Discrete Mathematics(H)
Southern University of Science and Technology
Mengxuan Wu
12212006

Midterm Review

Mengxuan Wu

1 Logic

1.1 Propositional Logic

1.1.1 Propositions

A proposition is a **declarative** sentence that is **either true or false**, but not both. For example, "*SUSTech is in Shenzhen*" is a proposition, while "*No parking on campus*" is not a proposition.

1.1.2 Logical Connectives

There are six logical connectives in propositional logic, which are **negation**($\neg$), **conjunction**($\wedge$), **disjunction**($\vee$), **exclusive or**($\oplus$), **implication**($\rightarrow$), and **biconditional**($\leftrightarrow$).

For implication $p \rightarrow q$, we call p the **hypothesis** and q the **conclusion**.

The **converse** of $p \rightarrow q$ is $q \rightarrow p$. The **inverse** of $p \rightarrow q$ is $\neg p \rightarrow \neg q$. The **contrapositive** of $p \rightarrow q$ is $\neg q \rightarrow \neg p$.

1.1.3 Tautologies and Contradictions

A **tautology** is a proposition that is always true, regardless of the truth values of its individual components. A **contradiction** is a proposition that is always false. A **contingency** is a proposition that is neither a tautology nor a contradiction.

1.1.4 Logical Equivalences

Two propositions are **logically equivalent** if they have the same truth values for all possible combinations of truth values of their component propositions.

The propositions p and q are logically equivalent if and only if $p \leftrightarrow q$ is a tautology, denoted by $p \equiv q$ or $p \Leftrightarrow q$.

1.1.5 Important Logical Equivalences

- **Identity laws**

$$p \wedge T \equiv p$$
$$p \vee F \equiv p$$

- **Domination laws**

$$p \vee T \equiv T$$
$$p \wedge F \equiv F$$

- **Idempotent laws**

$$p \vee p \equiv p$$
$$p \wedge p \equiv p$$

- **Double negation laws**

$$\neg(\neg p) \equiv p$$

- **Commutative laws**

$$p \vee q \equiv q \vee p$$
$$p \wedge q \equiv q \wedge p$$

- **Associative laws**

$$(p \vee q) \vee r \equiv p \vee (q \vee r)$$
$$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$$

- **Distributive laws**

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

- **De Morgan's laws**

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$
$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

- **Absorption laws**

$$p \vee (p \wedge q) \equiv p$$
$$p \wedge (p \vee q) \equiv p$$

- **Negation laws**

$$p \vee \neg p \equiv T$$
$$p \wedge \neg p \equiv F$$

- **Useful law**

$$p \to q \equiv \neg p \vee q$$

1.2 Predicate Logic

1.2.1 Predicates and Quantifiers

A **constant** models a specific object. A **variable** represents objects of specific type. A **predicate** represents properties or relations among objects.

However, a predicate is not a proposition, because it is not a declarative sentence. It becomes a proposition when the variable is assigned a value. Additionally, the universal quantification and existential quantification of a predicate is a proposition. For example, $\text{Prime}(x)$ **is not** a proposition, while $\text{Prime}(3)$ and $\exists x \text{Prime}(x)$ **are** propositions.

The **universe(domain)** D of a predicate is the set of all objects that can be substituted for the variables in a predicate. The **truth set** of a predicate $P(x)$ is the set of objects in the universe that can be substituted for x in $P(x)$ to make the resulting proposition true.

The truth values of $\forall x P(x)$ and $\exists x P(x)$ depend on both the **universe** and the **predicate** $P(x)$.

1.2.2 Precedence of Quantifiers

The quantifiers $\forall$ and $\exists$ have higher precedence than all the logical operators. For example, $\forall x P(x) \wedge Q(x)$ means $(\forall x P(x)) \wedge Q(x)$, not $\forall x(P(x) \wedge Q(x))$.

1.2.3 Translation with Quantifiers

Universal quantification

Sentence: All SUSTech students are smart.

