MFCS

Modular Arithmetic

Klaus Sutner
Carnegie Mellon University
Fall 2022

Total Recall: Divisibility

For $a, b \in \mathbb{Z}$, a divides b iff $\exists\, c \in \mathbb{Z}\,(a \cdot c = b)$.

This is usually written $a \mid b$.

Proposition

Note that $\pm 1 \mid a$ and $a \mid 0$ for all $a \in \mathbb{Z}$.
Divisibility is reflexive, transitive and almost antisymmetric.

Lemma (Linear Combinations)

If $d \mid a$ and $d \mid b$ then $d \mid (xa + yb)$ for all $x, y \in \mathbb{Z}$.

Quotients and Remainders

Theorem (Division Theorem)

Let b be positive, and a an arbitrary integer. Then there exist integers q and r such that

$$a = q \cdot b + r, \text{ where } 0 \leq r < b.$$

Moreover, the numbers q and r are uniquely determined (quotient and remainder).

In older literature this is often called the "Division Algorithm," though no algorithm is anywhere in sight.

Notation:

$r = a \bmod b$	remainder
$q = a \operatorname{div} b$	quotient

Actual Algorithm

Suppose both a and b and given in binary, and have thousands of digits. The data type used to store the bits is an array,

Find a fast algorithm to determine the quotient and remainder.

What is the running time of your algorithm in terms of the number of digits of the input?

Primes

$p \geq 2$ is prime if its only positive divisors are 1 and p.

Lemma

For every $n \geq 2$ there is a prime p such that $p \mid n$.

Theorem (Euclid, c. 300 BCE)

There are infinitely many primes.

Lemma

If p is prime and $p \mid ab$ then $p \mid a$ or $p \mid b$.

Sample Proof

The last lemma is easily handled with a forward link to the greatest common divisor, see below.

Let $d = \gcd(a, p)$.

If $d = p$ then clearly d divides a.

But otherwise $d = 1$, hence $xa + yp = 1$ for some integers x and y.

It follows that $xab + ypb = b$ and p divides b.

□

The Fundamental Theorem

Theorem

Let $n \geq 2$. Then there exist distinct primes $p_1, \ldots, p_k$ such that

$$n = p_1^{e_1} \, p_2^{e_2} \, \ldots \, p_k^{e_k}$$

where $e_i > 0$. The decomposition is unique up to order.

Proof.

Induction using the last lemma to split off prime factors. □

But beware, finding this prime decomposition seems very hard algorithmically. This (presumptive) difficulty turns out to be a blessing in disguise, it is currently exploited in cryptography.

The greatest common divisor is defined by

$$\gcd(a, b) = \max\big(d \mid d \text{ divides } a, b \big)$$

a and b are coprime (relatively prime) if $\gcd(a, b) = 1$.

The basic properties of the GCD:

Lemma

- $\gcd(x, 0) = x$
- $\gcd(x, y) = \gcd(y, x)$
- $\gcd(x, y) = \gcd(y, x \bmod y)$

Example

These properties produce an algorithm to compute it the GCD. Typical run: $a = 4234$ and $b = 4693286$.

$$
\begin{aligned}
4234 &= 0 \cdot 4693286 + 4234 \\
4693286 &= 1108 \cdot 4234 + 2014 \\
4234 &= 2 \cdot 2014 + 206 \\
2014 &= 9 \cdot 206 + 160 \\
206 &= 1 \cdot 160 + 46 \\
160 &= 3 \cdot 46 + 22 \\
46 &= 2 \cdot 22 + 2 \\
22 &= 11 \cdot 2 + 0
\end{aligned}
$$

The table provides a complete proof that $\gcd(4234, 4693286) = 2$, verification comes down to checking the arithmetic.

Linear Combinations

The last example suggests to take a closer look at linear combinations

$$c = x \cdot a + y \cdot b$$

where $x, y \in \mathbb{Z}$.

Obviously c is divisible by $\gcd(a, b)$.

