forthright48
learning never ends

Modular Multiplicative Inverse

forthright48 on September 23, 2015

Problem

Given value of A and M, find the value of X such that $AX \equiv 1 \pmod{\mathrm{M}}$.

For example, if $A = 2$ and $M = 3$, then $X = 2$, since $2 \times 2 = 4 \equiv 1 \pmod{3}$.

We can rewrite the above equation to this:

$$AX \equiv 1 \pmod{\mathrm{M}}$$
$$X \equiv \frac{1}{A} \pmod{\mathrm{M}}$$
$$X \equiv A^{-1} \pmod{\mathrm{M}}$$

Hence, the value X is known as Modular Multiplicative Inverse of A with respect to M.

How to Find Modular Inverse?

First we have to determine whether Modular Inverse even exists for given A and M before we jump to finding the solution. Modular Inverse doesn't exist for every pair of given value.

Modular Inverse of A with respect to M, that is, $X = A^{-1} (\text{mod M})$ exists, if and only if A and M are coprime.

Why is that?

$$AX \equiv 1 \ (\text{mod M})$$
$$AX - 1 \equiv 0 \ (\text{mod M})$$

Therefore, M divides $AX - 1$. Since M divides $AX - 1$, then a divisor of M will also divide $AX - 1$. Now suppose, A and M are not coprime. Let D be a number greater than 1 which divides both A and M. So, D will divide $AX - 1$. Since D already divides A, D must divide 1. But this is not possible. Therefore, the equation is unsolvable when A and M are not coprime.

From here on, we will assume that A and M are coprime unless state otherwise.

Using Fermat's Little Theorem

Recall Fermat's Little Theorem from a previous post, "Euler's Theorem and Fermat's Little Theorem". It stated that, if A and M are coprime and M is a prime, then, $A^{M-1} \equiv 1 (\text{mod M})$. We can use this equation to find the modular inverse.

$$A^{M-1} \equiv 1 \ (\text{mod M}) \text{ (Divide both side by } A)$$
$$A^{M-2} \equiv \frac{1}{A} \ (\text{mod M})$$
$$A^{M-2} \equiv A^{-}1 \ (\text{mod M})$$

Therefore, when M is prime, we can find modular inverse by calculating the value of A^{M-2}. How do we calculate this? Using Modular Exponentiation.

This is the easiest method, but it doesn't work for non-prime M. But no worries since we have other ways to find the inverse.

Using Euler's Theorem

It is possible to use Euler's Theorem to find the modular inverse. We know that:

$$A^{\phi(M)} \equiv 1 (\text{mod M})$$
$$\therefore A^{\phi(M)-1} \equiv A^{-1} (\text{mod M})$$

This process works for any M as long as it's coprime to A, but it is rarely used since we have to

calculate Euler Phi value of M which requires more processing. There is an easier way.

Using Extended Euclidean Algorithm

We are trying to solve the congruence, $AX \equiv 1 (\mod M)$. We can convert this to an equation.

$$AX \equiv 1 (\mod M)$$
$$AX + MY = 1$$

Here, both X and Y are unknown. This is a linear equation and we want to find integer solution for it. Which means, this is a Linear Diophantine Equation.

Linear Diophantine Equation can be solved using Extended Euclidean Algorithm. Just pass $\text{ext\_gcd}()$ the value of A and M and it will provide you with values of X and Y. We don't need Y so we can discard it. Then we simply take the mod value of X as the inverse value of A.

Code

A and M need to be coprime. Otherwise, no solution exists. The following codes do not check if A and M are coprime. The checking is left of the readers to implement.

When M is Prime

We will use Fermat's Little Theorem here. Just call the $bigmod()$ function from where you need the value.

```
1 int x = bigmod( a, m - 2, m ); // (ax)%m = 1
```

Here x is the modular inverse of a which is passed to $bigmod()$ function.

When M is not Prime

For this, we have to use a new function.

```
1 int modInv ( int a, int m ) {
2     int x, y;
3     ext_gcd( a, m, &x, &y );
4
5     // Process x so that it is between 0 and m-1
6     x %= m;
7     if ( x < 0 ) x += m;
8     return x;
```

```
9 | }
```

I wrote this function since after using $\text{ext\_gcd}()$ we need to process x so that it's value is between 0 and $M-1$. Instead of doing that manually, I decided to write a function.

So, if we want to find the modular inverse of A with respect to M, then the result will be $X = modInv(A, M)$.

Complexity

Repeated Squaring method has a complexity of $O(logP)$, so the first code has complexity of $O(logM)$, whereas Extended Euclidean has complexity of $O(log_{10}A + log_{10}B)$ so second code has complexity $O(log_{10}A + log_{10}M)$.

Why Do We Need Modular Inverse?

We need Modular Inverse to handle division during Modular Arithmetic. Suppose we are trying to find the value of the following equations:

$\frac{4}{2} \% 3$ - This is simple. We just simplify the equation and apply normal modular operation. That is, it becomes $\frac{4}{2} \% 3 = 2 \% 3 = 2$.

Then what happens when we try to do same with $\frac{12}{9} \% 5$? First we simply. $\frac{12}{9} \% 5 = \frac{4}{3} \% 5$. Now we are facing an irreducible fraction. Should we simply perform the modular operation with numerator and denominator? That doesn't help since both of them are smaller than 5.

This is where Modular Inverse comes to the rescue. Let us solve the equation $X \equiv 3^{-1} \pmod 5$. How do we find the value of X? We will see that on the later part of the post. For now, just assume that we know the value of X.

Now, we can rewrite the above equation in the following manner:

$\frac{12}{9} \% 5$
$\frac{4}{3} \% 5$
$(4 \times 3^{-1}) \% 5$
$((4 \% 5) \times (3^{-1} \% 5)) \% 5$
$\therefore 4X \% 5$

So, now we can easily find the value of $\frac{A}{B} \% M$ by simply calculating the value of $(A \times B^{-1}) \% M$

.

Conclusion

Modular Inverse is a small topic but look at the amount of background knowledge it requires to understand it! Euler's Theorem, Euler Phi, Modular Exponentiation, Linear Diophantine Equation, Extended Euclidian Algorithm and other small bits of information. We covered them all before, so we can proceed without any hitch.

Hopefully, you understood how Modular Inverse works. If not, make sure to revise the articles in the "Reference" section below.

Reference

1. Wiki - Modular Multiplicative Inverse
2. forthright48 - Euler's Theorem and Fermat's Little Theorem
3. forthright48 - Modular Exponentiation
4. forthright48 - Euler Phi
5. forthright48 - Linear Diophantine Equation
6. forthright48 - Extended Euclidean Algorithm

Category: CPPS, Number Theory

Previous:
Repeated Squaring Method for Modular Exponentiation

Next:
Euler Phi Extension and Divisor Sum Theorem

Guest

Archives

September 2018 (2)

February 2018 (1)

January 2018 (1)

November 2017 (2)

September 2015 (7)

August 2015 (13)

July 2015 (15)

Categories

CPPS (40)

- Combinatorics (3)
- Number Theory (33)

Meta (1)

Recent Comments

forthright48 on Number of Divisors of an Integer

Sohana Akter on Number of Divisors of an Integer

forthright48 on Number of Divisors of an Integer

forthright48 on Prufer Code: Linear Representation of a Labeled Tree

Mohammad Samiul Islam on SPOJ LCMSUM – LCM Sum

Privacy Policy