Universe: all students

Translation: $\forall x(\text{At}(x, \text{SUSTech}) \rightarrow \text{Smart}(x))$

Typical error: $\forall x(\text{At}(x, \text{SUSTech}) \wedge \text{Smart}(x))$, which means all students are at SUSTech and smart.

Existential quantification

Sentence: Some SUSTech students are smart.

Universe: all students

Translation: $\exists x(\text{At}(x, \text{SUSTech}) \wedge \text{Smart}(x))$

Typical error: $\exists x(\text{At}(x, \text{SUSTech}) \rightarrow \text{Smart}(x))$, this is true if there is anyone who is not at SUSTech.

1.2.4 Nested Quantifiers

The order of nested quantifiers is important. For example, let $L(x, y)$ denotes "x loves y", then $\forall x \exists y L(x, y)$ means "Everyone loves someone", while $\exists y \forall x L(x, y)$ means "There is someone whom everyone loves".

However, the order of nested quantifiers does not matter if quantifiers are of the same type.

1.2.5 Negating Quantifiers

- $\neg \forall x P(x) \equiv \exists x \neg P(x)$
- $\neg \exists x P(x) \equiv \forall x \neg P(x)$

- $\neg\forall x\exists y P(x,y) \equiv \exists x\forall y\neg P(x,y)$

2 Mathematical Proofs

2.1 Theorems and Proofs

2.1.1 Definitions

An **axiom** or **postulate** is a statement or proposition that is regarded as being established, accepted, or self-evidently true. A **theorem** is a statement or proposition that can be proved to be true. A **lemma** is a statement that can be proved to be true, and is used in proving a theorem.

In **formal proofs**, steps follow logically from the set of premises, axioms, lemmas, and other previously proved theorems.

2.1.2 Rules of inference

- **Modus Ponens**: $((p \to q) \wedge p) \to q$
- **Modus Tollens**: $((p \to q) \wedge \neg q) \to \neg p$
- **Hypothetical Syllogism**: $((p \to q) \wedge (q \to r)) \to (p \to r)$
- **Disjunctive Syllogism**: $((p \vee q) \wedge \neg p) \to q$
- **Addition**: $p \to (p \vee q)$
- **Simplification**: $(p \wedge q) \to p$
- **Conjunction**: $((p) \wedge (q)) \to (p \wedge q)$
- **Resolution**: $((p \vee q) \wedge (\neg p \vee r)) \to (q \vee r)$
- **Universal instantiation**: $\forall x P(x) \to P(c)$
- **Universal generalization**: $P(c)$ for an arbitrary $c \to \forall x P(x)$
- **Existential instantiation**: $\exists x P(x) \to P(c)$ for some element c
- **Existential generalization**: $P(c) \to \exists x P(x)$

2.1.3 Methods of Proof

To proof $p \to q$:

- **Direct proof:** Show that if p is true then q follows.
- **Proof by contrapositive:** Show that $\neg q \to \neg p$.
- **Proof by contradiction:** Show that $p \wedge \neg q$ contradicts the assumptions.
- **Proof by cases:** Give proofs for all possible cases.
- **Proof of equivalence** $p \leftrightarrow q$: Prove $p \to q$ and $q \to p$.

3 Sets and Functions

3.1 Sets

3.1.1 Definitions

A **set** is an unordered collection of objects, called **elements** or **members** of the set. We can represent a set by listing its elements between braces, or defining a property that its elements satisfy.

3.1.2 Important Sets

- $\mathbb{N}$ is the set of natural numbers.
- $\mathbb{Z}$ is the set of integers. $\mathbb{Z}^+$ is the set of positive integers.
- $\mathbb{Q}$ is the set of rational numbers.
- $\mathbb{R}$ is the set of real numbers.
- $\mathbb{C}$ is the set of complex numbers.
- $\mathbb{U}$ is the set of all objects under consideration.
- $\emptyset$ is the empty set. *Note:* $\emptyset \neq \{\emptyset\}$
- $P(S)$ is the power set of S, which is the set of all subsets of S.
- Disjoint sets A and B are sets that have no elements in common, i.e., $A \cap B = \emptyset$.

