SUDDHANANDA SCHOOL OF MANAGEMENT & COMPUTER SCIENCE

LECTURE NOTES ON

Discrete Mathematics (MCBS1001)

Unit – I

SET THEORY, RELATIONS AND FUNCTIONS:

INTRODUCTION TO DISCRETE MATHEMATICS

Discrete Mathematics is the branch of mathematics that studies discrete or distinct objects. It forms the mathematical foundation of computer science and information technology.

The word "discrete" means separate or distinct. Therefore, discrete mathematics deals with countable values instead of continuous values.

Examples of discrete objects:

- Integers
- Graphs
- Statements
- Logical values
- Finite sets

Continuous mathematics deals with:

- Distance
- Temperature
- Speed
- Real numbers

while discrete mathematics deals with:

- Binary numbers
- Algorithms
- Networks
- Data structures

Importance of Discrete Mathematics in Computer Science

Discrete mathematics is very important because almost every area of computer science uses it.

Applications

1. Programming Languages
2. Data Structures
3. Artificial Intelligence
4. Cryptography

5. Networking
6. Compiler Design
7. Database Systems
8. Operating Systems
9. Software Engineering
10. Digital Electronics

SET THEORY

Set theory is one of the fundamental concepts of discrete mathematics.

A set is a collection of distinct and well-defined objects.

The objects in a set are called elements or members.

Examples of Sets

- A = {1,2,3,4}
- B = {a,e,i,o,u}
- C = {red, blue, green}

Characteristics of a Set

1. Elements must be distinct.
2. Elements must be well-defined.
3. Order of elements does not matter.

Example:

- {1,2,3} = {3,2,1}

Representation of Sets

1. Roster Method

Elements are listed within braces.

Example:

- A = {2,4,6,8}

2. Set Builder Method

A rule or property is used.

Example:

- A = {x | x is an even number less than 10}

Meaning:

- x such that x is even and less than 10.

TYPES OF SETS

1. Empty Set

A set with no elements.

Symbol:

- Φ or { }

Example:

- A = {x | x is a natural number less than 0}

Since no natural number is less than 0:

- A = Φ

2. Singleton Set

A set containing only one element.

Example:

- A = {5}

3. Finite Set

A set with limited number of elements.

Example:

- A = {1,2,3,4,5}

4. Infinite Set

A set with unlimited elements.

Example:

- N = {1,2,3,…}

5. Equal Sets

Two sets are equal if they contain exactly the same elements.

Example:

- A = {1,2,3}
- B = {3,2,1}

Therefore:

- A = B

6. Equivalent Sets

Two sets having same number of elements.

Example:

- A = {1,2,3}
- B = {a,b,c}

Both contain 3 elements.

7. Universal Set

The set containing all possible elements under discussion.

Symbol:

- U

Example:

- U = {1,2,3,4,5,6,7,8,9}

8. Subset

If every element of set A belongs to set B, then A is subset of B.

Symbol:

- $A \subseteq B$

Example:

- A = {1,2}
- B = {1,2,3,4}

Then:

- $A \subseteq B$

Proper Subset

If:

- $A \subseteq B$
  and
- $A \neq B$

then A is proper subset.

Cardinality of Set

The number of elements in a set is called cardinality.

Example:

- A = {1,2,3,4}

Then:

$$n(A) = 4$$

Power Set

The set of all subsets of a set is called power set.

Example:
If:

- A = {1,2}

Subsets are:

- Φ
- {1}
- {2}
- {1,2}

Therefore:

- P(A) = {Φ,{1},{2},{1,2}}

Number of subsets:

$$| P(A) | = 2^n$$

Where:

- n = number of elements.

OPERATIONS ON SETS

1. Union of Sets

The union contains all elements belonging to A or B or both.

Symbol:

- ∪

Example:

- A = {1,2,3}
- B = {3,4,5}

Then:

- A ∪ B = {1,2,3,4,5}

2. Intersection of Sets

Contains common elements.

