WEB3CLUBS FOUNDATION LIMITED

Course Instructor: DR. Cyprian Omukhwaya Sakwa
PHONE: +254723584205 Email: cypriansakwa@gmail.com

Foundational Mathematics for Web3 Builders

Implemented in RUST

Lecture 32

July 11, 2024

Modular arithmetic

There are many applications of modular arithmetic in computer science. Some of the applications include the construction of pseudo-random number generators, hashing Functions and Cryptology.

Definition 6

Let m be a positive integer. We say that the integers a and b are congruent modulo m (or $\mathrm{mod} m$) if $m \mid (a-b)$ and we write $a \equiv b (\mathrm{mod}\ m)$. If $m \nmid (a-b)$, then we write $a \not\equiv b (\mathrm{mod}\ m)$.

The relation $a \equiv b (\mathrm{mod}\ m)$ is a congruence relation, or simply, a congruence. The number m is called the modulus of the congruence. Two numbers are are said to be incongruent with respect to a given modulus m if they are not congruent with respect to that modulus m.

Example 21

a) $10 \equiv 4 \pmod 3$ since $3 \mid (10 - 4)$

b) $10 \equiv 1 \pmod 3$ since $3 \mid (10 - 1)$

c) $15 \not\equiv -5 \pmod 3$ since $3 \nmid (15 - -5)$ or $3 \nmid 20$

d) $22 \not\equiv 4 \pmod 5$ since $5 \nmid 18$

Given $a, b, m \in \mathbb{Z}$ with $m > 0$, we also say that b is congruent to $a \bmod m$ if $b = a + mt$ for some integer t.

Example 22

Which numbers are congruent to $3 \bmod 7$?

Solution

From above, $a \equiv 3 \bmod 7$ if $a = 3 + 7t$ for some integer t. Taking $t = \cdots, -4, -3, -2, -1, 0, 1, 2, 3, 4, \cdots$ we get

$\{\cdots, -25, -18, -11, -4, 3, 10, 17, 24, 31, \cdots\}$

Notice that all these numbers leave a remainder of 3 on division by 7.

Here's a Rust program that generates a specified number of positive and negative integers congruent to $3 \bmod 7$.

```rust
fn main() {
        let count = 9;
        // Specify the +ve and negative numbers to be generated

        let positive_numbers = generate_positive_congruent_numbers(3, 7, count);
        let negative_numbers = generate_negative_congruent_numbers(3, 7, count);

println!("Positive numbers congruent to 3 mod 7: {:?}", positive_numbers);
println!("Negative numbers congruent to 3 mod 7: {:?}", negative_numbers);
}

fn generate_positive_congruent_numbers(congruent: i32, modulus: i32, count: usize)
-> Vec<i32> {
        (0..count).map(|k| congruent + k as i32 * modulus).collect()
}

fn generate_negative_congruent_numbers(congruent: i32, modulus: i32, count: usize)
-> Vec<i32> {
        (1..=count).map(|k| congruent - k as i32 * modulus).collect()
}
```

Understanding the Rust code

1. The main Function
   - Sets the count (the +ve and -ve numbers to be generated).
   - generate_positive_congruent_numbers and generate_negative_congruent_numbers are called with the arguments 3 (congruence), 7 (modulus), and count
   - The generated sets of numbers are stored in positive_numbers and negative_numbers respectively. The results are printed.
2.

```
fn generate_positive_congruent_numbers(congruent: i32, modulus: i32, count: usize)
-> Vec<i32> {
        (0..count).map(|k| congruent + k as i32 * modulus).collect()
}
```

   - Generates a vector of positive numbers congruent to congruent modulo modulus.
   - It uses a range from 0 to count-1.
   - For each value k in the range, it computes the number as congruent + k * modulus.

Understanding the Rust code (conti...)

3.

```
fn generate_negative_congruent_numbers(congruent: i32, modulus: i32, count: usize)
-> Vec<i32> {
        (1..=count).map(|k| congruent - k as i32 * modulus).collect()
}
```

- generates a vector of negative numbers congruent to congruent modulo modulus.
- It uses a range from 1 to count.
- For each value k in the range, it computes the number as congruent - k * modulus.
- The result is collected into a vector and returned.

Definition 7

Given integers a and m, with $m > 0$, $a \bmod m$ is defined to be the remainder when a is divided by m.

Example 23

a) $14 \bmod 5 = 4$

b) $139 \bmod 10 = 9$

c) $-14 \bmod 5 = 1$

d) $1148 \bmod 5 = 3$

e) $-4 \bmod 9 = 5$

f) $(17 + 23) \bmod 5 \equiv 2 + 3 = 0$

g) $(18 + 23) \bmod 4 \equiv 2 + 3 = 1$

h) $(19 \times 288) \bmod 5 \equiv 4 \times 3 \equiv 12 \bmod 5 = 2$

i) $(11^2 \times 13^3) \bmod 4 \equiv (3^2 \times 1^3) \bmod 4 \equiv 1 \times 1 = 1$

Here is Rust code to obtain $a \bmod b$.

```rust
fn main() {
        // Use i64 to handle large integers
        let a: i64 = 14;
        let b: i64 = 5;

        // Compute a % b
        let result = a % b;//If a is negative you can modify this part as
        //let result=(a % b +b) % b//
//This is because, for instance -4 mod 5 would be -4 and so adding b (which is 5)
// ensures the result is non-negative. Applying the modulo operation again handles
//any possible overflow (this is necessary when the negative value is very large).


        println!("14 % 5 = {}", result);
}
```

This code can also handle $(a + c) \bmod b$.

The following Rust code does $(a^n \times b^t) \bmod m$.

```rust
fn main() {
        // Calculate (11^2 * 13^2) % 4
        let result = (11_i32.pow(2) * 13_i32.pow(2)) % 4;
        println!("The result is: {}", result);
}
```

For bigger values of a and b we can modify it to

```rust
use num::bigint::BigInt;
use num::traits::Pow;

fn main() {
        // Initialize BigInt values for base, exponent, and modulus
        let base1 = BigInt::from(19);
        let exp1 = 4_u32;
        let modulus1 = BigInt::from(5);

        let base2 = BigInt::from(288);
        let exp2 = 8_u32;
        let modulus2 = BigInt::from(5);

        // Calculate (base1^exp1) % modulus1
        let result1 = base1.pow(exp1 as usize) % &modulus1;

        // Calculate (base2^exp2) % modulus2
        let result2 = base2.pow(exp2 as usize) % &modulus2;
```

```
19
20          // Multiply the results and take the modulus again
21          let final_result = (result1 * result2) % modulus1.clone();
22          // Since modulus1 == modulus2
23
24          println!("The result is: {}", final_result);
25  }
```