Chapter Notes: Set Theory & Relations

Set Theory

Introduction

Set Theory is a branch of mathematical logic that deals with the study of sets, which are well-defined collections of objects or elements. The elements of a set can be anything: numbers, letters, objects, etc. Set theory forms the foundation for many areas of mathematics and computer science.

- **Set Notation**: Sets are typically denoted using curly braces {}. For example, a set of natural numbers less than 5 can be written as $A = \{1, 2, 3, 4\}$.
- **Types of Sets**:
  - **Finite Set**: A set with a limited number of elements, e.g., $A = \{1, 2, 3\}$.
  - **Infinite Set**: A set with an unlimited number of elements, e.g., the set of natural numbers $\mathbb{N}$.
  - **Null or Empty Set**: A set with no elements, denoted as $\emptyset$.
  - **Singleton Set**: A set containing only one element, e.g., $A = \{a\}$.
  - **Universal Set**: A set containing all the elements under consideration, typically denoted as U.

Combination of Sets

There are various operations that can be performed on sets to create new sets:

- **Union**: The union of two sets A and B is the set of all elements that are in either A or B (or both). Denoted as $A \cup B$.
  - Example: $A = \{1, 2\}, B = \{2, 3\} \Rightarrow A \cup B = \{1, 2, 3\}$.
- **Intersection**: The intersection of two sets A and B is the set of all elements that are in both A and B. Denoted as $A \cap B$.
  - Example: $A = \{1, 2\}, B = \{2, 3\} \Rightarrow A \cap B = \{2\}$.
- **Difference**: The difference of two sets A and B, denoted as $A - B$, is the set of elements that are in A but not in B.
  - Example: $A = \{1, 2, 3\}, B = \{2, 3\} \Rightarrow A - B = \{1\}$.
- **Complement**: The complement of a set A, denoted as A', is the set of all elements in the universal set U that are not in A.
  - Example: $U = \{1, 2, 3, 4\}, A = \{1, 2\} \Rightarrow A' = \{3, 4\}$.

Relations

Definition

A relation on a set A is a subset of the Cartesian product $A \times A$. It defines a relationship between elements of the set. A relation R on A is represented as: $R \subseteq A \times A$ For example, if $A = \{1, 2, 3\}$, a relation R can be $\{(1, 2), (2, 3)\}$, indicating that 1 is related to 2, and 2 is related to 3.

Operations on Relations

- **Union of Relations**: If R_1 and R_2 are two relations on A, their union $R_1 \cup R_2$ is the relation containing all pairs that belong to either R_1 or R_2.
- **Intersection of Relations**: The intersection $R_1 \cap R_2$ consists of all pairs that belong to both R_1 and R_2.

- **Difference of Relations**: The difference R1−R2R_1 - R_2 consists of all pairs in R1R_1 that are not in R2R_2.
- **Complement of a Relation**: The complement of RR is the set of all pairs in A×AA \times A that do not belong to RR.

Properties of Relations

- **Reflexive**: A relation RR is reflexive if for every element a∈Aa \in A, (a,a)∈R(a, a) \in R.
- **Symmetric**: A relation RR is symmetric if for every pair (a,b)∈R(a, b) \in R, (b,a)∈R(b, a) \in R.
- **Antisymmetric**: A relation RR is antisymmetric if for every pair (a,b)∈R(a, b) \in R and (b,a)∈R(b, a) \in R, it must be the case that a=ba = b.
- **Transitive**: A relation RR is transitive if whenever (a,b)∈R(a, b) \in R and (b,c)∈R(b, c) \in R, (a,c)∈R(a, c) \in R.

Composite Relations

The composite of two relations R1R_1 and R2R_2 is a relation that connects elements via a third element. It is denoted as R1∘R2R_1 \circ R_2, and it is defined by: (a,c)∈R1∘R2 if there exists a b such that (a,b)∈R1 and (b,c)∈R2(a, c) \in R_1 \circ R_2 \text{ if there exists a } b \text{ such that } (a, b) \in R_1 \text{ and } (b, c) \in R_2

Equality of Relations

Two relations R1R_1 and R2R_2 on a set AA are equal if they contain the same pairs: R1=R2 ⟺ ∀(a,b)∈A×A,(a,b)∈R1 ⟺ (a,b)∈R2R_1 = R_2 \iff \forall (a, b) \in A \times A, (a, b) \in R_1 \iff (a, b) \in R_2

Recursive Definition of Relation

A relation RR can be recursively defined using a base case and a recursive case. For example:

- Base Case: If (a,a)∈R(a, a) \in R, then aa is related to itself.
- Recursive Case: If (a,b)∈R(a, b) \in R and (b,c)∈R(b, c) \in R, then (a,c)∈R(a, c) \in R.