More interestingly, we could run through the equations above backwards and write $2 = \gcd(a, b)$ as a linear combination of a and b:

$$\gcd(a, b) = 2 = 205068 \cdot a - 185 \cdot b$$

Lemma (Extended Euclidean Algorithm)

There exist integers x, y such that

$$\gcd(a, b) = x \cdot a + y \cdot b.$$

These so-called cofactors can be computed along with the GCD.

Trace of the Euclidean algorithm. Wlog $a \geq b \geq 0$.

$$r_{i-2} = q_i \cdot r_{i-1} + r_i \quad \text{where } r_0 = a, r_1 = b$$

Hence, $r_n = 0$ for some n, and $r_{n-1} = \gcd(a, b)$. Define

$$\begin{aligned} x_0 &= 1 & y_0 &= 0 \\ x_1 &= 0 & y_1 &= 1 \\ x_i &= x_{i-2} - q_i \cdot x_{i-1} & y_i &= y_{i-2} - q_i \cdot y_{i-1} \end{aligned}$$

A simple induction shows that

$$r_i = a \cdot x_i + b \cdot y_i.$$

Example

q_i	r_i	x_i	y_i
$-$	1233	1	0
$-$	1000	0	1
1	233	1	-1
4	68	-4	5
3	29	13	-16
2	10	-30	37
2	9	73	-90
1	1	-103	127
9	0	1000	-1233

We have

$$-103 \cdot 1233 + 127 \cdot 1000 = 1 = \gcd(1233, 1000)$$

Diophantine Equation

We can also think of

$$a \cdot x + b \cdot y = c$$

as an equation, we want solutions for x and y.

Again, we clearly need $d = \gcd(a, b) \mid c$ for any solution to exist.

We can divide by the GCD and use the extended Euclidean algorithm as before. But note that the solution is not unique: for any solution (x_0, y_0) we get infinitely many other solutions of the form

$$(x_0 + tb/d, y_0 - ta/d)$$

where $t \in \mathbb{Z}$. In fact, these are all the solutions.

Efficiency of EEA

One can implement all the necessary arithmetic in $O(k^2)$ steps for k-bit numbers. In fact addition is only $O(k)$, but for mods and remainders we need $O(k^2)$ steps.

But how often does the while-loop execute? Trivially no more than $a \geq b$ times, but that's no good at all.

Note that one must lose one bit at least at every other step. This follows from

$$r_{i-2} = q_i \cdot r_{i-1} + r_i$$

Hence total running time is $O(k^3)$ steps for k-bit inputs.

Incidentally, the worst possible input is two consecutive Fibonacci numbers. In this case, $q_i = 1$ at all times, and the algorithm just runs backwards through the Fibonacci numbers.

Definition

Let p prime. The p-adic valuation of an integer $n \neq 0$ is the largest e such that p^e divides n, in symbols $\nu_p(n)$; we set $\nu_p(0) = \infty$.

$$\nu_p(ab) = \nu_p(a) + \nu_p(b)$$

$$a \mid b \iff \forall\, p\, (\nu_p(a) \leq \nu_p(b))$$

$$\gcd(a, b) = \prod_p p^{\min(\nu_p(a), \nu_p(b))}$$

Alas, the last formula does not yield an efficient way to compute gcds: we have no good way to produce the prime decomposition of the numbers.

The Lattice View

The natural numbers with division $\langle \mathbb{N}, | \rangle$ form a so-called lattice: a partial order where any two elements have a join (supremum) and and a meet (infimum).

