Introduction to Number Theory and Cryptography

Chapter 4, *Number Theory and Cryptography*, from "Discrete Mathematics and Its Applications" covers the following key points:

1. **Divisibility and Modular Arithmetic**: Introduces concepts like divisibility, with modular arithmetic playing a crucial role in computer science applications. It discusses remainders when dividing integers and modular arithmetic's importance in encryption and generating pseudo-random numbers.
2. **Integer Representations and Algorithms**: Highlights how integers can be represented in various bases (binary, octal, hexadecimal) and explores algorithms for these representations.
3. **Prime Numbers and GCD**: Discusses the fundamental theorem of arithmetic, properties of primes, and algorithms for computing the greatest common divisor (Euclidean algorithm).
4. **Solving Congruences**: Covers methods for solving linear congruences and applying them to modular arithmetic problems.
5. **Applications of Congruences**: Describes how congruences are used in pseudorandom number generation, memory allocation, and error detection in identification numbers.
6. **Cryptography**: Introduces classical and modern cryptography, including private and public key cryptosystems like RSA, key sharing protocols, and signature verification. RSA's strength is discussed along with modular exponentiation and key management protocols for secure communication.

Divisibility and Modular Arithmetic

Divisibility and Division Algorithm

The ideas that we will develop in this section are based on the notion of divisibility. Division of an integer by a positive integer produces a quotient and a remainder.

Divisibility

- For integers a and b with $a \neq 0$, we say a divides b (written $a \mid b$) if there exists an integer c such that $b = ac$.
- If $a \mid b$, then a is a **divisor** of b, and b is a **multiple** of a.
- Properties:
  - If $a \mid b$ and $a \mid c$, then $a \mid (b + c)$.
  - If $a \mid b$, then $a \mid bc$ for any integer c.
  - If $a \mid b$ and $b \mid c$, then $a \mid c$.

Division Algorithm

- For any integer a and positive integer d, there exist unique integers q and r such that:

$$a = dq + r, \quad \text{where} \quad 0 \leq r < d$$

- Here, $q = a \text{ div } d$ and $r = a \bmod d$
- Example: 101 = 11×9 + 2 ⇒ 101 div 11 = 9, 101 mod 11 = 2

Modular Arithmetic and Congruences

Modular Arithmetic

The **modulus** *m* is a fixed positive integer. Two integers *a* and *b* are **congruent modulo m** (written $a \equiv b \pmod{m}$) if *m* divides $(a - b)$.

Key Theorems

- $a \equiv b \pmod{m} \Leftrightarrow a \bmod m = b \bmod m$
- If $a \equiv b \pmod{m}$ and $c \equiv d \pmod{m}$, then:
  - $a + c \equiv b + d \pmod{m}$
  - $ac \equiv bd \pmod{m}$

Arithmetic in $\mathbb{Z}_m$

For elements in $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$:

- Addition: $a +_m b = (a + b) \bmod m$
- Multiplication: $a \square_m b = (a \square b) \bmod m$

Example

In $\mathbb{Z}_{11}$:

$7 +_{11} 9 = 16 \bmod 11 = 5$

$7 \square_{11} 9 = 63 \bmod 11 = 8$

Integer Representations and Algorithms

Integer Representations in Different Bases

Base b Representation

Any integer *n* can be represented in base *b* (where *b* > 1) as:

$$n = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0, \quad \text{where} \quad 0 \leq a_i < b \quad \text{and} \quad a_k \neq 0$$

Common Bases:

- **Binary (base 2)**: Used by computers, digits are 0 and 1.
- **Octal (base 8)**: Groups of 3 binary digits.
- **Hexadecimal (base 16)**: Uses digits 0–9 and letters A–F for 10–15; corresponds to 4-bit binary blocks.