Order of Relations

The order of a relation refers to the number of elements in the relation. If a relation RR is a subset of A×AA \times A, its order is the number of pairs it contains.

POSET & Lattices

Hasse Diagram

A **Hasse Diagram** is a graphical representation of a partially ordered set (POSET). In this diagram:

- Each element is represented by a vertex.
- An edge from element aa to element bb is drawn if a≤ba \leq b and there is no element cc such that a<c<ba < c < b.

POSET (Partially Ordered Set)

A POSET is a set AA equipped with a binary relation ≤\leq (or any other suitable ordering relation) that satisfies the following properties:

- **Reflexive**: a≤aa \leq a for all a∈Aa \in A.
- **Antisymmetric**: If a≤ba \leq b and b≤ab \leq a, then a=ba = b.
- **Transitive**: If a≤ba \leq b and b≤cb \leq c, then a≤ca \leq c.

Lattices

A **lattice** is a special type of POSET where every pair of elements has both a least upper bound (supremum) and a greatest lower bound (infimum). A lattice is defined as:

- **Join**: The least upper bound (supremum) of two elements, denoted a∨ba \vee b.
- **Meet**: The greatest lower bound (infimum) of two elements, denoted a∧ba \wedge b.

Properties of Lattices

- **Bounded Lattice**: A lattice that has a greatest element (denoted 11) and a least element (denoted 00).
- **Complemented Lattice**: A lattice in which every element has a complement. For an element aa, there exists an element bb such that a∨b=1a \vee b = 1 and a∧b=0a \wedge b = 0.
- **Distributed Lattice**: A lattice where the meet and join operations distribute over each other, i.e., a∧(b∨c)=(a∧b)∨(a∧c)a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) and a∨(b∧c)=(a∨b)∧(a∨c)a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c).
- **Modular Lattice**: A lattice where the distributive property holds only in certain cases. Specifically, a∨(b∧c)=(a∨b)∧(a∨c)a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) holds when a≤ca \leq c.
- **Complete Lattice**: A lattice where every subset has both a supremum and an infimum.

Conclusion

Understanding **Set Theory** and **Relations** is crucial for solving many computer science problems, such as database theory, algorithms, and graph theory. The concept of **POSETs** and **Lattices** is also fundamental in areas like logic, algebra, and discrete mathematics, where the organization and relationship of elements play a key role in the design and analysis of systems.

These concepts form the mathematical backbone of many computational structures, algorithms, and data representations used in modern computing.

Chapter Notes: Functions & Boolean Algebra

Functions

Definition of a Function

A function is a relation between two sets, typically denoted as AA and BB, where each element of the set AA (called the domain) is associated with exactly one element of the set BB (called the codomain). A function from set AA to set BB is denoted as:

f:A→Bf: A \to B

The element f(a)f(a) is the image of a∈Aa \in A under the function ff.

- **Domain**: The set of all possible inputs to the function.
- **Codomain**: The set of all possible outputs.
- **Range**: The set of actual outputs produced by the function for all elements in the domain.

Classification of Functions

Functions can be classified based on their properties or their behavior. Some common classifications include:

1. **One-to-One (Injective) Function**:
    - A function f:A→Bf: A \to B is **one-to-one** if different elements in the domain map to different elements in the codomain.
    - Formally, f(a1)=f(a2)f(a_1) = f(a_2) implies a1=a2a_1 = a_2.

2. **Onto (Surjective) Function**:
   - A function f:A→Bf: A \to B is **onto** if every element of BB is the image of at least one element in AA.
   - For every b∈Bb \in B, there exists at least one a∈Aa \in A such that f(a)=bf(a) = b.
3. **One-to-One Correspondence (Bijective) Function**:
   - A function f:A→Bf: A \to B is **bijective** if it is both one-to-one and onto.
   - A bijective function establishes a perfect pairing between the elements of AA and BB, with no elements left unpaired in either set.
4. **Constant Function**:
   - A function f:A→Bf: A \to B is **constant** if it maps every element of AA to the same element of BB, i.e., f(a1)=f(a2)f(a_1) = f(a_2) for all a1,a2∈Aa_1, a_2 \in A.
5. **Identity Function**:
   - A function f:A→Af: A \to A is the **identity function** if f(a)=af(a) = a for all a∈Aa \in A.
6. **Inverse Function**:
   - If a function f:A→Bf: A \to B is bijective, its inverse f−1:B→Af^{-1}: B \to A exists, and for every element b∈Bb \in B, f−1(f(a))=af^{-1}(f(a)) = a and f(f−1(b))=bf(f^{-1}(b)) = b.
7. **Many-to-One Function**:
   - A function f:A→Bf: A \to B is **many-to-one** if two or more elements in AA map to the same element in BB.