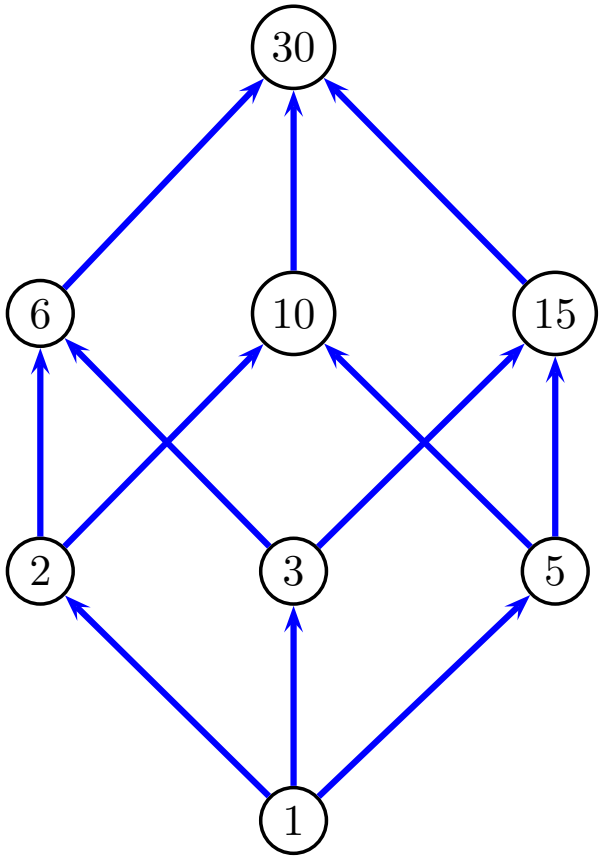
In this case, the join is simply the least common multiple, and the meet is the greatest common divisor.

Taking an algebraic perspective, one can think of a structure $\langle A, \sqcup, \sqcap \rangle$ with two binary operations where

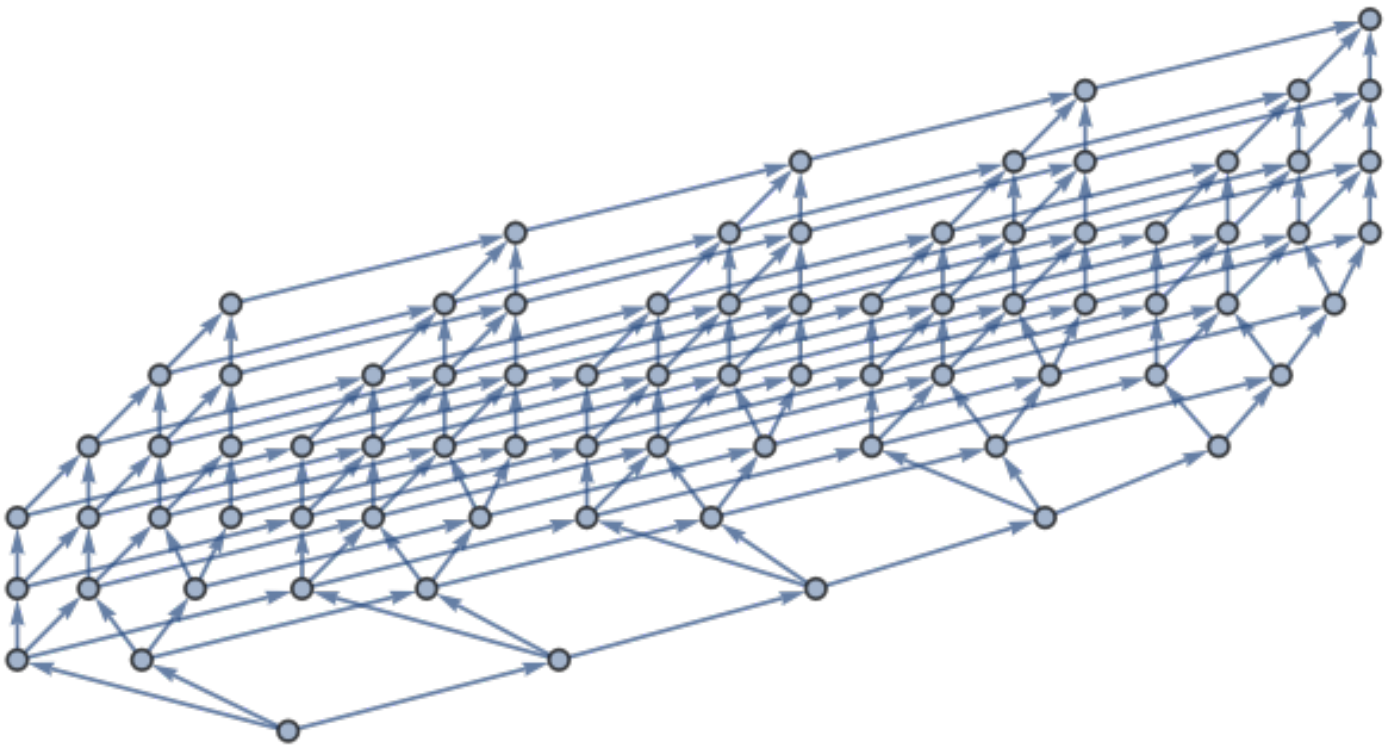
- $\sqcup$ and and $\sqcap$ are associative and commutative
- absorption holds:

