Modular Arithmetic

If \( a \mod n = b \), then \( a = c \times n + b \). When you reduce a number \( a \) modulo \( n \) you usually want \( 0 \leq b < n \).

**Division Principle** [Bar02, pg. 61]: Let \( n \) be a positive integer and let \( a \) be any integer. Then there is exactly one pair of integers \( (c, b) \), \( 0 \leq b < n \) such that

\[
a = c \times n + b
\]

Examples:

- \( 17 \mod 5 = 2 \).
- \( 5 \mod 17 = 5 \).
- \( -8 \mod 3 = 1 \).

Some interesting properties of modular arithmetic:

\[
\begin{align*}
(a + b) \mod n &= (a \mod n + b \mod n) \mod n \\
(a \times b) \mod n &= (a \mod n \times b \mod n) \mod n \\
a^{-1} \mod n &= (a \mod n)^{-1}
\end{align*}
\]

which implies for example that

\[
(a \times (b+c)) \mod n = (a \mod n \times (b+c) \mod n) \mod n = (a \mod n \times (b \mod n \times c \mod n) \mod n) \mod n
\]
Modular Arithmetic

Example:

\[
(1234^{103} \times (1234^{32} + 1004^{245})) \mod 7
\]

\[
= 2^{103} \times (2^{32} + 3^{245})) \mod 7
\]

\[
= 2^{102} \times 2 \times (2^{32} + 3^{245})) \mod 7
\]

\[
= 2^{3 \cdot 34} \times 2 \times (2^{32} + 3^{245})) \mod 7
\]

\[
= 8^{34} \times 2 \times (2^{32} + 3^{245})) \mod 7
\]

\[
= 1^{34} \times 2 \times (2^{32} + 3^{245})) \mod 7
\]

\[
= 2 \times (4 + 5) \mod 7
\]

\[
= 2 \times 2 \mod 7
\]

\[
= 4
\]

Or, it is

\[
6764272377039604808006178055144906284633965782655606023664537316976\]

\[
752951384676463207919355564693975100855374063429268655061579855616\]

\[
806952088963846232974182202848803884955876363418030350483247250724631\]

\[
48332580139601163758807163959980616799419330958377856305601238263592\]

\[
0726053970067991456773244997104100369143911024550323643899333412749\]

\[
847654642971626166584986296154744033730885175975556976620658033217438\]

\[
80280868182620586591868607914549064740934594906378968122996574072724\]

\[
0610788091704653742699714387717546200022361247224368645062455882516\]

\[
77860769297702055524007172037257055742380864415433040887992580892514\]

\[
085538198662824039695765574178668960149972025379896072951585262587618\]

\[
46453044514479205381938683422173039265005451812870791033921812833308\]

\[
34131979868926531264458479736358778622572499449415763943865945787807\]

\[
55954244411694235864300346590674911568973317438026358845496672381789\]

\[
09903984749431006079030838865685491827363683331151586843871472933347\]

\[
39082872093966419871034772779648311073868559479294419934485808969958\]

\[
7734429853257643035321271289118720 \mod 7
\]

\[
= 4
\]
Modular Exponentiation

Say you want to compute $6^{469} \mod 7$. You could compute

$$6 \times 6 \times 6 \times \cdots \cdots 469 \text{ times}$$

Or, observe that $469 = 111010101_2 = 2^8 + 2^7 + 2^6 + 2^4 + 2^2 + 2^0$. That is

$$6^{469} = 6^{2^8} \times 6^{2^7} \times 6^{2^6} \times 6^{2^4} \times 6^{2^2} \times 6^{2^0}$$

So instead compute each term individually with one multiply each. That is, compute $6^2, 6^4, 6^8, 6^{16}, 6^{32}, 6^{64}, 6^{128}, 6^{256}$ by repeated squaring.
The GCD of two numbers $a$ and $b$ is the largest integer that divides both $a$ and $b$.

\[ \text{GCD}(a, b) = \text{GCD}(b, a \mod b) \]

Why doesn’t GCD(a, b) = GCD(a, a mod b) work? Because $d | a \land d | b \Rightarrow d | (a \mod b)$. Similarly, $d | b \land d | (a \mod b) \Rightarrow d | a$. This proves why the known algorithm for GCD works. For the latter algorithm, you can’t go in the reverse direction, i.e., $d | a \land d | (a \mod b) \not\Rightarrow$ that $d | b$. This is because if $d | (a \mod b)$ then $d | (a - kb)$ but it may be that $d | k$ instead of $b$. 
The inverse of an element $x \mod n$ is the element $y$ such that

$$xy = 1 \mod n$$

Consider the set of numbers modulo 9.

