Modular Multiplicative Inverse

Table of Contents

Definition

A modular multiplicative inverse of an integer a is an integer x such that $a \cdot x$ is congruent to 1 modular some modulus m. To write it in a formal way: we want to find an integer x so that

$$a \cdot x \equiv 1 \mod m.$$

We will also denote x simply with a^{-1}.

We should note that the modular inverse does not always exist. For example, let $m = 4$, $a = 2$. By checking all possible values modulo m is should become clear that we cannot find a^{-1} satisfying the above equation. It can be proven that the modular inverse exists if and only if a and m are relatively prime (i.e. $\gcd(a, m) = 1$).

In this article, we present two methods for finding the modular inverse in case it exists, and one method for finding the modular inverse for all numbers in linear time.

Finding the Modular Inverse using Extended Euclidean algorithm

Consider the following equation (with unknown x and y):

$$a \cdot x + m \cdot y = 1$$

This is a Linear Diophantine equation in two variables. As shown in the linked article, when $\gcd(a, m) = 1$, the equation has a solution which can be found using the extended Euclidean algorithm. Note that $\gcd(a, m) = 1$ is also the condition for the modular inverse to exist.

Now, if we take modulo m of both sides, we can get rid of $m \cdot y$, and the equation becomes:

$$a \cdot x \equiv 1 \mod m$$

Thus, the modular inverse of a is x.

The implementation is as follows:

```
int x, y;
int g = extended_euclidean(a, m, x, y);
if (g != 1) {
    cout << "No solution!";
}
else {
    x = (x % m + m) % m;
    cout << x << endl;
}
```

Notice that we way we modify `x`. The resulting `x` from the extended Euclidean algorithm may be negative, so `x % m` might also be negative, and we first have to add `m` to make it positive.

Finding the Modular Inverse using Binary Exponentiation

Another method for finding modular inverse is to use Euler's theorem, which states that the following congruence is true if a and m are relatively prime:

$$a^{\phi(m)} \equiv 1 \mod m$$

ϕ is Euler's Totient function. Again, note that a and m being relative prime was also the condition for the modular inverse to exist.

If m is a prime number, this simplifies to Fermat's little theorem:

$$a^{m-1} \equiv 1 \mod m$$

Multiply both sides of the above equations by a^{-1}, and we get:

- For an arbitrary (but coprime) modulus m: $a^{\phi(m)-1} \equiv a^{-1} \mod m$
- For a prime modulus m: $a^{m-2} \equiv a^{-1} \mod m$

From these results, we can easily find the modular inverse using the binary exponentiation algorithm, which works in $O(\log m)$ time.

Even though this method is easier to understand than the method described in previous paragraph, in the case when m is not a prime number, we need to calculate Euler phi function, which involves factorization of m, which might be very hard. If the prime factorization of m is known, then the complexity of this method is $O(\log m)$.

Finding the modular inverse for every number modulo m

The problem is the following: we want to compute the modular inverse for every number in the range $[1, m-1]$.

Applying the algorithms described in the previous sections, we can obtain a solution with complexity $O(m \log m)$.

Here we present a better algorithm with complexity $O(m)$. However for this specific algorithm we require that the modulus m is prime.

We denote by $\text{inv}[i]$ the modular inverse of i. Then for $i > 1$ the following equation is valid:

$$\text{inv}[i] = - \left\lfloor \frac{m}{i} \right\rfloor \cdot \text{inv}[m \bmod i] \bmod m$$

Thus the implementation is very simple:

```
inv[1] = 1;
for(int i = 2; i < m; ++i)
    inv[i] = (m - (m/i) * inv[m%i] % m) % m;
```

Proof

We have:

$$m \bmod i = m - \left\lfloor \frac{m}{i} \right\rfloor \cdot i$$

Taking both sides modulo m yields:

$$m \bmod i \equiv - \left\lfloor \frac{m}{i} \right\rfloor \cdot i \mod m$$

Multiply both sides by $i^{-1} \cdot (m \bmod i)^{-1}$ yields

$$(m \bmod i) \cdot i^{-1} \cdot (m \bmod i)^{-1} \equiv -\left\lfloor \frac{m}{i} \right\rfloor \cdot i \cdot i^{-1} \cdot (m \bmod$$

which simplifies to:

$$i^{-1} \equiv -\left\lfloor \frac{m}{i} \right\rfloor \cdot (m \bmod i)^{-1} \mod m,$$

Practice Problems

- UVa 11904 - One Unit Machine
- Hackerrank - Longest Increasing Subsequence Arrays
- Codeforces 300C - Beautiful Numbers
- Codeforces 622F - The Sum of the k-th Powers
- Codeforces 717A - Festival Organization
- Codeforces 896D - Nephren Runs a Cinema