Lesson Fifteen

Math 6080 (for the Masters Teaching Program), Summer 2020

15. Parallel Processing Arithmetic of Large Numbers. The remainders:

$$\{0, ..., m-1\} \text{ modulo } m$$

have an arithmetic: they add and multiply, obeying all the associative, commutative and distributive laws, via the "clock arithmetic" rule:

$$r + s = (r+s)\%m \ \text{ and } \ rs = (rs)\%m$$

Exampls. (i) Modulo 13, we have:

$$7 + 8 = 15\%13 = 2 \text{ and } 7 * 8 = 56\%13 = 4$$

(ii) Draw addition and muliplication tables for numbers modulo 5 and 6.

Python Exercise. Prompt the user for a modulus and output the addition and multiplication tables with that modulus. To make this look nice, we should probably import some Python code, e.g. tabulate.

Proposition. A remainder r has a multiplicative inverse (i.e. a reciprocal) among the numbers modulo m if and only if r and m are relatively prime.

Proof. Suppose r and m are relatively prime. Then the enhanced Eucild's algorithm produces an equation of the form:

$$ar + bm = 1$$

But this means that $ar\%m = 1$, i.e. a is the multiplicative inverse of r.

On the other hand, if r has a multiplicative inverse, i.e. if there is an s in the numbers modulo m with the property that $rs\%m = 1$, then

$$rs = 1 + bm$$

for some integer b (by definition of %) and then every common divisor of r and m is a divisor of 1, which means that 1 is the greatest common divisor.

Corollary. If $\gcd(r, m) = 1$, then for every remainder b, the equation:

$$rx = b$$

has a unique solution among the numbers modulo m.

Proof. Multiply both sides by the multiplicative inverse of r.

Fix relatively prime numbers x and y.Then:

Chinese Remainder II. The function

$$f(r) = (r\%x, r\%y)$$

is an **isomorphism** with inverse function g as in Lesson Thirteen.

By this we mean that f is a bijection and:

$$f(q + r) = f(q) + f(r) \text{ and } f(qr) = f(q)f(r)$$

where the arithmetic on $\{0, ..., x-1\} \times \{0, ..., y-1\}$ is defined by:

$$(s, t) + (u, v) = (s + u, t + v) \text{ and } (s, t)(u, v) = (su, tv)$$

i.e. the arithmetic modulo xy transfers over to the arithmetics modulo x and y.

This is used in computer science (and certainly by Python) to "parallel process" the arithmetic of large numbers.

Example. Let $x = 30$ and $y = 31$. Then:

$$(-1)30 + (1)31 = 1$$

and using Lesson Thirteen, $g(s, t) = -30t + 31s$ is the inverse function of $f(r)$. Since $30 \cdot 31 = 930$, we can "parallel process" arithmetic of numbers, provided the answer doesn't exceed 930. For example, to calculate:

$$15 * 33 - 400$$

we parallel process it to:

$$f(15 * 33 - 217) = f(15) * f(33) - f(217) =$$
$$= (15, 15)(3, 2) - (10, 28) = (15, 30) - (10, 28) = (5, 2)$$

and

$$g(5, 2) = -30(2) + 31(5) = -60 + 155 = 95$$

This is very useful when doing many operations with large numbers, because the final recovery of the number via g is the only "large number" calculation that needs to be done.

Exercise. Given a number $n = x * y$ written as a product of relatively prime numbers x and y, write a Python function to multiply any pair of numbers $a * b$ by multiplying them first modulo x and then modulo y, and then recovering the product moduli $x * y$. Use your function to compute products of numbers modulo:

$$2 * 3 * 5 * 7 * 11 * 13 * 17 * 19 = 9,699,690 \text{ (basically 10 million)}$$

(1) Call the function to farm the arithmetic out to the two factors:

$$2 * 19 * 3 * 17 \text{ and } 5 * 13 * 7 * 11$$

(2) Recursively, farm out each of these two arithmetics to their factors:

$$2 * 19 \text{ and } 3 * 17 \quad \text{and} \quad 5 * 13 \text{ and } 7 * 11$$

and then finally farm each of these out to their factors.