$$x \sqcup (x \sqcap y) = x \qquad\qquad x \sqcap (x \sqcup y) = x$$

Divisor Lattice



A good way to visualize a divisor lattice is to draw a little diagram like this one.



Exercises

Exercise

Verify that $\langle \mathbb{N}, \mathrm{lcm}, \gcd \rangle$ *really forms lattice.*

Exercise

How are lcm *and* $\gcd$ *expressed in the picture of the divisor lattice of* 30*?*

Exercise

How is the structure of prime divisors of $148176 = 2^4 3^3 7^3$ *expressed in the picture of the divisor lattice?*

Odds and Evens

Distinguishing between even and odd integers may sound trivial, but it is often quite useful. Here is a simple example.

Suppose we have a polynomial with integer coefficients

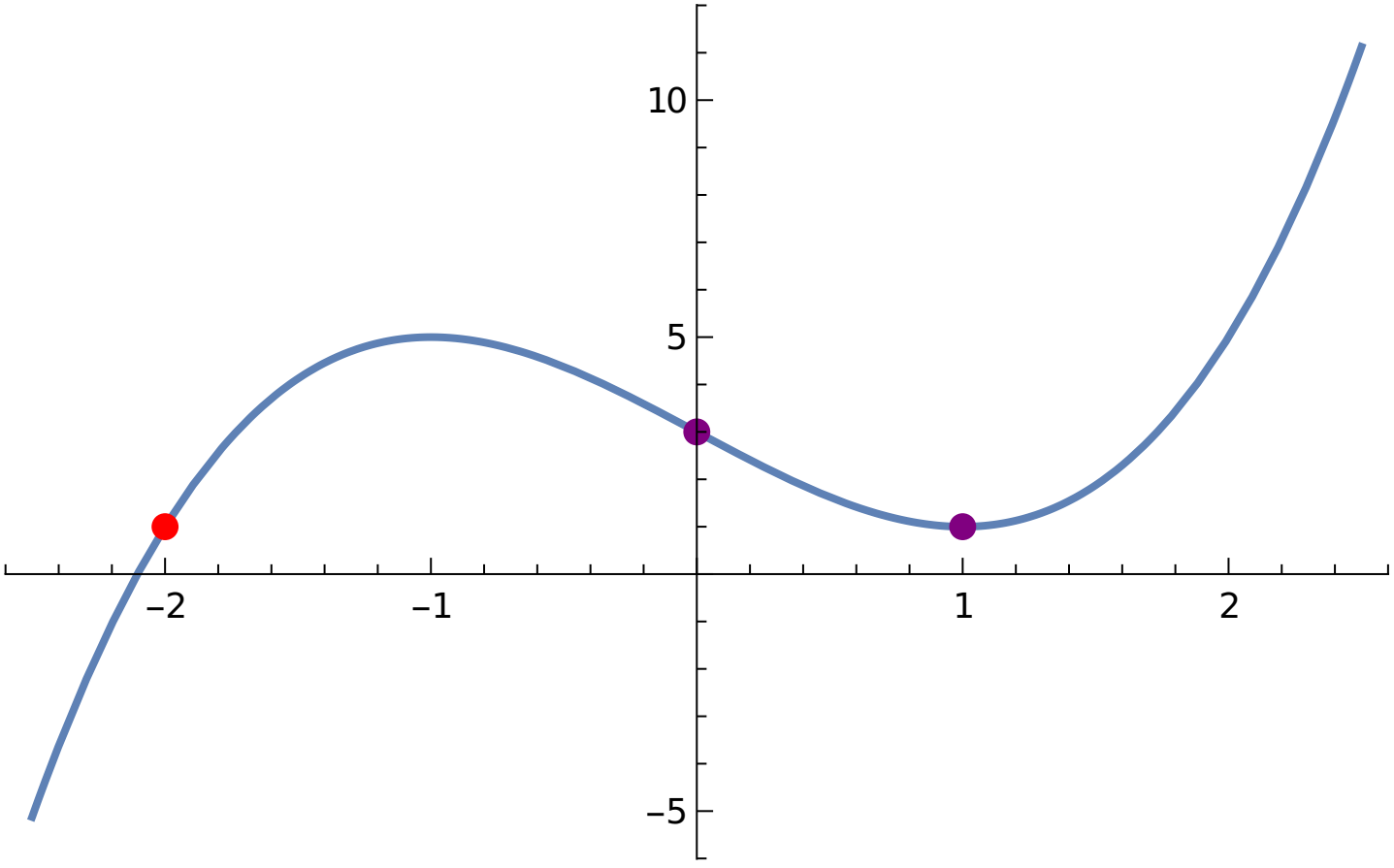
$$p(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d.$$