Symbol:

- ∩

Example:

- A ∩ B = {3}

3. Difference of Sets

Contains elements present in first set but absent in second.

Example:

- A − B = {1,2}

4. Complement of Set

Contains elements not present in the set.

Example:
If:

- U = {1,2,3,4,5}
- A = {1,2}

Then:

- A' = {3,4,5}

5. Symmetric Difference

Contains elements belonging to either A or B but not both.

Example:

- A = {1,2,3}
- B = {3,4,5}

Then:

- A Δ B = {1,2,4,5}

Laws of Set Theory

Commutative Laws

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

Associative Laws

$$(A \cup B) \cup C = A \cup (B \cup C)$$

Distributive Laws

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

De Morgan's Laws for Sets

$$(A \cup B)' = A' \cap B'$$

$$(A \cap B)' = A' \cup B'$$

VENN DI.AGRAM

A Venn diagram is the pictorial representation of sets using circles.

It helps in:

- Understanding set operations
- Solving problems visually
- Comparing relationships between sets

CARTESIAN PRODUCT

The Cartesian product of two sets A and B is the set of ordered pairs.

Definition:

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

Example

If:

- A = {1,2}
- B = {a,b}

Then:

- A × B = {(1,a),(1,b),(2,a),(2,b)}

Properties of Cartesian Product

1. Order matters.
2. Number of ordered pairs:

$$n(A \times B) = n(A) \times n(B)$$

RELATIONS

A relation is a subset of Cartesian product.

If:

- A = {1,2}
- B = {3,4}

Then:

- $A \times B = \{(1,3),(1,4),(2,3),(2,4)\}$

Possible relation:

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

Domain and Range of Relation

Domain

Set of first elements.

Example:

- $R = \{(1,2),(2,3)\}$

Domain:

- $\{1,2\}$

Range

Set of second elements.

Range:

- $\{2,3\}$

Types of Relations

1. Reflexive Relation

A relation R on set A is reflexive if:

- $(a,a) \in R$ for every $a \in A$

Example:

- $\{(1,1),(2,2)\}$

2. Symmetric Relation

If:

- $(a,b) \in R$

then:

- $(b,a) \in R$

Example:

- $\{(1,2),(2,1)\}$

3. Transitive Relation

If:

- (a,b) and (b,c) belong to R

then:

- (a,c) must belong to R.

4. Equivalence Relation

A relation that is:

1. Reflexive

2. Symmetric
3. Transitive

is called equivalence relation.

FUNCTIONS

A function is a special relation where every input has exactly one output.

Definition:

$$f: A \to B$$

Where:

- A = domain
- B = codomain

Example of Function

$$f(x) = x + 2$$

If:

- x = 1 → 3
- x = 2 → 4

Domain, Codomain and Range

Domain

Input values.

Codomain

Possible output values.

Range

Actual output values.

Types of Functions

1. One-One Function

Different inputs produce different outputs.

Example:

$$f(x) = 2x$$

2. Many-One Function

Different inputs may give same output.

Example:

- $f(x)=x^2$

Since:

- f(2)=4
- f(-2)=4

3. Onto Function

Every element in codomain has at least one pre-image.

4. Into Function

Some codomain elements are not mapped.

5. Bijective Function

A function that is both:

- One-One
- Onto

Composition of Functions

If:

- f:A→B
- g:B→C

then:

- gof:A→C

Defined as:

$$(g \circ f)(x) = g(f(x))$$

Inverse Function

If:

- f(a)=b

then inverse function:

- $f^{-1}(b)=a$

Exists only for bijective functions.

Unit – II

MATHEMATICAL LOGIC AND BOOLEAN ALGEBRA:

INTRODUCTION TO MATHEMATICAL LOGIC

Mathematical Logic is a branch of discrete mathematics that deals with logical reasoning and truth values. It is widely used in:

- Computer Programming

- Digital Electronics
- Artificial Intelligence
- Database Systems
- Circuit Design

Logic helps a computer make decisions based on conditions.

