Introduction to Modular Arithmetic

forthright48 on July 25, 2015

> **"In mathematics, modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" upon reaching a certain value—the modulus." – Wiki**

The clock is a good example for modular arithmetic. Let's say 12'o clock means 0. Then clock shows the following values, $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0, 1, 2, 3 \ldots$. Every time clock hits 12, it wraps around to 0. Since clock wraps around 12, we say that 12 is the Modulus.

General Concepts

In order to understand modular arithmetic, we need to know the following:

Modulo Operation $\%$

Modular arithmetic deals with Modulo operation. It has the symbol, "$\%$". The expression $A\%B = C$ means that when we divide A with B we get the remainder C. In other words, the modulo operation finds the remainder. For example:

$5 \ \% \ 3 = 2$
$15 \ \% \ 5 = 0$
$24 \ \% \ 30 = 24$
$30 \ \% \ 24 = 6$
$4 \ \% \ 4 = 0$

Congruence Relation $\equiv$

In mathematics, saying that $A \equiv B \pmod{M}$ means that both A and B has same remainder when they are divided by M. For example,

$5 \equiv 10 \pmod{5}$
$2 \equiv 7 \pmod{5}$
$4 \equiv 10 \pmod{3}$

Range

In modular arithmetic with modulus M, we only work with values ranging from 0 to $M-1$. All other values can be converted to the corresponding value from the range using Modulo operation. For example for $M = 3$:

Type							
Normal Arithmetic	-3	-2	-1	0	1	2	3
Modular Arithmetic	0	1	2	0	1	2	0

Properties of Modular Arithmetic

There are four properties of Modular Arithmetic that we need to learn.

Addition

$$\text{if, } a \equiv b \pmod{m} \text{ and, } c \equiv d \pmod{m} \text{ then, } a + c \equiv b + d \pmod{m}$$

Meaning, if we have to find $(a_1 + a_2 + a_3 \ldots + a_n) \ \% \ m$, we can instead just find $((a_1 \ \% \ m) + (a_2 \ \% \ m) + \ldots (a_n \ \% \ m)) \ \% \ m$.

For example, $(13 + 24 + 44) \ \% \ 3 = (1 + 0 + 2) \ \% \ 3 = 3 \ \% \ 3 = 0$.

Subtraction

$$\text{if, } a \equiv b \pmod{m} \text{ and, } c \equiv d \pmod{m} \text{ then, } a – c \equiv b – d (\text{mod m})$$

Meaning, if we have to find $(a_1 - a_2 - a_3 \ldots - a_n) \ \% \ m$, we can instead just find $((a_1 \ \% \ m) – (a_2 \ \% \ m) - \ldots (a_n \ \% \ m)) \ \% \ m$.

For example, $(-14 – 24 – 44) \ \% \ 3 = (-2 – 0 – 2) \ \% \ 3 = -4 \ \% \ 3 = -1 \ \% \ 3 = 2$.

Multiplication

$$\text{if, } a \equiv b \pmod{m} \text{ and, } c \equiv d \pmod{m} \text{ then, } a \times c \equiv b \times d \pmod{m}$$

Meaning, if we have to find $(a_1 \times a_2 \times a_3 \ldots \times a_n) \ \% \ m$, we can instead just find $((a_1 \ \% \ m) \times (a_2 \ \% \ m) \times \ldots (a_n \ \% \ m)) \ \% \ m$.

For example, $(14 \times 25 \times 44) \ \% \ 3 = (2 \times 1 \times 2) \ \% \ 3 = 4 \ \% \ 3 = 1 \ \% \ 3 = 1$.

Division

Modular Division does not work same as the above three. We have to look into a separate topic known as Modular Inverse in order to find $\frac{a}{b} \ \% \ m$. Modular Inverse will be covered in a separate post.

For now just know that, $\frac{a}{b} \ not \equiv \frac{a \ \% \ m}{b \ \% \ m} \pmod{m}$. For example, $\frac{6}{3} \ \% \ 3$ should be 2. But using the formula above we get $\frac{0}{0} \ \% \ 3$ which is undefined.

Coding Tips

Handling Negative Numbers

In some programming language, when modulo operation is performed on negative numbers the result is also negative. For example in C++ we will get $-5 \ \% \ 4 = -1$. But we need to convert this into legal value. We can convert the result into legal value by adding M.

Therefore, in C++ it is better to keep a check for a negative number when doing modulo operation.

```
res = a % m;
if ( res < 0 ) res += m;
```

Modulo Operation is Expensive

Modulo Operation is as expensive as a division operator. It takes more time to perform division and modulo operations than to perform addition and subtraction. So it is better to avoid using modulo operation where possible.

One possible situation is when we are adding lots of values. Let's say that we have two variables a and b both between 0 to $m-1$. Then we can write:

```
res = a + b;
if ( res >= m ) res -= m;
```

This technique will not work when we are multiplying two values.

Resource

1. Wiki - Modular Arithmetic - https://en.wikipedia.org/wiki/Modular_arithmetic
2. AoPS - Introduction to Modular Arithmetic - http://www.artofproblemsolving.com/wiki/index.php/Modular_arithmetic/Introduction

Category: CPPS, Number Theory

Previous:
Divisor Summatory Function

Next:
Extended Euclidean Algorithm

Archives

- May 2019 (1)
- April 2019 (1)
- March 2019 (1)
- December 2018 (2)
- November 2018 (4)
- September 2018 (2)
- February 2018 (1)
- January 2018 (1)
- November 2017 (2)
- September 2015 (7)
- August 2015 (13)
- July 2015 (15)

Categories

- CPPS (45)
  - Combinatorics (4)
  - Data Structure (1)
  - Number Theory (36)
- Meta (1)
- Misc (4)

Recent Comments

My Shopee Interview – Shadman Protik on My Interview Experience with Shopee / Garena / Sea Group

Istiad Hossain Akib on SPOJ LCMSUM – LCM Sum

Rifat Chowdhury on MyStory#02 – Deciding Where to Study CS

Salman Farsi on Leading Digits of Factorial

Learning Notes on Multiplicative Functions, Dirichlet Convolution, Mobius Inversion, etc – RobeZH's Blog on SPOJ LCMSUM – LCM Sum