CSC110 Lecture 17: Modular Arithmetic

Print this handout

For your reference, here is the definition of modular equivalence.

Let $a, b, n \in \mathbb{Z}$, with $n \neq 0$. We say that a **is equivalent to** b **modulo** n when $n \mid a - b$. In this case, we write $a \equiv b \pmod{n}$.

Exercise 1: Modular arithmetic practice

1. Expand the statement $14 \equiv 9 \pmod{5}$ into a statement using the divisibility predicate. Is this statement True or False?

   $5 \mid (14-9)$

   $5 \mid 5$

   True since $5 = 1 \cdot 5$

2. Expand the statement $9 \equiv 4 \pmod{3}$ into a statement using the divisibility predicate. Is this statement True or False?

   $3 \mid (9-4)$

   $3 \mid 5$

   False since $5 = 3k$ for no $k \in \mathbb{Z}$.

3. Prove the following statement using *only* the definitions of divisibility and modular equivalence (and no other statements/theorems):

   - $\forall a, b, c \in \mathbb{Z},\ \forall n \in \mathbb{Z}^+,\ a \equiv b \pmod{n} \Rightarrow ca \equiv cb \pmod{n}$

Let $a, b, c \in \mathbb{Z}$ and let $n \in \mathbb{Z}^+$.

Assume $a \equiv b \pmod{n}$

That is $n \mid a-b$

or $\exists k_1 \in \mathbb{Z},\ a-b = k_1 n$

WTS $ca \equiv cb \pmod{n}$ That is, $n \mid (ca-cb)$

or $\exists k_2 \in \mathbb{Z},\ ca-cb = k_2 n$

Since $a-b = k_1 n$, $c(a-b) = ck_1 n$

or $ca-cb = (ck_1)n$. Let $k_2 = ck_1$

then $ca-cb = k_2 n$, as req'd.

Exercise 2: Modular division

In lecture, we proved the following theorem about the existence of modular inverses. For your reference, we've also included an abridged proof with just the key steps shown.

Modular inverse theorem:

$\forall n \in \mathbb{Z}^+,\ \forall a \in \mathbb{Z},\ \gcd(a,n) = 1 \Rightarrow (\exists p \in \mathbb{Z},\ ap \equiv 1 \pmod{n})$.

the modular inverse

Key proof steps:

- Assuming $\gcd(a,n) = 1$, by the GCD Characterization Theorem there exist $p, q \in \mathbb{Z}$ such that $1 = ap + qn$.
- Then $qn = 1 - ap$
- Then $ap \equiv 1 \pmod{n}$.

Now, you'll turn this proof into an algorithm. In the code below, we've provided the `extended_euclidean_gcd` function from last class, as well as the specification for a new `modular_inverse` function. Your task is to complete `modular_inverse` by *writing appropriate precondition(s)* and then *writing the function body*. Recall that last class, we implemented the following function:

```python
def extended_euclidean_gcd(a: int, b: int) -> tuple[int, int, int]:
    """Return the gcd of a and b, and integers p and q such that
    gcd(a, b) == p * a + b * q.

    Preconditions:
    - a >= 0
    - b >= 0

    >>> extended_euclidean_gcd(10, 3)
    (1, 1, -3)
    """
    x, y = a, b

    px, qx = 1, 0
    py, qy = 0, 1

    while y != 0:
        # assert math.gcd(x, y) == math.gcd(a, b)  # L.I. 1
        assert x == px * a + qx * b                # L.I. 2
        assert y == py * a + qy * b                # L.I. 3

        q, r = divmod(x, y)  # equivalent to q, r = (x // y, x % y)
        x, y = y, r

        px, qx, py, qy = py, qy, px - q * py, qx - q * qy

    return x, px, qx


def modular_inverse(a: int, n: int) -> int:
    """Return the inverse of a modulo n, in the range 0 to n - 1 inclusive.

    Preconditions:  (TODO: fill this in!)
```

- $\gcd(a,n) == 1$
- $n > 0$

what do we need to be guaranteed that it exists?

```
>>> modular_inverse(10, 3)  # 10 * 1 is equivalent to 1 modulo 3
1
>>> modular_inverse(3, 10)  # 3 * 7 is equivalent to 1 modulo 10
7
"""
# TODO: implement this function!
```

Since $\gcd(a,n) = 1$, we know $\exists\, p, q \in \mathbb{Z}$
s.t. $1 = \boxed{p} \cdot a + q \cdot n$ ← unique p, q

or $pa = 1 - q \cdot n$

or $pa \equiv 1 - qn \pmod{n}$

or $pa \equiv 1 \pmod{n}$ since $qn \equiv 0 \bmod n$

↑ is a modular inverse but not nec. in range(0, n−1)

so result = extended_euclidean_gcd(a, n)

p = result[1]

The returned p may not be in $\{0, \ldots, n-1\}$ but p % n will be so and $p \equiv p\%n \pmod{n}$

Additional exercises

return p % n

see below for justification.