\[
\begin{array}{c|cccccccc}
\times & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
2 & 0 & 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 \\
3 & 0 & 3 & 6 & 0 & 3 & 6 & 0 & 3 & 6 \\
4 & 0 & 8 & 3 & 7 & 2 & 6 & 1 & 5 & 0 \\
5 & 0 & 5 & 1 & 6 & 2 & 7 & 3 & 8 & 4 \\
6 & 0 & 6 & 3 & 0 & 6 & 3 & 0 & 6 & 3 \\
7 & 0 & 7 & 5 & 3 & 1 & 8 & 6 & 4 & 2 \\
8 & 0 & 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

Not every number has an inverse modulo 9. In fact, only numbers coprime to 9 have inverses!
The Extended Euclidean Algorithm \( EGCD(f, d) \) permits one to find \( d^{-1} \mod f \) and \( f^{-1} \mod d \) [provided that \( GCD(f, d) = 1 \)] in addition to \( GCD(f, d) \). Start with the vectors

\[
(1, 0, f) \& (0, 1, d)
\]

and reduce one vector with another by subtracting a multiple of one from the second until the result has the third component 1. Both vectors maintain the invariant

\[
fx_1 + dx_2 = x_3
\]

Eventually, you get an equation of the form

\[
fx_1 + dx_2 = 1
\]

This gives \( x_2 = d^{-1} \mod f \) and \( x_1 = f^{-1} \mod d \).

Show examples of GCD & EGCD using RSA.pm and `~/bin/perl/egcd`. 
An integer $a > 1$ whose only divisors are the trivial divisors 1 and $a$ is said to be a prime number [CLRS01].

Example: $2, 3, 5, 7, 11, 13, 17, 19, 23, 29, \ldots \ldots$  

- If $n$ is a composite integer, then $n$ has a prime factor not exceeding $\sqrt{n}$.
  What this means is that in order to test a number $n$ for primality, it’s sufficient to try dividing it by all primes $\leq \sqrt{n}$.

- There are infinitely many primes [Ros93, Theorem 1.17].
  $n! + 1$ cannot have a prime divisor $\leq n$.

- $\pi(x)$, the numbers of primes $\leq n \rightarrow n/\log n$ as $n \rightarrow \infty$.
  1. $n/\log n \rightarrow \infty$ as $n \rightarrow \infty$.
  2. Even though the # of primes is $\infty$, its density gets sparser and sparser as $n \rightarrow \infty$.
  3. Approximately speaking, one would need to sample $\log n$ numbers to find a prime close to $n$.  

Consider finding all primes \( \leq 25 \):

\[
\begin{array}{ccccc}
1 & 2 & 3 & 4 & 5 \\
6 & 7 & 8 & 9 & 10 \\
11 & 12 & 13 & 14 & 15 \\
16 & 17 & 18 & 19 & 20 \\
21 & 22 & 23 & 24 & 25 \\
\end{array}
\]
1. Find a factor of $n$ by successively dividing $n$ by primes $2 \ldots \sqrt{n}$.

2. To find the factors of $n$, find $x^2 - y^2 = n$ [Bre89, Chapter 5]. The non-trivial factors of $n$ are then $(x - y)$ and $(x + y)$. 
Fermat’s Theorem

For prime $p$ and integer $b$ not divisible by $p$,

\[ b^{p-1} \equiv 1 \pmod{p} \]

Proof: Consider the product

\[ P = 1b \cdot 2b \cdot 3b \cdots (p-2)b \cdot (p-1)b = b^{p-1}(p-1)! \]

But $1b \neq 2b \neq 3b \neq \cdots \neq (p-1)b$ because the residue system mod $p$ is a field and $b$ has an inverse. Thus $1b, 2b, \ldots$ merely enumerate the numbers $1 \ldots (p-1)$ in some order. Canceling out $(p-1)!$ from both sides [because $(p-1)!$ is coprime to $p$] of the equation we get $b^{p-1} = 1$.

Consider the prime 17. Then

- $2^{16} = 65536 = 1 \mod 17$.
- $4^{16} = 4294967296 = 1 \mod 17$.
- $15^{16} = 6568408355712890625 = 1 \mod 17$.

Let’s try the same exercise with the prime 19.

- $2^{18} = 262144 = 1 \mod 19$.
- $4^{18} = 68719476736 = 1 \mod 19$.
- $15^{18} = 1477891880035400390625 = 1 \mod 19$.

If $x^{p-1} = 1 \mod p$, then $x \cdot x^{p-2} = 1 \mod p$. This means that $x^{p-2}$ is the inverse of $x \pmod{p}$.
Euler’s theorem

It is a generalization of Fermat’s theorem.