Claim

If both $p(0)$ and $p(1)$ are odd, then $p(x) \neq 0$ for all integers x.

Here is a trick to prove this: it suffices to show that $p(x)$ is always odd. "Suffices" sounds weird, this is actually a stronger assertion.

Picture



Careful, though, there is a root over $\mathbb{R}$, just not an integral one.

Even/Odd Arithmetic

To do this, we need to understand how the even/odd classification interacts with arithmetic. No problem.

We write e for even, and o for odd. Here are the Cayley tables for even/odd addition and multiplication:

$+$	e	o
e	e	o
o	o	e

$\cdot$	e	o
e	e	e
o	e	o

It follows that n even (odd) implies n^k even (odd) for all $k \geq 1$. So we can eliminate higher-degree terms in the polynomial.

So, now we are dealing with the linear polynomial $(a + b + c)x + d$.

The Argument

First note that $p(0) = d$ and $p(1) = a + b + c + d$ are both odd.

Case 1: So for even x we get

$$p(e) = (a + b + c) \cdot e + o = o$$

Case 2: For odd x we have

$$p(o) = (a + b + c)o + o = e \cdot o + o = o.$$

In both cases $p(x)$ is odd, done.

Carl Friedrich Gauss, 1777-1855.

Fix some positive modulus m. We can define an equivalence relation $\equiv_m$ on $\mathbb{Z}$ as follows:

$$x \equiv_m y \iff m \text{ divides } x - y$$

We say that x is congruent to y modulo m.

Notation: we usually write

$$x = y \pmod{m}$$

instead of $x \equiv_m y$. This looks more algebraic and turns out to be more convenient in a lot of arguments.

As usual, equivalence classes are expressed by brackets: $[x]$ or $[x]_{\equiv_m}$.

Don't confuse this with $[n] = \{1, 2, \ldots, n\}$.

Two Notations

Recall: for a given modulus m, $a \bmod m$ denotes the remainder function, a map on the integers:

$$\bmod m : \mathbb{Z} \longrightarrow \{0, 1, \ldots, m-1\}$$

The kernel relation of this function is none other than $\equiv_m$.

On the other hand, the notation

$$a = b \pmod{m}$$

indicates that a and b are congruent modulo m. Essentially this says: do not interpret the equation over $\mathbb{Z}$, but over a different algebraic structure, the modular numbers, that we will introduce shortly.

Congruences

There are lots of equivalence relations on the integers, but ours is particularly interesting since it coexists peacefully with addition and multiplication (this is the meaning of congruence in algebra in general).

