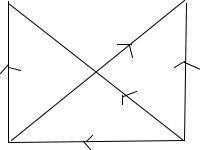
1. If xiRxi, then we get an arc which starts from node xi and returns to node xi. This arc is called a loop.

**Example:** Let X = {1, 2, 3, 4} and R={(x, y)| x > y}. Draw the graph of R and also give its matrix.

Solution: R = {(4, 1), (4, 3), (4, 2), (3, 1), (3, 2), (2, 1)}.

The graph of R and the matrix of R are

2



1

3 4

Graph of R

0 0 0 0

 

*MR* = 1 0 0 0



 1 0 

0

1



1 1 1 0

* 1. **Define Equivalence Relation with example**

ANS A relation R in a set X is called an equivalence relation if it is reflexive, symmetric and transitive.

The following are some examples of equivalence relations:

1. Equality of numbers on a set of real numbers.
2. Equality of subsets of a universal set.

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

Solution: (i). For any x ∈ X, x − x = 0 is divisible by 3.

∴ xRx

⇒ R is reflexive.

(ii) For any x, y ∈ X, if xRy, then x − y is divisible by 3.

−(x − y) is divisible by 3. 88

y − x is divisible by 3. yRx

Thus, the relationR is symmetric.

1. For any x, y, z ∈

X, let xRy and yRz.

(x − y) + (y − z) is divisible by 3 x − z is divisible by 3

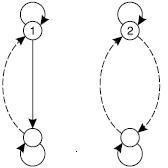
xRz

Hence, the relation R is transitive.

Thus, the relation R is an equivalence relation.

Example: Let X = {1, 2, 3, 4} and R == {(1, 1), (1, 4), (4, 1), (4, 4), (2, 2), (2, 3), (3, 2), (3, 3)}.

Prove that R is an equivalence relation.



* 1. Write a short notes on equivalence classes with example ANS

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, …}

89

#### Define a Composition relation with example ANS

**Definition** :Let R be a relation from X to Y and S be a relation from Y to Z. Then the relation RoS is given by

R o S = {(x, z) / xÎX 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 relation, then Ř is also an equivalence relation.

+ 2 3 n

**Definition** Let X be any finite set and R be a relation in X. Therelation *R* = *R*𝖴*R* 𝖴*R* 𝖴*· · ·*𝖴*R* in X.

is called the transitive closure of R in X

**Example**: Let the relation *R* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 3)*}* on the set *{*1*,* 2*,* 3*}*.

What is the transitive closure of *R*?

**Solution:** Given that *R* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 3)*}*.

The transitive closure of *R* is

*R*+ = *R* 𝖴 *R*2 𝖴 *R*3 𝖴 *· · ·* = R= *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 3)*}*

*R*2 = *R ◦ R* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 3)*} ◦ {*(1*,* 2)*,* (2*,* 3)*,* (3*,* 3)*}* = *{*(1*,* 3)*,*

(2*,* 3)*,* (3*,* 3)*}*

*R*3 = *R*2 *◦ R* = *{*(1*,* 3)*,* (2*,* 3)*,* (3*,* 3)*}*

*R*4 = *R*3 *◦ R* = *{*(1*,* 3)*,* (2*,* 3)*,* (3*,* 3)*}*

*R*+ = *R* 𝖴 *R*2 𝖴 *R*3 𝖴 *R*4 𝖴 *...* 90

= *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 3)*}* 𝖴 *{*(1*,* 3)*,* (2*,* 3)*,* (3*,* 3)*}* 𝖴 *{*(1*,* 3)*,* (2*,* 3)*,* (3*,* 3)*}* 𝖴 *...*

*={(1, 2), (1, 3), (2, 3), (3, 3)}.*

Therefore *R*+ = *{*(1*,* 2)*,* (1*,* 3)*,* (2*,* 3)*,* (3*,* 3)*}*.

**Example:** Let *X* = *{*1*,* 2*,* 3*,* 4*}* and *R* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 4)*}* be a relation on *X*. Find *R*+.

Solution: Given *R* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 4)*}*

*R*2 = *{*(1*,* 3)*,* (2*,* 4)*}*

*R*3 = *{*(1*,* 4)*}*

*R*4 = *{*(1*,* 4)*}*

*R*+ = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 4)*,* (1*,* 3)*,* (2*,* 4)*,* (1*,* 4)*}*.

**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.

* 1. **Write a short notes on Partial Ordering and POSET explain with example ANS.** A binary relation R in a set P is called a partial order relation or a partial

ordering in

**P iff R** is reflexive, antisymmetric, and transitive. i.e.,

* + - aRa for all a ∈ P
    - aRb and bRa ⇒ a = b
    - aRb and bRc ⇒ aRc

A set P together with a partial ordering R is called a partial ordered set or poset.

The relation R is often denoted by the symbol ≤ which is diﬀ erent from the usual less than equal to symbol.

Thus, if ≤ is a partial order in P , then the ordered pair (P, ≤) is called a poset.

**Example:** Show that the relation ‖greater than or equal to‖ is a partial ordering on the set of integers.

**Solution:** Let Z be the set of all integers and the relation R =′≥′ (i). Since a ≥ a for every integer a, the relation ′ ≥′ is reflexive. (ii). Let a and b be any two integers.

Let aRb and bRa ⇒ a ≥ b and b ≥ a

⇒ a = b

∴ The relation ′ ≥′ is antisymmetric.

(iii). Let a, b and c be any three integers. Let aRb and bRc ⇒ a ≥ b and b ≥ c

⇒ a ≥ c

∴ The relation ′ ≥′ is transitive.

Since the relation ′ ≥′ is reflexive, antisymmetric and transitive, ′ ≥′ is partial ordering on the set of integers. Therefore, (Z, ≥) is a **poset.**

**Example**: Show that the inclusion ⊆ is a partial ordering on the set power set of a set S.

**Solution**: Since (i). A ⊆ A for all A ⊆ S, ⊆ is reflexive. (ii). A ⊆ B and B ⊆ A ⇒ A = B, ⊆ is antisymmetric.

(iii). A ⊆ B and B ⊆ C ⇒ A ⊆ C, ⊆ is transitive.

Thus, the relation ⊆ is a partial ordering on the power set of S.

**Example:** Show that the divisibility relation ′/′ is a partial ordering on the set

of positive

integers.

**Solution**: Let Z+ be the set of positive integers.

Since (i). *a/a* for all *a* ∈ *Z*+, */* is reflexive. (ii). *a/b* and *b/a* ⇒ *a* = *b*, */* is

antisymmetric. (iii). *a/b* and *b/c* ⇒

*a/c*, */* is transitive.

It follows that */* is a partial ordering on *Z*+ and (*Z*+*, /*) is a poset.

which case we say that *x* and *y* are **incomparable**. **Examples:**

1. The poset (*Z, ≤*) is a **totally ordered**.

Since *a ≤ b* or *b ≤ a* whenever *a* and *b* are integers.

1. The divisibility relation */* is a partial ordering on the set of positive integers.

Therefore (*Z*+*, /*) is a poset and it is not a totally ordered, since it contain elements that are

**incomparable**, such as 5 and 7, 3 and 5.

**Definition:** In a poset (*P, ≤*), an element *y* ∈ *P* is said to ***cover*** an element *x* ∈ *P* if *x < y* and if there does not exist any element *z* ∈ *P* such that *x ≤ z* and *z ≤ y*;

that is, *y* covers *x* ⇔ (*x < y* 𝖠 (*x ≤ z≤ y* ⇒ *x* = *z* ∨ *z* = *y*)).

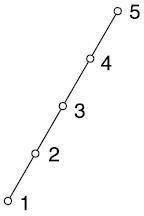
#### Define Hasse diagram and how to construct the Hasse diagram with example ANS;

A partial order *≤* on a set *P* can be represented by means of a diagram known as Hasse diagram of (*P, ≤*). In such a diagram,

1. Each element is represented by a small circle or dot.
2. The circle for *x* ∈ *P* is drawn below the circle for *y* ∈ *P* if *x < y*, and a line is drawn between *x* and *y* if *y* covers *x*.
3. If *x < y* but *y* does not cover *x*, then *x* and *y* are not connected directly by a single line.

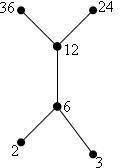
Note: For totally ordered set (*P, ≤*), the Hasse diagram consists of circles one below the other. The poset is called a chain.

**Example:** Let *P* = *{*1*,* 2*,* 3*,* 4*,* 5*}* and *≤* be the relation ‖less than or equal to‖ then the Hasse diagram is:



It is a totally ordered set.

Example: Let *X* = *{*2*,* 3*,* 6*,* 12*,* 24*,* 36*}*, and the relation *≤* be such that *x ≤ y* if *x* divides *y*. Draw the Hasse diagram of (*X, ≤*). Solution: The Hasse diagram is is shown below:



It is not a total order set.

Example: Draw the Hasse diagram for the relation *R* on *A* = *{*1*,* 2*,* 3*,* 4*,* 5*}* whose relation matrix given below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1   | 0 | 1 | 1 | 1   |
|  0 | 1 | 1 | 1 | 1 |

0 0 1 1 1

*M* = 

*R*  0 0 0 1 0

 0 0 0 0 1 

Solution:

 

 

R= *{*(1*,* 1)*,* (1*,* 3)*,* (1*,* 4)*,* (1*,* 5)*,* (2*,* 2)*,* (2*,* 3)*,* (2*,* 4)*,* (2*,* 5)*,* (3*,* 3)*,* (3*,* 4)*,* (3*,* 5)*,* (4*,* 4)*,* (5*.*5)*}.*

Hasse diagram for *MR* is



4

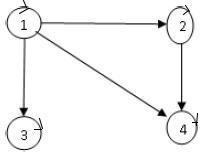
5

3

1

2

Example: A partial order *R* on the set *A* = *{*1*,* 2*,* 3*,* 4*}* is represented by the following digraph. Draw the Hasse diagram for R.

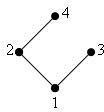


Solution: By examining the given digraph , we find that

R= *{*(1*,* 1)*,* (1*,* 2)*,* (1*,* 3)*,* (1*,* 4)*,* (2*,* 2)*,* (2*,* 4)*,* (3*,* 3)*,* (4*,* 4)*}*.

We check that *R* is reflexive, transitive and antisymmetric. Therefore, *R* is partial order relation on *A*.

The hasse diagram of *R* is shown below:



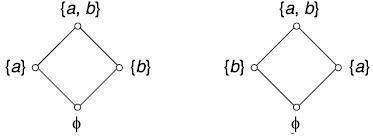
Example: Let *A* be a finite set and *ρ*(*A*) be its power set. Let ⊆ be the inclusion relation on the elements of *ρ*(*A*). Draw the Hasse diagram of *ρ*(*A*)*,* ⊆) for

* *A* = *{a}*
* *A* = *{a, b}*. Solution: (i). Let *A* = *{a}*

*ρ*(*A*) = *{ϕ, a}*

Hasse diagram of (*ρ*(*A*)*,* ⊆) is shown in Fig:

(ii). Let *A* = *{a, b}*. *ρ*(*A*) = *{ϕ, {a}, {b}, {a, b}}*. The Hasse diagram for (*ρ*(*A*)*,* ⊆) is shown in fig:

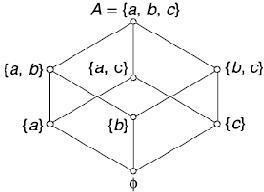


Example: Draw the Hasse diagram for the partial ordering ⊆ on the power set *P* (*S*) where *S* = *{*9*a*5*, b, c}*.

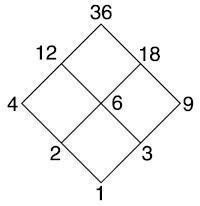
Solution: *S* = *{a, b, c}*.

*P* (*S*) = *{ϕ, {a}, {b}, {c}, {a, b}, {a, c}, {b, c}, {a, b, c}}*.

Hasse diagram for the partial ordered set is shown in fig:



Example: Draw the Hasse diagram representing the positive divisions of 36 (i.e., *D*36).

