WEB3CLUBS FOUNDATION LIMITED

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

Foundational Mathematics for Web3 Builders

Implemented in RUST

Lecture 39

August 1, 2024

Computing inverses of elements of a group

In the context of the additive group Z_n which consists of integers modulo n, the additive inverse of an element $a \in Z_n$ is an element $b \in Z_n$ such that $a + b \equiv 0 \bmod n$.

Example 69

Find the inverse of $7 \in Z_{19}$.

Solution

Inverse of 7 is $-7 \bmod 19 = 12 \bmod 19$.

Generally, for any element $a \in Z_n$, its additive inverse $b \in Z_n$ is given by $b = (n - a) \bmod n$

Here is the Rust code.

```rust
fn additive_inverse(a: i32, n: i32) -> i32 {
        (n - a % n) % n
}

fn main() {
        let n = 19;
        let a = 7;
        let inverse = additive_inverse(a, n);
        println!("The additive inverse of {} in Z_{} is {}", a, n, inverse);
}
```
.

Rust code for Inverses of elements of the group $\mathbb{Z}_n^*$

Consider the group of units modulo 42 given by $\mathbb{Z}_{42}^* = \{1, 5, 11, 13, 17, 19, 23, 25, 29, 31, 37, 41\}$. We can multiply and divide the elements of this set without leaving the set. It is simple to verify the first three properties of the group. The identity element equals 1. Each element has an inverse, which may be obtained through the extended Euclidean algorithm.

Let us for instance find the inverse of $41 \in \mathbb{Z}_{42}^*$.

Solution

$$42 = 41(1) + 1$$

$$41 = 1(41) + 0$$

So the $\gcd(41, 42) = 1$ and reversing the first equation we obtain;
$1 = 42 - 41(1)$ or $1 = 42 + 41(-1)$ or $1 = 42 + 41(41)$
Thus, $41^{-1} \equiv 41 \mod 42$.

Example 70

Find $7^{-1} \in \mathbb{Z}_{19}^*$

Solution

By Euclid's algorithm algorithm we have

$19 = 7(2) + 5$

$7 = 5(1) + 2$

$5 = 2(2) + 1$

$2 = 1(2) + 0$

We solve for $\gcd$

$1 = 5 - 2(2) = 5 - [7 - 5(1)](2) = 5 - 7(2) + 5(2) = -7(2) + 5(3)$

$= -7(2) + [19 - 7(2)](3) = -7(2) + 19(3) - 7(6)$

$= 19(3) + 7(-8)$

That is, $1 = 19(3) + 7(-8)$.

Thus, $7^{-1} = -8 \equiv 11 \mod 19$

Example 71

Find $364^{-1} \in \mathbb{Z}^*_{765}$

Solution

This question wants us to find $364^{-1} \bmod 765$

$$765 = 364(2) + 37$$

$$364 = 37(9) + 31$$

$$37 = 31(1) + 6$$

$$31 = 6(5) + 1$$

Now solve for $\gcd$

$1 = 31 - 6(5) = 31 - [37 - 31(1)](5)$

$= -37(5) + 31(6) = -37(5) + [364 - 37(9)](6)$

$= 364(6) - 37(59) = 364(6) - [765 - 364(2)](59)$

$= 765(-59) + 364(124)$

Hence $1 = 765(-59) + 364(124)$. Thus $x = 124$

The following Rust code defines two functions, extended_gcd and mod_inverse, and demonstrates their use in the main function to compute multiplicative modular inverses.

```
        fn extended_gcd(a: i64, b: i64) -> (i64, i64, i64) {
                if b == 0 {
                        (a, 1, 0)
                }
                else {
                        let (g, x, y) = extended_gcd(b, a % b);
                        (g, y, x - (a / b) * y)
                }
        }

        fn mod_inverse(a: i64, m: i64) -> Option<i64> {
                let (g, x, _) = extended_gcd(a, m);
                if g != 1 {
                        None // No modular inverse if gcd(a, m) != 1
                } else {
                        Some((x % m + m) % m) // Ensure the result is positive
                }
        }

        fn main() {
                let a = 364;
                let m = 765;
                match mod_inverse(a, m) {
Some(inv) => println!("The␣modular␣inverse␣of␣{}␣modulo␣{}␣is␣{}", a, m, inv),
        None => println!("The␣modular␣inverse␣does␣not␣exist"),
                }
        }
```

Example 72

compute inverse of $1234567890123456789012345678901234567891$ modulo $9876543210987654321098765432109876543219$ if it exists.