Operations on Functions

Functions can be combined or operated upon in several ways:

1. **Function Composition**:
   - Given two functions f:A→Bf: A \to B and g:B→Cg: B \to C, the composition of ff and gg, denoted by g∘fg \circ f, is a function from AA to CC defined by: (g∘f)(a)=g(f(a))(g \circ f)(a) = g(f(a))
2. **Inverse of a Function**:
   - As mentioned earlier, if ff is a bijective function, its inverse f−1f^{-1} is defined such that: f−1(f(a))=aandf(f−1(b))=bf^{-1}(f(a)) = a \quad \text{and} \quad f(f^{-1}(b)) = b
3. **Scalar Multiplication of Functions**:
   - For a constant kk, the scalar multiplication of a function ff is the function k⋅fk \cdot f where (k⋅f)(a)=k⋅f(a)(k \cdot f)(a) = k \cdot f(a).

Growth of Functions

The **growth rate** of a function describes how quickly the value of the function increases as its input grows. In computer science, this concept is important for analyzing the efficiency of algorithms, particularly in terms of time complexity.

- **Big-O Notation**: Represents the upper bound of the growth rate of a function, i.e., the worst-case time complexity.
  - For example, if f(n)=O(g(n))f(n) = O(g(n)), then there exists a constant cc such that f(n)≤c⋅g(n)f(n) \leq c \cdot g(n) for large nn.
- **Common Growth Rates**:
  - **Constant**: O(1)O(1)
  - **Logarithmic**: O(log n)O(\log n)

- **Linear**: O(n)O(n)
- **Quadratic**: O(n2)O(n^2)
- **Cubic**: O(n3)O(n^3)
- **Exponential**: O(2n)O(2^n)

Boolean Algebra

Introduction

Boolean Algebra is a branch of algebra that deals with logical variables and operators. It was introduced by George Boole in the 19th century and is fundamental in digital circuit design, computer science, and logic theory.

In Boolean Algebra, the variables can take only two values: **True (1)** or **False (0)**. The operations on these values include AND, OR, and NOT.

Axioms and Theorems of Boolean Algebra

Boolean Algebra operates on a set of axioms and theorems that govern its operations. The axioms of Boolean algebra are:

1. **Identity Law**:
   - A·1=AA \cdot 1 = A
   - A+0=AA + 0 = A
2. **Null Law**:
   - A·0=0A \cdot 0 = 0
   - A+1=1A + 1 = 1
3. **Idempotent Law**:
   - A·A=AA \cdot A = A
   - A+A=AA + A = A
4. **Complement Law**:
   - A·A¯=0A \cdot \overline{A} = 0
   - A+A¯=1A + \overline{A} = 1
5. **Distributive Law**:
   - A·(B+C)=(A·B)+(A·C)A \cdot (B + C) = (A \cdot B) + (A \cdot C)
   - A+(B·C)=(A+B)·(A+C)A + (B \cdot C) = (A + B) \cdot (A + C)
6. **Commutative Law**:
   - A·B=B·AA \cdot B = B \cdot A
   - A+B=B+AA + B = B + A
7. **Associative Law**:
   - A·(B·C)=(A·B)·CA \cdot (B \cdot C) = (A \cdot B) \cdot C
   - A+(B+C)=(A+B)+CA + (B + C) = (A + B) + C

Algebraic Manipulation of Boolean Expressions

Boolean expressions can be simplified using algebraic rules, much like arithmetic algebra. This is done to make logical circuits more efficient.

- **Example**: Simplify A·(A+B)A \cdot (A + B): Using the Distributive Law, we get: A·(A+B)=A·A+A·B=A+A·B=AA \cdot (A + B) = A \cdot A + A \cdot B = A + A \cdot B = A (since A·A=AA \cdot A = A).

Simplification of Boolean Functions

The goal of simplifying Boolean functions is to reduce the number of terms and operations, which in turn simplifies the implementation of digital circuits.