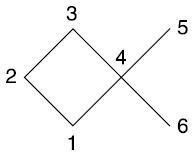
Solution: We have *D*36 = *{*1*,* 2*,* 3*,* 4*,* 6*,* 9*,* 12*,* 18*,* 36*}* if and only *a* divides *b*. The Hasse diagram for *R* is shown in Fig.

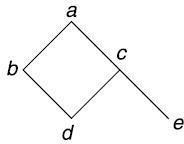
**Minimal and Maximal elements(members):** Let (*P, ≤*) denote a partially or-dered set. An element *y* ∈ *P* is called a *minimal member* of *P* relative to *≤* if for no *x* ∈ *P* , is *x < y*.

Similarly an element *y* ∈ *P* is called a maximal member of *P* relative to the partial ordering *≤* if

for no *x* ∈ *P* , is *y < x*. Note:

1. The minimal and maximal members of a partially ordered set need not unique.
2. Maximal and minimal elements are easily calculated from the Hasse diagram. They are the *'top'* and *'bottom'* elements in the diagram.

Example:

In the Hasse diagram, there are two maximal elements and two minimal elements. The elements 3, 5 are maximal and the elements 1 and 6 are minimal.

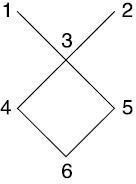
Example: Let *A* = *{a, b, c, d, e}* and let the partial order on *A* in the natural way.

The element *a* is maximal.

The elements *d* and *e* are minimal.

Upper and Lower Bounds: Let (*P, ≤*) be a partially ordered set and let *A* ⊆ *P* . Any element *x* ∈ *P*96

is called an *upper bound* for *A* if for all *a* ∈ *A*, *a ≤ x*. Similarly, any element *x* ∈ *P* is called a

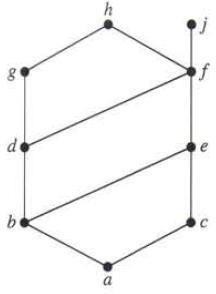
*lower bound* for *A* if for all *a* ∈ *A*, *x ≤ a*. Example: *A* = *{*1*,* 2*,* 3*, ...,* 6*}* be ordered as pictured in figure.

If *B* = *{*4*,* 5*}* then the upper bounds of *B* are 1, 2, 3. The lower bound of *B* is 6.

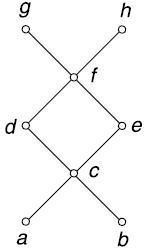
Least Upper Bound and Greatest Lower Bound:

Let (*P, ≤*) be a partial ordered set and let *A* ⊆ *P* . An element *x* ∈ *P* is a *least upper bound* or *supremum* for *A* if *x* is an upper bound for *A* and *x ≤ y* where *y* is any upper bound for *A*. Similarly, the *the greatest lower bound* or *in mum* for *A* is an element *x* ∈ *P* such that *x* is a lower bound and *y ≤ x* for all lower bounds *y*.

Example: Find the great lower bound and the least upper bound of *{b, d, g}*, if they exist in the

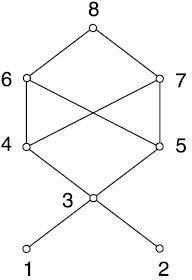
poset shown in fig:

Solution: The upper bounds of *{b, d, g}* are *g* and *h*. Since *g < h*, *g* is the least upper bound. The lower bounds of *{b, d, g}* are *a* and *b*. Since *a < b*, *b* is the greatest lower bound.

Example: Let *A* = *{a, b, c, d, e, f, g, h}* denote a partially ordered set whose Hasse diagram is shown in Fig:

If *B* = *{c, d, e}* then *f, g, h* are upper bounds of *B*. The element *f* is least upper bound.

Example: Consider the poset *A* = *{*1*,* 2*,* 3*,* 4*,* 5*,* 6*,* 7*,* 8*}* whose Hasse diagram is shown in Fig and

let *B* = *{*3*,* 4*,* 5*}*

The elements 1, 2, 3 are lower bounds of *B*. 3 is greatest lower bound.

97

#### Define the Function, Domain and Range with example ANS

A function is a special case of relation.

**Definition**: Let *X* and *Y* be any two sets. A relation *f* from *X* to *Y* is called a function if for every *x* ∈ *X*, there is a unique element *y* ∈ *Y* such that (*x, y*) ∈ *f*.

Note: The definition of function requires that a relation must satisfies two additional conditions in order to qualify as a function.

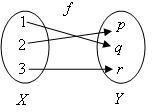
These conditions are as follows:

For every x ∈ X must be related to some y ∈ Y , i.e., the domain of f must be X and nor merely a subset of X.

Uniqueness, i.e., (*x, y*) ∈ *f* and (*x, z*) ∈ *f* ⇒ *y* = *z*.

The notation *f* : *X → Y* , means *f* is a function from *X* to*Y* .

Example: Let *X* = *{*1*,* 2*,* 3*}*, *Y* = *{p, q, r}* and *f* = *{*(1*, p*)*,* (2*, q*)*,* (3*, r*)*}* then

*f*(1) = *p, f*(2) = *q, f*(3) = *r*. Clearly *f* is a function from *X* to *Y* .

**Domain and Range of a Function:** If *f* : *X → Y* is a function, then *X* is called the Domain of *f* and the set *Y* is called the codomain of *f*.

The range of *f* is defined as the set of all images under *f*.

It is denoted by *f*(*X*) = *{y|* for some *x* in *X, f*(*x*) = *y}* and is called the image of *X* in *Y* . The Range *f* is also denoted by *Rf* .

**Example:** If the function *f* is defined by *f*(*x*)=*x*2 + 1 on the set *{−*2*, −*1*,* 0*,* 1*,* 2*}*, find the range of *f*.

Solution: *f*(*−*2) = (*−*2)2 + 1 = 5

*f*(*−*1) = (*−*1)2 + 1 = 2

*f*(0) = 0 + 1 = 1

*f*(1) = 1 + 1 = 2

*f*(2) = 4 + 1 = 5

Therefore, the range of *f* = *{*1*,* 2*,* 5*}*.

98

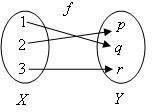
#### Write types of functions with examples

**One-to-one(Injection):** A mapping *f* : *X → Y* is called *one-to-one* if distinct elements of *X* are mapped into distinct elements of *Y* , i.e., *f* is one-to-one if

*x*1 =*̸ x*2 ⇒ *f*(*x*1) =*̸*

equivalently *f*(*x*1) = *f*(*x*2) ⇒ *x*1 = *x*2 for *x*1*, x*2 ∈ *X*.

(*fx*2)or



**Example**: *f* : *R → R* defined by *f*(*x*) = 3*x*, ∀*x* ∈*R* is one-one, since

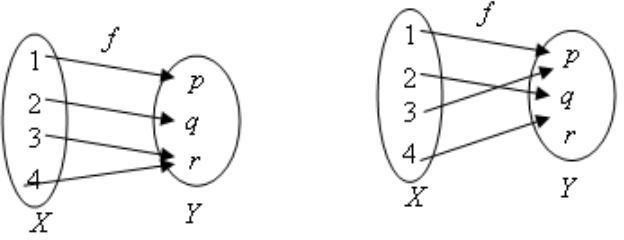
*f*(*x*1) = *f*(*x*2) ⇒ 3*x*1 = 3*x*2 ⇒ *x*1 = *x*2*,* ∀*x*1*, x*2 ∈*R.*

**Example:** Determine whether *f* : *Z → Z* given by *f*(*x*) = *x*2, *x* ∈ *Z* is a one-to-One function. Solution: The function *f* : *Z → Z* given by *f*(*x*) = *x*2, *x* ∈ *Z* is not a one-to-one function. This is because both 3 and -3 have 9 as their image, which is against the definition of a one-to-one function.

**Onto(Surjection):** A mapping *f* : *X → Y* is called *onto* if the range set *Rf* = *Y* .

If *f* : *X → Y* is onto, then each element of *Y* is *f*-image of atleast one element of *X*. i.e., *{f*(*x*) : *x* ∈ *X}* = *Y* .

If *f* is not onto, then it is said to be *into*.

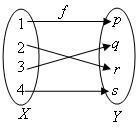


Surjective Not Surjective

**Example**: *f* : *R → R*, given by *f*(*x*) = 2*x,* ∀*x* ∈ *R* is onto.

**Bijection or One-to-One, Onto:** A mapping f : X → Y is called one-to-one, onto or bijective if it is both one-to-one and onto. Such a mapping is also called a

one-to-one correspondence between X and Y .



**Example**: Show that a mapping *f* : *R → R* defined by *f*(*x*) = 2*x* + 1 for *x* ∈ *R* is a bijective map from *R* to *R*.

**Solution**: Let *f* : *R → R* defined by *f*(*x*) = 2*x* + 1 for *x* ∈ *R*. We need to prove that f is a

bijective map, i.e., it is enough to prove that *f* is one-one and onto.

* Proof of *f* being one-to-one

Let *x* and *y* be any two elements in *R* such that *f*(*x*) = *f*(*y*)

⇒ 2*x* + 1 = 2*y* + 1

⇒ *x* = *y*

Thus, *f*(*x*) = *f*(*y*) ⇒ *x* = *y*

This implies that *f* is one-to-one.

* Proof of *f* being onto

Let *y* be any element in the codomain *R*

⇒ *f*(*x*) = *y*

⇒ 2*x* + 1 = *y*

⇒ *x* = *(y-1)/2*

Clearly, *x* = *(y-1)/2*∈ *R*

Thus, every element in the codomain has pre-image in the domain. This implies that *f* is onto

Hence, *f* is a bijective map.

**Identity function:** Let *X* be any set and *f* be a function such that *f* : *X → X* is defined by *f*(*x*) = *x*

for all *x* ∈ *X*. Then, *f* is called the identity function or identity transformation on *X*. It can be

denoted by *I* or *Ix*.

Note: The identity function is both one-to-one and onto.

Let *Ix*(*x*) = *Ix*(*y*)

⇒ *x* = *y*

⇒ *Ix* is one-to-one

* 1. **Explain omposite and Inverse functions ANS**

Let f : X → Y and g : Y → Z be two functions. Then the composition of f and g denoted by g ◦ f, is the function from X to Z defined as (g ◦ f)(x) = g(f(x)), for all x ∈ X.

Note. In the above definition it is assumed that the range of the function f is a subset of Y (the Domain of g), i.e., Rf ⊆ Dg. g ◦ f is called the left composition g with f.

**Example:** Let X = {1, 2, 3}, Y = {p, q} and Z = {a, b}. Also let f : X → Y be

f = {(1, p), (2, q), (3,} and g : Y → Z be given by g = {(p, b), (q, b)}. Find g ◦ f.

**Solution:** g ◦ f = {(1, b), (2, b), (3, b).

**Example:** Let *X* = *{*1*,* 2*,* 3*}* and *f, g, h* and *s* be the functions from *X* to *X* iven by *f* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 1)*} g* = *{*(1*,* 2)*,* (2*,* 1)*,* (3*,* 3)*}*

*h* = *{*(1*,* 1)*,* (2*,* 2)*,* (3*,* 1)*} s* = *{*(1*,* 1)*,* (2*,* 2)*,* (3*,* 3)*}*

Find *f ◦ f*; *g ◦ f*; *f ◦ h ◦ g*; *s ◦ g*; *g ◦ s*; *s ◦ s*; and *f ◦ s*.

Solution:

*f ◦ g* = *{*(1*,* 3)*,* (2*,* 2)*,* (3*,* 1)*}*

*g ◦ f* = *{*(1*,* 1)*,* (2*,* 3)*,* (3*,* 2)*} ̸*= *f ◦ g*

*f ◦ h ◦ g* = *f ◦* (*h ◦ g*) = *f ◦ {*(1*,* 2)*,* (2*,* 1)*,* (3*,* 1)*}*

*= {*(1*,* 3)*,* (2*,* 2)*,* (3*,* 2)*}*

*s ◦ g* = *{*(1*,* 2)*,* (2*,* 1)*,* (3*,* 3)*}* = *g*

*g ◦ s* = *{*(1*,* 2)*,* (2*,* 1)*,* (3*,* 3)*}*

∴ *s ◦ g* = *g ◦ s* = *g*

*s ◦ s* = *{*(1*,* 1)*,* (2*,* 2)*,* (3*,* 3)*}* = *s*

*f ◦ s* = *{*(1*,* 2)*,* (2*,* 3)*,* (3*,* 1)*}*

Thus, s ◦ s = s, f ◦ g ≠g ◦ f, s ◦ g = g ◦ s = g and h ◦ s = s ◦ h = h.