1. Using only the definition of divisibility and the definition of congruence modulo n, prove the following statements.

   let b represent the reciprocal of a (modulo n)

   a. $\forall a, b, c, d \in \mathbb{Z},\ \forall n \in \mathbb{Z}^+,\ a \equiv b \pmod{n} \land c \equiv d \pmod{n} \Rightarrow a + c \equiv b + d \pmod{n}$

   then $a \cdot b \equiv 1 \pmod{n}$

   b. $\forall a, b \in \mathbb{Z}, \forall n \in \mathbb{Z}^+,\ (0 \le a < n) \land (0 \le b < n) \land (a \equiv b \pmod{n}) \Rightarrow a = b.$

   but $a \cdot b + q \cdot n$ need not $= 1$

2. Our version of the *Modular inverse theorem* used an implication: *if* $\gcd(a, n) = 1$ then a has an inverse modulo n. But it turns out that the converse is true as well, so the full Modular inverse theorem is really an if and only if!

   Using the GCD characterization theorem precisely, prove this converse form:

   $$\forall n \in \mathbb{Z}^+,\ \forall a \in \mathbb{Z},\ \big(\exists p \in \mathbb{Z},\ ap \equiv 1 \pmod{n}\big) \Rightarrow \gcd(a, n) = 1$$

3. Implement the following function, which is the modular analog of division. Use your

↳ The following argument may help to understand why we return p%n and not p.

The extended gcd algorithm will return the unique p, q such that

$$1 = p \cdot a + q\,n$$

but that p might not be in

$$\{0, 1, 2, \ldots, n-1\}$$

Let's apply the quotient remainder Theorem to p, n

We can write

this is == p%n

$$p = \hat{q} \cdot n + \hat{r}$$

for some $\hat{q} \in \mathbb{Z}$ and $\hat{r} \in \{0, 1, 2, \ldots, n-1\}$

We know $pa \equiv 1 \pmod{n}$

so $(\hat{q}n + \hat{r})\,a \equiv 1 \pmod{n}$

and $\hat{q}na + \hat{r}\,a \equiv 1 \pmod{n}$

but $\quad \hat{q}na \equiv 0 \pmod{n}$

so $\quad \hat{q}na + \hat{r}a - \hat{q}na \equiv 1 - 0 \pmod{n}$

or $\quad \hat{r}a \equiv 1 \pmod{n}$

or $\quad a\hat{r} \equiv 1 \pmod{n}$

Also $\quad \hat{r} \in \{0, 1, 2, \ldots, n-1\}$

so the function should return

$$\hat{r}$$

(which is equal to P % n

or result[1] % n)

!!!

`modular_inverse` function from above. Once again, figure out what the necessary precondition(s) are for this function.

```
def modular_divide(a: int, b: int, n: int) -> int:
    """Return an integer k such that ak is equivalent to b modulo n.

    The return value k should be between 0 and n-1, inclusive.

    Preconditions:



    >>> modular_divide(7, 6, 11)  # 7 * 4 is equivalent to 6 modulo 11
    4
    """
```

4. (*Modular exponentiation and order*) Consider modulo 5, which has the possible remainders $0, 1, 2, 3, 4$. In each table, fill in the value for remainder b, where $0 \le b < 5$, that makes the modular equivalence statement in each row True. The first table is done for you.

   Use Python as a calculator if you would like to. (Or write a comprehension to calculate them all at once!)

   a. Powers of 2.

Power of 2	**Value for b**
$2^1 \equiv b \pmod 5$	2
$2^2 \equiv b \pmod 5$	4
$2^3 \equiv b \pmod 5$	3
$2^4 \equiv b \pmod 5$	1
$2^5 \equiv b \pmod 5$	2
$2^6 \equiv b \pmod 5$	4

   b. Powers of 3.

Power of 3	**Value for b**
$3^1 \equiv b \pmod 5$	

Power of 3	**Value for b**
$3^2 \equiv b \pmod{5}$	
$3^3 \equiv b \pmod{5}$	
$3^4 \equiv b \pmod{5}$	
$3^5 \equiv b \pmod{5}$	
$3^6 \equiv b \pmod{5}$	

c. Powers of 4.

Power of 4	**Correct value for b**
$4^1 \equiv b \pmod{5}$	
$4^2 \equiv b \pmod{5}$	
$4^3 \equiv b \pmod{5}$	
$4^4 \equiv b \pmod{5}$	
$4^5 \equiv b \pmod{5}$	
$4^6 \equiv b \pmod{5}$	

d. Using the tables above, write down the *order* of 2, 3, and 4 modulo 5:

n	$\mathrm{ord}_5(n)$
2	
3	
4	