Solution

The following Rust code offers functions for calculating the modular inverse of huge integers with the num-bigint crate, which can handle arbitrarily large integers. Running this code we find that the modular inverse is $775570818839977857515581324586738336533$.

```
use num_bigint::BigInt;
use num_traits::{One, Zero};
use std::str::FromStr;

// Function to calculate the modular inverse
fn mod_inverse(a: &BigInt, m: &BigInt) -> Option<BigInt> {
        let (g, x, _) = extended_gcd(a.clone(), m.clone());
        if g.is_one() {
                Some((x % m + m) % m)  // Ensure the result is positive
        } else {
                None  // No inverse exists if gcd(a, m) != 1
        }
}
```

```

        // Extended Euclidean Algorithm
        fn extended_gcd(a: BigInt, b: BigInt) -> (BigInt, BigInt, BigInt) {
                if b.is_zero() {
                        (a, BigInt::one(), BigInt::zero())
                } else {
let (g, x, y) = extended_gcd(b.clone(), a.clone() % b.clone());
                        (g, y.clone(), x - (a / b) * y)
                }
        }

        fn main() {
let a = BigInt::from_str("123456789012345678901234567890123456789 1").unwrap();
let m = BigInt::from_str("987654321098765432109876543210987654319").unwrap();

                match mod_inverse(&a, &m) {
Some(inv) => println!("The␣modular␣inverse␣is:␣{}", inv),
None => println!("No␣modular␣inverse␣exists␣for␣the␣given␣input."),
                }
        }
```

Understanding the Rust code

1. extended_gcd function uses the Extended Euclidean Algorithm to compute the greatest common divisor $\gcd$ of two integers a and b, as well as determine coefficients x and y such that $\gcd(a, b) = ax + by$.

```rust
fn extended_gcd(a: i64, b: i64) -> (i64, i64, i64) {
        if b == 0 {
                (a, 1, 0)
        } else {
                let (g, x, y) = extended_gcd(b, a % b);
                (g, y, x - (a / b) * y)
        }
}
```

- Base case: If b is zero, the function returns (a, 1, 0) because:
  - ✓ For instance, the $\gcd$ of a and 0 is a
  - ✓ The coefficients are 1 and 0 because $a \cdot 1 + 0 \cdot 0 = a$

```rust
        if b == 0 {
                (a, 1, 0)
        }
```

Understanding the Rust code (conti...)

- Recursive Case: The function calls itself with b and a % b. This continues until b becomes zero.
- Calculating Coefficients:
  - ✓ The recursive call returns the $\gcd$ and the coefficients x_1 and y_1 for the equation involving b and a % b.
  - ✓ The new coefficients x and y are calculated using

    $$x = y_1$$

    $$y = x_1 - (a/b) \cdot y_1$$
- The base case ensures the function terminates when the second number becomes zero.

Understanding the Rust code (conti...)

(2) mod_inverse function computes the modular inverse of a modulo m, which is a number x with $ax \equiv 1 \bmod m$.

```
fn mod_inverse(a: i64, m: i64) -> Option<i64> {
        let (g, x, _) = extended_gcd(a, m);
        if g != 1 {
                None // No modular inverse if gcd(a, m) != 1
        } else {
                Some((x % m + m) % m) // Ensure the result is positive
        }
}
```

- It uses the extended_gcd function to get the $\gcd$ and the coefficient x.

```
        let (g, x, _) = extended_gcd(a, m);
```

- Checks for Inverse Existence. If the $\gcd$ is not 1, then a and m are not coprime, and there is no modular inverse.

```
        if g != 1 {
                None // No modular inverse if gcd(a, m) != 1
```

Understanding the Rust code (conti...)