3.1.3 Set Operations

- **Union:** $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- **Intersection:** $A \cap B = \{x | x \in A \text{ and } x \in B\}$
- **Difference:** $A - B = \{x | x \in A \text{ and } x \notin B\}$
- **Complement:** $\overline{A} = \{x | x \in U \text{ and } x \notin A\}$
- **Cartesian product:** $A \times B = \{(a, b) | a \in A \text{ and } b \in B\}$

3.1.4 Cardinality

The **cardinality** of a set S, denoted by $|S|$, is the number of distinct elements in the set. The sets A and B have the same cardinality if there is a one-to-one correspondence between them. If there is a one-to-one function from A to B, the cardinality of A is less than or equal to the cardinality of B, denoted by $|A| \leq |B|$.

A set that is either finite or has the same cardinality as the set of positive integers $\mathbb{Z}^+$ is called **countable**. A set that is not countable is called **uncountable**.

Here are some examples of countable and uncountable sets:

- **Countable Sets:** $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{Z}^+$, the set of finite strings S over a finite alphabet A
- **Uncountable Sets:** $\mathbb{R}$, $P(\mathbb{N})$

The subset of a countable set is still countable.

To prove a set is **countable**, we can use Schröder-Bernstein theorem. If there are one-to-one functions $f : A \to B$ and $g : B \to A$, then there is a one-to-one correspondence between A and B, and $|A| = |B|$.

To prove a set is **uncountable**, we can use Cantor's diagonalization method. Assume that S is countable, then we can list all elements of S in a table. Then we can construct a new element that is not in the table, which contradicts the assumption that S is countable.

For power set, we have the following formula:

$$|P(S)| = 2^{|S|}$$

For union and intersection, we have the following formulas:

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

For Cartesian product, we have the following formula:

$$|A \times B| = |A| \times |B|$$

Note: $|\emptyset| = 0$ *and* $|\{\emptyset\}| = 1$

3.1.5 Computable vs. Uncomputable

We say a function is **computable** if there is an algorithm that can compute the function's value for any input in a finite amount of time.

There are functions that are not computable. This is true because the set of computer programs is countable, while the set of functions from $\mathbb{N}$ to the set $\{0, 1, 2, ..., 9\}$ is uncountable. (Cantor's diagonalization method)

3.1.6 Cantor's Theorem

For any set S, $|S| < |P(S)|$.

This is obviously true for finite sets, because $|P(S)| = 2^{|S|}$. (Note that $|\emptyset| = 0, |P(\emptyset)| = 1$)

For infinite sets, we can prove this by contradiction. Assume that $|S| = |P(S)|$, then there is a one-to-one correspondence between S and $P(S)$. Let f be a function from S to $P(S)$, then we can construct a set $T = \{s \in S | x \notin f(s)\}$. Since f is a one-to-one correspondence, there must be an element $s_0 \in S$ such that $f(s_0) = T$. However, $s_0 \in T$ implies $s_0 \notin T$, which is a contradiction.

To build a one-to-one function from $P(S)$ to S is trivial, because we can just map each subset of S to its smallest element.

Hence, we know that $|S| \neq |P(S)|$ and $|S| \leq |P(S)|$. Therefore, $|S| < |P(S)|$.

3.1.7 Set Identities

- **Identity laws**

$$A \cup \emptyset = A$$
$$A \cap U = A$$

- **Domination laws**

$$A \cup U = U$$
$$A \cap \emptyset = \emptyset$$

- **Idempotent laws**

$$A \cup A = A$$
$$A \cap A = A$$

- **Complementation laws**

$$\overline{\overline{A}} = A$$

- **Commutative laws**

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

- **Associative laws**

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

- **Distributive laws**

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

- **De Morgan's laws**

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

- **Absorption laws**

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

- **Complement laws**

$$A \cup \overline{A} = U$$
$$A \cap \overline{A} = \emptyset$$

3.2 Tuples

An **ordered n-tuple** is a sequence of n elements, where n is a positive integer.