Examples

- $(101011111)_2 = 351_{10}$
- $(7016)_8 = 3598_{10}$
- $(2AE0B)_{16} = 175627_{10}$

Algorithm 1: Base b Expansion

procedure *base b expansion*(n, b: positive integers with $b > 1$)
$q := n$
$k := 0$
while $q \neq 0$
    $a_k := q$ **mod** b
    $q := q$ **div** b
    $k := k + 1$
return $(a_{k-1}, \ldots, a_1, a_0)$ $\{(a_{k-1} \ldots a_1 a_0)_b$ is the base b expansion of $n\}$

Used to find the base-b digits of *n* by successive division

Arithmetic Algorithms (Addition & Multiplication)

Addition Algorithm

1. **Start from the rightmost bits** of the two binary numbers, a and b.
2. **Add the rightmost bits**:
   - Compute: $\mathbf{a_0 + b_0 = 2 \times c_0 + s_0}$
   - Here, s_0 is the rightmost bit of the sum, and c_0 is the carry (0 or 1).
3. **Move to the next pair of bits** (moving left), and **add along with the previous carry**:
   - Compute: $\mathbf{a_1 + b_1 + c_0 = 2 \times c_1 + s_1}$
   - $\mathbf{s_1}$ is the next bit of the sum, and $\mathbf{c_1}$ is the new carry.
4. **Repeat this process** for each bit pair, always adding the two bits and the current carry.
5. **At the last bit (leftmost bits)**:
   - Add: $\mathbf{a_{n-1} + b_{n-1} + c_{n-2} = 2 \times c_{n-1} + s_{n-1}}$
   - Here, $\mathbf{c_{n-1}}$ becomes the final carry.
6. **The final sum** is formed by placing the final carry $\mathbf{s_n = c_{n-1}}$ at the front, followed by the bits $\mathbf{s_{n-1}, s_{n-2}, \ldots, s_1, s_0}$.

Multiplication Algorithm

Multiply using:

$$ab = a(b_0 2^0 + b_1 2^1 + \cdots + b_{n-1} 2^{n-1}) = a(b_0 2^0) + a(b_1 2^1) + \cdots + a(b_{n-1} 2^{n-1})$$

Each term $a \cdot b_j 2^j$ is calculated by bit shift.

Time complexity: $O(n^2)$ shifts and bit additions

Division, Modular Arithmetic & Efficient Exponentiation

Division Algorithm

1. **Initialize:**
   - Set $q = 0$ (quotient)
   - Set $r = |a|$ (absolute value of a)
2. **Repeat subtraction:**
   - While $r \geq d$:
     - Subtract d from r (i.e., $r := r - d$)
     - Increase q by 1 (i.e., $q := q + 1$)
3. **Adjust for negative a:**
   - If $a < 0$ **and** $r > 0$:
     - Set $r := d - r$
     - Set $q := -(q + 1)$
4. **Return result:**
   - Return the pair (q, r), where:
     - $q = a \text{ div } d$ (the quotient)
     - $r = a \bmod d$ (the remainder)

Modular Exponentiation

Efficiently computes $b^n \bmod m$:

- Uses successive squaring and reduction.
- Time: $O((\log m)^2 \log n)$ bit operations

Primes and Greatest Common Divisors

Introduction to Primes and the Fundamental Theorem

Definition of Prime:

- An integer $p > 1$ is **prime** if its only positive divisors are 1 and p.
- A **composite** number has more than two positive divisors.

Fundamental Theorem of Arithmetic:

- Every integer > 1 is either a prime or a **unique product of primes** (up to order).

Examples:

- $100 = 2^2 \cdot 5^2$
- 641 = prime
- $999 = 3^3 \cdot 37$
- $1024 = 2^{10}$

Testing for Primes and Trial Division

Trial Division Theorem:

- If *n* is composite, it has a **prime factor ≤ √n**.

Trial Division Algorithm:

1. Check divisibility by all primes ≤ √n.
2. If *n* not divisible by any of them → *n* is **prime**.

Example:

- 101 is not divisible by 2, 3, 5, or 7 → **prime**

Greatest Common Divisor (gcd)

Definition:

- *gcd(a, b)* is the **largest integer** that divides both *a* and *b*.

