基础数论讲义

本讲义主要介绍以下内容：

- 整除、同余、素数、唯一分解定理等概念
- GCD 及相关知识：辗转相除法、裴蜀定理、扩展欧几里得算法（exgcd）
- 线性筛
- 乘法逆元及其求解方法（扩展欧几里得算法和快速幂）

---

1. 整除、同余、素数、唯一分解定理

1.1 整除

- **定义**：
  对于两个整数 a 和 b（$b \neq 0$），如果存在整数 k 使得
  $$a = b \times k, \tag{1}$$
  则称 b **整除** a，记作 $b \mid a$。
- **性质**：
  - 如果 $b \mid a$，则 a 为 b 的整数倍。
  - 整除关系满足传递性：若 $c \mid b$ 且 $b \mid a$，则 $c \mid a$。

1.2 同余

- **定义**：
  给定模数 m（$m > 0$），如果两个整数 a 和 b 满足
  $$a - b = km \quad (k \in \mathbb{Z}), \tag{2}$$
  则称 a 与 b **模 m 同余**，记作
  $$a \equiv b \pmod{m}. \tag{3}$$
- **性质**：
  - **反身性**：$a \equiv a \pmod{m}$。
  - **对称性**：如果 $a \equiv b \pmod{m}$，则 $b \equiv a \pmod{m}$。
  - **传递性**：如果 $a \equiv b \pmod{m}$ 且 $b \equiv c \pmod{m}$，则 $a \equiv c \pmod{m}$。
  - 同余运算满足加、减、乘法的性质：
    若 $a \equiv b \pmod{m}$ 且 $c \equiv d \pmod{m}$，则
    - $a + c \equiv b + d \pmod{m}$
    - $a - c \equiv b - d \pmod{m}$

        - $ac \equiv bd \pmod{m}$

1.3 素数

- **定义**：
  大于 1 的整数 p，若其只有两个正约数，即 1 和 p 本身，则称 p 为 **素数**（或质数）。
- **注意**：
  - 1 不是素数。
  - 素数是数论的基石，所有大于 1 的整数都可以唯一分解成素数的乘积。

1.4 唯一分解定理

- **定理**：
  基本定理说明：**任何大于 1 的正整数都可以唯一地（忽略因子顺序）表示为素数的乘积。**
- **意义**：
  - 这一定理为数论提供了坚实的基础，常用于求解整除问题、最大公约数（GCD）、最小公倍数等。

---

2. GCD 及相关知识：辗转相除法、裴蜀定理与扩展欧几里得算法

2.1 GCD 的定义

- **定义**：
  对于两个整数 a 和 b（不全为零），它们的 **最大公约数**（GCD，Greatest Common Divisor）记作 $\gcd(a, b)$，即同时整除 a 和 b 的最大正整数。

2.2 辗转相除法（欧几里得算法）的原理

- **基本思想**：
  假设 a 和 b 是两个正整数，并且可以写成

  $$a = b \times q + r \quad (0 \le r < b), \tag{4}$$

  则有

  $$\gcd(a, b) = \gcd(b, r). \tag{5}$$

- **证明概要**：
  1. 设 d 是 a 和 b 的公约数，则 d 整除 a 和 b。
  2. 根据 $r = a - bq$，则 d 同时整除 r。
  3. 同理，任何整除 b 和 r 的数也整除 a。
  4. 因此，所有同时整除 a 和 b 的正整数与同时整除 b 和 r 的正整数集合相同，故 $\gcd(a, b) = \gcd(b, r)$。
- **伪代码实现**：

```
function gcd(a, b):
    while b ≠ 0:
        r = a mod b
        a = b
        b = r
    return a
```

2.3 裴蜀定理（Bezout's Identity）

- **定理**：
  对于任意整数 a 和 b，存在整数 x 和 y 使得
  $ax + by = \gcd(a, b).$
- **意义**：
  - 裴蜀定理证明了最大公约数可以表示为 a 与 b 的线性组合。
  - 这一结果为求解线性不定方程、计算乘法逆元等提供了理论支持。

2.4 扩展欧几里得算法（exgcd）

扩展欧几里得算法不仅能求 $\gcd(a, b)$，还能同时求得一组整数 x 和 y，使得：

$ax + by = \gcd(a, b).$

算法思路

1. **递归终止条件**：
   当 $b = 0$ 时，有
   $a \times 1 + 0 \times 0 = a,$
   所以返回 $(\gcd(a, 0) = a, ; x = 1, ; y = 0)$。
2. **递归关系**：
   若 $a = b \times q + r$（其中 $r = a \mod b$），则
   $\gcd(a, b) = \gcd(b, r).$
   假设递归调用返回一组解 (x_1, y_1)，使得
   $bx_1 + ry_1 = \gcd(b, r),$
   则将 $r = a - bq$ 代入可得：
   $bx_1 + (a - bq)y_1 = ay_1 + b(x_1 - qy_1) = \gcd(a, b).$
   因此，可令：
   $x = y_1, \quad y = x_1 - q \times y_1.$