3.3 Functions

3.3.1 Definitions

Let A and B be sets. A **function** f from A to B, denoted by $f : A \to B$, is an assignment of exactly one element of B to each element of A.

We represent a function by a formula or explicitly state the assignments between elements of A and B. For example, let $A = \{1, 2, 3\}$ and $B = \{a, b, c, d\}$, then $f : A \to B$ can be represented by $1 \mapsto a, 2 \mapsto b, 3 \mapsto c$.

Let $f : A \to B$ be a function. We say that A is the **domain** of f and B is the **codomain** of f. If $f(a) = b$, b is the **image** of a under f, and a is a **preimage** of b. The **range of** f is the set of all images of elements of A, denoted by $f(A)$.

3.3.2 Injective, Surjective, and Bijective Functions

A function $f : A \to B$ is **injective (one-to-one)** if and only if $f(a_1) = f(a_2)$ implies $a_1 = a_2$ for all $a_1, a_2 \in A$. A function $f : A \to B$ is **surjective (onto)** if and only if $f(A) = B$. A function $f : A \to B$ is **bijective (one-to-one correspond)** if and only if f is both injective and surjective.

The inverse of a function, denoted by $f^{-1}(x)$, is only defined for bijective functions.

3.3.3 Sequences

A **sequence** is a function from a subset of integers (typically $\{0, 1, 2, ...\}$ or $\{1, 2, 3, ...\}$) to a set S. We use the notation a_n to denote the image of n under the function, and we call a_n the nth term of the sequence.

4 Complexity of Algorithms

4.1 Big-O Notation

We say that $f(n) = O(g(n))$ (reads as "$f(n)$ is big-O of $g(n)$") if there are positive constants c and n_0 such that $|f(n)| \leq |c \cdot g(n)|$ for all $n \geq n_0$.

Important big-O estimation:

- $n! = O(n^n)$
- $\log n! = O(n \log n)$ (Actually we can prove $\log n! < n \log n < 2 \log n!$)
- $\log_a n = O(n)$ for any $a \geq 2$
- $n^k = O(a^n)$ for any k and $a > 1$

Similarly, we can define big-Ω and big-Θ.

4.2 Algorithms

An **algorithm** is a finite set of precise instructions for performing a computation or for solving a problem. A **computational problem** is a specification of the desired input-output relationship. An **instance** of a computational problem is a specific input. A **correct algorithm** halts with the correct output for every instance of the problem, and we can say that the algorithm **solves** the problem.

4.2.1 Time and Space Complexity

The **time complexity** of an algorithm is the number of machine operations it performs on an instance. The **space complexity** of an algorithm is the number of cells of memory it uses on an instance.

4.2.2 Input Size

The input size is the number of bits needed to represent the input. For an integer n, the input size actually is $\lceil \log_2(n+1) \rceil$ (or just $\log_2 n$). Therefore, an algorithm that runs in $\Theta(n)$ seems to be linear and efficient, but it is actually exponential ($\Theta(n) = \Theta(2^{size(n)})$).

We say two positive functions $f(n)$ and $g(n)$ are of the same type if and only If

$$c_1 g(n^{a_1})^{b_1} \leq f(n) \leq c_2 g(n^{a_2})^{b_2}$$

for all large n and some positive constants $c_1, c_2, a_1, a_2, b_1, b_2$.

Therefore, all polynomial functions are of the same type, and all exponential functions are of the same type. But polynomial functions are not of the same type as exponential functions.

4.2.3 Decision Problems and Optimization Problems

A **decision problem** is a question that has two possible answers: yes or no. An **optimization problem** is a question that requires an answer that is an optimal configuration.

If L is a decision problem and x is the input, we often write $x \in L$ to mean that the answer to the decision problem is yes, and $x \notin L$ to mean that the answer is no.

An optimization problem usually has a corresponding decision problem.

4.2.4 Complexity Classes

We divide the set of all decision problems into three complexity classes.

A problem is **solvable (tractable)** in **polynomial time** if there is an algorithm that solves the problem and the number of steps required by the algorithm on any instance of size n is $O(n^k)$ for some constant k.