- Else, Return Positive Modular Inverse.

```
else {
        Some((x % m + m) % m) // Ensure the result is positive
```

Example 73

Find $31^{-1} \bmod 60$.

- **Main function.**

✓ The main function initializes a to 31 and m to 60.

✓ It calls mod_inverse(a, m) with $a = 31$ and $m = 60$.

✓ mod_inverse Function calls extended_gcd(a, m) to get the $\gcd$ and coefficients.

Understanding the Rust code (conti...)

- **Extended Euclidean Function works recursively.**

1. Initial Call: extended_euclidean(31, 60)

✓ Since $b \neq 0$, the function makes a recursive call:
extended_euclidean(60, 31 % 60)
31 % 60 is 31, so it calls extended_euclidean(60, 31)

2. First Recursive Call: extended_euclidean(60, 31)

✓ Since $b \neq 0$, the function makes first recursive call:
extended_euclidean(31, 60 % 31)
60 % 31 is 29, so it calls extended_euclidean(31, 29)

3. Second Call: extended_euclidean(31, 29)

✓ Since $b \neq 0$, the function makes another recursive call:
extended_euclidean(29, 31 % 29)
31 % 29 is 2, so it calls extended_euclidean(29, 2)

Understanding the Rust code (conti...)

4. Third Call: extended_euclidean(29, 2)

✓ Since $b \neq 0$, the function makes another recursive call: extended_euclidean(2, 29 % 2)
29 % 2 is 1 so it calls extended_euclidean(2, 1)

5. Fourth Call: extended_euclidean(2, 1)

✓ Since $b \neq 0$, the function makes another recursive call: extended_euclidean(1, 2 % 1)
2 % 1 is 0, so it calls extended_euclidean(1, 0)

6. Base Case: extended_euclidean(1, 0)

- Since $b = 0$ it returns (1, 1, 0)
- This means the $\gcd$ is 1, and the coefficients x and y are 1 and 0, respectively.

Understanding the Rust code (conti...)

Unwinding the Recursion: Now unwind the recursion and calculate the coefficients for each step.

a) Returning from extended_euclidean(2, 1)

✓ It receives (1, 1, 0) from the base case.

✓ It calculates new coefficients x and y.

$$x = y_1 = 0$$

$$y = x_1 - (a/b) * y_1 = 1 - (2/1) * 0 = 1$$

It returns (1, 0, 1)

b) Returning from extended_euclidean(29, 2)

✓ It receives (1, 0, 1) from the previous step..

✓ It calculates new coefficients x and y.

$$x = y_1 = 1, \quad y = x_1 - (a/b) * y_1 = 0 - (29/2) * 1 = -14$$

It returns (1, 1, -14)

Understanding the Rust code (conti...)

c) Returning from extended_euclidean(31, 29)

✓ It receives (1, 1, -14) from the previous step..

✓ It calculates new coefficients x and y.

$x = y_1 = -14$

$y = x_1 - (a/b) * y_1 = 1 - (31/29) * -14 = 15$

It returns (1, -14, 15)

d) Returning from extended_euclidean(60, 31)

✓ It receives (1, -14, 15) from the previous step..

✓ It calculates new coefficients x and y.

$x = y_1 = 15$

$y = x_1 - (a/b) * y_1 = -14 - (60/31) * 15 = -29$

Returns (1, 15, -29)

So, extended_gcd(31, 60) returns (1, -29, 15)

Understanding the Rust code (conti...)

Back to mod_inverse

Since g = 1, it computes the modular inverse

(x % m + m) % m = (-29 % 60 + 60) % 60 = 31

The modular inverse is 31

Example 74

Find all solutions to $17x \equiv 1 \pmod{29}$

Solution

We are required to determine $17^{-1} \bmod 29$

- **Main function.**

✓ The main function initializes a to 17 and m to 29.

