**Chinese Remainder Theorem**

**Theory:**

The Chinese Remainder Theorem (CRT) is a mathematical concept that provides a method for solving a system of simultaneous linear congruences. In essence, it allows you to find a unique solution to a set of modular equations by combining solutions from simpler, individual modular equations.

Here's a brief theory of CRT:

1. System of Congruences:

CRT is used to solve a system of congruences of the form:

```

x ≡ a\_1 (mod n\_1)

x ≡ a\_2 (mod n\_2)

...

x ≡ a\_k (mod n\_k)

```

2. Coprime Moduli:

For CRT to work, the moduli (n\_1, n\_2, ..., n\_k) should be pairwise coprime, meaning that their greatest common divisors (GCD) are all 1.

3. Unique Solution:

CRT guarantees a unique solution for the value of "x" within a specific range.

4. Solution Calculation:

- Calculate the product of all moduli: N = n\_1 \* n\_2 \* ... \* n\_k.

- For each congruence, compute N\_i = N / n\_i.

- Find the modular inverses (x\_i) of each N\_i modulo n\_i.

- The solution "x" is given by:

```

x = a\_1 \* N\_1 \* x\_1 + a\_2 \* N\_2 \* x\_2 + ... + a\_k \* N\_k \* x\_k (mod N)

```

5. Applications:

CRT has applications in number theory, modular arithmetic, cryptography, and computer science. It is used in algorithms for efficient arithmetic in finite fields and modular integer operations.

CRT is a powerful tool for solving systems of modular equations, and it plays a crucial role in various computational and cryptographic applications.

**Code:**

def extended\_gcd(a, b):

    if a == 0:

        return (b, 0, 1)

    else:

        g, x, y = extended\_gcd(b % a, a)

        return (g, y - (b // a) \* x, x)

def mod\_inverse(a, m):

    g, x, \_ = extended\_gcd(a, m)

    if g != 1:

        raise ValueError("The modular inverse does not exist.")

    return x % m

def chinese\_remainder\_theorem(n, a):

    N = 1

    N\_i = []

    x\_i = []

    b\_i = []

    for ni in n:

        N \*= ni

    for ni in n:

        N\_i.append(N // ni)

        x\_i.append(mod\_inverse(N\_i[-1], ni))

    for i in range(len(n)):

        b\_i.append(a[i] \* N\_i[i] \* x\_i[i])

    print("Intermediate Steps:")

    for i in range(len(n)):

        print(f"N\_{i} = {N\_i[i]}, x\_{i} = {x\_i[i]}, b\_{i} = {b\_i[i]}")

    result = sum(b\_i) % N

    return result

# Input from the user

n = []

a = []

num\_congruences = int(input("Enter the number of congruences: "))

for i in range(num\_congruences):

    n\_i = int(input(f"Enter the modulus (n\_{i}): "))

    a\_i = int(input(f"Enter the remainder (a\_{i}): "))

    n.append(n\_i)

    a.append(a\_i)

result = chinese\_remainder\_theorem(n, a)

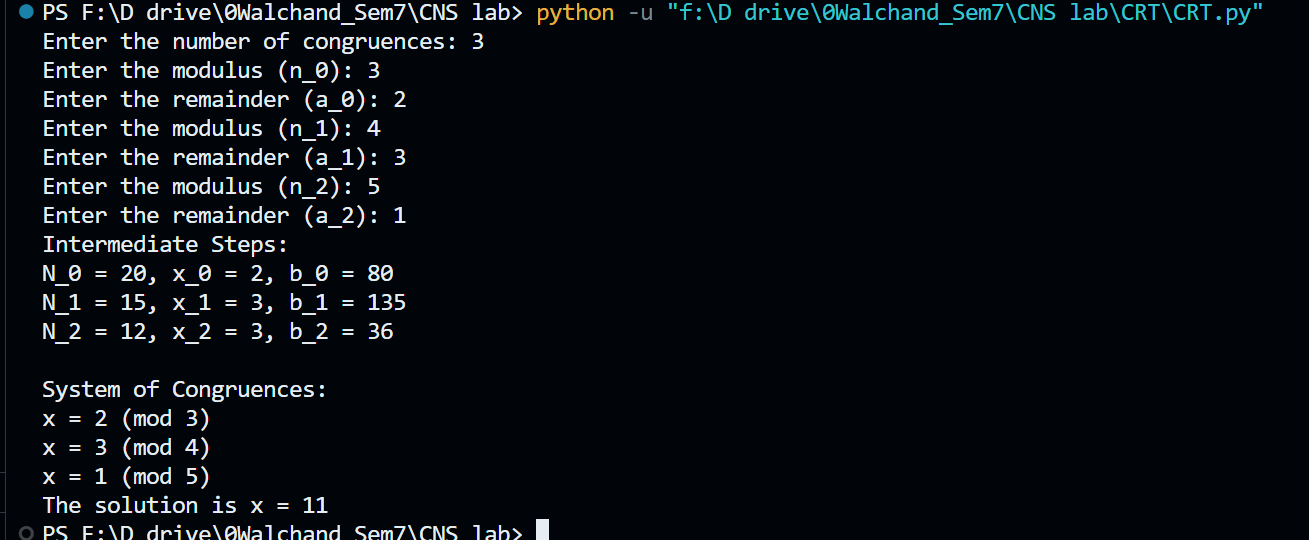
print("\nSystem of Congruences:")

for i in range(len(n)):

    print(f"x = {a[i]} (mod {n[i]})")

print(f"The solution is x = {result}")

**Screenshot:**

****