Example:

- If password is correct → Login successful
- If marks > 40 → Pass

These conditions are based on logical statements.

PROPOSITIONS

A proposition is a declarative statement that is either true or false, but not both.

Examples of Propositions

1. "5 is greater than 3" → True
2. "7 is an even number" → False
3. "India is in Asia" → True

Statements Which Are NOT Propositions

1. Open the door.
2. What is your name?
3. x + 5 = 10

These are not propositions because truth value cannot be determined directly.

TYPES OF PROPOSITIONS

1. Simple Proposition

A proposition containing only one statement.

Example:

- "10 is greater than 5"

2. Compound Proposition

A proposition formed by combining two or more propositions using logical operators.

Example:

- "5 > 3 and 8 > 4"

LOGICAL CONNECTIVES

Logical connectives are symbols used to combine propositions.

Connective	Symbol	Meaning
AND	∧	Both statements true
OR	∨	At least one true
NOT	¬	Opposite truth value

Connective	Symbol	Meaning
Implication	→	If…then
Biconditional	↔	Both statements same

1. Conjunction (AND)

The conjunction of two propositions is true only when both propositions are true.

Symbol:

- ∧

Example:

- P = "5 > 3"
- Q = "8 > 2"

Then:

- P ∧ Q = True

Truth Table of AND

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

2. Disjunction (OR)

The OR operation is true if at least one proposition is true.

Symbol:

- ∨

Truth Table of OR

P	Q	P ∨ Q
T	T	T
T	F	T
F	T	T
F	F	F

3. Negation (NOT)

Negation changes truth value.

Symbol:

- ¬

Example:
If:

- P = True

Then:

- ¬P = False

Truth Table of NOT

P	**¬P**
T	F
F	T

4. Implication

"If P then Q"

Symbol:

- →

Example:

- If it rains, then roads become wet.

Truth Table of Implication

P	**Q**	**P → Q**
T	T	T
T	F	F
F	T	T
F	F	T

5. Biconditional

"P if and only if Q"

Symbol:

- ↔

It is true when both statements have same truth value.

Truth Table of Biconditional

P	**Q**	**P ↔ Q**
T	T	T

P	Q	P ↔ Q
T	F	F
F	T	F
F	F	T

TAUTOLOGY

A tautology is a proposition that is always true for every possible truth value.

Example:

P \vee \neg P

This statement is always true.

Truth Table

P	¬P	P V ¬P
T	F	T
F	T	T

CONTRADICTION

A contradiction is a proposition that is always false.

Example:

P \wedge \neg P

Truth Table

P	¬P	P Λ ¬P
T	F	F
F	T	F

CONTINGENCY

A proposition which is neither always true nor always false.

Example:

- P V Q

LOGICAL EQUIVALENCE

Two propositions are logically equivalent if they have identical truth tables.

Symbol:

- ≡

Example:

P \rightarrow Q \equiv \neg P \vee Q

LAWS OF LOGIC

1. Identity Laws

$P \vee F = P$

$P \wedge T = P$

2. Domination Laws

$P \vee T = T$

$P \wedge F = F$

3. Idempotent Laws

$P \vee P = P$

$P \wedge P = P$

4. Double Negation Law

$\neg(\neg P)=P$

5. Commutative Laws

$P \vee Q = Q \vee P$

$P \wedge Q = Q \wedge P$

6. Associative Laws

$(P \vee Q) \vee R = P \vee (Q \vee R)$

7. Distributive Laws

$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$

DE MORGAN'S LAWS

Very important laws in logic and digital electronics.

First Law

$\neg(P \vee Q)=\neg P \wedge \neg Q$

Meaning:
Negation of OR becomes AND of negations.

Second Law

$\neg(P \wedge Q)=\neg P \vee \neg Q$

Meaning:
Negation of AND becomes OR of negations.

Example Using De Morgan's Law

Simplify:

$\neg(A+B)$

Solution:

$A'B'$

BOOLEAN ALGEBRA

Boolean Algebra is the algebra of logic developed by George Boole.

