Modular Arithmetic

Modular arithmetic is a useful algebraic transformation technique. It allows operations like addition, subtraction and multiplication to be performed parallely whereas the normal methods for doing arithmetic are serial. Efficient algorithms to manipulate large numbers can be found using modular arithmetic.

In this section we study how modular arithmetic could be used to efficient multiplication of large integers. The process for integer multiplication using modular arithmetic is shown in figure given below.

$$\begin{tikzcd}
\text{integers} \arrow[r, "\substack{\text{conventional}\\ \text{multiplication}}"] \arrow[d, "\text{division}"'] & \substack{\text{integer}\\ \text{products}} \\
\substack{\text{integers}\\ \bmod p} \arrow[r, "\substack{\bmod p\\ \text{multiplication}}"'] & \substack{\text{products}\\ \bmod p} \arrow[u, "\substack{\text{Chinese Remainder}\\ \text{Algorithm}}"']
\end{tikzcd}$$

Figure 9.3 Integer multiplication by mod p transformations

Introduction:

In modular arithmetic, the numbers we are dealing with are just integers and the operations used are addition, subtraction, multiplication and division. The only difference between modular arithmetic and the arithmetic you learned in your primary school is that in modular arithmetic all operations are performed regarding a positive integer, i.e. the modulus.

Before going into modular arithmetic, let's review some basic concepts. The division theorem tells us that for two integers a and b where $b \neq 0$, there always exists unique integers q and r such that $a = qb + r$ and $0 \le r < |b|$.

For example, $a = 17$, b=3, we can find $q = 5$ and $r = 2$ so that 17 = 3*5+2. a is called the **dividend**, b is called the **divisor**, q is called the **quotient** and r is called the **remainder**. If $r = 0$, then we say b **divides** a or a is **divisible** by b. This establishes a natural congruence relation on the integers. For a positive integer n, two integers a and b are said to be **congruent modulo** n (or a is congruent to b modulo n), if a and b have the same remainder when divided by n (or equivalently if $a - b$ is divisible by n). It can be expressed as $a \equiv b \bmod n$. n is called the **modulus**.

Example :

- Two odd numbers are congruent modulo 2 because all odd numbers can be written as 2n+1;
- Two even numbers are congruent modulo 2 because all even numbers can be written as 2n+0;
- 38 ≡ 23 mod 15 because 38 = 15*2 + 8 and 23 = 15 +8;
- -1 ≡ 1 mod 2 because -1 = -1*2+1 and 1 = 0*2+1;
- 8 ≡ 3 mod 5 because 8 = 5+3 and 3 = 0*5+3;
- -8 ≡ 2 mod 5 because -8 = -2*5+2 and 2 = 0*5+2;
- 8 ≢ -8 mod 5 because 8 = 5+3 and -8 = -2*5+2. The remainders 3 and 2 are not the same.

In modular arithmetic, the following are the definitions of addition, subtraction and multiplication.

The set of integers (0,1,2,……,p-1} where p is prime form the Galois field of p denoted as GF(p). If a,b € GF(p) then:

$$(a+b) \bmod p = \begin{cases} a+b & \text{if } a+b<p \\ a+b-p & \text{if } a+b \geq p \end{cases}$$
$$(a-b) \bmod p = \begin{cases} a-b & \text{if } a-b \geq 0 \\ a-b+p & \text{if } a-b<0 \end{cases}$$

(3+2) mod 7 =5

$(ab) \bmod p = r$ such that r is the remainder when the product ab is divided by p; $ab = qp + r$, where $0 \leq r < p$.

$(a/b) \bmod p = (ab^{-1}) \bmod p = r$, the unique remainder when ab^{-1} is divided by p; $ab^{-1} = qp + r,\ 0 \leq r < p$.

The factor b^{-1} is the *multiplicative inverse* of b in GF(p). For every element b in GF(p) except zero, there exists a unique element called b^{-1} such that $bb^{-1} \bmod p = 1$.

Example:

By definition we know that to find $x = b^{-1}$, there must exist an integer k, $0 \leq k < p$, such that $bx = kp + 1$. For example, if $p = 7$,

b :	1	2	3	4	5	6	(element)
b^{-1} :	1	4	5	2	3	6	(inverse)
k :	0	1	2	1	2	5	

Integer arithmetic using modular arithmetic:

Steps:

1. Represent integers as a set of moduli.

2. Perform arithmetic on the moduli

3. Convert the result of step2 to integer

Now let's see how we can use modular arithmetic as a transformation technique to help us work with integers. We begin by looking at how we can represent integers using a set of moduli, then how we perform arithmetic on this representation, and finally how we can produce the proper integer result.

