Last update: October 16, 2023 Translated From: e-maxx.ru

Chinese Remainder Theorem

The Chinese Remainder Theorem (which will be referred to as CRT in the rest of this article) was discovered by Chinese mathematician Sun Zi.

Formulation

Let $m = m_1 \cdot m_2 \cdots m_k$, where m_i are pairwise coprime. In addition to m_i, we are also given a system of congruences

$$\left\{\begin{array}{rcl} a & \equiv & a_1 \pmod{m_1} \\ a & \equiv & a_2 \pmod{m_2} \\ & \vdots & \\ a & \equiv & a_k \pmod{m_k} \end{array}\right.$$

where a_i are some given constants. The original form of CRT then states that the given system of congruences always has *one and exactly one* solution modulo m.

E.g. the system of congruences

$$\left\{\begin{array}{rcl} a & \equiv & 2 \pmod{3} \\ a & \equiv & 3 \pmod{5} \\ a & \equiv & 2 \pmod{7} \end{array}\right.$$

has the solution 23 modulo 105, because $23 \bmod 3 = 2$, $23 \bmod 5 = 3$, and $23 \bmod 7 = 2$. We can write down every solution as $23 + 105 \cdot k$ for $k \in \mathbb{Z}$.

Corollary

A consequence of the CRT is that the equation

$$x \equiv a \pmod{m}$$

is equivalent to the system of equations

$$\left\{\begin{array}{rcl} x & \equiv & a_1 \pmod{m_1} \\ & \vdots & \\ x & \equiv & a_k \pmod{m_k} \end{array}\right.$$

(As above, assume that $m = m_1 m_2 \cdots m_k$ and m_i are pairwise coprime).

Solution for Two Moduli

Consider a system of two equations for coprime m_1, m_2:

$$\begin{cases} a \equiv a_1 \pmod{m_1} \\ a \equiv a_2 \pmod{m_2} \end{cases}$$

We want to find a solution for $a \pmod{m_1 m_2}$. Using the Extended Euclidean Algorithm we can find Bézout coefficients n_1, n_2 such that

$$n_1 m_1 + n_2 m_2 = 1.$$

In fact n_1 and n_2 are just the modular inverses of m_1 and m_2 modulo m_2 and m_1. We have $n_1 m_1 \equiv 1 \pmod{m_2}$ so $n_1 \equiv m_1^{-1} \pmod{m_2}$, and vice versa $n_2 \equiv m_2^{-1} \pmod{m_1}$.

With those two coefficients we can define a solution:

$$a = a_1 n_2 m_2 + a_2 n_1 m_1 \bmod m_1 m_2$$

It's easy to verify that this is indeed a solution by computing $a \bmod m_1$ and $a \bmod m_2$.

$$\begin{array}{rcll} a & \equiv & a_1 n_2 m_2 + a_2 n_1 m_1 & \pmod{m_1} \\ & \equiv & a_1 (1 - n_1 m_1) + a_2 n_1 m_1 & \pmod{m_1} \\ & \equiv & a_1 - a_1 n_1 m_1 + a_2 n_1 m_1 & \pmod{m_1} \\ & \equiv & a_1 & \pmod{m_1} \end{array}$$

Notice, that the Chinese Remainder Theorem also guarantees, that only 1 solution exists modulo $m_1 m_2$. This is also easy to prove.

Lets assume that you have two different solutions x and y. Because $x \equiv a_i \pmod{m_i}$ and $y \equiv a_i \pmod{m_i}$, it follows that $x - y \equiv 0 \pmod{m_i}$ and therefore $x - y \equiv 0 \pmod{m_1 m_2}$ or equivalently $x \equiv y \pmod{m_1 m_2}$. So x and y are actually the same solution.

Solution for General Case

Inductive Solution

As $m_1 m_2$ is coprime to m_3, we can inductively repeatedly apply the solution for two moduli for any number of moduli. First you compute $b_2 := a \pmod{m_1 m_2}$ using the first two congruences, then you can compute $b_3 := a \pmod{m_1 m_2 m_3}$ using the congruences $a \equiv b_2 \pmod{m_1 m_2}$ and $a \equiv a_3 \pmod{m_3}$, etc.

Direct Construction

A direct construction similar to Lagrange interpolation is possible.

Let $M_i := \prod_{i \neq j} m_j$, the product of all moduli but m_i, and N_i the modular inverses $N_i := M_i^{-1} \bmod m_i$. Then a solution to the system of congruences is:

$$a \equiv \sum_{i=1}^{k} a_i M_i N_i \pmod{m_1 m_2 \cdots m_k}$$

We can check this is indeed a solution, by computing $a \bmod m_i$ for all i. Because M_j is a multiple of m_i for $i \neq j$ we have

$$\begin{aligned} a &\equiv \sum_{j=1}^k a_j M_j N_j &\pmod{m_i} \\ &\equiv a_i M_i N_i &\pmod{m_i} \\ &\equiv a_i M_i M_i^{-1} &\pmod{m_i} \\ &\equiv a_i &\pmod{m_i} \end{aligned}$$

Implementation

```
struct Congruence {
    long long a, m;
};

long long chinese_remainder_theorem(vector<Congruence> const& congruences) {
    long long M = 1;
    for (auto const& congruence : congruences) {
        M *= congruence.m;
    }

    long long solution = 0;
    for (auto const& congruence : congruences) {
        long long a_i = congruence.a;
        long long M_i = M / congruence.m;
        long long N_i = mod_inv(M_i, congruence.m);
        solution = (solution + a_i * M_i % M * N_i) % M;
    }
    return solution;
}
```

Solution for not coprime moduli

As mentioned, the algorithm above only works for coprime moduli $m_1, m_2, \dots m_k$.