It deals with only two values:

- 0 → False
- 1 → True

Boolean algebra is widely used in:

- Computer circuits
- Logic gates
- Digital systems
- Microprocessors

Boolean Variables

Variables that can take only:

- 0
  or
- 1

Example:

- A = 0
- B = 1

Basic Boolean Operations

Operation	Symbol	Meaning
OR	+	Addition
AND	.	Multiplication
NOT	'	Complement

Boolean OR Operation

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

Boolean AND Operation

A	B	A.B
0	0	0
0	1	0

A	B	A.B
1	0	0
1	1	1

Boolean NOT Operation

A	A'
0	1
1	0

Laws of Boolean Algebra

1. Identity Law

A+0=A

A.1=A

2. Null Law

A+1=1

A.0=0

3. Idempotent Law

A+A=A

A.A=A

4. Complement Law

A+A'=1

A.A'=0

5. Commutative Law

A+B=B+A

AB=BA

6. Associative Law

(A+B)+C=A+(B+C)

7. Distributive Law

A(B+C)=AB+AC

Principle of Duality

In Boolean algebra:

- Replace OR by AND
- Replace 0 by 1

to obtain dual expression.

Example:

Original:

- A + 0 = A

Dual:

- A . 1 = A

BOOLEAN FUNCTIONS

A Boolean function produces output either 0 or 1.

Example:

F(A,B)=A+B

Unit – III

GRAPH THEORY:

INTRODUCTION TO GRAPH THEORY

Graph Theory is an important branch of discrete mathematics that studies graphs and their properties.

A graph is a mathematical structure used to represent relationships between objects.

In graph theory:

- Objects are represented by vertices (nodes)
- Connections are represented by edges (links)

Graph theory is widely used in:

- Computer networks
- Social media
- Google maps

- Routing algorithms
- Artificial intelligence
- Database systems

HISTORY OF GRAPH THEORY

Graph theory was introduced by the famous mathematician Leonhard Euler in 1736 while solving the “Seven Bridges of Königsberg Problem.”

This problem became the foundation of graph theory.

BASIC TERMINOLOGIES OF GRAPH THEORY

Graph

A graph consists of:

- Set of vertices
- Set of edges

Mathematical representation:

G=(V,E)

Where:

- V = set of vertices
- E = set of edges

Example

Suppose:

- V = {A,B,C,D}
- E = {(A,B),(B,C),(C,D)}

Then:

- G = (V,E)

Vertex

A vertex is a node in a graph.

Example:

- A, B, C, D

Edge

An edge connects two vertices.

Example:

- (A,B)

Adjacent Vertices

Two vertices connected by an edge are called adjacent vertices.

Example:
If:

- (A,B) exists

Then:

- A and B are adjacent.

Incident Edge

An edge connected to a vertex is called incident edge.

DEGREE OF A VERTEX

The number of edges connected to a vertex is called degree.

Symbol:

- deg(v)

Example

If three edges are connected to vertex A:

Then:

deg(A)=3

Isolated Vertex

A vertex having degree zero.

Pendant Vertex

A vertex having degree one.

TYPES OF GRAPHS

1. Simple Graph

A graph without:

- Self-loops
- Parallel edges

2. Multigraph

A graph containing multiple edges between same vertices.

3. Pseudograph

A graph containing self-loops.

4. Directed Graph (Digraph)

A graph where edges have directions.

Example:

- A → B

5. Undirected Graph

A graph where edges have no direction.

Example:

- A — B

6. Weighted Graph

A graph where edges contain weights or costs.

Used in:

- Shortest path algorithms
- Network routing

7. Complete Graph

A graph where every vertex is connected to every other vertex.

Notation:

- Kn

Example:

- K4 → complete graph with 4 vertices.

Number of edges in complete graph:

\frac{n(n-1)}{2}

8. Null Graph

A graph having vertices but no edges.

9. Connected Graph

A graph where every pair of vertices has a path between them.

10. Disconnected Graph

A graph where some vertices are not connected.