Example: Let *f*(*x*) = *x* + 2, *g*(*x*) = *x −* 2 and *h*(*x*) = 3*x* for *x* ∈ *R*, where *R* is the set of real numbers. Find *g ◦ f*; *f ◦ g*; *f ◦ f*; *g ◦ g*; *f ◦ h*; *h ◦ g*; *h ◦ f*; and *f ◦ h ◦ g*.

Solution: *f* : *R → R* is defined by *f*(*x*) = *x* + 2

f: *R → R* is defined by *g*(*x*) = *x −* 2

*h* : *R → R* is defined by *h*(*x*) = 3*x*

* *g ◦ f* : *R → R*

Let *x* ∈ *R*. Thus, we can write

(*g ◦ f*)(*x*) = *g*(*f*(*x*)) = *g*(*x* + 2) = *x* + 2 *−* 2 = *x*

∴ (*g ◦ f*)(*x*) = *{*(*x, x*)*| x* ∈ *R}*

* + (*f ◦ g*)(*x*) = *f*(*g*(*x*)) = *f*(*x −* 2) = (*x −* 2) + 2 = *x*

∴ *f ◦ g* = *{*(*x, x*)*| x* ∈ *R}*

* + (*f ◦ f*)(*x*) = *f*(*f*(*x*)) = *f*(*x* + 2) = *x* + 2 + 2 = *x* + 4

∴ *f ◦ f* = *{*(*x, x* + 4)*| x* ∈ *R}*

* + (*g ◦ g*)(*x*) = *g*(*g*(*x*)) = *g*(*x −* 2) = *x −* 2 *−* 2 = *x −* 4

⇒ *g ◦ g* = *{*(*x, x −* 4)*| x* ∈ *R}*

* + (*f ◦ h*)(*x*) = *f*(*h*(*x*)) = *f*(3*x*) = 3*x* + 2

∴ *f ◦ h* = *{*(*x,* 3*x* + 2)*| x* ∈ *R}*

* + (*h ◦ g*)(*x*) = *h*(*g*(*x*)) = *h*(*x −* 2) = 3(*x −* 2) = 3*x −* 6

∴ *h ◦ g* = *{*(*x,* 3*x −* 6)*| x* ∈ *R}*

* + (*h ◦ f*)(*x*) = *h*(*f*(*x*)) = *h*(*x* + 2) = 3(*x* + 2) = 3*x* + 6 *h ◦ f* =

*{*(*x,* 3*x* + 6)*| x* ∈ *R}*

* + (*f ◦ h ◦ g*)(*x*) = [*f ◦* (*h ◦ g*)](*x*)

*f*(*h ◦ g*(*x*)) = *f*(3*x −* 6) = 3*x −* 6 + 2 = 3*x −* 4

∴ *f ◦ h ◦ g* = *{*(*x,* 3*x −* 4)*| x* ∈ *R}*.

#### Example: What is composition of functions? Let *f* and *g* be functions from *R* to *R*, where *R* is a set of real numbers defined by *f*(*x*) = *x*2 + 3*x* + 1 and *g*(*x*) = 2*x −* 3.

**Find the composition of functions: i) *f ◦ f* ii) *f ◦ g* iii) *g ◦ f*.**

***Inverse Functions***

A function *f* : *X → Y* is aid to be *invertible* of its inverse function *f−*1 is also function from the range of *f* into *X*.

Theorem: A function *f* : *X → Y* is invertible ⇔ *f* is one-to-one and onto.

**Example:** Let *X* = *{a, b, c, d}* and *Y* = *{*(1*,* 2*,* 3*,* 4*}* and let *f* : *X → Y* be given by

f = *{*(*a,* 1)*,* (*b,* 2)*,* (*c,* 2)*,* (*d,* 3)*}*. Is *f−*1 a function?

Solution: *f−*1 = *{*(1*, a*)*,* (2*, b*)*,* (2*, c*)*,* (3*, d*)*}*. Here, 2 has two distinct images *b* and *c*. 10

Therefore, *f−*1 is not a function.

. **Example:** Let *R* be the set of real numbers and *f* : *R → R* be given by

*f* = *{*(*x, x*2)*| x* ∈ *R}*. Is *f−*1 a function?

**Solution:** The inverse of the given function is defined as *f−*1 =

*{*(*x*2*, x*)*| x* ∈ *R}*.

and *g* are both

Therefore, it is not a function.

Theorem: If *f* : *X → Y* and *g* : *Y → X* be such that *g ◦ f* = *Ix* and *f ◦ g* = *Iy*, then f invertible. Furthermore, *f−*1 = *g* and *g−*1 = *f*.

#### Example: Let X = {1, 2, 3, 4} and f and g be functions from X to X given by f = {(1, 4), (2, 1), (3,, (4, 3)} and g = {(1, 2), (2, 3), (3, 4), (4, 1)}.

**Prove that f and g are inverses of each other.**

#### Solution: We check that

|  |  |  |  |
| --- | --- | --- | --- |
| (*g ◦ f*)(1) = *g*(*f*(1)) = *g*(4) = 1 | = *Ix*(1)*,* | (*f ◦ g*)(1) | = *f*(*g*(1)) = *f*(2) = 1 = *Ix*(1)*.* |
| (*g ◦ f*)(2) = *g*(*f*(2)) = *g*(1) = 2 | = *Ix*(2)*,* | (*f ◦ g*)(2) | = *f*(*g*(2)) = *f*(3) = 2 = *Ix*(2)*.* |
| (*g ◦ f*)(3) = *g*(*f*(3)) = *g*(2) | = 3= *Ix*(3)*,* | (*f ◦ g*)(3) | = *f*(*g*(3)) = *f*(4) = 3 = *Ix*(3)*.* |
| (*g ◦ f*)(4) = *g*(*f*(4)) = *g*(3) | = 4= *Ix*(4)*,* | (*f ◦ g*)(4) | = *f*(*g*(4)) = *f*(1) = 4 = *Ix*(4)*.* |

another.

Thus, for all *x* ∈ *X*, (*g ◦ f*)(*x*) = *Ix*(*x*) and (*f ◦ g*)(*x*) = *Ix*(*x*). Therefore *g* is inverse of *f* and *f* is inverse of *g*.

**Example:** Show that the functions *f*(*x*) = *x*3 and *g*(*x*) = *x*1*/*3 for *x* ∈*R* are inverses of one

Solution: *f* : *R → R* is defined by *f*(*x*) = *x*3 ;

f: *R → R* is defined by *g*(*x*) = *x*1*/*3

(*f ◦ g*)(*x*) = *f*(*g*(*x*)) = *f*(*x*1*/*3) = *x*3(1*/*3) = *x* = *Ix*(*x*)

i.e., (*f ◦ g*)(*x*) = *Ix*(*x*)

and (*g ◦ f*)(*x*) = *g*(*f*(*x*)) = *g*(*x*3) = *x*3(1*/*3) = *x* = *Ix*(*x*) i.e., (*g ◦ f*)(*x*) = *Ix*(*x*)

Thus, *f* = *g−*1 or *g* = *f−*1

i.e., *f* and *g* are inverses of one other.

**\*\*\*Example**: *f* : *R → R* is defined by *f*(*x*) = *ax* + *b*, for *a, b* ∈ *R* and *a* =*̸* 0.

Show that *f*is

invertible and find the inverse of *f*.

* 1. First we shall show that *f* is one-to-one Let *x*1*, x*2 ∈ *R* such that *f*(*x*1) = *f*(*x*2)

⇒ *ax*1 + *b* = *ax*2 + *b*

⇒ *ax*1 = *ax*2

⇒ *x*1 = *x*2

∴ *f* is one-to-one.

To show that *f* is onto.

Let *y* ∈ *R*(codomain) such that *y* = *f*(*x*) for some *x* ∈ *R*.

⇒ *y* = *ax* + *b*

⇒ *ax* = *y − b*

⇒ *x* = *(y-b)/a*

Given *y* ∈ *R*(codomain), there exists an element *x* = *(y-b)/a* ∈ *R* s~~uch~~ that *f*(*x*) = *y*.

∴ *f* is onto

*f* is invertible and *f−*1(*x*)= *(x-b)/a*

Example: Let *f* : *R → R* be given by *f*(*x*) = *x*3 *−* 2. Find *f−*1.

(i) First we shall show that *f* is one-to-one Let *x*1*, x*2 ∈ *R* such that *f*(*x*1) = *f*(*x*2)

⇒ *x*31 *−* 2 = *x*32 *−*

2 ⇒ *x*31 = *x*32

⇒ *x*1 = *x*2

∴ *f* is one-to-one.

the

called the

#### Write a short notes on Floor and Ceiling functions: with example

**ANS**. Let *x* be a real number, then the least integer that is not less than *x* is called

**CEILING** of *x*. The CEILING of *x* is denoted by ⌈x⌉. Examples: ⌈2.15⌉ = 3,⌈ √ 5⌉ = 3,⌈ −7.4⌉ = −7, ⌈−2⌉ = −2

Let *x* be any real number, then the greatest integer that does not exceed *x* is

**Floor** of *x*. The FLOOR of *x* is denoted by ⌊x⌋.

Examples: ⌊5.14⌋ = 5, ⌊ √5⌋ = 2,⌊ −7.6⌋ = −8,⌊6⌋ = 6,⌊ −3⌋ = −3

#### Example: Let *f* and *g* abe functions from the positive real numbers to positive real numbers

**defined by *f*(*x*) =** ⌊**2*x***⌋**, *g*(*x*) = *x*2. Calculate *f ◦ g* and *g ◦ f*.**

Solution: f ◦ g(x) = f(g(x)) =f(x2)=⌊2x2⌋

g ◦ f(x) = g(f(x))=g(⌊2x⌋)=(⌊2x⌋)2

#### What is Lattice and write properties with example

**Definition:** A lattice is a partially ordered set (*L, ≤*) in which every pair of elements

*a, b* ∈ *L* has a greatest lower bound and a least upper bound.

**Example:** Let *Z*+ denote the set of all positive integers and let *R* denote the relation ‘division‘ In *Z*+, such that for any two elements *a, b* ∈ *Z*+, *aRb*, if *a* divides *b*.

Then (*Z*+*, R*) is a lattice in which the join of *a* and *b* is the least common multiple of *a* and *b*,

i.e *a* ∨ *b* = *a* ⊕ *b* = LCM of *a* and *b,*

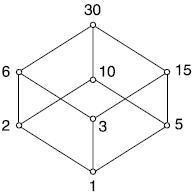
and the meet of *a* and *b*, i.e. *a* ∗ *b* is the greatest common divisor (GCD) of *a* and *b*

i.e., *a* 𝖠*b* = *a* ∗ *b* = GCD of *a* and *b.*

We can also write *a*+*b* = *a*∨*b* = *a*⊕*b*=LCM of *a* and *b* and *a.b* = *a*𝖠*b* = *a*∗*b*=GCD of *a* and *b*.

**Example:** Let *n* be a positive integer and *Sn* be the set of all divisors of *n* If *n* = 30, *S*30 = *{*1*,* 2*,*

3*,* 5*,* 6*,* 10*,* 15*,* 30*}*. Let *R* denote the relation division as defined in Example 1. Then (*S*30*, R*) is a Lattice see Fig:



Example: Let *A* be any set and *P* (*A*) be its power set. The poset *P* (*A*)*,* ⊆) is a lattice in which the meet and join are the same as the operations *∩* and 𝖴 on sets respectively.

*S* = *{a}*, *P* (*A*) = *{ϕ, {a}}*



*S* = *{a, b}*, *P* (*A*) = *{ϕ, {a}, {a}, S}*.

*≤*).