Let a and b be integers and suppose that a is represented by the r-tuple $(a_1, \ldots, a_r)$, where $a_i = a \bmod p_i$, and b is represented by $(b_1, \ldots, b_r)$, where $b_i = b \bmod p_i$. The p_i are typically single precision primes. This is called a *mixed radix* representation which contrasts with the conventional representation of integers using a single radix such as 10 (decimal) or 2 (binary). The rules for addition, subtraction, and multiplication using a mixed radix representation are as follows:

$$\begin{aligned}(a_1, \ldots, a_r) + (b_1, \ldots, b_r) &= ((a_1 + b_1) \bmod p_1, \ldots, (a_r + b_r) \bmod p_r) \\ (a_1, \ldots, a_r)(b_1, \ldots, b_r) &= ((a_1 b_1) \bmod p_1, \ldots, (a_r b_r) \bmod p_r)\end{aligned}$$

Example 9.9 For example, let the moduli be $p_1 = 3$, $p_2 = 5$, and $p_3 = 7$ and suppose we start with the integers 10 and 15.

$$\begin{aligned} 10 &= (10 \bmod 3, 10 \bmod 5, 10 \bmod 7) = (1, 0, 3) \\ 15 &= (15 \bmod 3, 15 \bmod 5, 15 \bmod 7) = (0, 0, 1) \end{aligned}$$

Then

$$\begin{aligned} 10 + 15 &= (25 \bmod 3, 25 \bmod 5, 25 \bmod 7) = (1, 0, 4) \\ &= (1 + 0 \bmod 3, 0 + 0 \bmod 5, 3 + 1 \bmod 7) = (1, 0, 4) \end{aligned}$$

$$\begin{aligned} \text{Also } \ 15 - 10 &= (5 \bmod 3, 5 \bmod 5, 5 \bmod 7) = (2, 0, 5) \\ &= (0 - 1 \bmod 3, 0 - 0 \bmod 5, 1 - 3 \bmod 7) = (2, 0, 5) \end{aligned}$$

$$\begin{aligned} \text{Also } \ 10 * 15 &= (150 \bmod 3, 150 \bmod 5, 150 \bmod 7) = (0, 0, 3) \\ &= (1 * 0 \bmod 3, 0 * 0 \bmod 5, 3 * 1 \bmod 7) = (0, 0, 3) \ \square \end{aligned}$$

After we have performed some desired sequence of arithmetic operations using these r-tuples, we are left with some r-tuple $(c_1, \ldots, c_r)$. We now need some way of transforming back from modular form with the assurance that the resulting integer is the correct one.

Converting $(c_1,c_2,c_3,\ldots c_r)$ to integer form can be done by using One Step Chinese Remainder Algorithm

Euclidean Algorithm:

The *Euclidean algorithm* is arguably one of the oldest and most widely known algorithms. It is a method of computing the greatest common divisor (GCD) of two integers *a* and *b*. It allows computers to do a variety of simple number-theoretic tasks, and also serves as a foundation for more complicated algorithms in number theory.

The Euclidean algorithm is basically a continual repetition of the division algorithm for integers. The point is to repeatedly divide the divisor by the remainder until the remainder is 0. The GCD is the last non-zero remainder in this algorithm. The example below demonstrates the algorithm to find the GCD of 102 and 38:

102=2*38+26
38=1*26+12
26=2*12+2
12=6*2+0

The GCD is 2 because it is the last non-zero remainder that appears before the algorithm terminates.

Recursive Implementation of Euclid's Algorithm

```
Algorithm GCD($a$, $b$)
// Assume $a > b \geq 0$.
{
    if $b \neq 0$ then return GCD($b, a$ mod $b$);
    else return $a$;
}
```

$$\gcd(22,8) = \gcd(8,6) = \gcd(6,2) = \gcd(2,0) = 2$$

$$\text{and } \gcd(21,13) = \gcd(13,8) = \gcd(8,5) = \gcd(5,3) = \gcd(3,2) = \gcd(2,1) = \gcd(1,0) = 1 \quad \square$$

Extended Euclidean Algorithm

```
Algorithm ExEuclid($b$, $p$)
// $b$ is in GF($p$), $p$ being a prime. ExEuclid is a function
// whose result is the integer $x$ such that $bx + kp = 1$.
{
    $c := p$; $d := b$; $x := 0$; $y := 1$;
    while ($d \neq 1$) do
    {
        $q := \lfloor c/d \rfloor$;
        $e := c - d * q$;
        $w := x - y * q$;
        $c := d$; $d := e$; $x := y$; $y := w$;
    }
    if $y < 0$ then $y := y + p$;
    return $y$;
}
```

ExEuclid(7, 11) trace:

b	p	c	d	x	y	q	e	w
7	11	11	7	0	1	1	4	-1 (0-1*1)
7	11	7	4	1	-1	1	3	2 (1-(-1*1))
7	11	4	3	-1	2	1	1	-3 (-1 –(2*1))
7	11	3	1	2	-3			

When d=1, we return y if y>=0 or else we return (y+p) .In the above example as y<0 the algorithm returns (y+p) i.e. -3+11 =8.

```
Algorithm OneStepCRA($a$, $p$, $b$, $q$)
// $a$ and $b$ are in GF($p$), $\gcd(p,q) = 1$. This function
// returns a $c$ such that $c \bmod p = a$ and $c \bmod q = b$.
{
    $t := a$ mod $q$; $pb := p$ mod $q$; $s :=$ ExEuclid($pb, q$);
    $u := ((b - t) * s)$ mod $q$; if ($u < 0$) then $u := u + q$;
    $t := u * p + a$; return $t$;
}
```

Algorithm 9.12 One-step Chinese Remainder Algorithm

In the algorithm 9.12, Extended Euclidean Algorithm is used to return multiplicative inverse of a number ‘b’ w.r.t. a prime number ‘p’. One Step CRA is used to convert the result of mod p multiplication to integer product.

