ECE 404 Homework 3

Ranjan Behl (rbehl)

February 11, 2021

1. Written HW: Check next page

Ranjan Behl

ECE 404 Homework #3

Due: Thursday 02/11/2021 at 5:59PM

This homework covers topics related to finite fields.

Theory Problems

Solve the following problems.

1. Show whether or not the set of remainders Z_{18} forms a group with the modulo *addition* operator. Then show whether or not Z_{18} forms a group with the modulo *multiplication* operator.

2. Compute $gcd(36459, 27828)$ using Euclid's algorithm. Show all of the steps.

3. Is the set of all unsigned integers W a group under the $gcd(\cdot)$ operation? Why or why not? (**Hint**: Find the identity element for $\{W, gcd(\cdot)\}$.)

4. Use the Extended Euclid's Algorithm to compute by hand the multiplicative inverse of 27 in Z_{32} . List all of the steps.

5. In the following, find the smallest possible integer x. Briefly explain (i.e. you don't need to list out all of the steps) how you found the answer to each. You should solve them *without* simply plugging in arbitrary values for x until you get the correct value :

   (a) $9x \equiv 11 \pmod{13}$

   (b) $6x \equiv 3 \pmod{23}$

   (c) $5x \equiv 9 \pmod{11}$

Programming Problem

Rewrite and extend the Python (or Perl) implementation of the *binary* GCD algorithm presented in Section 5.4.4 so that it incorporates the Bezout's Identity to yield multiplicative inverses. In other words, create a binary version of the multiplicative-inverse script of Section 5.7 that finds the answers by implementing the multiplications and division as bit shift operations.

Your script should be named `mult_inv.py/pl` and accept two command-arguments:

```
mult_inv.py a b
```

Which should print the multiplicative inverse of `a` mod `b`

$\forall a, b \in Z_{18}$

$(a + b) \in Z_{18}$

1. Show whether or not the set of remainders Z_{18} forms a group with the modulo *addition* operator. Then show whether or not Z_{18} forms a group with the modulo *multiplication* operator.

Is $Z_{18} = \{Z_{18}, +\}$

1) Closure ?

$\forall a, b, n \in Z_{18}$,

$(a+b) \in Z_{18} \because (n+b) = (a+b) \bmod n$

$= [a \bmod n + b \bmod n] \bmod n$

2) Associativity ?

$\forall a, b, c, n \in Z_{18}$,

$(a+b)+c = [[(a \bmod n + b \bmod n) \bmod n] \bmod n + c \bmod n] \bmod n$

$a+(b+c) = [a \bmod n + [b \bmod n + c \bmod n) \bmod n]] \bmod n$

This shows + is associativity

3) Identity element ?

$\exists e \in Z_{18}$ s.t.

$\forall a, n Z_{18}$,

$a+e \equiv a \pmod n$

e.g.) let $a = 5$, $n = 2$, and $e = 0$

then $5 \equiv 5 \pmod 2 \Rightarrow 5 \bmod 2 = 5 \bmod 2 \Rightarrow 3 = 3$ ✓

$\therefore$ the identity element e must be 0.

4) Inverse element ?

$\forall a, n \in Z_{18}, \exists b \in Z_{18}$ s.t.

$n + b = b + a = e$

since

$a + b = (a+b) \bmod n = e = 0 \therefore b = -a \because$

$[a + (-a)] \bmod n = 0 \bmod n = 0$

Thus Z_{18} has been shown to have the properties of closure, associativity, identity element and an inverse element. Therefore it is a group under modular addition.

Is $Z_{18} = \{Z_{18}, *\}$?

Z_{18} is not a group under modular multiplication because it has no inverse element.

let $b = a^{-1}$ then $(a)(b) \equiv ? 1 \bmod(n)$ where 1 is the identity element (same as regular multiplication).

$(a)(b) \equiv ? 1 \bmod(n)$

e.g) let $a = 4$, $n = 9$ then $b^{-1} = ?$

$4(b) \equiv 1 \bmod 9$

$4b \bmod 9 = 1 \bmod 9$

$b = 7$

$4(9) \bmod 9 = 28 \bmod 9 = 1 = 1 \bmod 9$ ✓

however if $a = 6$ then no such b can be found. Therefore since the inverse elements exists only for certain elements and not for Z_{18}. Therefore Z_{18} can't be a group under modular multiplication.