Here is the critical idea: we can define arithmetic on the equivalence classes to obtain a new algebraic structure $\mathbb{Z}_m$ of modular numbers.

$$[x] + [y] = [x + y]$$

$$[x] \cdot [y] = [x \cdot y]$$

A potential problem with this type of definition is that it needs to be well-defined: we must be able to change the representatives without breaking the identities.

So we have to check the following:

$$x = x', y = y' \pmod{m}$$

implies

$$x + y = x' + y' \pmod{m}$$
$$x \cdot y = x' \cdot y' \pmod{m}$$

This is a huge restriction compared to arbitrary equivalence relations. There are uncountably many equivalence relations on $\mathbb{Z}$, but it turns out that all non-trivial congruences are Gaussian congruences.

Proof

Let's check for addition. We have to show that

$$x = x', y = y' \pmod{m} \quad \text{implies} \quad x + y = x' + y' \pmod{m}$$

Let $x = qm + r$, $x' = q'm + r$, $y = pm + s$, $y' = p'm + s$. Then

$$x + y = (q + p)m + r + s \qquad x' + y' = (q' + p')m + r + s$$

and our claim holds.

Multiplication is entirely similar.

Remainders

We could express the critical arithmetic properties also in terms of the remainder function like so:

$$(x + y) \bmod m = ((x \bmod m) + (y \bmod m)) \bmod m$$

$$(x \cdot y) \bmod m = ((x \bmod m) \cdot (y \bmod m)) \bmod m$$

Note that this is a bit clumsy, we have to remainder twice (at least in general).

All Congruences are Gaussian

Claim: Let $\equiv$ be a non-trivial congruence on the integers. Then $\equiv$ is a Gaussian congruence.

Proof. Let $a < b$ such that $a \equiv b$. Then $0 = a - a \equiv b - a$. Let $m > 0$ be the least such difference and note that $m\mathbb{Z} \subseteq [0]$.

We claim that $[0] = m\mathbb{Z}$.

Suppose otherwise and let $0 \equiv b$ where $b \notin m\mathbb{Z}$. Let p such that $pm < b < (p+1)m$. Then $0 < b < m$, but $b \equiv 0$, contradicting our choice of m.

It follows that $[a] = a + m\mathbb{Z}$ and we are done. $\square$

Careful notations that keep track of lots of details like

$$[x]_{\equiv_m} \qquad [x]_{\mathbb{Z}_m} \qquad +_m \qquad \cdot_m \qquad +_{\mathbb{Z}_m}$$

are perfectly correct, but they get very tedious in actual use.

So, if it is clear from context what we mean, we may drop all these decorations and just write x instead of $[x]_{\equiv_m}$. Similarly we just use $+$ and $\cdot$ for addition and multiplication of modular numbers.

In effect, we load a different math library and work in

$$\mathbb{Z}_m = \langle \{0, 1, \ldots, m-1\}, +, \cdot \rangle$$

If you feel nervous about this initially, add brackets and subscripts to your heart's content.