- **Common Techniques**:
  - **Boolean Identities**: Using laws like the Idempotent Law and the Complement Law.
  - **Consensus Theorem**: A term can be removed from a Boolean expression if its presence does not affect the result. For example: A·B+A¯·C+B·C=A·B+A¯·CA \cdot B + \overline{A} \cdot C + B \cdot C = A \cdot B + \overline{A} \cdot C
  - **Quine–McCluskey Algorithm**: A tabular method used for minimizing Boolean functions.

Karnaugh Maps (K-Maps)

Karnaugh maps (or K-maps) are a graphical method for simplifying Boolean functions. It is particularly useful for functions with up to six variables.

- **Procedure**:
  - Plot the truth table of a Boolean function in a grid.
  - Group adjacent cells containing 1s (ones) in powers of two (i.e., 1, 2, 4, 8, etc.).
  - For each group, write the simplified Boolean expression for the variables that are constant within the group.
  - The final result is the sum (OR) of these simplified expressions.
- **Example**: Consider the function f(A,B,C)=A¯·B+A·C¯f(A, B, C) = \overline{A} \cdot B + A \cdot \overline{C}. By plotting this in a K-map and grouping the 1s, we can reduce the expression.

Conclusion

Functions and **Boolean Algebra** are essential topics in discrete mathematics and computer science. Functions help describe relationships between sets and are foundational to areas such as algorithms and data structures. Boolean Algebra, on the other hand, plays a crucial role in simplifying logical expressions and designing efficient digital circuits, which are integral to computing systems. Understanding these concepts enables better problem-solving, algorithm optimization, and circuit design.

Chapter Notes: Theory of Logic & Predicate Logic

Theory of Logic

Proposition

A **proposition** (also known as a statement) is a declarative sentence that is either true or false, but not both. In logic, propositions are the basic building blocks of logical reasoning. They are denoted by symbols such as PP, QQ, RR, etc.

- **Example**:
  - "The sky is blue" is a proposition because it can be true or false.

- "2 + 2 = 5" is also a proposition, but it is false.

A proposition is often represented in logical notation as PP, and its truth value is either **True (T)** or **False (F)**.

Truth Tables

A **truth table** is a tabular representation of all possible truth values for a set of propositions. It shows the outcome of logical operations (like AND, OR, NOT, etc.) on those propositions. Truth tables help in analyzing the validity of logical expressions.

- **Example of a Truth Table for AND (P∧QP \land Q)**:

PP	QQ	P∧QP \land Q
T	T	T
T	F	F
F	T	F
F	F	F

This truth table shows that P∧QP \land Q is true only when both PP and QQ are true.

Tautology

A **tautology** is a logical expression that is always true, regardless of the truth values of the individual propositions. In other words, a tautology is a formula that evaluates to true for every possible combination of truth values of its propositions.

- **Example**:
  - P∨¬PP \lor \neg P (Law of excluded middle): This is a tautology because, whether PP is true or false, P∨¬PP \lor \neg P will always be true.

Satisfiability

A logical expression is **satisfiable** if there exists at least one combination of truth values for its variables that makes the expression true. If an expression can be made true for some assignment of truth values, it is said to be satisfiable.

- **Example**:
  - The expression P∧QP \land Q is satisfiable because there is a combination of PP and QQ (specifically, P=TP = T and Q=TQ = T) that makes it true.

Contradiction

A **contradiction** is a logical expression that is always false, regardless of the truth values of the propositions involved. A contradiction cannot be satisfied under any conditions.

- **Example**:
  - P∧¬PP \land \neg P is a contradiction because it asserts that PP is both true and false at the same time, which is impossible.

Algebra of Propositions

The **algebra of propositions** refers to the set of operations and rules used to manipulate and simplify logical expressions. It involves a set of logical connectives, such as **AND**, **OR**, **NOT**, and **implication**, which follow specific algebraic rules (analogous to the laws of algebra in arithmetic).

- **Basic Operations**:
  - **Conjunction (AND)**: P∧QP \land Q
  - **Disjunction (OR)**: P∨QP \lor Q

  - **Negation (NOT)**: ¬P\neg P
  - **Implication**: P→QP \to Q
  - **Biconditional**: P↔QP \leftrightarrow Q
- **Laws in the Algebra of Propositions**:
  - **Identity Law**: P∧T=PP \land T = P, P∨F=PP \lor F = P
  - **Domination Law**: P∧F=FP \land F = F, P∨T=TP \lor T = T
  - **Idempotent Law**: P∧P=PP \land P = P, P∨P=PP \lor P = P
  - **Double Negation Law**: ¬(¬P)=P\neg(\neg P) = P
  - **De Morgan's Laws**:
    - ¬(P∧Q)=¬P∨¬Q\neg(P \land Q) = \neg P \lor \neg Q
    - ¬(P∨Q)=¬P∧¬Q\neg(P \lor Q) = \neg P \land \neg Q

Theory of Inference

In logic, **inference** refers to the process of deriving new propositions from existing ones based on logical rules. An **inference rule** allows you to make conclusions from premises.