2. Compute $gcd(36459, 27828)$ using Euclid's algorithm. Show all of the steps.

$gcd(36459, 27828) = gcd(27828, 36459 \bmod 27828)$
$= gcd(27828, 8631) = gcd(8631, 27828 \bmod 8631)$
$= gcd(8631, 1935) = gcd(1935, 8631 \bmod 1935)$
$= gcd(1935, 891) = gcd(891, 1935 \bmod 891)$
$= gcd(891, 153) = gcd(153, 891 \bmod 153)$
$= gcd(153, 126) = gcd(126, 153 \bmod 126)$
$= gcd(126, 27) = gcd(27, 126 \bmod 27)$
$= gcd(27, 18) = gcd(18, 27 \bmod 18)$
$= gcd(18, 9) = gcd(9, 18 \bmod 9)$

$18 \bmod 9 = 0$

$\Rightarrow \boxed{gcd(36459, 27828) = 9}$

3. Is the set of all unsigned integers $\mathbb{W}$ a group under the $gcd(\cdot)$ operation? Why or why not? (**Hint**: Find the identity element for $\{\mathbb{W}, gcd(\cdot)\}$.)

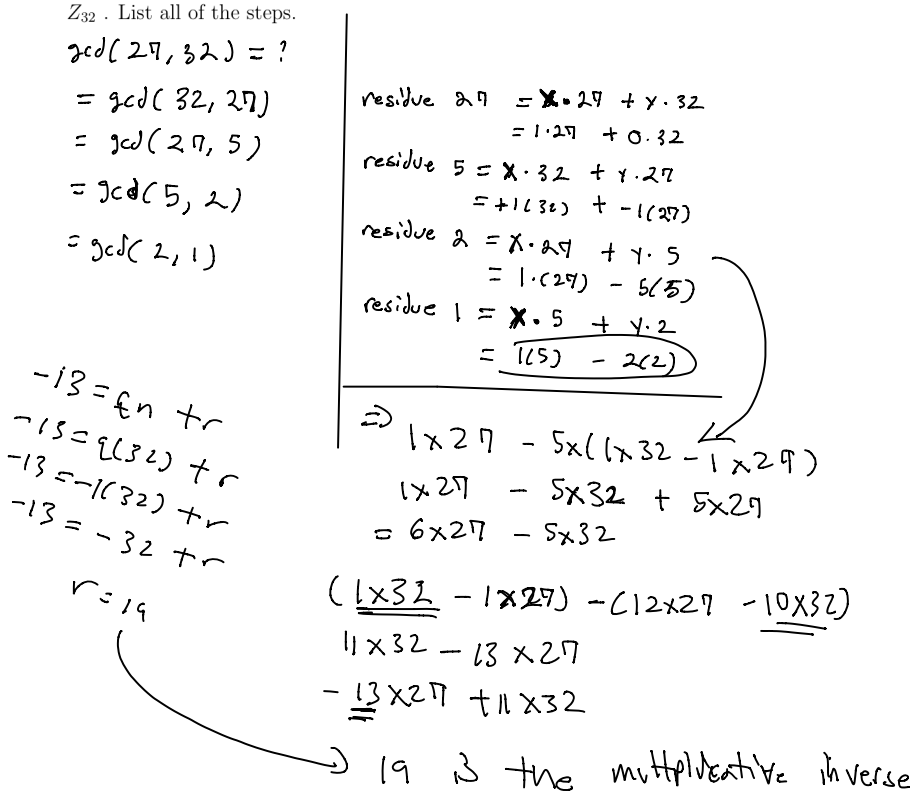
$\exists e \in W$ s.t.
$\forall a \in W$, where $gcd(a,e) = gcd(e,a) = a$.

let $a = 5$
$gcd(5,e) = gcd(e,5) = gcd(5, e \bmod 5)$
$= gcd(e \bmod 5, 5 \bmod (e \bmod 5)) = \ldots = a$

based on this $a|a$ & $a|e$ must be true.
$a|e \Rightarrow e = ax$ for some integer X.
Since $X \in \mathbb{Z}$, X can be signed, making e be signed. $\therefore$ W ~~can~~ cannot be a group

under gcd(·) b/c the identity element
can be signed.

4. Use the Extended Euclid's Algorithm to compute by hand the multiplicative inverse of 27 in Z_{32} . List all of the steps.