The class P is the set of all decision problems that are solvable in polynomial time. The class NP is the set of all decision problems for which there exists a certificate for each yes-input that can be verified in polynomial time.

4.2.5 NP Completeness

Reduction is a relationship between two problems. We say Q can be reduced to Q' if every instance of Q can be transformed into an instance of Q' such that the answer to the transformed instance is yes if and only if the answer to the original instance is yes.

A polynomial-time reduction from Q to Q' is a reduction that can be performed in polynomial time, denoted by $Q \leq_p Q'$. Intuitively, this means Q is no harder than Q'.

A problem Q is **NP-complete** if and only if

- $Q \in NP$
- Every problem $L \in NP$, $L \leq_p Q$

Therefore, if we can find a polynomial-time algorithm for an NP-complete problem, then we can solve all NP problems in polynomial time.

5 Number Theory

5.1 Divisibility and Modular Arithmetic

5.1.1 Divisibility

Let a, b, c be integers. Then the following holds:

- If $a|b$ and $a|c$, then $a|(b+c)$ and $a|(b-c)$.
- If $a|b$, then $a|bc$.
- If $a|b$ and $b|c$, then $a|c$.
- Corollary: If a, b, c are integers, where $a \neq 0$, such that $a|b$ and $a|c$, then $a|(mb+nc)$ for any integers m and n.

5.1.2 Modular Arithmetic

Let a, b, n be integers, where $n > 0$. We say that a is **congruent to** b **modulo** n, denoted by $a \equiv b \pmod{n}$, if and only if $n|(a-b)$. This is called **congruence** and n is called the **modulus**.

The following properties hold:

- If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $a + c \equiv b + d \pmod{n}$.
- If $a \equiv b \pmod{n}$ and $c \equiv d \pmod{n}$, then $ac \equiv bd \pmod{n}$.
- Corollary: $(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$, and $(a \cdot b) \bmod m = ((a \bmod m) \cdot (b \bmod m)) \bmod m$.

5.2 Prime

An integer $p > 1$ is **prime** if and only if its only positive divisors are 1 and p. An integer $n > 1$ is **composite** if and only if it is not prime.

The **fundamental theorem of arithmetic** states that every integer greater than 1 is either prime or can be written as a unique product of prime numbers.

GCD of two integers a and b, denoted by $\gcd(a, b)$, is the largest integer that divides both a and b. If we factorize a and b into prime numbers, then $\gcd(a, b) = p_1^{\min(e_1, f_1)} \cdot p_2^{\min(e_2, f_2)} \cdot \ldots \cdot p_k^{\min(e_k, f_k)}$, where e_i and f_i are the exponents of p_i in the factorization of a and b. Two integers are **relatively prime** if and only if their GCD is 1.

Similarly, we can define **LCM** of two integers a and b, denoted by $\text{lcm}(a, b)$, is the smallest positive integer that is divisible by both a and b.

5.3 Eculidean Algorithm

The **Eculidean algorithm** is an efficient method for computing the GCD of two integers. It can solve the problem in $O(\log n)$ time, where n is the smaller of the two integers.

The central idea is that if $a = bq + r$, then $\gcd(a, b) = \gcd(b, r)$. Here is the proof: If d is a common divisor of a and b, we can write $d \mid a$ and $d \mid b$. By the corollary, we know that $d \mid (a - bq)$, which implies $d \mid r$. Therefore, d is a common divisor of b and r.

For example, to compute $\gcd(287, 91)$:

$$\begin{aligned} 287 \bmod 91 &= 14 \\ 91 \bmod 14 &= 7 \\ 14 \bmod 7 &= 0 \end{aligned}$$

Therefore, $\gcd(287, 91) = 7$.

5.4 Bezout's Theorem

Let a and b be integers, not both zero. Then there exist integers x and y such that $\gcd(a, b) = ax + by$.

We can use the extended Eculidean algorithm to find x and y.