#### Some Properties of Lattice

Let (*L, ≤*) be a lattice and ∗ and ⊕ denote the two binary operation meet and join on (*L,*

Then for any *a, b, c* ∈ *L*, we have

(L1)*: a*∗a = a, *(L1)′ : a*⊕a = a (Idempotent laws)

(*L*2): *b*∗*a* = *b*∗*a*, (*L*2)*′* : *a* ⊕*b* = *b* + *a* (Commutative laws)

(*L*3) : (*a*∗*b*)∗*c* = *a*∗(*b*∗*c*)*,* (*L*3)*′* : (*a*⊕*b*)⊕*c* = *a*⊕(*b* + *c*) (Associative laws)

(*L*4) : *a*∗(*a* + *b*) = *a,*(*L*4)*′* : *a*⊕(*a*∗*b*) = *a* (Absorption laws).

The above properties (*L*1) to (*L*4) can be proved easily by using definitions of

meet and join.

We can apply the principle of duality and obtain (*L*1)*′* to (*L*4)*′*.

#### Define the terms with examples

1. ***Bounded Lattice (ii) Distributive lattice (iii)* Complemented lattice:**

#### Bounded Lattice:

**A bounded lattice** is an algebraic structure (L,,,0,1) sucha that (L,,) is a lattice, and the constants 0,1∈ L satisfy the following:

* 1. for all x∈ L, x1=x and x1=1
  2. for all x∈ L, x0=0 and x0=x.

The element 1 is called the upper bound, or top of L and the element 0 is called the lower bound or bottom of L.

*Distributive lattice:*

A lattice (*L*,∨,𝖠) is **distributive** if the following additional identity holds for all *x*, *y*, and *z* in *L*: *x* 𝖠 (*y* ∨ *z*) = (*x* 𝖠 *y*) ∨ (*x* 𝖠 *z*)

Viewing lattices as partially ordered sets, this says that the meet peration preserves nonempty

finite joins. It is a basic fact of lattice theory that the above condition is equivalent to its dual

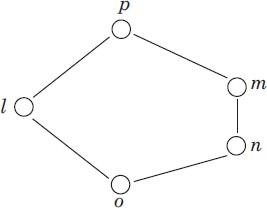
*x* ∨ (*y* 𝖠 *z*) = (*x* ∨ *y*) 𝖠 (*x* ∨ *z*) for all *x*, *y*, and *z* in *L*.

**Example:** Show that the following simple but significant lattices are not distributive.

Solution a) To see that the diamond lattice is not distributive, use the middle elements of the lattice: a 𝖠 (b ∨ c) = a 𝖠 1 = a, but (a 𝖠 b) ∨ (a 𝖠 c) = 0 ∨ 0 = 0, and a ≠0. 10

Similarly, the other distributive law fails for these three elements.

1. The pentagon lattice is also not distributive

**Example:** Show that lattice is not a distributive lattice.

**Sol.** A lattice is distributive if all of its elements follow distributive property so let we verify the distributive property between the elements *n*, *l* and *m*.

GLB(*n*, LUB(*l*, *m*)) = GLB(*n*, *p*) [∴ LUB(*l*, *m*) = *p*]

= *n* (LHS)

also LUB(GLB(*n*, *l*), GLB(*n*, *m*)) = LUB(*o*, *n*); [∴ GLB(*n*, *l*) = *o* and GLB(*n*, *m*) = *n*]

= *n* (RHS)

so LHS = RHS.

But GLB(*m*, LUB(*l*, *n*)) = GLB(*m*, *p*) [∴ LUB(*l*, *n*) = *p*]

= *m* (LHS)

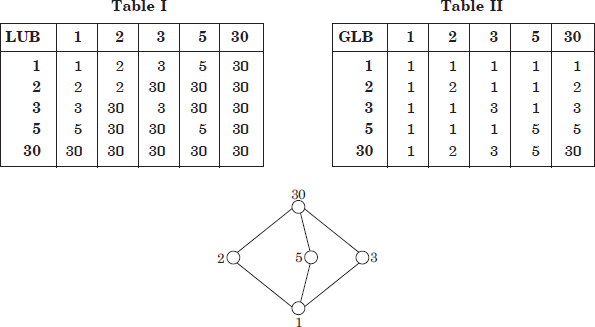
also LUB(GLB(*m*, *l*), GLB(*m*, *n*)) = LUB(*o*, *n*); [∴ GLB(*m*, *l*) = *o* and GLB(*m*, *n*) = *n*]

= *n* (RHS)

Thus, LHS ≠ RHS hence distributive property doesn‘t hold by the lattice so lattice is not distributive.

**Example:** Consider the poset (X, ≤ ) where X = {1, 2, 3, 5, 30} and the partial ordered relation ≤ is defined as i.e. if x and y ∈X then x ≤ y means ‗x divides y‘. Then show that poset (I+, ≤) is a lattice.

**Sol.** Since GLB(*x*, *y*) = *x* 𝖠 *y* = *lcm*(*x*, *y*) and LUB(*x*, *y*) = *x* ∨ *y* = *gcd*(*x*, *y*)

Now we can construct the operation table I and table II for GLB and LUB respectively and the Hasse diagram is shown in Fig.

Test for distributive lattice, i.e.,

GLB(*x*, LUB(*y*, *z*)) = LUB(GLB(*x*, *y*), GLB(*x*, *z*))

Assume *x* = 2, *y* = 3 and *z* = 5, then

*RHS:*GLB(2, LUB(3, 5)) = GLB(2, 30) = 2

*LHS:* LUB(GLB(2, 3), GLB(2, 5)) = LUB(1, 1) = 1 10

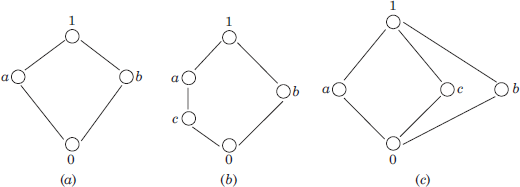
Since*RHS* ≠ *LHS*, hence lattice is not a distributive lattice.

**Complemented lattice:**

A complemented lattice is a bounded lattice (with least element 0 and greatest element 1), in which every element a has a complement, i.e. an element b satisfying

a ∨ b = 1 and a 𝖠 b = 0. Complements need not be unique.

**Example:** Lattices shown in Fig (a), (b) and (c) are complemented lattices.



***Sol.***

For the lattice (*a*) GLB(*a*, *b*) = 0 and LUB(*x*, *y*) = 1. So, the complement *a* is *b* and vise versa. Hence, a complement lattice.

For the lattice (*b*) GLB(*a*, *b*) = 0 and GLB(*c*, *b*) = 0 and LUB(*a*, *b*) = 1 and LUB(*c*, *b*) = 1; so both *a* and *c* are complement of *b.*

Hence, a complement lattice.

In the lattice (*c*) GLB(*a*, *c*) = 0 and LUB(*a*, *c*) = 1; GLB(*a*, *b*) = 0 and LUB(*a*, *b*) = 1.

So, complement of *a* are *b* and *c*.

Similarly complement of *c* are *a* and *b* also *a* and *c* are complement of *b*.

Hence lattice is a complement lattice.

11

UNIT –III

* 1. **Define algorithm and properties with example**

**Definition** An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

**Example:** Describe an algorithm for finding the maximum value in a finite sequence of integers.

#### Description of algorithms in pseudocode:

* + - Intermediate step between English prose and formal coding in a programming language.
    - Focus on the fundamental operation of the program, instead of peculiarities of a given programming language.
    - Analyze the time required to solve a problem using an algorithm, independent of the actual programming language.

#### Properties of Algorithms

**Input:** An algorithm has input values from a specified set.

**Output:** From the input values, the algorithm produces the output values from a specified set. The output values are the solution.

**Correctness:** An algorithm should produce the correct output values for each set of input values. **Finiteness:** An algorithm should produce the output after a finite number of steps for any input. **Effectiveness:** It must be possible to perform each step of the algorithm correctly and in a finite amount of time.

**Generality:** The algorithm should work for all problems of the desired forms

#### Example: Linear Search

**Prose:** Locate an item in a list by examining the sequence of list elements one at a time, starting at the beginning.

**More formal prose:** Find item x in the list [a1, a2, . . . , an].

* + - First compare x with a1. If they are equal, return the position 1.
    - If not, try a2. If x = a2, return the position 2.
    - Keep going, and if no match is found when the entire list is scanned, return 0.

#### Pseudocode: Algorithm 1: Linear Search

Input: x : integer, [a1, . . . , an] : list of distinct integers Output: Index i s.t. x = ai or 0 if x is not in the list.

i := 1;

while i ≤ n and x 6= ai do i := i + 1;

if i ≤ n then result := i else result := 0; return result;

#### Write Binary search pseudocode with example Binary Search Prose description:

* Assume the input is a list of items in increasing order, and the target element to be found.
* The algorithm begins by comparing the target with the middle element.
  + If the middle element is strictly lower than the target, then the search proceeds with the upper half of the list.
  + I Otherwise, the search proceeds with the lower half of the list (including the middle).

Repeat this process until we have a list of size 1.

I If target is equal to the single element in the list, then the position is returned. Otherwise, 0 is returned to indicate that the element was not found.

#### Binary Search

**Pseudocode: Algorithm 2: Binary Search**

**Input:** x : integer, [a1, . . . , an] : strictly increasing list of integers

**Output:** Index i s.t. x = ai or 0 if x is not in the list. i := 1; // i is the left endpoint of the interval

j := n; // j is the right endpoint of the interval while i < j do

m := b(i + j)/2;

if x > am then i := m + 1 else j := m; if x = ai then result := i else result := 0; return result;

#### Example: Binary Search

**Find target 19 in the list: 1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22**

The list has 16 elements, so the midpoint is 8. The value in the 8th position is 10. As 19>10, search is restricted to positions 9-16.

2 3 5 6 7 8 10 12 13 15 16 18 19 20 22 2

The midpoint of the list (positions 9 through 16) is now the 12th position with a value of 16. Since 19 > 16, further search is restricted to the 13th position and above.

2 3 5 6 7 8 10 12 13 15 16 18 19 20 22 3

The midpoint of the current list is now the 14th position with a value of 19. Since 19 6> 19, further search is restricted to the portion from the 13th through the 14th positions.

2 3 5 6 7 8 10 12 13 15 16 18 19 20 22 4

The midpoint of the current list is now the 13th position with a value of 18. Since 19 > 18, search is restricted to position 14.

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22 5 Now the list has a single element and the loop ends.

Since 19 = 19, the location 14 is returned

#### Write about Greedy Algorithms

**Optimization problems** minimize or maximize some parameter over all possible inputs.

#### Examples of optimization problems:

* + - Finding a route between two cities with the smallest total mileage.
    - Determining how to encode messages using the fewest possible bits.
    - Finding the fiber links between network nodes using the least amount of fiber. Optimization problems can often be solved using a greedy algorithm, which makes the “best” (by a local criterion) choice at each step.

This does not necessarily produce an optimal solution to the overall problem, but in many instances, it does.

After specifying what the “best choice” at each step is, we try to prove that this approach always produces an optimal solution, or find a counterexample to show that it does not.

#### Write a short notes on the Growth of Functions

Given functions f : N → R or f : R → R.

Analyzing how fast a function grows. Comparing two functions. Comparing the efficiently of different algorithms that solve the same problem.

Applications in number theory

#### Big-O Notation Definition

Let f, g : R → R. We say that f is O(g) if there are constants C and k such that

#### ∀x > k. |f(x)| ≤ C|g(x)|

This is read as **“f is big-O of g” or “g asymptotically dominates f”.**

The constants **C** and **k** are called **witnesses** to the relationship between f and g.

Only one pair of witnesses is needed. (One pair implies many pairs, since one can always make k or C larger.)

#### Illustration of Big-O Notation

f(x) = x 2 + 2x + 1, g(x) = x 2 . f is O(g) witness k = 1 and C = 4. Abusing notation, this is often written as f(x) = x 2 + 2x + 1 is O(x 2 ). **Example**

#### Bounds on functions. Prove that

f(x) = anx n + an−1x n−1 + · · · + a1x + a0 is O(x n ). 1 + 2 + · · · + n is O(n 2 ).

n! = 1 × 2 × · · · × n is O(n n ). log(n!) is O(n log(n)).