PATH IN GRAPH

A path is a sequence of connected vertices.

Example:

- A → B → C → D

Length of Path

The number of edges in the path.

Simple Path

A path where vertices are not repeated.

CIRCUIT OR CYCLE

A closed path where starting and ending vertices are same.

Example:

- A → B → C → A

Simple Cycle

A cycle without repeating vertices except starting and ending vertex.

SUBGRAPH

A graph formed from another graph using some vertices and edges.

GRAPH REPRESENTATION

Graphs can be represented in computers using:

1. Adjacency Matrix
2. Adjacency List

1. Adjacency Matrix

A square matrix representing connections.

If edge exists:

- 1

Otherwise:

- 0

Example

For graph:

- A connected to B

Matrix:

	A	**B**
A	0	1
B	1	0

Advantages

- Easy representation
- Simple implementation

Disadvantages

- Requires more memory

2. Adjacency List

Each vertex stores list of adjacent vertices.

Example:

- A → B,C
- B → A,D

Advantages

- Memory efficient

Disadvantages

- Searching becomes slower

EULER GRAPH

A graph containing Euler circuit is called Euler graph.

Euler Path

A path that visits every edge exactly once.

Euler Circuit

A closed path visiting every edge exactly once.

Conditions for Euler Graph

1. Graph must be connected.
2. Every vertex should have even degree.

Example

If all vertices have even degree:

- Euler circuit exists.

HAMILTONIAN GRAPH

A graph containing Hamiltonian cycle.

Hamiltonian Path

A path visiting every vertex exactly once.

Hamiltonian Cycle

A cycle visiting every vertex exactly once.

Difference Between Euler and Hamiltonian Graph

Euler Graph	Hamiltonian Graph
Visits every edge once	Visits every vertex once
Based on edges	Based on vertices

TREE

A tree is a connected graph without cycles.

Properties of Tree

1. Connected graph
2. No cycles
3. n vertices contain n−1 edges
4. Exactly one path between two vertices

Example

If a tree contains:

- 5 vertices

Then:

- Number of edges = 4

Using:

e=n-1

ROOTED TREE

A tree having one special node called root.

TERMINOLOGIES OF TREES

Parent Node

A node having child nodes.

Child Node

A node connected below parent.

Leaf Node

A node with no children.

Sibling Nodes

Nodes having same parent.

Level of Node

Distance from root node.

Height of Tree

Maximum level in tree.

BINARY TREE

A tree where each node has at most two children.

Children:

- Left child
- Right child

Types of Binary Trees

1. Full Binary Tree

Every node has either:

- 0 children
  or
- 2 children

2. Complete Binary Tree

All levels completely filled except possibly last level.

3. Perfect Binary Tree

All internal nodes have two children and all leaf nodes are at same level.

BINARY SEARCH TREE (BST)

A binary tree where:

- Left subtree contains smaller values
- Right subtree contains larger values

Applications of BST

1. Searching
2. Sorting
3. Databases
4. File systems

SPANNING TREE

A spanning tree connects all vertices without cycles.

Properties of Spanning Tree

1. Includes all vertices
2. Contains n−1 edges
3. No cycles

Minimum Spanning Tree

A spanning tree having minimum total weight.

Used in:

- Telephone networks
- Road networks
- Cable connections

GRAPH TRAVERSAL

Traversal means visiting all vertices systematically.

Two important methods:

1. BFS
2. DFS

BREADTH FIRST SEARCH (BFS)

BFS visits nodes level by level.

Uses:

- Queue data structure

BFS Algorithm

1. Start from source vertex.
2. Visit adjacent vertices.

3. Add visited vertices into queue.
4. Repeat until queue becomes empty.

Applications of BFS

1. Shortest path
2. Social networking
3. Broadcasting systems
4. GPS navigation

DEPTH FIRST SEARCH (DFS)

DFS visits deeply before backtracking.