- **Example of Inference Rules**:
  - **Modus Ponens**: If P→QP \to Q and PP, then QQ.
  - **Modus Tollens**: If P→QP \to Q and ¬Q\neg Q, then ¬P\neg P.
  - **Hypothetical Syllogism**: If P→QP \to Q and Q→RQ \to R, then P→RP \to R.

Inferences can be made through **deductive reasoning**, where the conclusion necessarily follows from the premises, or **inductive reasoning**, where the conclusion is likely but not certain.

Predicate Logic

First Order Predicate

Predicate logic extends propositional logic by involving **predicates** and **quantifiers**. A **predicate** is a function that takes one or more arguments and returns a proposition. A **first-order predicate** is a predicate that takes one argument, often represented by a variable.

- **Example**:
  - P(x)P(x) could represent "x is a prime number."
  - ∀x P(x)\forall x \, P(x) means "For all x, x is a prime number."

A predicate logic expression involves not only propositions but also variables and functions that can take values from a specific domain.

Well-formed Formula (WFF) of Predicate Logic

A **well-formed formula (WFF)** in predicate logic is a syntactically correct expression constructed from predicates, variables, logical connectives, and quantifiers. The structure of a WFF follows rules similar to those in propositional logic, but with added complexity due to the involvement of predicates and quantifiers.

- **Example of WFF**:
  - ∀x (P(x)→Q(x))\forall x \, (P(x) \to Q(x)): "For all x, if x is a prime, then x is greater than 1."
  - ∃x P(x)\exists x \, P(x): "There exists an x such that x is prime."

Quantifiers

In predicate logic, **quantifiers** are used to specify the scope of a variable in a logical expression. There are two main types of quantifiers:

1. **Universal Quantifier (∀\forall)**:
    - The universal quantifier ∀x\forall x indicates that the statement it precedes is true for all values of xx in the domain.
    - Example: ∀x P(x)\forall x \, P(x) means "For all xx, P(x)P(x) is true."
2. **Existential Quantifier (∃\exists)**:
    - The existential quantifier ∃x\exists x asserts that there is at least one value of xx for which the statement is true.
    - Example: ∃x P(x)\exists x \, P(x) means "There exists at least one xx such that P(x)P(x) is true."

Inference Theory of Predicate Logic

Inference in predicate logic involves deriving conclusions from a set of premises using logical rules. Just like in propositional logic, predicate logic has inference rules that allow you to derive conclusions from given premises. However, predicate logic also includes rules for handling quantifiers and variables.

- **Rules of Inference for Quantifiers**:
    - **Universal Instantiation (UI)**: From ∀x P(x)\forall x \, P(x), you can infer P(a)P(a) for any particular aa.
    - **Existential Generalization (EG)**: From P(a)P(a), you can infer ∃x P(x)\exists x \, P(x).
    - **Universal Generalization (UG)**: From a conclusion about a particular element, you can generalize to all elements, i.e., from P(a)P(a) for an arbitrary aa, infer ∀x P(x)\forall x \, P(x).
- **Example of Inference**:
    - Given the premises ∀x (P(x)→Q(x))\forall x \, (P(x) \to Q(x)) and P(a)P(a), we can infer Q(a)Q(a) by **Universal Instantiation** and **Modus Ponens**.

Conclusion

Theory of Logic and **Predicate Logic** provide the foundational structures for reasoning about truth, validity, and inference in mathematics, computer science, and artificial intelligence. While propositional logic focuses on simple statements and their relationships, predicate logic allows for more complex statements involving variables and quantifiers, making it a powerful tool for formal reasoning. The rules and principles discussed in this chapter are essential for understanding how logical systems work, forming the basis for logical proofs, programming languages, and reasoning systems.

Chapter Notes: Algebraic Structures

Algebraic Structures: Definition

In mathematics, an **algebraic structure** is a set of elements equipped with one or more operations that satisfy specific axioms or rules. These operations are typically binary (involving two elements from the set), and the structures are fundamental in various branches of mathematics, including group theory, ring theory, and field theory.