#### Big-Omega Notation Definition

Let f, g : R → R. We say that f is Ω(g) if there are constants C and k such that

#### ∀x > k. |f(x)| ≥ C|g(x)|

This is read as **“f is big-Omega of g”.**

The constants **C** and **k** are called witnesses to the relationship between f and g.

Big-O gives an upper bound on the growth of a function, while Big-Omega gives **a lower bound**. Big-Omega tells us that a function grows at least as fast as another.

#### Big-Theta Notation Definition

Let f, g : R → R. We say that f is Θ(g) if f is O(g) and f is Ω(g).

#### We say that “f is big-Theta of g” and also that “f is of order g” and also that “f and g are of the same order”.

f is Θ(g) if and only if there exists constants C1, C2 and k such that

#### C1g(x) < f(x) < C2g(x) if x > k.

**Example**

**Show that the sum 1 + 2 + · · · + n of the first n positive integers is Θ(n 2 ). Solution:** Let f(n) = 1 + 2 + · · · + n.

We have previously shown that f(n) is O(n 2 ).

To show that f(n) is Ω(n 2 ), we need a positive constant C such that f(n) > Cn2 for sufficiently large n.

Summing only the terms greater than n/2 we obtain the inequality 1+ 2 + · · · + n ≥ dn/2e + (dn/2e + 1) + · · · + n

≥ dn/2e + dn/2e + · · · + dn/2e = (n − dn/2e + 1)dn/2e

≥ (n/2)(n/2) = n 2/4

Taking C = 1/4, f(n) > Cn2 for all positive integers n.

Hence, f(n) is Ω(n 2 ), and we can conclude that f(n) is Θ(n 2 ).

#### Write a short notes on Complexity of Algorithms

* + - Given an algorithm, how efficient is this algorithm for solving a problem given input of a particular size?

How much time does this algorithm use to solve a problem?

How much computer memory does this algorithm use to solve a problem?

* + - We measure time complexity in terms of the number of operations an algorithm uses and use big-O and big-Theta notation to estimate the time complexity.
    - Compare the efficiency of different algorithms for the same problem.
    - We focus on the **worst-case time complexity** of an algorithm. Derive an upper bound on the number of operations an algorithm uses to solve a problem with input of a particular size. (As opposed to the average-case complexity.)
    - Here: Ignore implementation details and hardware properties. −→ See courses on algorithms and complexity.

#### Worst-Case Complexity of Linear Search Algorithm 4: Linear Search

**Input**: x : integer, [a1, . . . , an] : list of distinct integers

**Output:** Index i s.t. x = ai or 0 if x is not in the list. i := 1;

while i ≤ n and x 6= ai do i := i + 1;

if i ≤ n then result := i else result := 0; return result;

#### Count the number of comparisons.

* + - At each step two comparisons are made; i ≤ n and x 6= ai .
    - To end the loop, one comparison i ≤ n is made.
    - After the loop, one more i ≤ n comparison is made. If x = ai , 2i + 1 comparisons are used.

If x is not on the list, 2n + 1 comparisons are made and then an additional comparison is used to exit the loop. So, in the worst case 2n + 2 comparisons are made.

Hence, the complexity is Θ(n).

#### Average-Case Complexity of Linear Search

For many problems, determining the average-case complexity is very difficult.

(And often not very useful, since the real distribution of input cases does not match the assumptions.)

However, for linear search the average-case is easy.

Assume the element is in the list and that the possible positions are equally likely.

By the argument on the previous slide, if x = ai , the number of comparisons is 2i + 1. Hence, the average-case complexity of linear search is n

∑ Xn 2i + 1 = n + 2

Which is Θ(n). i=1

#### Write Worst-Case Complexity of Binary Search Algorithm 5: Binary Search

**Input:** x : integer, [a1, . . . , an] : strictly increasing list of integers

**Output:** Index i s.t. x = ai or 0 if x is not in the list. i := 1; // i is the left endpoint of the interval

j := n; // j is the right endpoint of the interval while i < j do

m := b(i + j)/2c;

if x > am then i := m + 1 else j := m;

if x = ai then result := i else result := 0; return result;

Assume (for simplicity) n = 2 k elements.

Note that k = log n. Two comparisons are made at each stage; i < j, and x > am.

At the first iteration the size of the list is 2k and after the first iteration it is 2 k−1 . Then 2k−2 and so on until the size of the list is 21 = 2.

At the last step, a comparison tells us that the size of the list is the size is 2 0 = 1 and the element is compared with the single remaining element.

Hence, at most 2k + 2 = 2 log n + 2 comparisons are made. Θ(log n).

#### Define the Mathematical induction with example

**Definition Mathematical Induction** To show that a propositional function P(n) is true for all positive integers n≥1 follow these steps:

* ***Basis Step***: Verify that P(1) is true.
* ***Inductive Step***: Show that if P(k) is true for some integer k≥1, then P(k+1) is also true.

To prove the implication

P(k)⇒P(k+1)

in the inductive step, we need to carry out two steps: assuming that P(k)P(k) is true, then using it to prove P(k+1)P(k+1) is also true.

So we can refine an induction proof into a 3-step procedure:

* Verify that P(1) is true.
* Assume that P(k) is true for some integer k≥1.
* Show that P(k+1) is also true.

The second step, the assumption that P(k) is true, is sometimes referred to as the

**inductive hypothesis**.

This is how a mathematical induction proof may look:

The idea behind mathematical induction is rather simple. However, it must be delivered with precision.

* Be sure to say “Assume the identity holds for some integer k≥1.” Do not say “Assume it holds

for all integers k≥1.” If we already know the result holds for all k≥1, then there is no need to prove anything at all.

* Be sure to specify the requirement k≥1. This ensures that the chain reaction of the falling dominoes starts with the first one.
* Do not say “let n=k” or “let n=k+1.” The point is, you are not assigning the value of k and

k+1 to n. Rather, you are assuming that the statement is true when nn equals kk, and using it to show that the statement also holds when nn equals k+1.

Example: Use mathematical induction to show that

## 1+2+3+⋯+n=n(n+1)/2

for all integers n≥1n≥1.

**Example 3:** For all n ≥ 1, prove that, 1 + 3 + 5 … 2n – 1 = n2

**Solution:**

*Let the given statement be S(n),*

*and S(n) = 1 + 3 + 5 … 2n – 1 = n2 For n = 1, 2 \* 1 – 1 = 12 Thus S(1) is true .*

*Now, let’s take a positive integer, k, and assume S(k) to be true i.e.,*

*S(k) = 1+ 3 + 5 .. (2k – 1) = k2*

*We shall now prove that S(k + 1) is also true, so now we have,*

*1 + 3 + 5 .. (2(k + 1) – 1) = (k + 1)2*

*L.H.S: 1 + 3 + 5 + …. (2k – 1 ) + 2k + 2 – 1*

*= S(k) + 2k + 1*

*= k2 + 2k + 1*

*= (k + 1)*

*= R.H.S*

*Thus S(k + 1) is true, whenever S(k) is true for all natural numbers. And we initially showed that S(1) is true thus S(n) is true for all natural numbers.*

**Example** Prove an = a1 + (n – 1) d, is the general term of any arithmetic sequence.

### Solution:

*For n = 1, we have an = a1 + (1 – 1) d = a1, so the formula is true for n = 1,*

*Let us assume that the formula ak = a1 + (k – 1) is true for all natural numbers.*

*We shall now prove that the formula is also true for k+1, so now we have,*

*ak + 1 = a1 + [(k + 1) – 1] d = a1 + k · d.*

*We assumed that ak = a1 + (k – 1) d, and by the definition of an arithmetic sequence ak+ 1 – ak = d,*

*then, ak + 1 – ak*

*= (a1 + k · d) – (a1 + (k – 1)d)*

*= a1 – a1 + kd – kd + d*

*= d*

*Thus the formula is true for k + 1, whenever it is true for k. And we initially showed that the formula is true for n = 1. Thus the formula is true for all natural numbers.*

Strong Induction is another form of mathematical induction. Through this induction technique, we can prove that a propositional function, P(n) is true for all positive integers

, n, using the following steps −

* **Step 1(Base step)** − It proves that the initial proposition P(1)P(1) true.
* **Step2(Inductivestep)** –It proves that the

conditional statement [P(1)𝖠P(2)𝖠P(3)𝖠⋯𝖠P(k)]→P(k+1) is true for positive integers k

#### Recursion

Sometimes it is difficult to define an object explicitly. It may be easy to define this object in terms of itself. This process is called recursion.

We can use recursion to define sequences, functions, and sets

Example: an=2n for n = 0,1,2,… 1,2,4,8,16,32,… After giving the first term, each term of the sequence can be defined from the previous term

a1=1 an+1 = 2an

When a sequence is defined recursively, mathematical induction can be used to prove results about the sequence.

Let P(k) be proposition about ak. Basis step: Verify P(1).

Inductive step: Show k1 (P(k) P(k+1)).

#### Recursively defined functions

Assume f is a function with the set of nonnegative integers as its domain We use two steps to define f.

Basis step: Specify the value of f(0).

Recursive step: Give a rule for f(x) using f(y) where 0y Such a definition is called a recursive or inductive definition. **Example**

Suppose f(0) = 3 f(n+1) = 2f(n)+2, n 0. Find f(1), f(2) and f(3).

#### Solution:

f(1) = 2f(0) + 2 = 2(3) + 2 = 8

f(2) = 2f(1) + 2 = 2(8) + 2 = 18

f(3) = 2f(2) + 2 = 2(18) + 2 = 38 6

#### Example Give an inductive definition of the factorial function Solution:

Basis step: (Find F(0).) F(0)=1

Recursive step: (Find a recursive formula for F(n+1).) F(n+1) = (n+1) F(n)

What is the value of F(5)? F(5) = 5F(4)

= 5 . 4F(3)

= 5 . 4 . 3F(2)

= 5 . 4 . 3 . 2F(1)

= 5 . 4 . 3 . 2 . 1F(0)

= 5 . 4 . 3 . 2 . 1 . 1 = 120

#### Recursively defined sets and structures

Assume S is a set. We use two steps to define the elements of S. Basis step: Specify an initial collection of elements.

Recursive step: Give a rule for forming new elements from those already known to be in S.

**Example** Consider S Z defined by Basis step: (Specify initial elements.) 3 S

Recursive step: (Give a rule using existing elements) If x S and y S, then x+y S.

3 S

3 + 3 = 6 S 6 + 3 = 9 S

6 + 6 = 12 S …

#### Example Show that the set S defined in previous , is the set of all positive integers that are multiples of 3.

**Solution:**

Let A be the set of all positive integers divisible by 3.

We want to show that A=S Part 1: (Show AS using mathematical induction.)

Show

x (x A x S).

Define P(n). P(n) is “3n S”.

Basis step: (Show P(1).) P(1) is “3 S”.

By recursive definition of S, 3 S, so P(1) is true.

#### Example Show that the set S defined in previous is the set of all positive integers that are multiples of 3.

**Solution:** Part 1: (Show A

S using mathematical induction.) Inductive step: (Show k1 P(k)P(k+1).)

Define inductive hypothesis: P(k) is “3k S”. Show k1 P(k+1) is true. P(k+1) is “3(k+1) S”. 3(k+1) = 3k + 3

By recursive definition of S, since 3 S and 3k S, (3k+3) S. By mathematical induction, n1 3n S.



UNIT –IV

#### Explain the Generating Functions

In mathematics, a **generating function** is a formal power series in one indeterminate, whose

coefficients encode information about a sequence of numbers *an* 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 *an* is

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

If *an* 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 *am, n* (where *n* and *m* are natural numbers) is

#### Example:



**Exponential generating function**

The *exponential generating function* of a sequence *an* is



#### Example:

**Function of Sequences:**

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

series

1

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

#### (2)

**(3)**

#### (4)

for the partition function P, where is a *q*-Pochhammer symbol,and

#### (5)

**(6)**

#### (7)

for the Fibonacci numbers .

Generating functions are very useful in combinatorial enumeration problems. For example, the subset sum problem, which asks the number of ways to select out of given integers such that their sum equals , can be solved using generatingfunctions.