Properties:

- *gcd(24, 36) = 12*
- If *gcd(a, b) = 1*, then *a* and *b* are **relatively prime**.
- A set is **pairwise relatively prime** if every pair in it is relatively prime.

Using Prime Factorization:

- If:
  $a = 2^3 \cdot 3 \cdot 5$, $b = 2^2 \cdot 5^3$
  → $gcd(a, b) = 2^2 \cdot 5 = 20$

Euclidean Algorithm

Goal: Efficiently compute gcd(a, b) using repeated division.

Steps (Example: gcd(287, 91)):

1. $287 = 91 \cdot 3 + 14$
2. $91 = 14 \cdot 6 + 7$
3. $14 = 7 \cdot 2 + 0 \rightarrow \gcd = 7$

General Rule:

If $a = bq + r$, then $gcd(a, b) = gcd(b, r)$

Bézout's Identity and Extended Euclidean Algorithm

Bézout's Theorem: There exist integers *s*, *t* such that:

$$\gcd(a, b) = sa + tb$$

Example:

- *gcd(252, 198) = 18*
- *18 = 4·252 – 5·198*

Extended Euclidean Algorithm: Tracks *s*, *t* values during gcd steps to solve linear Diophantine equations and find modular inverses

Solving Congruences

Linear Congruences:

- A congruence of the form **ax ≡ b (mod m)** is called a *linear congruence*.
- To solve:
  - Find the inverse of a modulo m, say a^{-1}, such that $\mathbf{a \cdot a^{-1} \equiv 1 \pmod{m}}$.
  - Multiply both sides: $\mathbf{x \equiv a^{-1} \cdot b \pmod{m}}$.
- If **gcd(a, m) = d > 1**, solutions exist **only if d ∣ b**, and there will be d distinct solutions modulo m.

Chinese Remainder Theorem (CRT):

Solves simultaneous systems:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_n \pmod{m_n} \end{aligned}$$

Where $m_1, m_2, \ldots, m_n$ are pairwise relatively prime.

Construct:

- Let $M = m_1 \cdot m_2 \cdot \ldots \cdot m_n$
- Let $M_i = M/m_i$, and find inverse y_i of $M_i \bmod m_i$
- Solution: $\mathbf{x \equiv \Sigma\, a_i \cdot M_i \cdot y_i \pmod{M}}$

Finding Inverses (Extended Euclidean Algorithm):

Use the Euclidean algorithm to compute **gcd(a, m)**.

Back-substitute to find integers s, t such that **sa + tm = 1** → s is the inverse of a mod m.

Example: To solve **3x ≡ 4 (mod 7)**:

- Inverse of 3 mod 7 is -2 ≡ 5 (since -2·3 ≡ 1 mod 7),
- Multiply: **x ≡ -2·4 ≡ -8 ≡ 6 (mod 7)**

Applications of Congruences

1 Hashing Functions

- Used to assign memory locations efficiently in databases.
- A common method: **h(k) = k mod m**, where *k* is a key (e.g., a Social Security number) and *m* is the number of memory slots.
- Handles **collisions** using **linear probing**: h(k, i) = (h(k) + i) mod m, for *i = 0, 1, 2, ...*Mission

Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim

2 Pseudorandom Number Generation

- Uses **linear congruential generators (LCG)**: $x_{n+1} = (a \cdot x_n + c) \bmod m$
  where:
  - *a* = multiplier, *c* = increment, *m* = modulus, x_0 = seed
- Example:
  For m = 9, a = 7, c = 4, x_0 = 3, the sequence is generated as:
  $x_1 = (7 \cdot 3 + 4) \bmod 9 = 25 \bmod 9 = 7$
  $x_2 = (7 \cdot 7 + 4) \bmod 9 = 53 \bmod 9 = 8$, and so on

3 Check Digits

- Used to detect errors in ID numbers (e.g., barcodes, ISBNs).
- A check digit is calculated using a modular formula.
- Example: ISBN-10 uses weighted sum mod 11: $(d_1 + 2d_2 + 3d_3 + \ldots + 10d_{10}) \bmod 11 \equiv 0$