Key Types of Algebraic Structures:

- **Group**: A set with a binary operation that satisfies four properties: closure, associativity, identity element, and invertibility.
- **Ring**: A set equipped with two operations (addition and multiplication) that satisfies certain properties, such as associativity and distributivity.
- **Field**: A ring where every non-zero element has a multiplicative inverse.

Groups

Definition of a Group

A **group** is an algebraic structure GG with a binary operation (often denoted as ·\cdot or ++) that satisfies four fundamental properties:

1. **Closure**: For all a,b∈Ga, b \in G, the result of the operation a·ba \cdot b must also belong to GG.
2. **Associativity**: For all a,b,c∈Ga, b, c \in G, the equation (a·b)·c=a·(b·c)(a \cdot b) \cdot c = a \cdot (b \cdot c) must hold.
3. **Identity Element**: There exists an identity element e∈Ge \in G such that for every element a∈Ga \in G, a·e=e·a=aa \cdot e = e \cdot a = a.
4. **Inverse Element**: For every element a∈Ga \in G, there exists an inverse element a−1∈Ga^{-1} \in G such that a·a−1=a−1·a=ea \cdot a^{-1} = a^{-1} \cdot a = e.

Subgroups and Order of Groups

- A **subgroup** HH of a group GG is a subset of GG that itself forms a group under the same operation as GG. For HH to be a subgroup, it must satisfy the group properties:
    1. Closure: If a,b∈Ha, b \in H, then a·b∈Ha \cdot b \in H.
    2. Identity: The identity element of GG must be in HH.
    3. Inverses: If a∈Ha \in H, then a−1∈Ha^{-1} \in H.
- The **order** of a group is the number of elements in the group, denoted |G||G|. The **order** of an element a∈Ga \in G, denoted o(a)o(a), is the smallest positive integer nn such that an=ea^n = e (where ee is the identity element).

Cyclic Groups

A group GG is called **cyclic** if there exists an element g∈Gg \in G such that every element of GG can be written as gng^n for some integer nn. The group is said to be generated by gg, and gg is called a **generator** of the group. A cyclic group is isomorphic to the additive group of integers modulo some nn, denoted Zn\mathbb{Z}_n.

- **Example**: Z5\mathbb{Z}_5 under addition modulo 5 is a cyclic group generated by 11.

Cosets

A **coset** is a subset of a group formed by adding (or multiplying) a fixed element aa to each element of a subgroup HH of GG. There are two types of cosets:

1. **Left Coset**: A left coset of HH with respect to a∈Ga \in G is the set aH={a·h|h∈H}aH = \{ a \cdot h \mid h \in H \}.
2. **Right Coset**: A right coset of HH with respect to a∈Ga \in G is the set Ha={h·a|h∈H}Ha = \{ h \cdot a \mid h \in H \}.

Cosets of a subgroup partition the group into disjoint subsets.

Lagrange's Theorem

Lagrange's Theorem states that if GG is a finite group and HH is a subgroup of GG, then the order (number of elements) of HH divides the order of GG. That is,

$$|G| = |H| \cdot [G:H]$$

where $[G:H]$ is the index of H in G, representing the number of distinct cosets of H in G.

- **Example**: In the group $\mathbb{Z}_6$ (integers modulo 6), the subgroups are $\{0\}$, $\{0, 3\}$, and $\{0, 1, 2, 3, 4, 5\}$. The orders of these subgroups divide 6.

Normal Subgroups

A **normal subgroup** N of a group G is a subgroup that is invariant under conjugation by elements of G. That is, for all $g \in G$ and $n \in N$, the element $g \cdot n \cdot g^{-1} \in N$. Normal subgroups are important because the quotient group G/N can be formed.

- **Example**: In $\mathbb{Z}_6$, the subgroup $\{0, 3\}$ is normal because $3 + x \equiv x + 3$ modulo 6 for all x.

Permutation Groups and Symmetric Groups

Permutation Groups

A **permutation** is a rearrangement of elements in a set. The set of all permutations of a finite set S forms a group called the **symmetric group**, denoted S_n, where n is the number of elements in S.

- **Example**: The symmetric group S_3 consists of all possible permutations of three elements $\{1, 2, 3\}$, which are the elements: $\{ (1), (12), (13), (23), (132), (123) \}$.

The operation in a symmetric group is **composition** of permutations, which is associative, has an identity permutation (which leaves elements unchanged), and each permutation has an inverse.

Symmetric Groups

The **symmetric group** S_n is the group of all permutations of n elements. The order of S_n is $n!$ (factorial), and the group consists of all possible bijections (one-to-one and onto functions) from the set $\{1, 2, \ldots, n\}$ to itself.

