**Final Year B. Tech., Sem VII 2022-23**

**Cryptography And Network Security**

**PRN/ Roll No: 2020BTECS00206**

**Full Name: SAYALI YOGESH DESAI**

**Batch: B4**

**Assignment No. 12**

1. **Aim:**

Implementation of Chinese Remainder Theorem

1. **Theory:**

Chinese Remainder Theorem:

If m1, m2, .., mk are pairwise relatively prime positive integers, and if a1, a2, .., ak are any integers, then the simultaneous congruences x ≡ a1 (mod m1), x ≡ a2 (mod m2), ..., x ≡ ak (mod mk) have a solution, and the solution is unique modulo m, where m = m1m2⋅⋅⋅mk .

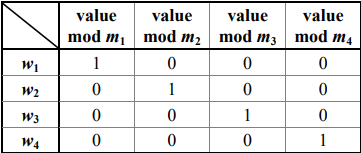
Proof that a solution exists:

To keep the notation simpler, we will assume k = 4. Note the proof is constructive, i.e., it shows us how to actually construct a solution.

Our simultaneous congruences are

x ≡ a1 (mod m1), x ≡ a2 (mod m2), x ≡ a3 (mod m3), x ≡ a4 (mod m4).

Our goal is to find integers w1, w2, w3, w4 such that:



Once we have found w1, w2, w3, w4, it is easy to construct x:

x = a1w1 + a2w2 + a3w3 + a4w4.

Moreover, as long as the moduli (m1, m2, m3, m4) remain the same, we can use the same w1, w2, w3, w4 with any a1, a2, a3, a4.

First define:

z1 = m / m1 = m2m3m4

z2 = m / m2 = m1m3m4

z3 = m / m3 = m1m2m4

z4 = m / m4 = m1m2m3

Note that

1. z1 ≡ 0 (mod mj) for j = 2, 3, 4.
2. ii) gcd(z1, m1) = 1.

(If a prime p dividing m1 also divides z1= m2m3m4, then p divides m2, m3, or m4.) and likewise for z2, z3, z4.

Next define:

y1 ≡ z1–1 (mod m1)

y2 ≡ z2 –1 (mod m2)

y3 ≡ z3 –1 (mod m3)

y4 ≡ z4 –1 (mod m4)

The inverses exist by (ii) above, and we can find them by Euclid’s extended algorithm.

Note that

1. y1z1 ≡ 0 (mod mj) for j = 2, 3, 4. (Recall z1 ≡ 0 (mod mj) )
2. iv) y1z1 ≡ 1 (mod m1) and likewise for y2z2, y3z3, y4z4.

Lastly define:

w1 ≡ y1z1 (mod m)

w2 ≡ y2z2 (mod m)

w3 ≡ y3z3 (mod m)

w4 ≡ y4z4 (mod m)

Then w1, w2, w3, and w4 have the properties in the above table.

1. **Code:**

#include <bits/stdc++.h>

using namespace std;

// Function for extended Euclidean Algorithm

int ansS, ansT;

int findGcdExtended(int r1, int r2, int s1, int s2, int t1, int t2)

{

// Base Case

if (r2 == 0)

{

ansS = s1;

ansT = t1;

return r1;

}

int q = r1 / r2;

int r = r1 % r2;

int s = s1 - q \* s2;

int t = t1 - q \* t2;

cout << q << " " << r1 << " " << r2 << " " << r << " " << s1 << " " << s2 << " " << s << " " << t1 << " " << t2 << " " << t << endl;

return findGcdExtended(r2, r, s2, s, t2, t);

}

int modInverse(int A, int M)

{

int x, y;

int g = findGcdExtended(A, M, 1, 0, 0, 1);

if (g != 1) {

cout << "\n Inverse doesn't exist";

return 0;

}

else {

// m is added to handle negative x

int res = (ansS % M + M) % M;

cout << "\n Inverse is " << res << endl;

return res;

}

}

int findX(vector<int> num, vector<int> rem, int k)

{

// Compute product of all numbers

int prod = 1;

for (int i = 0; i < k; i++)

prod \*= num[i];

// Initialize result

int result = 0;

// Apply above formula

for (int i = 0; i < k; i++) {

int pp = prod / num[i];

result += rem[i] \* modInverse(pp, num[i]) \* pp;

}

return result % prod;

}

int main()

{

// 3

// 3 4 5

// 2 3 1

int k;

cout << "\n Enter total count of equations : ";

cin >> k;

vector<int> num(k), rem(k);

cout<<"\n Enter divisors : ";

for (int i = 0; i < k; i++)

cin >> num[i];

cout<<"\n Enter remainders : ";

for (int i = 0; i < k; i++)

cin >> rem[i];

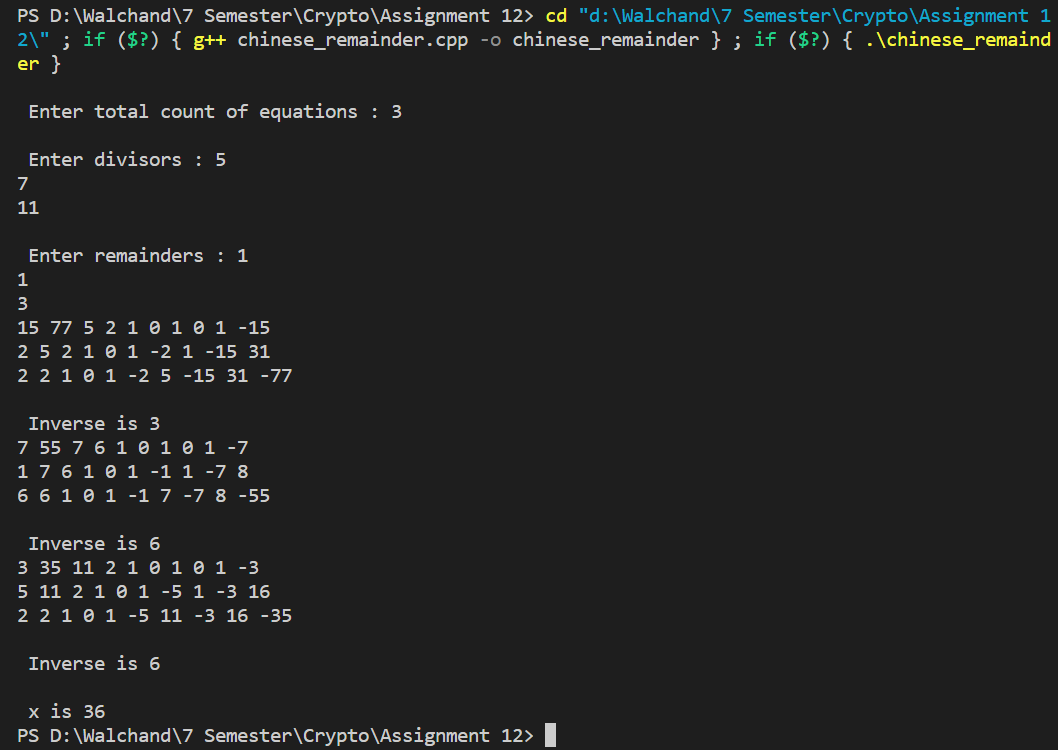
int x = findX(num, rem, k);

cout << "\n x is " << x;

return 0;

}

1. **Output:**

****

1. **Conclusion:**

Successfully implemented Chinese Remainder Theorem.