C++ 实现

```cpp
#include <tuple>

// 扩展欧几里得算法：返回 (g, x, y)，满足
// x * a + y * b = g = gcd(a, b)
std::tuple<long long, long long, long long> extendedGCD(long long a, long long b) {
    if (b == 0)
        return {a, 1, 0};
    auto [g, x1, y1] = extendedGCD(b, a % b);
    long long x = y1;
    long long y = x1 - (a / b) * y1;
    return {g, x, y};
}
```

使用示例 —— 解二元一次不定方程

在数论中，一个典型的二元一次不定方程形式为

$$ax + by = c, \tag{6}$$

其中 a, b, c 为已知整数，要求解整数解 x 和 y。利用扩展欧几里得算法（exgcd），我们可以求出一组特解，并给出该方程的一般解。

解题思路

1. **求** $\gcd(a, b)$
   使用扩展欧几里得算法求出
   $$d = \gcd(a, b) \tag{7}$$
   以及一组系数 (x_0, y_0) 使得
   $$ax_0 + by_0 = d. \tag{8}$$
2. **判定方程是否有解**
   若 c 不能被 d 整除，则方程无整数解；否则有解。
3. **构造特解**
   将上面的等式乘以 $\frac{c}{d}$ 得到
   $$a\left(x_0 \cdot \frac{c}{d}\right) + b\left(y_0 \cdot \frac{c}{d}\right) = c. \tag{9}$$
   因此，特解可以写为
   $$x = x_0 \cdot \frac{c}{d}, \quad y = y_0 \cdot \frac{c}{d}. \tag{10}$$
4. **构造一般解**
   对于任意整数 t，一般解为
   $$x = x_0 \cdot \frac{c}{d} + \frac{b}{d}t, \quad y = y_0 \cdot \frac{c}{d} - \frac{a}{d}t. \tag{11}$$

C++ 模板代码

下面给出一个封装性较好的 C++ 模板，用于利用扩展欧几里得算法求解二元一次不定方程：

```
#include <tuple>
#include <iostream>
using namespace std;

// 扩展欧几里得算法：返回 (g, x, y)，满足
// a * x + b * y = g = gcd(a, b)
std::tuple<long long, long long, long long> extendedGCD(long long a, long long b) {
    if (b == 0)
        return {a, 1, 0};
    auto [g, x1, y1] = extendedGCD(b, a % b);
    long long x = y1;
    long long y = x1 - (a / b) * y1;
    return {g, x, y};
}

// 求解二元一次不定方程: a * x + b * y = c
// 若无解返回 false，否则返回 true，并将一组特解存入 x, y，同时 d 为 gcd(a, b)
bool solveDiophantine(long long a, long long b, long long c, long long &x, long long &y,
long long &d) {
    auto [g, x0, y0] = extendedGCD(a, b);
    d = g;
    // 无解条件：c 必须能被 gcd(a, b) 整除
    if (c % d != 0)
        return false;
    // 特解
    x = x0 * (c / d);
    y = y0 * (c / d);
    return true;
}

int main() {
    long long a, b, c;
    cout << "请输入方程 a*x + b*y = c 中的 a, b, c: ";
    cin >> a >> b >> c;

    long long x, y, d;
    if (!solveDiophantine(a, b, c, x, y, d)) {
        cout << "方程无整数解。" << endl;
    } else {
        cout << "方程有解，特解为:" << endl;
        cout << "x = " << x << ", y = " << y << endl;
        cout << "一般解为:" << endl;
        cout << "x = " << x << " + " << (b / d) << " * t" << endl;
        cout << "y = " << y << " - " << (a / d) << " * t" << endl;
    }
    return 0;
}
```

使用说明

- **输入**：用户输入整数 a, b, c。
- **输出**：若方程无解，程序输出提示；否则输出一组特解及一般解的表达形式。
- **参数说明**：
  - 特解 (x, y) 满足 $ax + by = c$；
  - 一般解中，$t \in \mathbb{Z}$ 为任意整数；
  - 注意：当 c 时，无整数解。

使用示例 —— 求乘法逆元

假设需要计算整数 a 在模 m 下的乘法逆元，即求 x 使得

$ax \equiv 1 \pmod{m}$,

前提条件是 $\gcd(a, m) = 1$。利用扩展欧几里得算法可求得如下：

```
// 计算 a 关于模 m 的乘法逆元，若不存在则返回 -1
long long modInverse(long long a, long long m) {
    auto [g, x, y] = extendedGCD(a, m);
    if (g != 1) {
        // a 与 m 不互质，逆元不存在
        return -1;
    }
    // 保证 x 为正
    x %= m;
    if (x < 0)
        x += m;
    return x;
}
```