Group Homomorphisms

Definition of Group Homomorphisms

A **group homomorphism** is a function $\varphi: G \to H$ between two groups G and H that preserves the group operation. That is, for all $a, b \in G$,

$$\varphi(a \cdot b) = \varphi(a) \cdot \varphi(b)$$

A homomorphism must map the identity element of G to the identity element of H, and it must map the inverse of each element in G to the inverse of the image in H.

- **Example**: The map $\varphi: \mathbb{Z}_6 \to \mathbb{Z}_3$, given by $\varphi(x) = x \mod 3$, is a homomorphism because it preserves addition modulo 6 and modulo 3.

Kernel and Image of a Homomorphism

- The **kernel** of a homomorphism $\varphi: G \to H$ is the set of elements in G that map to the identity element of H, i.e., $\text{ker}(\varphi) = \{g \in G \mid \varphi(g) = e_H\}$.
- The **image** of a homomorphism is the set of all elements in H that are the image of some element of G, i.e., $\text{im}(\varphi) = \{\varphi(g) \mid g \in G\}$.

Rings and Fields

Rings

A **ring** is an algebraic structure consisting of a set RR equipped with two operations: addition (+) and multiplication (×). A ring must satisfy the following properties:

1. (R,+)(R, +) is an abelian group (i.e., addition is commutative and associative, and there is an additive identity).
2. Multiplication is associative.
3. **Distributivity**: Multiplication distributes over addition, i.e., a·(b+c)=a·b+a·ca \cdot (b + c) = a \cdot b + a \cdot c and (a+b)·c=a·c+b·c(a + b) \cdot c = a \cdot c + b \cdot c.
4. **Multiplicative identity** (optional): If there is an element 11 such that a·1=aa \cdot 1 = a for all a∈Ra \in R, the ring is called a **unital ring**.

- **Example**: The set of integers Z\mathbb{Z} with usual addition and multiplication is a ring.

Fields

A **field** is a ring with the additional property that every non-zero element has a multiplicative inverse. Fields have the following properties:

1. (F,+)(F, +) is an abelian group.
2. (F\{0},·)(F \setminus \{0\}, \cdot) is an abelian group (multiplicative inverses exist for all non-zero elements).
3. Distributivity of multiplication over addition holds.

- **Example**: The set of rational numbers Q\mathbb{Q}, real numbers R\mathbb{R}, and complex numbers C\mathbb{C} are fields.

Conclusion

Algebraic structures like groups, rings, and fields form the foundation of many areas of mathematics and computer science. Groups are fundamental to symmetry, algebraic operations, and cryptography, while rings and fields play vital roles in algebraic number theory, polynomial equations, and coding theory. Understanding these structures and their properties is crucial for solving complex mathematical problems and modeling various systems.

Chapter Notes: Graphs and Combinatorics

Graphs

Definition and Terminology

A **graph** GG is a mathematical structure consisting of a set of **vertices** (also called nodes) and a set of **edges** (also called arcs or links), where each edge connects a pair of vertices. The basic components of a graph can be formally defined as follows:

- **Vertices**: The individual points in the graph, typically represented by V(G)V(G).
- **Edges**: The connections between vertices, often represented as E(G)E(G). Each edge connects two vertices, and may be directed (in a directed graph) or undirected (in an undirected graph).

The graph can be represented as a pair G=(V,E)G = (V, E), where VV is the set of vertices, and EE is the set of edges.

- **Degree**: The **degree** of a vertex vv in an undirected graph is the number of edges incident to it. In a directed graph, a vertex has an **in-degree** (the number of edges directed towards it) and an **out-degree** (the number of edges directed away from it).
- **Adjacency**: Two vertices are said to be **adjacent** if there is an edge connecting them.
- **Path**: A **path** in a graph is a sequence of vertices such that each adjacent pair is connected by an edge.

Representation of Graphs

Graphs can be represented in several ways:

1. **Adjacency Matrix**: An n×nn \times n matrix where nn is the number of vertices. The element aija_{ij} is non-zero if there is an edge between vertices ii and jj.
   - For undirected graphs, the matrix is symmetric.
2. **Adjacency List**: A collection of lists, where each list corresponds to a vertex and contains all the vertices adjacent to it.
3. **Edge List**: A list of pairs of vertices where each pair represents an edge between two vertices.

Multigraphs

A **multigraph** is a type of graph that allows multiple edges (also called parallel edges) between the same pair of vertices. This means two vertices can be connected by more than one edge.