#### How Calculating Coefficient of generating function:

*ANS.*

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

Polynomial Expansions:

1. 1 *xm* 1

1 *x x*... *x*

1 *x*

1 1 *x x* 2 ...

1. 1 *x*

*n* 2 *r n*

3) (1 *x*) 1 *C*( *n*,1) *x C*( *n*, 2) *x* ... *C*( *n*, *r*) *x* ... *C*( *n*, *n*)*x*

*nm*

*m* 2*m k km*

4) (1 *m n*

*x* ) 1*C*( *n*,1)*x C*(*n*,2) *x* ... ( 1) *C* ( *n*, *k* ) *x* ...(

*n*

1) *C* ( *n*, *n* )*x*

1 2 *r*



5) (1 *x*) *n* 1 *C* (1 *n* 1,1) *x C*(2 *n* 1, 2) *x* ... *C*( *r n* 1, *r*) *x* ...

**6) If h(x)=f(x)g(x), where f(x) 0 1 2** and g(x) 0 1 2 , then

*a a x ax*2 ... *b b x b x*2 ... h(x) *a b* ( *a b a b* ) *x* ( *a b a b a b* ) *x* 2 ... ( *a b a ba b* ... *a b* ) *xr* ...

0 0 1 0 0 1 2 0 1 1 *r* 0 *r* 1 1 *r* 2 2 0 *r*

0 2

























#### Explain Recurrence relations with example

**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

arithmetic

with

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

an = an -1 + d. L kewise if pn denotes the nth term of a geometric progression 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 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.

#### Solve the Recurrence relations by substitution and generating functions

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

1. Substitution (also called iteration),
2. Generatingfunctions,
3. Characteristics roots,

In the substitution method the recurrence relation 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.

#### 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)

#### 5. Solution of Linear Inhomogeneous Recurrence Relations:

The equation + 1 −1+ 2 −2=( ), where 1and 2 are constant, and ( ) 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, occurswhen ( )≡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 is a solution. Now suppose that ( ) is any particular solution of the inhomogeneous equation.

(That is, it solves the equation, but does not necessarily match the initial data.)

Then = +( ) is a solution to the inhomogeneous equation, which you can see simply by substituting  into the equation.

On the other hand, every solution  of the inhomogeneous equation is of the form = +( ) where  is a solution of the homogeneous equation, and ( ) is a particular solution of the inhomogeneous equation.

The proof of this is straightforward. If we have two solutions to the inhomogeneous equation, say  1 and 2, then their difference 1− 2=  is a solution to the homogeneous equation, which you can check by substitution.

But then 1= +  2, and we can set  2=( ), since by assumption, 2 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:

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

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.<

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

above.

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

To solve the equation − 6 −1 + 8 −2 = 3.

Let< s suppose that we are also given the initial data 0 = 3, 1 = 3. The associated homogeneous equation is − 6 −1 + 8 −2 = 0, so the

characteristic equation is 2 − 6 + 8 = 0, which has roots 1 = 2 and 2 = 4.

Thus, the general solution to the associated homogeneous equation is 12 + 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  that solves the homogeneous equation.

If that is so, then = −1 = −2 = , andsubstituting intotheequation gives  − 6  + 8  = 3, and we find that  = 1.

Now, the general solution to the inhomogeneous equations is 12 + 24 + 1. Reassuringly, this is the answer given in the back of the book. Our initial data lead to the equations 1 + 2 + 1 = 3 and 2 1 + 4 2 + 1 = 3,

whose solution is 1 = 3, 2 = −1.

Finally, the solution to the inhomogeneous equation, with the initial condition given, is = 3 · 2 − 4 + 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 − 6 −1 + 5 −2 = 3,

when we guessed that the particular solution was a constant , we< d have arrived at the equation − 6  + 5  = 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 theform

 = . Then wehave −6 −1 +5 −2 =3, which simplifies to 6 −10 =3 sothat

=−34 . Thus,  = −3 4 . This won< t be enough if 1 is a root of multiplicity 2, that is, if

−1 2 is a factor of the characteristic polynomial. Then there is a particular solution of the form  =

2. 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  =2 −1, we would start by guessing a particular solution  = 1 + 2. If it turned out that 1 was a characteristic root, we would amend our guess to  = 1 2+ 2 + 3. If 1 is a double root, this will fail also, but  = 1 3+ 2 2+ 3 + 4 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  = . In that case, we guess that a particular solution is just a constant multiple of , say ( )= . Again, we gave trouble when 1 is a characteristic root. We then guess that  = , which will fail only if 1 is a double root. In that case we must use  = 2 , 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

1. **Write about two algorithms which gives minimum spanning tree**

A. Prim’s algorithm: a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges with minimal weight among all edges incident to a vertex already in

the tree so that no edge produces a simple circuit when it is added Kruskal’s algorithm: a procedure for producing a minimum spanning tree in a weighted graph that successively adds edges of least weight that are not already in the tree such that no edge produces a simple circuit when it is added

#### How to Represent 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 = (aij) of the graph G is the m x m matrix defined as follows:

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

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

#### Drawbacks

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

If the number of edges is 0(m) or 0(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 = (pij) 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:

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 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

# Simple Path

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

1. **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**

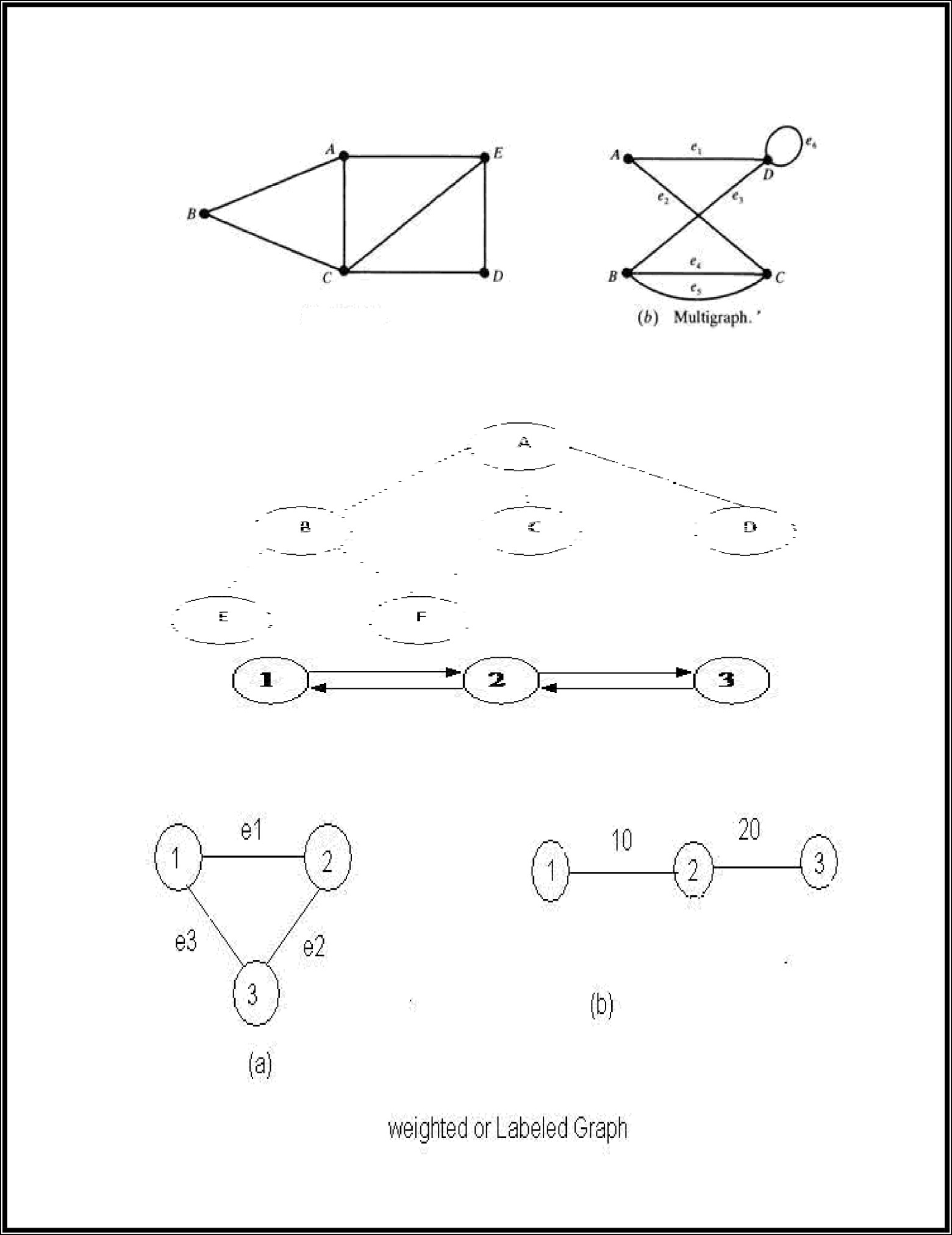
#### If the weight is assigned to each edge of the graph then it is called as Weighted or Labeled graph.

**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.



* 1. Graph.

# What is 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*



Simple Directed Graph

Outdegree of 1 =1

Outdegree of 2 =2

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.

* 1. Explain graph traversal methods 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:

* 1. Write DFS Algorithm with program

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

*Algorithmic Steps*

1. **Step 1**: Push the root node in the Stack.
2. **Step 2**: Loop until stack is empty.
3. **Step 3**: Peek the node of the stack.
4. **Step 4**: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
5. **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:

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)

{

}

else

{

child.visited=true; printNode(child); s.push(child);

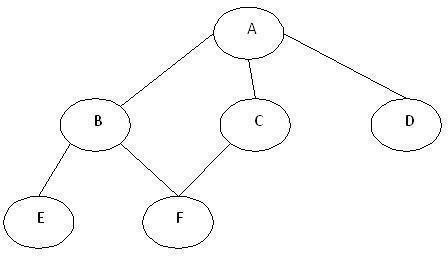
s.pop();

}

}

*//Clear visited property of nodes* clearNodes();

}



5. Write BFS Algorithm with program

# 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:

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)

{

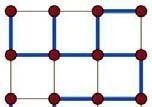
child.visited=true; printNode(child); q.add(child);

}

}

*//Clear visited property of nodes* clearNodes();

}



1. Explain Spanning trees

#### 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:

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).



#### How to count spanning trees

**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 *Cn* 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*

w*n*ith *n*

*n* − 2 vertices. The formula states that *t*(*Kn*) = *n n* − 2 Another way of stating Cayley's formula is that

Cayley's formula is a formula for the number of spanning trees in the complete graph *K* w*n*ith *n*

*n* − 2

vertices. The formula states that *t*(*Kn*) = *n n* − 2

Another way of stating Cayley's formula is that there are exactly *n* labelled trees with *n* vertices.

Cayley's formula can be proved using

Kirchhoff's matrix-tree theorem or via the Prüfer code.

*q* − 1 *p* − 1

If *G* is the complete bipartite graph *Kp*,*q*, then *t*(*G*) = *p q* , while if *G* is the *n*-dimensional

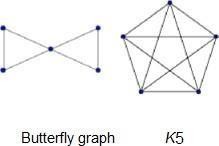
hypercube graph *Qn*, . These formulae are also consequences of the matrix-tree theorem.

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).

#### What is Planar Graph and applications with example

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.

# Example graphs

**planar non planar**

# 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

#### 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).

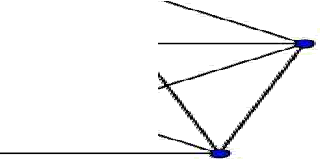
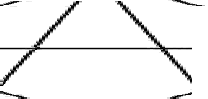
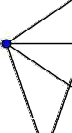
# Example:

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

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

1. **Explain Subgraphs, Multygraphs and Euler circuits 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.

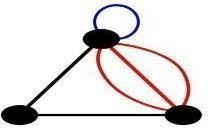
# Multi graphs:

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.