In the not coprime case, a system of congruences has exactly one solution modulo $\text{lcm}(m_1, m_2, \dots, m_k)$, or has no solution at all.

E.g. in the following system, the first congruence implies that the solution is odd, and the second congruence implies that the solution is even. It's not possible that a number is both odd and even, therefore there is clearly no solution.

$$\begin{cases} a \equiv 1 \pmod 4 \\ a \equiv 2 \pmod 6 \end{cases}$$

It is pretty simple to determine is a system has a solution. And if it has one, we can use the original algorithm to solve a slightly modified system of congruences.

A single congruence $a \equiv a_i \pmod{m_i}$ is equivalent to the system of congruences $a \equiv a_i \pmod{p_j^{n_j}}$ where $p_1^{n_1} p_2^{n_2}\cdots p_k^{n_k}$ is the prime factorization of m_i.

With this fact, we can modify the system of congruences into a system, that only has prime powers as moduli. E.g. the above system of congruences is equivalent to:

$$\begin{cases} a \equiv 1 & \pmod 4 \\ a \equiv 2 \equiv 0 & \pmod 2 \\ a \equiv 2 & \pmod 3 \end{cases}$$

Because originally some moduli had common factors, we will get some congruences moduli based on the same prime, however possibly with different prime powers.

You can observe, that the congruence with the highest prime power modulus will be the strongest congruence of all congruences based on the same prime number. Either it will give a contradiction with some other congruence, or it will imply already all other congruences.

In our case, the first congruence $a \equiv 1 \pmod 4$ implies $a \equiv 1 \pmod 2$, and therefore contradicts the second congruence $a \equiv 0 \pmod 2$. Therefore this system of congruences has no solution.

If there are no contradictions, then the system of equation has a solution. We can ignore all congruences except the ones with the highest prime power moduli. These moduli are now coprime, and therefore we can solve this one with the algorithm discussed in the sections above.

E.g. the following system has a solution modulo $\text{lcm}(10, 12) = 60$.

$$\begin{cases} a \equiv 3 \pmod{10} \\ a \equiv 5 \pmod{12} \end{cases}$$

The system of congruence is equivalent to the system of congruences:

$$\begin{cases} a \equiv 3 \equiv 1 \pmod 2 \\ a \equiv 3 \equiv 3 \pmod 5 \\ a \equiv 5 \equiv 1 \pmod 4 \\ a \equiv 5 \equiv 2 \pmod 3 \end{cases}$$

The only congruence with same prime modulo are $a \equiv 1 \pmod 4$ and $a \equiv 1 \pmod 2$. The first one already implies the second one, so we can ignore the second one, and solve the following system with coprime moduli instead:

$$\begin{cases} a \equiv 3 \equiv 3 \pmod 5 \\ a \equiv 5 \equiv 1 \pmod 4 \\ a \equiv 5 \equiv 2 \pmod 3 \end{cases}$$

It has the solution $53 \pmod{60}$, and indeed $53 \bmod{10} = 3$ and $53 \bmod{12} = 5$.

Garner's Algorithm

Another consequence of the CRT is that we can represent big numbers using an array of small integers.

Instead of doing a lot of computations with very large numbers numbers, which might be expensive (think of doing divisions with 1000-digit numbers), you can pick a couple of coprime moduli and represent the large number as a system of congruences, and perform all operations on the system of equations. Any number a less than $m_1 m_2 \cdots m_k$ can be represented as an array $a_1, \ldots, a_k$, where $a \equiv a_i \pmod{m_i}$.

By using the above algorithm, you can again reconstruct the large number whenever you need it.

Alternatively you can represent the number in the **mixed radix** representation:

$$a = x_1 + x_2 m_1 + x_3 m_1 m_2 + \ldots + x_k m_1 \cdots m_{k-1} \text{ with } x_i \in [0, m_i)$$

Garner's algorithm, which is discussed in the dedicated article Garner's algorithm, computes the coefficients x_i. And with those coefficients you can restore the full number.

Practice Problems:

- Google Code Jam - Golf Gophers

- Hackerrank - Number of sequences
- Codeforces - Remainders Game

Contributors:
jakobkogler (74.78%) jxu (13.04%) algmyr (5.65%) adamant-pwn (3.04%) thanhtnguyen (1.74%) vatsalsharma376 (0.87%) jatingaur18 (0.43%) MayankPratap (0.43%)