- **Example**: A multigraph can represent a situation where there are multiple routes between two cities in a transportation network.

Bipartite Graphs

A **bipartite graph** is a graph whose set of vertices can be divided into two disjoint sets UU and VV such that every edge connects a vertex in UU to a vertex in VV. In other words, there are no edges within a set UU or within a set VV.

- **Example**: A bipartite graph can represent a job assignment problem where one set represents workers and the other set represents jobs.

Planar Graphs

A **planar graph** is a graph that can be drawn on a plane without any edges crossing. In other words, it is possible to embed the graph in the plane such that no two edges intersect except at their vertices.

- **Example**: The graph representing the 4 vertices of a tetrahedron is a planar graph because it can be drawn on a plane without edge crossings.

Isomorphism and Homeomorphism of Graphs

1. **Graph Isomorphism**: Two graphs G1=(V1,E1)G_1 = (V_1, E_1) and G2=(V2,E2)G_2 = (V_2, E_2) are said to be **isomorphic** if there is a one-to-one correspondence between the vertices of G1G_1 and G2G_2, and there is an edge between two vertices in G1G_1 if and only if there is an edge between the corresponding vertices in G2G_2. Isomorphic graphs are essentially the same graph but may look different due to the arrangement of their vertices.
2. **Graph Homeomorphism**: Two graphs are **homeomorphic** if one graph can be transformed into the other by repeatedly replacing edges by paths with two vertices and a single edge, and vice versa.

Euler and Hamiltonian Paths

1. **Eulerian Path**: An **Eulerian path** in a graph is a path that visits every edge exactly once. A graph contains an Eulerian path if and only if it has exactly 0 or 2 vertices with an odd degree.
   - **Eulerian Circuit**: An Eulerian path that starts and ends at the same vertex is called an Eulerian circuit. A graph has an Eulerian circuit if and only if all its vertices have even degree.

2. **Hamiltonian Path**: A **Hamiltonian path** is a path that visits every vertex exactly once. A **Hamiltonian circuit** is a Hamiltonian path that starts and ends at the same vertex. Unlike Eulerian paths, there is no simple necessary and sufficient condition for a graph to have a Hamiltonian path.

Graph Coloring

Graph coloring is the assignment of labels (or "colors") to the vertices of a graph such that no two adjacent vertices share the same color. The **chromatic number** of a graph is the smallest number of colors required to color the graph.

- **Example**: A map coloring problem can be modeled as a graph coloring problem where each region is a vertex, and an edge connects two vertices if the regions share a border.

Combinatorics

Introduction

Combinatorics is the branch of mathematics focused on counting, arranging, and analyzing discrete structures. It has applications in computer science, cryptography, probability theory, and many other fields. Key topics in combinatorics include counting techniques, permutations, combinations, and the famous Pigeonhole Principle.

Counting Techniques

Combinatorics provides several methods for counting the number of ways to arrange or select objects. These techniques include:

1. **Factorial Notation**: The number of ways to arrange nn distinct objects is given by n!n!.
2. **Permutations**: A **permutation** of nn objects is an arrangement of those objects in a specific order. The number of permutations of nn objects is n!n!, and the number of permutations of rr objects selected from nn objects is given by P(n,r)=n!(n−r)!P(n, r) = \frac{n!}{(n-r)!}.
3. **Combinations**: A **combination** is a selection of objects without regard to order. The number of ways to choose rr objects from nn objects is given by the binomial coefficient C(n,r)=n!r!(n−r)!C(n, r) = \frac{n!}{r!(n-r)!}.

Pigeonhole Principle

The **Pigeonhole Principle** is a simple yet powerful principle in combinatorics. It states that if nn items are put into mm containers, and if n>mn > m, then at least one container must contain more than one item. This principle is used in many proofs and applications in combinatorics.

- **Example**: If 13 people are in a room, at least two people must have the same birth month, since there are only 12 months in a year. This is an application of the Pigeonhole Principle.

Conclusion

Graphs and combinatorics are fundamental areas of discrete mathematics with widespread applications in computer science, operations research, and network theory. Understanding the properties of graphs, such as isomorphism, Eulerian paths, and graph coloring, provides essential insights into problems related to networks, optimization, and scheduling. Combinatorics, with its powerful counting techniques and principles like the Pigeonhole Principle, is crucial for solving problems related to arrangement, selection, and optimization. Together, these topics form a critical foundation for advanced studies in algorithms, data structures, and mathematical modeling.