For example, since $\gcd(503, 286) = 1$, we can find x and y such that $503x + 286y = 1$.

$$\begin{aligned} 503 &= 1 \cdot 286 + 217 \\ 286 &= 1 \cdot 217 + 69 \\ 217 &= 3 \cdot 69 + 10 \\ 69 &= 6 \cdot 10 + 9 \\ 10 &= 1 \cdot 9 + 1 \\ 9 &= 9 \cdot 1 + 0 \end{aligned}$$

Eculidean algorithm

$$\begin{aligned} 1 &= 10 - 1 \cdot 9 \\ &= 10 - 1 \cdot (69 - 6 \cdot 10) = 7 \cdot 10 - 1 \cdot 69 \\ &= 7 \cdot (217 - 3 \cdot 69) - 1 \cdot 69 = 7 \cdot 217 - 22 \cdot 69 \\ &= 7 \cdot 217 - 22 \cdot (286 - 1 \cdot 217) = 29 \cdot 217 - 22 \cdot 286 \\ &= 29 \cdot (503 - 1 \cdot 286) - 22 \cdot 286 = 29 \cdot 503 - 51 \cdot 286 \end{aligned}$$

Extended Eculidean algorithm

Therefore, $x = 29$ and $y = -51$.

The corollaries of Bezout's theorem are:

- If $1 = \gcd(a, b)$ and $a|bc$, then $a|c$.
- If p is a prime and $p|a_1a_2...a_n$, then $p|a_i$ for some i.
- If $ac \equiv bc \pmod{n}$ and $\gcd(c, n) = 1$, then $a \equiv b \pmod{n}$.

The proof of the first corollary is as follows: Since $\gcd(a, b) = 1$, we can find x and y such that $ax + by = 1$. Then $c = cax + cby$. Since $a|bc$, we know that $a|cby$. Therefore, $a|(cax + cby) = c$.

5.5 Linear Congruence

A **linear congruence** is an equation of the form $ax \equiv b \pmod{n}$, where a, b, n are integers and $n > 0$.

5.5.1 Modular Inverse

Let a and n be integers, where $n > 0$. If there exists an integer $\overline{a}$ such that $a \cdot \overline{a} \equiv 1 \pmod{n}$, then $\overline{a}$ is called the **modular inverse** of a modulo n.

If a and n are relatively prime, then a has a modular inverse modulo n. Furthermore, the modular inverse of a modulo n is unique modulo n.

We can use the extended Eculidean algorithm to find the modular inverse of a modulo n.

5.5.2 Chinese Remainder Theorem

Let $n_1, n_2, ..., n_k$ be positive integers that are pairwise relatively prime, and let $a_1, a_2, ..., a_k$ be any integers. Then the system of linear congruences:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a solution, and any two solutions are congruent modulo $n_1 n_2 ... n_k$.

Let $n = n_1 n_2 ... n_k$. Then the solution is $x = a_1 y_1 \frac{n}{n_1} + a_2 y_2 \frac{n}{n_2} + ... + a_k y_k \frac{n}{n_k}$, where y_i is the modular inverse of $\frac{n}{n_i}$ modulo n_i.

For example, to solve the system of linear congruences:

$$\begin{aligned} x &\equiv 2 \pmod{3} \\ x &\equiv 3 \pmod{5} \\ x &\equiv 2 \pmod{7} \end{aligned}$$

We can find $y_1 = 2$, $y_2 = 1$, and $y_3 = 1$. Therefore, $x = 2 \cdot 35 \cdot 2 + 3 \cdot 21 \cdot 1 + 2 \cdot 15 \cdot 1 = 233$ is a solution.

5.6 Fermat's Little Theorem

Let p be a prime and a be an integer such that $a \not\equiv 0 \pmod{p}$. Then $a^{p-1} \equiv 1 \pmod{p}$.

5.7 Euler's Theorem

Euler's function $\phi(n)$ is the number of positive integers less than or equal to n that are relatively prime to n. The function has the following properties:

- If p is prime, then $\phi(p) = p - 1$.
- If p is prime and $k \geq 1$, then $\phi(p^k) = p^k - p^{k-1}$.
- If m and n are relatively prime, then $\phi(mn) = \phi(m)\phi(n)$.