✓ It calls mod_inverse(a, m) with $a = 17$ and $m = 29$.

✓ mod_inverse Function calls extended_gcd(a, m) to get the $\gcd$ and coefficients.

- **Extended Euclidean Function works recursively.**

1. Initial Call: extended_euclidean(17, 29)

✓ Since $b \neq 0$, the function makes a recursive call: extended_euclidean(29, 17 % 29)=(29,17)

Solution (conti...)

2. First Recursive Call: extended_euclidean(29, 17)

✓ Since $b \neq 0$, the function makes first recursive call: extended_euclidean(17, 29 % 17)=(17, 12)

3. Second Call: extended_euclidean(17, 12)

✓ Since $b \neq 0$, the function makes another recursive call: extended_euclidean(12, 17 % 12)=(12, 5)

4. Third Call: extended_euclidean(12, 5)

✓ Since $b \neq 0$, the function makes another recursive call: extended_euclidean(5, 12 % 5)=(5, 2)

5. Fourth Call: extended_euclidean(5, 2)

✓ Since $b \neq 0$, the function makes another recursive call: extended_euclidean(2, 5 % 2)=(2, 1)

Solution (conti...)

6. Fifth Call: extended_euclidean(2, 1)

✓ Since $b \neq 0$, the function makes another recursive call:
extended_euclidean(1, 2 % 1)
2 % 1 is 0, so it calls extended_euclidean(1, 0)

7. Base Case: extended_euclidean(1, 0)

- Since $b = 0$ it returns (1, 1, 0)
- This means the $\gcd$ is 1, and the coefficients x and y are 1 and 0, respectively.

Solution (conti...)

Unwinding the Recursion: Now unwind the recursion and calculate the coefficients for each step.

a) Returning from extended_euclidean(2, 1)

✓ It receives (1, 1, 0) from the base case.

✓ It calculates new coefficients x and y.

$x = y_1 = 0$

$y = x_1 - (a/b) * y_1 = 1 - (2/1) * 0 = 1$

It returns (1, 0, 1)

b) Returning from extended_euclidean(5, 2)

✓ It receives (1, 0, 1) from the previous step..

✓ It calculates new coefficients x and y.

$x = y_1 = 1, \quad y = x_1 - (a/b) * y_1 = 0 - (5/2) * 1 = -2$

It returns (1, 1, -2)

Solution (conti...)

c) Returning from extended_euclidean(12, 5)

✓ It receives (1, 1, -2) from the previous step..

✓ It calculates new coefficients x and y.

$x = y_1 = -2$

$y = x_1 - (a/b) * y_1 = 1 - (12/5) * -2 = 5$

It returns (1, -2, 5)

d) Returning from extended_euclidean(17, 12)

✓ It receives (1, -2, 5) from the previous step..

✓ It calculates new coefficients x and y.

$x = y_1 = 5$

$y = x_1 - (a/b) * y_1 = -2 - (17/12) * 5 = -7$

Returns (1, 5, -7)

Solution (conti...)

e) Returning from extended_euclidean(29, 17)

✓ It receives (1, 5, -7) from the previous step..

✓ It calculates new coefficients x and y.

$x = y_1 = -7$

$y = x_1 - (a/b) * y_1 = 5 - (29/17) * -7 = 12$

Returns (1, -5, 12)

So, extended_gcd(17, 29) returns (1, 12, -5)

Back to mod_inverse

Since g = 1, it computes the modular inverse

(x % m + m) % m = (12 % 29 + 29) % 29 = 12

The modular inverse is 12

Exercise 5

1. Use Rust code to find the inverses of the following modulo 779854321098765432109876543210987683313

   a) 79765431929 b) 98765432063 c) 79765431853

   d) 98765432081 e) 79765431901 f) 98765432099

2. Find the inverse of $21888242871839275222246405745257275088$ $696311157297823662689037894645226208583 \bmod$ $7133888888888882828282828777777777777774$.