22510064 Parshwa Herwade

B1 batch

Final Year CSE 2025-26

Experiment 03 – Euclidean and Extended Euclidean Algorithm

Objectives

To implement the Euclidean algorithm to compute the GCD of two numbers.

To implement the Extended Euclidean algorithm to find Bézout's coefficients.

To compute the modular inverse when possible.

Problem Statement

Implement a program in Java that finds the GCD of two numbers using the Euclidean algorithm, finds Bézout coefficients using the Extended Euclidean algorithm, and calculates modular inverse if gcd(a, m) = 1.

Equipment/Tools

Hardware: PC/Laptop

Software: JDK (Java Development Kit)

IDE/Text Editor: Eclipse/IntelliJ/Notepad++/VSCode

Theory

Euclidean Algorithm: Repeatedly divides until remainder is 0. The last non-zero remainder is GCD.

Extended Euclidean Algorithm: Provides integers (x, y) such that:

a*x + b*y = gcd(a, b).

Modular Inverse: If gcd(a, m) = 1, then the inverse exists and is x (mod m) from Extended Euclid.

Procedure

Input two numbers.

Apply Euclidean algorithm to compute GCD.

Apply Extended Euclidean algorithm to compute coefficients.

If GCD=1, compute modular inverse.

Print results.

Steps

Start program.

Take input numbers.

Apply Euclid & Extended Euclid.

Display GCD, coefficients, modular inverse.

End.

Observations & Conclusion

- The program consistently computes the GCD of two integers using the Euclidean algorithm.
- Extended Euclidean algorithm produces correct Bézout coefficients (x, y) such that $a \cdot x + b \cdot y = \gcd(a, b)$.
- Whenever the GCD is 1, the program successfully calculates the modular inverse of a (mod b).
- If GCD $\neq 1$, the program correctly reports that no modular inverse exists.
- The implementation is correct and reliable. It verifies mathematical properties of the Euclidean and Extended Euclidean algorithms and demonstrates their use in modular inverse calculation, which is crucial in cryptography (e.g., RSA).

CODE:

```
import java.util.*;

public class Exp03_Euclid {
    static class EResult {
        long gcd, x, y;
        EResult(long g, long x, long y) { this.gcd = g; this.x = x;
this.y = y; }
    }

    // Standard Euclidean algorithm
    static long gcd(long a, long b) {
        a = Math.abs(a); b = Math.abs(b);
        while (b != 0) {
            long t = a % b;
            a = b; b = t;
        }
        return a;
    }

    // Extended Euclidean algorithm: returns (gcd, x, y)
    static EResult egcd(long a, long b) {
        if (b == 0) return new EResult(Math.abs(a), a >= 0 ? 1 : -1,
0);
        EResult r = egcd(b, a % b);
        long g = r.gcd;
        long x1 = r.y;
        long y1 = r.x - (a / b) * r.y;
        return new EResult(g, x1, y1);
    }

    // Modular inverse of a mod m (if gcd(a,m)=1)
    static Optional<Long> modInverse(long a, long m) {
        EResult r = egcd(a, m);
        if (r.gcd != 1) return Optional.empty();
        long inv = r.x % m;
        if (inv < 0) inv += m;
        return Optional.of(inv);
    }
```

```
    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            System.out.println("=== Euclid & Extended Euclid ===");
            System.out.print("Enter a: ");
            long a = sc.nextLong();
            System.out.print("Enter b (or modulus m for inverse):
");
            long b = sc.nextLong();

            long g = gcd(a, b);
            EResult r = egcd(a, b);
            System.out.println("gcd(" + a + ", " + b + ") = " + g);
            System.out.println("Bezout coefficients (x, y): x=" +
r.x + ", y=" + r.y);
            System.out.println(a + "*(" + r.x + ") + " + b + "*(" +
r.y + ") = " + (a*r.x + b*r.y));

            if (g == 1) {
                Optional<Long> invAmodB = modInverse(a, b);
                invAmodB.ifPresent(inv ->
                    System.out.println("Inverse of " + a + " mod " +
b + " = " + inv)
                );
            } else {
                System.out.println("No modular inverse exists for "
+ a + " mod " + b + " (gcd != 1).");
            }
        }
    }
}
```

OUTPUT:

```
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab> cd "c:\Users\Parshwa\
=== Euclid & Extended Euclid ===
Enter a: 3
Enter b (or modulus m for inverse): 4
gcd(3, 4) = 1
Bezout coefficients (x, y): x=-1, y=1
3*(-1) + 4*(1) = 1
Inverse of 3 mod 4 = 3
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A3> cd "c
xp03_Euclid }
=== Euclid & Extended Euclid ===
Enter a: 240
Enter b (or modulus m for inverse): 46
gcd(240, 46) = 2
Bezout coefficients (x, y): x=-9, y=47
240*(-9) + 46*(47) = 2
No modular inverse exists for 240 mod 46 (gcd != 1).
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A3> cd "c
xp03_Euclid }
=== Euclid & Extended Euclid ===
Enter a: 120
Enter b (or modulus m for inverse): 23
gcd(120, 23) = 1
Bezout coefficients (x, y): x=-9, y=47
120*(-9) + 23*(47) = 1
Inverse of 120 mod 23 = 14
PS C:\Users\Parshwa\Desktop\ASSIGN\CNS lab\22510064_CNS_A3>
```