Chinese Remainder Theorem

Let's say we have a system of congruences like this:

x = a1(mod m1)

x = a2(mod m2)

x = a3(mod m3)

and so on...

CRT tells us that if all the moduli m1,m2,m3... are pairwise coprime (i.e. gcd(mi, mj) = 1 for all i ≠ j),

then there exists a unique solution modulo M = m1 × m2 × m3 × ...

That means there's exactly one number x between 0 and M-1 that satisfies all those congruences.

Let's Recall :-

x ≡ a (mod n)

(read as: "x is congruent to a modulo n")

-> x - a is a multiple of n (i.e. n divides (x-a))

-> x and a have the same remainder when dividing by n

You'll often see equations like:

x = 3 (mod 5)

which means all numbers that give remainder 3 when divided by 5.

So the set of all possible x is:

{n + 5k | k ∈ Z}

That is, x = n + 5k

Eg:-

x = 2 (mod 3)

x = 3 (mod 5)

x = 2 (mod 7)

equation 1 -> x = 2 (mod 3)

x = 2 + 3k -> 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, ...

equation 2 -> x = 3 (mod 5)

x = 3 + 5k -> 3, 8, 13, 18, 23, 28, 33, 38, 43, 48, ...

equation 3 -> x = 2 (mod 7)

x = 2 + 7k -> 2, 9, 16, 23, ...

so, 23 is the smallest integer to satisfy all the congruences

Now, we can say that next integer to satisfy will be

23 + LCM(3, 5, 7)**

And since 3, 5, 7 are coprime then LCM is just 3 * 5 * 7

so 23 + (105)**

=> x = 23 (mod 105)

But we need a better way to solve this so now let's look at this mathematically

CRT says that if all the moduli are pairwise coprime then a unique solution exist modulo M (m1 * m2 ... * mn)

Proof:-

We have, x = a1(mod m1)

x = a2(mod m2)

.

.

.

x = an(mod mn)

Let,

M = m1 * m2 * m3 ... mn

and,

Mi = M / mi

Note:

gcd(Mi, mi) = 1

ie:

Mi*xi + mi*yi = 1 { By Bezout's Identity }

By reducing the equation we get,

Mi*xi = 1 (mod mi)

Now, multiplying by ai on both sides we get,

(Mi*xi). ai = ai (mod mi)

Set,

x = M1*x1*1 + M2*x2*x2 + ... + Mn*xn*an

Observation,

Mi = 0 (mod mj), i is not equal to j

Mi has no reason to be 0 mod mi (since gcd(Mi, mi) = 1)

Fix,

We do this using a modular inverse.

xi = Mi^-1 (mod mi)

This means,

Mi * xi = 1 (mod mi)

Therefore,

xi = modular inverse of xi

Now we can say,

x = Mi.yi.ai (mod mi)

x = ai (mod mi)

This is how we prove that a solution exist

Now let's proof that the solution is unique

Proof:-

We already have a solution x

Suppose y is another solution, then we can say

x = a1(mod m1) y = a1(mod m1)

x = a2(mod m2) y = a2(mod m2)

. .

. .

. .

x = an(mod mn) y = an(mod mn)

Subtracting both the equations we get

(x - y) = 0 (mod m1)

(x - y) = 0 (mod m2)

.

.

.

(x - y) = 0 (mod mn)

This implies:

(x - y) | m1

(x - y) | m2

. => (x - y) | LCM(m1, m2, ..., mn)

.

(x - y) | mn

=> (x - y) | M

=> x = y (mod M)

This proves that x and y are in the same equivalence class

Final solution:

$$x \equiv \sum_i a_i M_i \operatorname{inv}_i \pmod{M}$$

So, we will simply calculate Mi and inverse then using the above formula we will get our X

Example:-

An old woman goes to market and horse steps on her basket and crushes the eggs.
The rider offers to pay the damages and asks her how many eggs she had brought.
She does not remember the exact number but when she had taken them out two at a time, there was one egg left.
The same happened when she picked 2,3,4,5,6 at a time, but when she took seven at a time they came out even.
What is the smallest number of eggs she could have had?

Let's rephrase it in mathematical notations.

Suppose, the old woman had n eggs in her basket.
Now she states that when she took out eggs from the basket 2 at a time,
there was only 1 egg remaining, meaning, n%2=1(mod 2). So basically, she is giving us various congruence equations.

From her statements, we get the following linear congruence equations:

n = 1 (mod 2)
n = 1 (mod 3)
n = 1 (mod 4)
n = 1 (mod 5)
n = 1 (mod 6)
n = 0 (mod 7)

Code

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to find gcd and x, y such that ax + by = gcd(a, b)
long long extended_gcd(long long a, long long b, long long &x, long long &y) {
    if (b == 0) {
        x = 1; y = 0;
        return a;
    }
    long long x1, y1;
    long long g = extended_gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return g;
}

// Modular inverse using Extended Euclid
long long mod_inverse(long long a, long long m) {
    long long x, y;
    long long g = extended_gcd(a, m, x, y);
    if (g != 1) return -1; // inverse doesn't exist
    return (x % m + m) % m;
}

// Chinese Remainder Theorem
long long chinese_remainder(vector<long long> a, vector<long long> m) {
    long long M = 1;
    for (auto val : m) M *= val;

    long long x = 0;
    for (int i = 0; i < a.size(); i++) {
        long long Mi = M / m[i];
        long long inv = mod_inverse(Mi, m[i]);
        x = (x + a[i] * Mi % M * inv % M) % M;
    }

    return (x + M) % M;  // ensure non-negative
}

int main() {
    vector<long long> a = {2, 3, 2};
    vector<long long> m = {3, 5, 7};

    cout << "x = " << chinese_remainder(a, m) << "\n";
}
```

Time Complexity:

$$T(n) = O(n) + \sum_{i=1}^{n} O(\log m_i)$$

$$T(n) = O(n \log M_{max})$$