22510064 Parshwa Herwade

Batch B1

Final Year CSE 2025-26

Experiment 04 – Chinese Remainder Theorem (CRT)

**Objectives**

* To implement the Chinese Remainder Theorem (CRT).
* To compute a unique solution modulo product of moduli.

**Problem Statement**

Given a system of simultaneous congruences:

x≡a1(modn1)

x≡a2(modn2)

.

.

x≡ak(modnk)

where the moduli n1, n2,…,nk are pairwise coprime positive integers, find the smallest non-negative integer x that satisfies all these congruences simultaneously.

**Equipment/Tools**

* Hardware: PC/Laptop
* Software: JDK
* IDE/Text Editor: Eclipse/IntelliJ/VSCode

**Theory**

* **CRT**: If we have congruences:

x ≡ a1 (mod n1)

x ≡ a2 (mod n2)

...

* where n1, n2,..., nk are pairwise coprime, then solution exists and is unique modulo N = n1\*n2\*...\*nk.
* Steps:
  1. Compute N = ∏ ni.
  2. For each i: Ni = N/ni, compute inverse of Ni mod ni.
  3. Final solution: x = Σ (ai \* Ni \* inv(Ni)) mod N.

**Procedure**

1. Input number of congruences and (ai, ni).
2. Validate moduli are pairwise coprime.
3. Apply CRT formula.
4. Print solution and modulus.

**Steps**

1. Start program.
2. Input congruences.
3. Check coprimality.
4. Apply CRT.
5. Print smallest solution.
6. End.

**Observations & Conclusion**

* CRT gives a unique solution modulo N.
* Useful in cryptography (RSA, Chinese RSA).
* Program verified with different sets of equations.
* For multiple congruences with pairwise coprime moduli, the program always computes a unique solution.
* The solution x satisfies all the given congruences when checked individually.
* The solution is always unique modulo N, where N is the product of all moduli.
* If moduli are not coprime, the program correctly detects this and reports an error.
* the CRT program works correctly for all valid inputs. It demonstrates that a system of congruences with coprime moduli has one and only one solution modulo N. This principle is widely used in number theory and cryptography (e.g., in RSA optimization).

CODE:

import java.util.\*;

public class Exp04\_CRT {

    static class EG { long g,x,y; EG(long g,long x,long y){this.g=g;this.x=x;this.y=y;} }

    static EG egcd(long a, long b){

        if(b==0) return new EG(Math.abs(a), a>=0?1:-1, 0);

        EG r = egcd(b, a%b);

        long g = r.g, x = r.y, y = r.x - (a/b)\*r.y;

        return new EG(g, x, y);

    }

    static Optional<Long> inv(long a, long m){

        EG r = egcd(a, m);

        if(r.g!=1) return Optional.empty();

        long v = r.x % m; if(v<0) v+=m; return Optional.of(v);

    }

    public static void main(String[] args){

        try (Scanner sc = new Scanner(System.in)) {

        System.out.println("=== Chinese Remainder Theorem ===");

        System.out.print("Enter number of congruences k: ");

        int k = sc.nextInt();

        long[] a = new long[k];

        long[] n = new long[k];

        for(int i=0;i<k;i++){

            System.out.print("Enter a" + (i+1) + ": ");

            a[i] = sc.nextLong();

            System.out.print("Enter n" + (i+1) + " (must be pairwise coprime): ");

            n[i] = sc.nextLong();

        }

        // Check pairwise coprime

        for(int i=0;i<k;i++){

            for(int j=i+1;j<k;j++){

                long g = gcd(n[i], n[j]);

                if(g != 1){

                    System.out.println("Error: n" + (i+1) + " and n" + (j+1) + " are not coprime (gcd="+g+").");

                    return;

                }

            }

        }

        long N = 1;

        for(long ni : n) N \*= ni;

        long x = 0;

        for(int i=0;i<k;i++){

            long Ni = N / n[i];

            Optional<Long> invNi = inv(Ni % n[i], n[i]);

            if(invNi.isEmpty()){

                System.out.println("No inverse for Ni mod n" + (i+1) + ", aborting.");

                return;

            }

            long term = (a[i] % N + N) % N;

            x = (x + term \* Ni % N \* invNi.get() % N) % N;

        }

        if(x<0) x += N;

        System.out.println("Smallest non-negative solution x = " + x);

        System.out.println("Solution is unique modulo N = " + N);

    }

    }

    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;

    }

}

OUTPUT:  
  