Uses:

- Stack

DFS Algorithm

1. Start from source node.
2. Visit one adjacent node deeply.
3. Continue until dead end.
4. Backtrack and continue.

Applications of DFS

1. Maze solving
2. Artificial Intelligence
3. Topological sorting
4. Cycle detection

DIFFERENCE BETWEEN BFS AND DFS

BFS	**DFS**
Uses Queue	Uses Stack
Level-wise traversal	Depth-wise traversal
Finds shortest path	Faster in some cases
More memory	Less memory

PLANAR GRAPH

A graph drawable without crossing edges.

Complete Bipartite Graph

Vertices divided into two sets.

Every vertex of one set connects to every vertex of another set.

Notation:

- Km,n

GRAPH COLORING

Assigning colors to vertices such that adjacent vertices have different colors.

Applications:

- Scheduling
- Register allocation
- Map coloring

CHROMATIC NUMBER

Minimum number of colors needed.

APPLICATIONS OF GRAPH THEORY

1. Computer Networks
2. Social Networks
3. Routing Algorithms
4. Web Page Ranking
5. Artificial Intelligence
6. Transportation Systems
7. Circuit Design
8. Database Query Optimization
9. Network Security
10. Operating Systems

REAL LIFE EXAMPLES OF GRAPHS

Real Life Object	**Graph Representation**
Facebook Users	Vertices
Friendships	Edges
Cities	Vertices
Roads	Edges
Computer Systems	Nodes
Network Cables	Links

ADVANTAGES OF GRAPH THEORY

1. Easy representation of relationships
2. Efficient path finding

3. Useful in networking
4. Helps in optimization problems

Unit – IV

RECURRENCE RELATIONS AND GENERATING FUNCTIONS:

INTRODUCTION TO RECURRENCE RELATIONS

A recurrence relation is an equation that defines a sequence using previous terms of the sequence.

In simple words, each term depends on earlier terms.

Recurrence relations are very important in:

- Computer algorithms
- Dynamic programming
- Artificial Intelligence
- Data structures
- Mathematical modeling

SEQUENCE

A sequence is an ordered list of numbers.

Examples:

- 2,4,6,8,10
- 1,1,2,3,5,8

TYPES OF SEQUENCES

1. Arithmetic Sequence

Difference between consecutive terms remains constant.

Example:

- 2,4,6,8,10

Common difference:

- 2

Formula:

a_n=a+(n-1)d

Where:

- a = first term
- d = common difference

Example

Find 5th term of:

- 2,4,6,8,…

Solution:

- a = 2
- d = 2
- n = 5

Using formula:

a_n=a+(n-1)d

Substituting values:

- $a_5 = 2 + (5-1)2$
- $a_5 = 10$

2. Geometric Sequence

Ratio between consecutive terms remains constant.

Example:

- 2,4,8,16,…

Common ratio:

- 2

Formula:

a_n=ar^{n-1}

Example

Find 4th term of:

- 3,6,12,…

Solution:

- a = 3
- r = 2
- n = 4

Then:

- $a_4 = 3(2)^3$
- $a_4 = 24$

RECURRENCE RELATION

A recurrence relation expresses nth term using previous terms.

General Form

a_n=f(a_{n-1},a_{n-2},...,a_{n-k})

Example

a_n=a_{n-1}+2

If:

- $a_1 = 1$

Then:

- $a_2 = 3$
- $a_3 = 5$
- $a_4 = 7$

Sequence:

- 1,3,5,7,…

ORDER OF RECURRENCE RELATION

The order is determined by how many previous terms are involved.

First Order Relation

Depends on one previous term.

Example:

a_n=a_{n-1}+1

Second Order Relation

Depends on two previous terms.

Example:

a_n=a_{n-1}+a_{n-2}

TYPES OF RECURRENCE RELATIONS

1. Linear Recurrence Relation

Variables appear linearly.

Example:

a_n=2a_{n-1}+3

2. Nonlinear Recurrence Relation

Variables appear with powers or products.

Example:

a_n=(a_{n-1})^2

3. Homogeneous Recurrence Relation

No additional independent term exists.