Example :

Example 9.10 Suppose we wish to take 4, 6, and 8 and compute $4 + 8 * 6 = 52$. Let $p_1 = 7$, and $p_2 = 11$.

$$\begin{array}{lcl}
4 & = & (4 \bmod 7, 4 \bmod 11) & = & (4,4) \\
6 & = & (6 \bmod 7, 6 \bmod 11) & = & (6,6) \\
8 & = & (8 \bmod 7, 8 \bmod 11) & = & (1,8) \\
8 * 6 & = & (6 * 1 \bmod 7, 8 * 6 \bmod 11) & = & (6,4) \\
4 + 8 * 6 & = & (4 + 6 \bmod 7, 4 + 4 \bmod 11) & = & (3,8)
\end{array}$$

So, we must convert the 2-tuple (3, 8) back to integer notation. Using OneStepCRA with $a = 3, b = 8, p = 7$, and $q = 11$, we get

$$\begin{array}{l}
t = a \bmod q = 3 \bmod 11 = 3 \\
pb = p \bmod q = 7 \bmod 11 = 7 \\
s = \text{ExEuclid}(pb,\ q) = 8;\ k = 5 \\
u = ((b - t)s) \bmod q = (8 - 3)8 \bmod 11 = 40 \bmod 11 = 7 \\
\textbf{return } (u * p + a) = 7 * 7 + 3 = 52 \qquad \square
\end{array}$$

So, the computation 4+8*6 is done in three steps using modular arithmetic:

Step 1: Divide the integers in the computation by the prime numbers p_1,p_2,p_3…. And store the corresponding remainders (i.e. representing the integers as moduli)

Step 2: Do the computation using ‘mod p’ addition and ‘mod p’ multiplication on the remainders obtained in step 1 (perform arithmetic on moduli)

Step 3: Apply one step CRA to convert the result of step 2 to integer product.

EXTRA READING MATERIAL

Extended Euclidean Algorithm:

The *Extended Euclidean algorithm* is an algorithm to compute integers x and y such that

$ax+by=\gcd(a,b)$

given a and b.

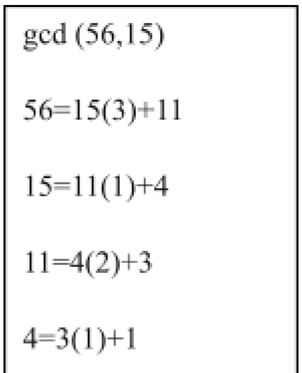
If we let ‘a’ to be a prime ‘p’ and b €GF(p) then GCD(p,b)=1. Extended Euclidean Algorithm finds integers ‘x’ and ‘y’ such that px+by=1 which implies that ‘y’ is multiplicative inverse of b mod p.

The existence of such integers is guaranteed by Bézout's lemma.

The extended Euclidean algorithm can be viewed as the reciprocal of modular exponentiation.

By reversing the steps in the Euclidean algorithm, it is possible to find these integers x and y. The whole idea is to start with the GCD and recursively work our way backwards. This can be done by treating the numbers as variables until we end up with an expression that is a linear combination of our initial numbers.

Example: 56s+15t =gcd(56,15) find s and t.

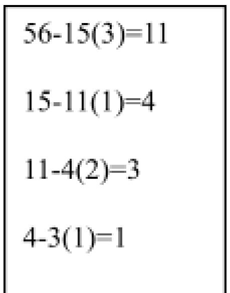

gcd (56,15)

56=15(3)+11

15=11(1)+4

11=4(2)+3

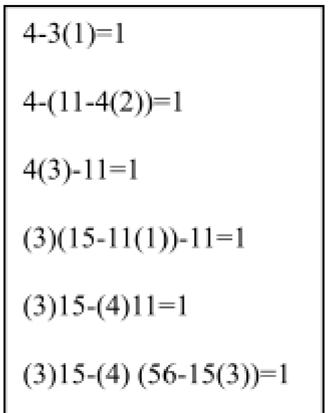
4=3(1)+1


56-15(3)=11

15-11(1)=4

11-4(2)=3

4-3(1)=1


4-3(1)=1

4-(11-4(2))=1

4(3)-11=1

(3)(15-11(1))-11=1

(3)15-(4)11=1

(3)15-(4) (56-15(3))=1

Rewrite