Euler's theorem states that if n is a positive integer and a is an integer such that a and n are relatively prime, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

It is worth noting that $\phi(n)$ might not be the smallest positive integer k such that $a^k \equiv 1 \pmod{n}$.

5.8 Primitive Roots

A **primitive root** modulo a prime p is an integer g such that every nonzero integer modulo p is congruent to a power of g modulo p.

6 Groups, Rings, and Fields

6.1 Groups

A **group** is a set G together with a binary operation $*$ on G such that the following axioms hold:

- **Closure:** For all $a, b \in G$, $a * b \in G$.
- **Associativity:** For all $a, b, c \in G$, $(a * b) * c = a * (b * c)$.
- **Identity:** There exists a **unique** element $1_e \in G$ such that for all $a \in G$, $a * 1_e = a$.
- **Inverse:** For each $a \in G$, there exists an element $a^{-1} \in G$ such that $a * a^{-1} = 1_e$.

6.1.1 Permutation Groups

A permutation group is a group whose elements are permutations of a given set S and whose operation is composition of permutations in S. Let $s_n =< 1, 2, ..., n >$ denotes a sequence of n elements, and P_n denotes the set of all permutations of s_n. Then for any two elements π and ρ, $\pi \circ \rho$ is also a permutation of s_n.

For example, $s_3 =< 1, 2, 3 >$, and $P_3 = \{< 1, 2, 3 >, < 1, 3, 2 >, < 2, 1, 3 >, < 2, 3, 1 > , < 3, 1, 2 >, < 3, 2, 1 >\}$. If $\pi =< 3, 2, 1 >$ and $\rho =< 1, 3, 2 >$, then $\pi \circ \rho =< 2, 1, 3 >$. The operation is $\pi \circ \rho[i] = \rho[\pi[i]]$.

Therefore, $(P_n, \circ)$ is a group.

6.1.2 Abeilian Groups

A group G is **abeilian** if and only if for all $a, b \in G$, $a * b = b * a$.

6.2 Rings

If $(R, +)$ is an abeilian group, we define a second binary operation $*$ on R such that the following axioms hold:

- **Closure:** For all $a, b \in R$, $a * b \in R$.
- **Associativity:** For all $a, b, c \in R$, $(a * b) * c = a * (b * c)$.
- **Distributivity:** For all $a, b, c \in R$, $a*(b+c) = a*b+a*c$ and $(a+b)*c = a*c+b*c$.

6.2.1 Commutative Ring

A ring R is **commutative** if and only if for all $a, b \in R$, $a * b = b * a$.

6.2.2 Integral Domain

A commutative ring R is an **integral domain** if the following axiom holds:

- **Identity:** There exists a **unique** element $1_m \in R$ such that for all $a \in R$, $a * 1_m = 1_m * a = a$.
- **Nonzero product:** For all $a, b \in R$, if $a * b = 0$, then $a = 0$ or $b = 0$.

6.3 Fields

A commutative ring F is a **field** if the following axiom holds:

- **Inverse:** For each $a \in F$, there exists an element $a^{-1} \in F$ such that $a * a^{-1} = a^{-1} * a = 1_m$.

6.4 Other Facts

- Z_m, the set of integers modulo m, is a commutative ring.
- $(GL(n), \cdot)$ is a group but not an abeilian group. (The set of all invertible $n \times n$ matrices)
- $(\mathbb{M}_{n\times n}, +, \cdot)$ is a ring but not a commutative ring.
- $(Z_m, +_m, \cdot_m)$ is a commutative ring but not an integral domain.
- $(\mathbb{Z}, +, \cdot)$ is an integral domain but not a field.

7 Cryptography

7.1 Public Key Cryptography

7.1.1 RSA Cryptosystem

The RSA public key cryptosystem works as follows:

1. Choose two large primes p and q.
2. Compute $n = pq$ and $\phi(n) = (p-1)(q-1)$.
3. Choose e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
4. Compute d such that $ed \equiv 1 \pmod{\phi(n)}$.
5. Publish the public key (n, e) and keep the private key d secret.