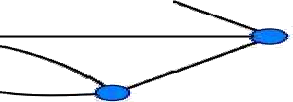
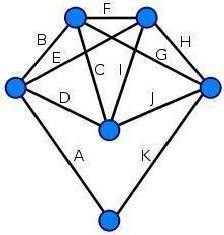
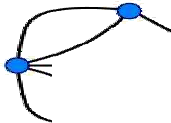
For the existence of Eulerian trails it is necessary that no more than two vertices have an odd degree; this means the Königsberg graph is *not* Eulerian. If there are no vertices of odd degree, all Eulerian trails are circuits. If there are exactly two vertices of odd degree, all Eulerian trails start 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.

# Explain Hamilton graphs 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 eve y trournament has an odd number of Hamiltonian paths every platonic solid,

considered as a graph, is Hamiltonian

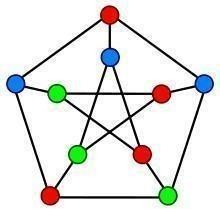
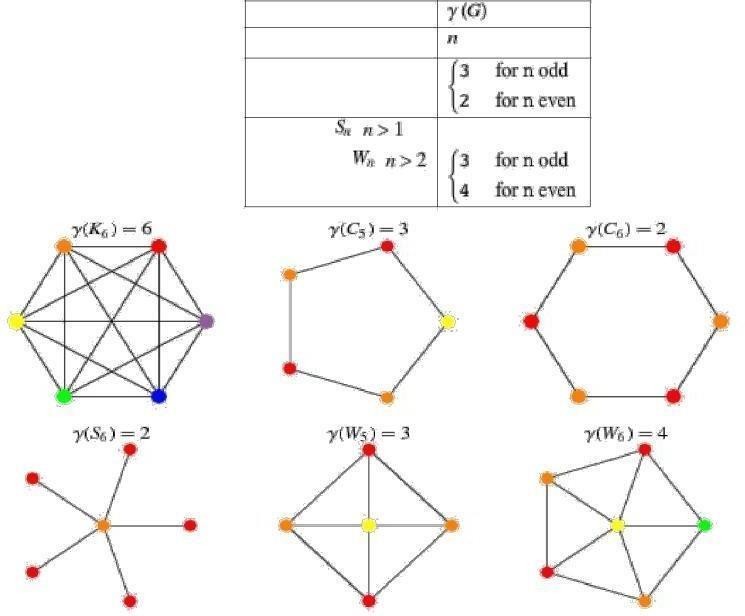
1. **Explain Chromatic Numbers**

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**

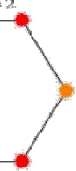
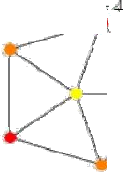
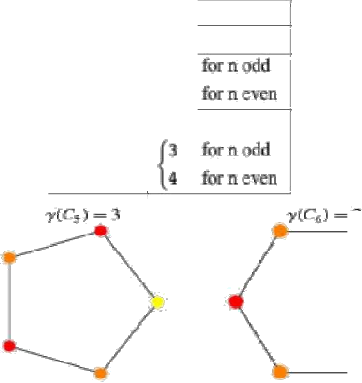
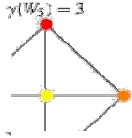
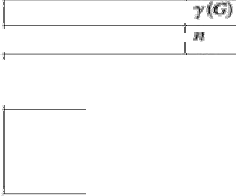
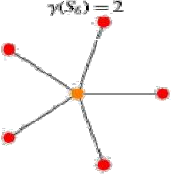
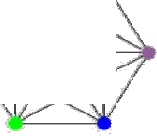
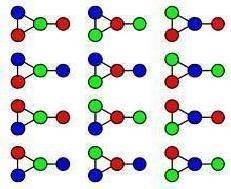
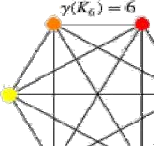
When used without any qualification, a coloring of a graph is almost always a *proper vertex coloring*, namely a labelling of the graph’s vertices with colors such that no two vertices sharing the same edge have the same color. Since a vertex with a loop could never be properly colored, it is understood 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, χ(*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 complete graph cycle graph ,

star graph , 2 wheel ,

graph

, 2

wheel , graph

#### PART- C

**UNIT WISE UNIVERSITY PREVIOUS QUESTION PAPER QUESTIONS**

#### UNIT –I PREVIOUS ASKED ESSAY QUESTIONS

* 1. a ) Obtain principal conjunctive normal form (PCNF) for the formula (~p → r)  (q ↔ p).

b) Show that the following is inconsistent P→Q, R→S, P  R, ~ (Q  S) [5+5]

* 1. a) Using indirect proof , derive P → ~ S from P → Q  R , Q → ~ P, S → ~ R , P.

b) Show that R → (S → Q), ~ P  R and S  P → Q [5+5]

* 1. a) Verify the validity of the following arguments. “Every living thing is a plant or an animal. Logu‟s dog is alive and it is not a plant.

All animals have heart. Therefore Logu‟s dog has a heart.”

b) Find the formulas in Disjunctive Normal Form equivalent to the following well formed formulas

(¬R) → (((P ˅ Q) → R) → ¬ Q [5+5]

4 Without using truth tables prove that

((P ∨ Q) ¬ (¬ P (¬ Q ∨ ¬ R ))) ∨ ( ¬ P¬ Q ) ∨ (¬ Q ¬ R) is a tautology. [10]

5 .a) 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.

b) Show that R → S can be drawn from the premises

P → (Q → S), ¬ R ˅ P and Q. [5+5]

6.a) Show that *x* *P*  *x*  *Q* *x*  *xP* *x* *xQ* *x* .

b) Obtain principle disjunctive normal form of the following: [5+5]

*p*  *p*  *q*  *q*  *q*

7.a) Assume x is a particular real number. Determine whether the following two statements are logically equivalent.

1. *x*  2 *or it is not the case that* 1 *x*  3
2. *x*  1 *or either x*  2 *or x*  3.

b). Translate the given statements into propositional logic using the propositions provided:

P: “The message is scanned for viruses”

Q:”The message was sent from an unknown system”

1. “The message is scanned for viruses whenever the message was sent from an unknown system.”
2. “It is necessary to scan the message for viruses whenever it was sent from an unknown

system.” [5+5]

8.a). Using automatic theorem proving show that (*Q*  (*P*  *Q*))  *P* . b). Explain the following:

**i)** Normal Forms ii) Free and bound variable

iii) Logical Equivalence iv) Resolution. [5+5]

9 . Obtain the principal disjunctive normal form of the following formula

*P*  (  *P* (*Q*  (  *Q* *R* )))

1. Verify whether the proposition

(( *P*   *q* ) *r* ) *s*   ((( *P*   *q* ) *r* ) *s* ) .[5+5]

1. a) Show that (  *x* )( *p* ( *x* )  *Q* ( *x* )) ((  *x* ) ( *p* ( *x* )  (  *x* ) ( *Q* ( *x* )) is a logically valid statement.

b) Show the following using the automatic theorem.

*i) P* (  *P*  *Q* )

*ii) P*   *P*  *Q*  *R* [5+5]

1. a) Show that ( ~ 𝑝 ∨ (∼ 𝑝 𝖠 𝑞 and( ∼ 𝑝 𝖠∼ 𝑞) are logically equivalent

b) Show that ∼ 𝑝 𝖠 , ∼ 𝑞 ∨ 𝑟 ∼ 𝑟 ⟹ ∼ 𝑝 .

13. Prove that ∀𝑥 𝑃 𝑥 ∨ 𝑄 𝑥 ⟹ 𝑥 𝑃 𝑥 ∨ ∃𝑥 (𝑥).

**UNIT –I PREVIOUS ASKED SHORT QUESTIONS**

* 1. Write converse and inverse for the statement “If Sun rises in the east then 3\*7=98”. [2]
  2. Express (PQ)  (P↔R) in terms of  ,  , ~ only. [3]
  3. Find the negations of the following quantified statements:

*x*, *y*,  *p*  *x*, *y*  *q*  *x*, *y**r*  *x*, *y* [2]

* 1. Construct a truth table to show that (p  q) → p is a tautology. [3]
  2. Define well-formed formulae and clause form. [2]
  3. Write the statement in symbolic form then negate statements:
     1. Some Drivers do not obey the speed limit.
     2. All dogs have fleas. [3]
  4. Write the converse and contrapositive of the statement:

“If *P* is a square, then *P* is a rectangle”. [2] 8 . Rewrite the following statement informally, without quantifiers or variables.

*x*  *R*, *if x*  2 *then x*2  4 [3]

1. Give the truth table for the propositional formula

( *P*  *Q* ) ( *P*  *Q* ) [2]

1. Write the sentence “It is not true that all roads lead to Rome” in the symbolic form. [3]

**UNIT –II PREVIOUS ASKED ESSAY QUESTIONS**

**1.**a) Explain properties of binary relations with examples.

b) Draw the Hasse diagram for the partial ordering {(A, B): A ≤ B} on the power set e(S)

where S={a, b, c} and ≤ is subset relation. [5+5]

2.a) Draw the Hasse diagram for the divisibility on the set {1,2,3,6,12,24,36,48,96}.

b) Define equivalence relation. Show that the relation equal on set of integers is equivalence relation [5+5]

1. a) Let *A*  *a*,*b*, *c*be a set and relation R on A is as  *a*, *a*,*a*,*b*,*b*, *c*, *c*, *c*. Is R.
   1. Reflexive ii) Symmetric iii)Transitive

b). Prove that *f* 1 *g*1  *gf* 1 , where *f* : *Q*  *Q* such that *f*  *x*  2*x* and *g* : *Q*  *Q*

Such that *g*  *x*  *x*  2 are two functions. [5+5]

1. a)Prove that the intersection of any two subgroups of a group G is again subgroup of G.

b) In a lattice (L, ≤ , 𝖠, ˅) state and prove the laws indempotent, commutative, association and absorption. [5+5]

1. Prove that the transitive closure R  of a relation R on a set A is the smallest transitive relation on A containing R.

6.a) If R and S are equivalence relations on a set A. Prove that *R* is an equivalence Relation.



*S*

b) Let *B*  *a*,*b*, *c*and *A*  *P* *B* be the power set of B. Draw the Hasse diagram for

 and poset A. [5+5]

* 1. Determine whether the following functions are injective, surjective or bijective. Also describe the inverses of the bijective functions.

3 *f* (*n*)  1 *if f* (*n*) *is odd*

* + 1. The function *f*:*N**N* with *f*(*a*)=9 and *f* (*n*)  



1 *f* (*n*) *if*

2

*f* (*n*) *is even*

* + 1. A function f:AA satisfying *f*(*f*(*x*))=*f*(*x*) for all *x*ϵ*A*
  1. Let *A*, *B* be finite sets with | *A* | *m* and | *B* | *n* . Determine the numbers of:

1. Functions *A**B*
2. Injective functions *A**B* (provided that *m*  *n*)
3. Surjective functions *A**B* (provided that *m*  *n*)
4. Bijective functions *A**B* (provided *m* = *n*)

Symmetric relations on *A* [5+5]

8 a). Show that the functions *f* : *R*  (1,  ) *and g* : (1,  )  *R* defined by *f* ( *x* )  3 2 *x*  1,

*g* ( *x* )= 1/2 lo g ( *x*  1) are inverses.

*b).* Prove that the transitive closure *R*  of a relation R on a set A is the smallest transitive relation on A containing R. [5+5]

Show that congruence modulo m is an equivalence relation on integers. A relation 𝑅 on 𝐴 is symmetric if and only if 𝑅 = 𝑅 −1 .

b) A relation 𝑅 on 𝐴 is reflexive if and only if 𝑅 −1 is reflexive.