5. In the following, find the smallest possible integer x. Briefly explain (i.e. you don't need to list out all of the steps) how you found the answer to each. You should solve them *without* simply plugging in arbitrary values for x until you get the correct value :

   (a) $9x \equiv 11 \pmod{13}$

   (b) $6x \equiv 3 \pmod{23}$

   (c) $5x \equiv 9 \pmod{11}$

a) $9x \bmod 13 = 11 \bmod 13 = 11$

$\gcd(9,13) = \gcd(13,9)$

$= \gcd(9,4)$

$= \gcd(4,1)$

$X = 3 \cdot 11 \bmod 13$

$= 33 \bmod 13$

$= \boxed{7}$

residue $4 = 1(13) + (-1)(9)$

residue $1 = 1(9) - 2(4)$

$\Rightarrow 1(9) - 2(1(13) + (-1)(9))$

$\Rightarrow 1(9) - 2(13) + 2(9)$

$\Rightarrow 3(9) - 2(13)$

$\Rightarrow m_1$

b) $6x \equiv 3 \pmod{23}$

$\gcd(6,23) = \gcd(23,6)$

$= \gcd(6,5)$

$= \gcd(5,1)$

$X = 4 \cdot 3 \bmod 23$

$= 12 \bmod 23 = \boxed{12}$

residue $5 = 1 \times 23 + (-3) \times 6$

residue $1 = 1 \times 6 + (-1)(5)$

$= 1 \times 6 + (-1)(1 \times 23 + (-3) \times 6)$

$= 4 \times 6 + (-1) \times 23$

$\Rightarrow m_1$

c) $5x \equiv 9 \pmod{11}$

$\gcd(5, 11) = \gcd(11, 5)$

$= \gcd(5, 1)$

$x = 9 \cdot 9 \mod 11$

$= 81 \mod 11$

$= \boxed{4}$

residue $1 = 1 \times 11 + (-2) \times 5$

$= -2 \times 5 + 1 \times 11$

$-2 \mod 11 = 9$

2. Coding HW

FOR HW 3 replace // and * with bitwise operations

3. Explanation of the Code

The code I used is shown below:

```
import sys
from operator import xor

def main():
    if len(sys.argv) != 3:
        sys.stderr.write("Usage: %s   <integer>   <modulus>\n" % sys.argv[0])
        sys.exit(1)

    NUM =  int(sys.argv[1])
    MOD =  int(sys.argv[2])
    MI(NUM,MOD)

def MI(num, mod):
    '''
    This function uses ordinary integer arithmetic implementation of the
    Extended Euclid's Algorithm to find the MI of the first-arg integer
    vis-a-vis the second-arg integer.
    '''
    NUM = num; MOD = mod
    x, x_old = 0, 1
    y, y_old = 1, 0
    while mod:
        q = divide(num,mod)
        num, mod = mod, num % mod
        x, x_old = x_old - multiply(q,x), x
        y, y_old = y_old - multiply(q,y), y
    if num != 1:
        print("\nNO MI. However, the GCD of %d and %d is %u\n" % (NUM, MOD, num))
    else:
        MI = (x_old + MOD) % MOD
        print("\nMI of %d modulo %d is: %d\n" % (NUM, MOD, MI))

#MI(NUM, MOD)

def multiply(x, y):
    #check for the sign
    if(xor((x < 0),(y < 0))):
        sign = -1
    else:
        sign = 1
```

```
    # since we have the sign, we can remove the negatives
    x = abs(x)
    y = abs(y)

    result = 0
    while(y != 0):
        if(y & 1):
            #result = add(result,x) #if y is odd add x to result
            if(sign == -1):
                result = result - x
            else:
                result = result + x
        #print("This is x\n",x)
        #if(x < 0):
         #   print("error")
        x  = x << 1
        y  = y >> 1
    return result


def divide(x,y):
     #check for the sign
    if(xor((x < 0),(y < 0))):
        sign = -1
    else:
        sign = 1

    # since we can the sign we can remove the negatives
    x = abs(x)
    y = abs(y)

    #edge case
    if(y == 1):
        return multiply(sign,x)
    #normal case
    result = 0
    while(x >= y):
        x = x - y
        result = result + 1

    return multiply(sign,result)


if __name__ == "__main__":
    main()
```

I reused all of the code provided in FINDMI.py and the only thing I changed was how // and * worked, also I added a main function for commandline arguments checking. I created a function called multiply that uses bit wise addiction to carry out the "multiplication". I was able to get it work for positive numbers however negatives numbers were causing me trouble so I had to google that portion. I found a article on geeks4geeks that kept the sign of final result in a separate variable. I really liked this idea as it allowed me to check for negative integers and instead of adding I would subtract x from the result. Where x was the first parameter

of the function. For division I used something similar with the sign, and I called my multiply function in division. The division function just carry out long division and in the end multiply the final result with the correct sign(if the two original numbers were negative then multiply by -1).