Def: $\phi(n)$ is the # of positive integers $< n$ that are relatively prime to $n$. 

$\phi(9) = 1, 2, 3, 4, 5, 6, 7, 8 = 6$.

Thm: If $GCD(x, n) = 1$, then $x^{\phi(n)} = 1 \pmod{n}$.

If $p$ is prime then $\phi(p) = (p - 1)$. 
Knapsack Encryption

Given a set of integers $a_1, a_2, \ldots a_n$, find whether a subset of them adds up to a given integer $t$. For example, for the set

$$A = \{4, 7, 33, 1, 12, 78, 11, 291\}$$

is there a subset that adds up to 17? To 129?

To encrypt text, say “NUS IS GREAT,” use the ASCII bit sequence of each character to select the set of numbers in the “knapsack” to add. So

- $N = 0x4E = 01001110 = 7 + 12 + 78 + 11 = 108.$
- $U = 0x55 = 01010101 = 7 + 1 + 78 + 291 = 377.$
- ........and so on......

Alternatively, the knapsack can have 16 numbers and you can encrypt two characters at a time. Suppose that

$$A = \{4, 7, 33, 1, 12, 78, 11, 291, 101, 29, 1101, 561, 487, 9826, 791, 893\}$$

Then you encrypt the message as

```
NU SI SG RE AT
```

The difficulty is that solving the general knapsack is as difficult for the recipient as it is for the enemy.
Merkle-Hellman Knapsack Encryption

Make the problem difficult for the enemy but easy for the recipient!

A superincreasing knapsack is one in which the integers in the knapsack form a superincreasing sequence. That is

\[ a_k > \sum_{j=1}^{k-1} a_j \]

Give a demo of superincreasing sequence using `printSuperIncreasingSeq(superIncreasingSeq(7))`.

An example is the sequence

\[ A = \{77, 105, 192, 392, 801, 1662, 3286, 6537\} \]

Now supposing one were to ask if there’s a subset of #s in the knapsack that add up to 2967 there’s an easy way to find it!

But the problem is that solving a superincreasing knapsack is as easy for the enemy as it is for the recipient!

So we try to confound the enemy by transforming a superincreasing knapsack into a “random” one.
Choose an \( m > \sum a_i \). Choose a \( w \) rel. prime to \( m \). Transform \( A \) into \( B \) such that \( b_i = w \times a_i \mod m \).

Let’s say that \( m = 13917 \) for the example above and that \( w = 269 \). Then

\[
B = \{6796, 411, 9897, 8029, 6714, 1734, 7163, 4911\}
\]

To encode the character ‘N’ which is 01001110, we might do \( 411 + 6714 + 1734 + 7163 = 16022 \).

To decode this number, the recipient does \( 269^{-1} \times 16022 \mod 13917 = 3725 \times 16022 \mod 13917 = 5854 \). Solving the superincreasing knapsack for 5854 gives the set 105 + 801 + 1662 + 3286!

**Practical Implementation**

Generate random numbers \( 0 < r_i < 2^{200} \) and choose

\[
a_i = 2^{200+i-1} + r_i
\]
Why is the Knapsack considered hard?

The general knapsack problem is NP Complete!
A little primer on complexity theory

\( f(n) \) is said to be \( O(g(n)) \) if \( \exists c, n_0 \) such that

\[
    f(n) \leq c|g(n)| \quad \forall n \geq n_0
\]

Example: \( f(n) = 17n + 10 \) is \( O(n) \) because

\[
    17n + 10 \leq 18n
\]

for \( n_0 = 10 \).

Demo plot \([n=1:100]\) 17*n+10, 18*n.

Example: \( f(n) = a_t n^t + a_{t-1} n^{t-1} + \cdots + a_0 \) is \( O(n^t) \).

- The class \( \mathbf{P} \): problems that can be solved in time bounded by a polynomial function of the problem size. For example, sorting, finding the max of a set of numbers, multiplication, exponentiation.

- The class \( \mathbf{NP} \): problems that can be verified in polynomial time. For example, Hamiltonian cycle, CNF satisfiability.

- The class \( \mathbf{NP Complete} \): Problems in NP to which every other problem in NP can be reduced in polynomial time.

If an NP Complete problem yields a polynomial time solution, then \( \mathbf{P} = \mathbf{NP} \). In some sense then, these are the hardest problems to solve in the class NP.

We know that \( \mathbf{P} \subseteq \mathbf{NP} \), that \( \mathbf{P} \neq \mathbf{EXP} \). But is \( \mathbf{P} = \mathbf{NP} \)?
References