3. 线性筛

3.1 概述

- **目标**：
  在 $O(n)$ 的时间复杂度内筛出区间 $[2, n]$ 内的所有素数，并记录每个合数的最小质因数。
- **对比埃拉托斯特尼筛法**：
  - 埃拉托斯特尼筛法的时间复杂度为 $O(n \log \log n)$。
  - 线性筛确保每个合数只被标记一次，严格达到 $O(n)$ 的时间复杂度。
- **基本思路**：
  1. 使用一个布尔数组标记每个数是否为合数（非素数）。
  2. 利用一个数组保存筛出的素数。
  3. 对于每个整数 i：

- 若 i 未被标记，则 i 为素数，加入素数列表。
- 遍历已知素数列表，对于每个素数 p：
  - 若 $i \times p \leq n$，则将 $i \times p$ 标记为合数。
  - 如果 i 能被 p 整除，则停止该次遍历（保证每个合数只被标记一次）。

3.2 C++ 封装模板

```cpp
#include <vector>
using namespace std;

class LinearSieve {
public:
    // 筛选出的素数列表
    vector<int> primes;
    // 标记数组：isComposite[i] 为 true 表示 i 是合数（非素数）
    vector<bool> isComposite;

    // 构造函数：筛选 2 到 n 范围内的所有素数
    LinearSieve(int n) {
        isComposite.assign(n + 1, false);
        // 从 2 开始筛
        for (int i = 2; i <= n; i++) {
            if (!isComposite[i]) {
                primes.push_back(i);
            }
            for (int p : primes) {
                if (i * p > n)
                    break;
                isComposite[i * p] = true;
                if (i % p == 0)
                    break;
            }
        }
    }
};
```

---

4. 乘法逆元

4.1 概念简介

- **定义**：
  给定模 m 和整数 a（满足 $\gcd(a, m) = 1$），称整数 x 为 a **关于模 m 的乘法逆元**，如果满足 $a \times x \equiv 1 \pmod{m}$.
- **存在条件**：
  乘法逆元存在的充要条件是 a 与 m 互质，即 $\gcd(a, m) = 1$。

4.2 计算方法

- **扩展欧几里得算法**：
  利用扩展欧几里得算法求得整数 x 和 y 满足
  $$ax + my = \gcd(a, m) = 1,$$
  其中 x 就是 a 在模 m 下的乘法逆元（经过适当取模处理）。
- **快速幂**：
  当 m 为质数时，可利用费马小定理计算逆元。根据费马小定理，
  $$a^{m-1} \equiv 1 \pmod{m},$$
  从而有
  $$a^{m-2} \equiv a^{-1} \pmod{m}.$$
  因此，我们可以利用快速幂算法计算 $a^{m-2} \bmod m$ 来求得 a 的乘法逆元。

4.3 C++ 模板代码 —— 扩展欧几里得算法求逆元

（注意：此处调用的 `extendedGCD` 函数已在上节中实现。）

```
// 计算 a 关于模 m 的乘法逆元，若不存在则返回 -1
long long modInverse(long long a, long long m) {
    auto [g, x, y] = extendedGCD(a, m);
    if (g != 1) {
        // a 与 m 不互质，逆元不存在
        return -1;
    }
    // 保证 x 为正
    x %= m;
    if (x < 0)
        x += m;
    return x;
}
```

4.4 C++ 模板代码 —— 快速幂求逆元（基于费马小定理）

当模 m 为质数时，我们可以利用费马小定理来计算乘法逆元。根据费马小定理，

$$a^{m-1} \equiv 1 \pmod{m},$$

所以

$$a^{m-2} \equiv a^{-1} \pmod{m}.$$

下面给出快速幂（又称模幂运算）的实现，以及利用快速幂求逆元的模板代码：

```
// 快速幂计算：计算 a^b mod mod
long long modExp(long long a, long long b, long long mod) {
    long long res = 1;
    a %= mod;
    while (b > 0) {
```

```
        if (b & 1)
            res = (res * a) % mod;
        a = (a * a) % mod;
        b >>= 1;
    }
    return res;
}

// 计算 a 关于模 m 的乘法逆元（m 为质数）
long long modInverseFast(long long a, long long m) {
    // 根据费马小定理，a^(m-2) mod m 即为 a 的逆元
    return modExp(a, m - 2, m);
}
```

总结

- **整除与同余**构成了整数运算的基本框架，为后续的数论知识打下基础。
- **素数与唯一分解定理**是数论的重要基石，帮助我们理解整数结构。
- **GCD** 通过辗转相除法求出，其核心思想基于 $a = b \times q + r$ 的关系；同时，**裴蜀定理**表明最大公约数可以表示为 a 与 b 的线性组合。
- **扩展欧几里得算法（exgcd）** 在求 $\gcd(a, b)$ 的同时给出系数 x 和 y，这一结果在求解乘法逆元以及线性不定方程中十分有用。
- **线性筛** 以线性时间筛选出所有素数，是高效处理素数问题的重要工具。
- **乘法逆元** 可以通过扩展欧几里得算法或快速幂求解，快速幂方法基于费马小定理，适用于模 m 为质数的情况。