**UNIT –II PREVIOUS ASKED SHORT QUESTIONS**

1. Define LUB and GLB of a lattice and give examples for each. [2]
2. Explain equivalence relation. Give suitable examples for a relation which is not equivalence relation. [3]
3. Let *X*  1, 2,3, 4,5,6 and R be a relation defined as (*x*, *y*)  *R* if and only if *x*  *y* is divisible by 3. Find the elements of relation of R. [2]
4. If R is set of real numbers, then show that the function: *F* : *R*  *R* defined by

*f*  *x*  5*x*3 1is one-one function. [2]

1. Define well-formed formulae and clause form. [2]
2. Write the statement in symbolic form then negate statements:
   1. Some Drivers do not obey the speed limit.
   2. All dogs have fleas. [3]
3. Let *I*  {0,1 ,2}and define functions *f* and *g* from *I* to *I* as follows:

For all *x* in *I*, *f* (*x*)  (*x*2  *x* 1) mod3 *and g*(*x*)  (*x*  2)2 mod 3

State whether *f*=*g*. [2]

8 Compute the transitive closure of the relation R={(1,1),(1,2),(1,3),(2,3),(3,1)} defined over a set S={1,2,3} [3]

1. State principle of inclusion. [2]
2. Define lattice.

**UNIT –III**

**PREVIOUS ASKED ESSAY QUESTIONS**

* 1. Prove by Mathematical induction that 6 𝑛+2 + 7 2𝑛+1 is divisible by43 for each positive integer n.
  2. Find the number of non negative integral solution for the equation X1+X2+X3+X4=50, where X1 >= 2, X2 >= 4, X3 >= -3, X4 >= 7
  3. a) Find the number of integers between 1 and 250 that are divisible by any of the integers 2, 3 and 6.

b) Find the coefficient of x9 y3 in the expansion of (2x - 3y)12. [5+5]

* 1. a). How many bit strings of length 10 contain:
     1. At most four 1‟s ii) At least four 1‟s iii) Exactly four 1‟s.

b) There are 40 computer programmers for a job. 25 know Java, 28 know Oracle and 7 know neither language. Using principle of inclusion exclusion find how many know

both languages. [5+5]

* 1. a) Find the solution for the Fibonacci series an=an-1+an-2, n>2 and a0=1, a1=1.

b) Using substitution method, find the solution for an=an-1+1/n(n-1) where a0=2 [5+5]

6.

Show that *n*

1

*iC* *n*,*i*  *n*2*n*1 .

6.b) In how many ways can 4 mathematics books, 3 history books, 3 chemistry books and 2 sociology books be arranged on the shelf so that all books of the same subject are together. [5+5]

**UNIT IV**

**PREVIOUS ASKED ESSAY QUESTIONS**

1. a). Solve the recurrence relation an - 7an-1 + 16an-2 - 12an-3 = 0 for n  3 with the initial conditions a0=1, a1=4, and a2=8.

b) Find the solution for an - 3an-1 - 4an-2 = 0 for n  2 and, a0 = a1 = 1. [5+5]

2.a) Find a generating function for the recurrence relation *an*  *an*1  6*an*2  0 for *n*  2

b) Express Fibonacci sequence of numbers 1, 1, 2, 3, 5, 8, 13, 21, 34,... in terms of general expression for the rth number ar and generating function. [5+5]

3.a) Find the number of integer solutions of the equation x1 + x2 + x3 + x4 + x5 = 30

Under the constraints xi ≥ 0 for all i = 1, 2, 3, 4, 5 and further x2 is even and x3 is odd.

b) Solve the recurrence relation *an*  6*an*1  9*an*2  0 for n  2 . [5+5]

4. a) Solve recurrence relation an=3an-1-2an-2 for *n*  2 .

b) Find the recurrence relation and initial condition for the following sequence: 6, -18, 54, -162 … [5+5]

5.a) Solve the following recurrence relation using substitution method:

*a*0  2, *a*1 3,*a*  *an*2  2*n*  *n*3*n*  *n*2 4*n*

*n*

1. Find a recursive relation for the following:
   1. The number of strings of length n over the lower-case Roman alphaphet

{a,b,c,…,z} containing two consecutive vowels.

* 1. The number of strings of length n over the lower-case Roman alphabet

{a,b,c,…,z} not containing two consecutive consonants. [5+5]

6.a) Solve the following recurrence relation by substitution

*a*  *a*  3 *n* 2  3 *n*  1 *W h e re a*  1 .

*n n*  1 0

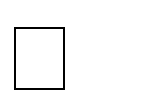
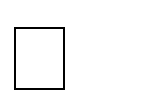
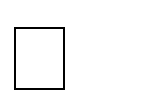
b). Find a general expression for an using generating function

[5+5]

*a*  7 *a*  1 6 *a*  1 2 *a*  0 , *n*  3 .

*n n*  1 *n*  2 *n*  3

7. a). Find the generating function for the following sequence: 1 , 1 3 , 1 3 5 , 1 3 5   (2*n*  1) ,



2 2  4 2  4  6 2  4  6   (2*n*  2)

1. Use generating function to solve the following recurrence relation:

*a*0  2, *a*1 3,*an* 5*an*1 6*an* 2 7*n for n*  2. [5+5]

7 a). If the person invests Rs.10, 000 at 10% annual interest compounded quarterly, in how Many months the money will become 15000.

* 1. Solve the following recurrence relation *an*1  2*an*  2*n*, *n*  0,*a*  1. [5+5]

**UNIT –IV PREVIOUS ASKED SHORT QUESTIONS**

* + 1. Write the characteristic equation for the following recurrence relation

an – 4an-4 = 0, n > 4 and solve it. [2]

* + 1. Find the generating function for the sequence A =

{ar} [3]

2, if 0 < r < 3

ar = 4, if 4 < r < 5

0, if r > 6

* + 1. What is homogeneous recurrence relation? [2]
    2. Find the number of non-negative integer solutions of the equation *x1* + *x2* + *x3* = 11.[3]
    3. From 6 boys and 4 girls, 5 are to be selected for admission for a particular course.

In How many ways can this be done if there must be exactly 2 girls? [2]

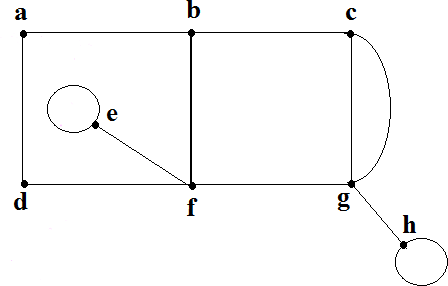
* + 1. Determine the co-efficient of x12 inx3 (1-2x)10. [3]
    2. Find the generating function for the following sequence 0, 1, -2, 3, -4…. [2]
    3. Define first order and second order recurrence relations. [3]
    4. What is a non-homogeneous recurrence relation? Give an example. [2]
    5. Give any three applications of generating fnctions.
    6. Give the disjunctive rule for counting problem.
    7. What is the closed form expression of the sequence *a*  9 .5 *n* , *n*  0 ? [2]

*n*

* + 1. Find the coefficient of *x* 9 *in* (1  *x* 3  *x* 8 )1 0 [3]

**UNIT-V JNTUH-PREVIOUSLY ASKED ESSAY QUESTIONS**

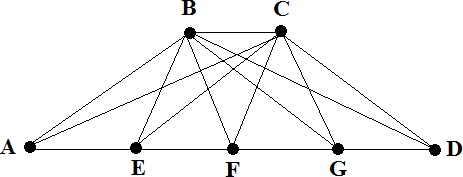
1. Give an example graph which is Hamiltonian but not Eulerian.
2. What is a Hamiltonian Cycle? Draw bipartite graph K3,4 and prove that this graph does not have a Hamiltonian cycle.
3. Define spanning tree. Apply Prim’s algorithm to find minimum spanning tree on the following weighted graph
4. a) Find the degree of each region in the following planar graph 2.



# Graph: 2

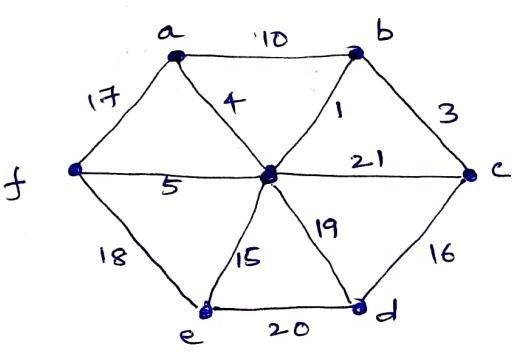
b) Show that the complete bi-partite graph **K3, 3** is notplanar graph. [5+5]

5 .a) Find the dual of the following graph 3.



# Graph: 3

b) Define spanning tree. Apply Prim’s algorithm to find minimum spanning tree on the following weighted graph 4. [5+5]



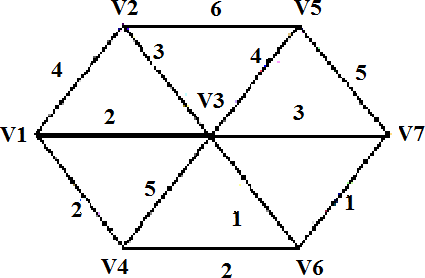
[

**Gr**s**ap**u**h:** l**4**

6.a) Show that a simple complete digraph with n nodes has the maximum number of edges *n* *n* 1 . Assuming that there are no loops.

b) State and explain graph coloring problem. Give its applications. [5+5]

7.a) Find the minimum spanning tree by using kruskal‟s algorithm.



b) Write short notes on DFS and BFS. [5+5]

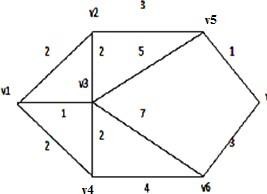
8 .a) Prove that a simple graph with n vertices and k components can have at most

*n*  *k* *n*  *k* 1 edges.

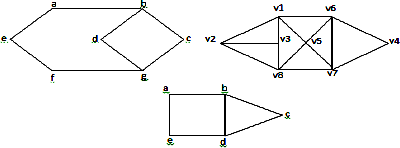
1 and 7 in the following weighted figure 1?

[5+5]

b) What is the shortest path between

Figure: 1

1. Determine whether the given graphs have Hamilton circuits. If it has for such circuits shown in figure 2. [10]



# Figure: 2

1. Use the algorithm BFS to find out whether the following graphs, given by their adjacency lists are connected, and otherwise determine their connected components. Consider that

the set of vertices is alphabetically ordered. [5+5]

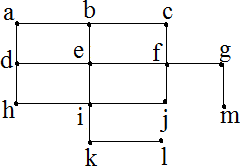
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a b c d e f g h i j | | | | | | | | | | | | | | |
| d | d | | h | | a | a | | a | | b | | c | b | b |
| e | g | |  | | b | d | | d | | i | |  | g | g |
| F | i | |  | | e |  | |  | | j | |  |  |  |
| J | f | |  | |  |  | |  | |  | |  |  |  |
| b) |  | |  | |  |  | |  | |  | |  |  |  |
| a b c d e f g h i j k l m | | | | | | | | | | | | | | |
| B | a | f | b | b | c | b | b | c | a | | c g | | | |
| J | d | i | h | g |  | e | d | k | b | | i | | | |
| E | k |  |  |  | m |  |  |  |  | |  | | | |
| G |  |  |  |  |  |  |  |  |  | |  | | | |
| H |  |  |  |  |  |  |  |  |  | |  | | | |
| J |  |  |  |  |  |  |  |  |  | |  | | | |

11.a) Let G be the non directed graph of order 9 such that each vertex has degree 5 or 6. Prove that atleast 5 vertices have degree 6 or atleast 6 vertices have degree 5.

b). Determine the number of edges in:

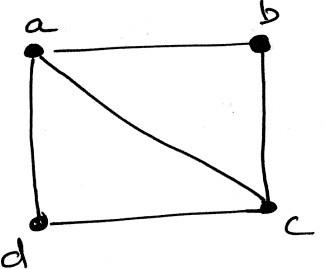
i) Kn ii) Km,n iii) Pn. [5+5]

12.a) Using depth first search method, determine the spanning tree T for the following graph with e as the root of T.



b). Give an example graph which is Hamiltonian but not Eulerian. [5+5]

**UNIT –V PREVIOUS ASKED SHORT QUESTIONS**

1. Give a general formula for Chromatic number of Cycle graph **Cn**. [2]
2. Find the Euler Path in the following graph 1. [3]

# Graph: 1

1. What is chromatic numbers? [2]
2. Define Euler‟s circuit and Give an example. [3]
3. What is minimum spanning tree? Give an example. [2]
4. Define Bipartite graph and Isomorphic graphs. [3]
5. What do you mean by isomorphism? Give examples of isomorphic graphs. [2] 8.What is a planar graph? Give examples of planar and non-planar graphs. [3]

9. What are the advantages of adjacency matrix representation? [2] 10.Define a spanning tree. [3]