Example:

a_n-2a_{n-1}=0

4. Non-Homogeneous Recurrence Relation

Contains extra term.

Example:

a_n-2a_{n-1}=5

FIBONACCI SEQUENCE

The Fibonacci sequence is one of the most important recurrence relations.

Definition:

F_n=F_{n-1}+F_{n-2}

Initial conditions:

- $F_0 = 0$
- $F_1 = 1$

Fibonacci Series

- 0,1,1,2,3,5,8,13,…

Calculation Example

Using:

F_n=F_{n-1}+F_{n-2}

Find:

- F_4

Solution:

- $F_2 = 1$
- $F_3 = 2$
- $F_4 = 3$

Applications of Fibonacci Sequence

1. Algorithm Design
2. Data Structures
3. Nature Modeling
4. Artificial Intelligence
5. Computer Graphics

SOLVING RECURRENCE RELATIONS

Several methods are used.

1. Iteration Method

Repeated substitution is performed.

Example

Solve:

a_n=2a_{n-1}

Given:

- $a_0 = 1$

Solution:

- $a_1 = 2$
- $a_2 = 4$
- $a_3 = 8$

Thus:

a_n=2^n

2. Characteristic Root Method

Used for linear homogeneous recurrence relations.

Example

Solve:

a_n-3a_{n-1}+2a_{n-2}=0

Auxiliary equation:

r^2-3r+2=0

Factoring:

(r-1)(r-2)=0

Roots:

- r = 1
- r = 2

General solution:

a_n=A(1)^n+B(2)^n

GENERATING FUNCTIONS

A generating function converts sequence into algebraic expression.

It helps solve recurrence relations and counting problems.

Definition

If:

- $a_0, a_1, a_2, \ldots$

Then generating function is:

$G(x)=a_0+a_1x+a_2x^2+a_3x^3+\cdots$

Example

Sequence:

- 1,1,1,1,…

Generating function:

$G(x)=1+x+x^2+x^3+\cdots$

Geometric Generating Function

Using geometric series:

$1+x+x^2+x^3+\cdots=\frac{1}{1-x}$

Example of Generating Function

Find generating function for:

- 1,2,3,4,…

Solution:

$G(x)=1+2x+3x^2+4x^3+\cdots$

APPLICATIONS OF GENERATING FUNCTIONS

1. Solving recurrence relations
2. Counting problems
3. Probability theory
4. Algorithm analysis
5. Combinatorics

COMBINATORICS

Combinatorics is the branch dealing with counting arrangements and selections.

FUNDAMENTAL PRINCIPLE OF COUNTING

If:

- One task can be done in m ways
- Another task in n ways

Then total ways:

$m \times n$

PERMUTATION

Arrangement of objects.

Formula:

${}^nP_r=\frac{n!}{(n-r)!}$

Example

Arrange 2 letters from A,B,C.

Solution:

{}^3P_2=\frac{3!}{1!}=6

COMBINATION

Selection of objects.

Formula:

{}^nC_r=\frac{n!}{r!(n-r)!}

Example

Choose 2 students from 4.

Solution:

{}^4C_2=\frac{4!}{2!2!}=6

BINOMIAL THEOREM

Expansion formula:

(a+b)^n

General term:

T_{r+1}=\binom{n}{r}a^{n-r}b^r

Example

Expand:

(x+y)^2

Result:

- $x^2 + 2xy + y^2$

APPLICATIONS OF RECURRENCE RELATIONS

1. Dynamic Programming
2. Algorithm Complexity
3. Artificial Intelligence
4. Computer Graphics
5. Financial Modeling
6. Population Growth Problems

APPLICATIONS OF COMBINATORICS

1. Cryptography
2. Probability
3. Network Design
4. Coding Theory

5. Scheduling Problems

ADVANTAGES OF RECURRENCE RELATIONS

1. Easy representation of sequences
2. Efficient algorithm analysis
3. Useful in dynamic programming
4. Simplifies complex calculations