To encrypt a message m, compute $c \equiv m^e \pmod{n}$. To decrypt a ciphertext c, compute $m \equiv c^d \pmod{n}$.

The correctness of the algorithm is as follows:

$$\begin{aligned} c^d &\equiv (m^e)^d && \pmod{n} \\ &\equiv m^{ed} && \pmod{n} \\ &\equiv m^{k\phi(n)+1} && \pmod{n} \\ &\equiv m \cdot (m^{\phi(n)})^k && \pmod{n} \\ &\equiv m \cdot 1^k && \pmod{n} \\ &\equiv m && \pmod{n} \end{aligned}$$

To leave a RSA signature, compute $s \equiv m^d \pmod{n}$. To verify a RSA signature, compute $m \equiv s^e \pmod{n}$.(d and e are interchangeable)

7.1.2 El Gamal Cryptosystem

The El Gamal public key cryptosystem works as follows:

1. Choose a large prime p and a primitive root g modulo p.
2. Choose x such that $1 < x < p - 2$.
3. Compute $y \equiv g^x \pmod{p}$.
4. Publish the public key (p, g, y) and keep the private key x secret.

To encrypt a message m, choose k such that $1 < k < p - 1$ and $\gcd(k, p - 1) = 1$, then compute $c_1 \equiv g^k \pmod{p}$ and $c_2 \equiv m \cdot y^k \pmod{p}$. To decrypt a ciphertext (c_1, c_2), compute $m \equiv c_2 \cdot (c_1^x)^{-1} \pmod{p}$.

The correctness of the algorithm is as follows:

$$\begin{aligned} c_2 \cdot (c_1^x)^{-1} &\equiv m \cdot y^k \cdot (g^{kx})^{-1} && \pmod{p} \\ &\equiv m \cdot g^{kx} \cdot g^{-kx} && \pmod{p} \\ &\equiv m && \pmod{p} \end{aligned}$$

7.2 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange works as follows:

1. Choose a large prime p and a primitive root g modulo p.
2. Alice chooses x such that $1 < x < p - 2$ and sends $y \equiv g^x \pmod{p}$ to Bob.
3. Bob chooses z such that $1 < z < p - 2$ and sends $w \equiv g^z \pmod{p}$ to Alice.
4. Alice computes $w^x \equiv (g^z)^x \equiv g^{zx} \pmod{p}$.
5. Bob computes $y^z \equiv (g^x)^z \equiv g^{xz} \pmod{p}$.
6. Now Alice and Bob share the secret key $g^{xz} \pmod{p}$.

8 Mathematical Induction

8.1 Proof by Smallest Counterexample

To prove a statement $P(n)$ for all $n \geq n_0$, we can use proof by smallest counterexample. We assume that $P(n)$ is false for some $n > 0$. Then there must be a smallest integer m such that $P(m)$ is false. Since $P(n_0)$ is true, $m > n_0$. Then we use the fact that $P(m')$ the for all $0 \leq m' < m$ is true to show that $P(m)$ is true, which is a contradiction.

8.2 Direct Proof

8.2.1 Weak Principle of Mathematical Induction

If the statement $P(b)$ is true, and the statement $P(n-1) \to P(n)$ is true for all integers $n > b$, then the statement $P(n)$ is true for all integers $n \geq b$.

We call $P(b)$ the **basis step (inductive hypothesis)** and $P(n-1) \to P(n)$ the **inductive step (inductive conclusion)**.

8.2.2 Strong Principle of Mathematical Induction

If the statement $P(b)$ is true, and the statement $P(b) \wedge P(b+1) \wedge \cdots \wedge P(n-1) \to P(n)$ is true for all integers $n > b$, then the statement $P(n)$ is true for all integers $n \geq b$.

We can see that the weak form is a special case of the strong form, and the strong form can be derived from the weak form.