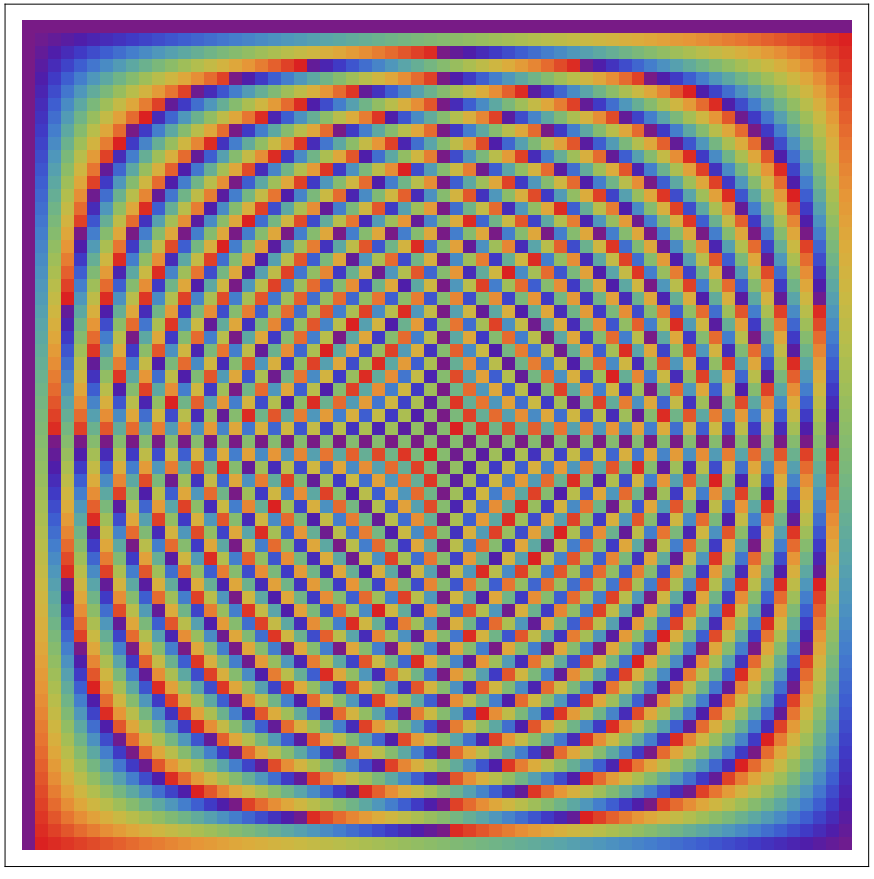
Cayley Tables

Example ($\mathbb{Z}_5$)

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\cdot$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Cayley Tables



Inevitable Clock Problem

A clock (which functions accurately) shows the hour hand positioned at a minute mark, and the the minute hand two marks away. What time is it?

Really have 60 possible positions. Equations:

$$m = h \pm 2 \pmod{60}$$
$$m = 12h \pmod{60}$$

By exploiting the congruence properties it follows that

$$11h = \pm 2 \pmod{60}$$

multiply by 11:

$$h = \pm 22 \pmod{60}$$

It's $4{:}24$ or $7{:}36$.

The Real Problem

The real question:
How does one solve equations modulo m?

Since there are only finitely many modular numbers one could, in principle, use brute force. Alas, for even slightly large moduli this is not a realistic option, we need some theory.

We know how to handle linear equations $ax = c \pmod{m}$, but life becomes already fairly difficult for quadratic ones. The good news: one can exploit this computational hardness for cryptography.

Simplifying Equations

Proposition

Let $ab = ac \pmod{m}$ *and* $m' = m/\gcd(a, m)$.
Then $b = c \pmod{m'}$.

In particular when a and m are coprime we can simply drop the a.

Exercise

Use p*-adic valuations to prove the proposition.*

Inhomogeneous Equations

First an important special case.

Lemma

The equation

$$a \cdot x = 1 \pmod{m}$$

has a solution if, and only if, a and m are coprime.
If a solution exists it is unique modulo m.

Proof.

A solution means that $ax - 1 = qm$, so a and m must be coprime.

In the opposite direction use the extended Euclidean algorithm to compute cofactors $ax + my = 1$.

□

Multiplicative Inverses

The situation in the lemma is very important.

The solution x such that $ax = 1 \pmod{m}$ is called the multiplicative inverse of a (modulo m).

Notation: $a^{-1} \pmod{m}$.

Example

$m = 11$.

x	1	2	3	4	5	6	7	8	9	10
x^{-1}	1	6	4	3	9	2	8	7	5	10

Note that $10 = 10^{-1}$ (no surprise, really: $10 = -1$).
So $1/2 = 6 \pmod{11}$.

Euler's Totient Function

The collection of all modular numbers that have a multiplicative inverse is usually written $\mathbb{Z}_m^\star$ and called the multiplicative subgroup.

Definition (Multiplicative Subgroup)

$$\mathbb{Z}_m^\star = \{\, a \in \mathbb{Z}_m \mid \gcd(a, m) = 1 \,\}$$

Definition (Euler's Totient Function)

The cardinality of $\mathbb{Z}_m^\star$ is written $\varphi(m)$.

Some Values

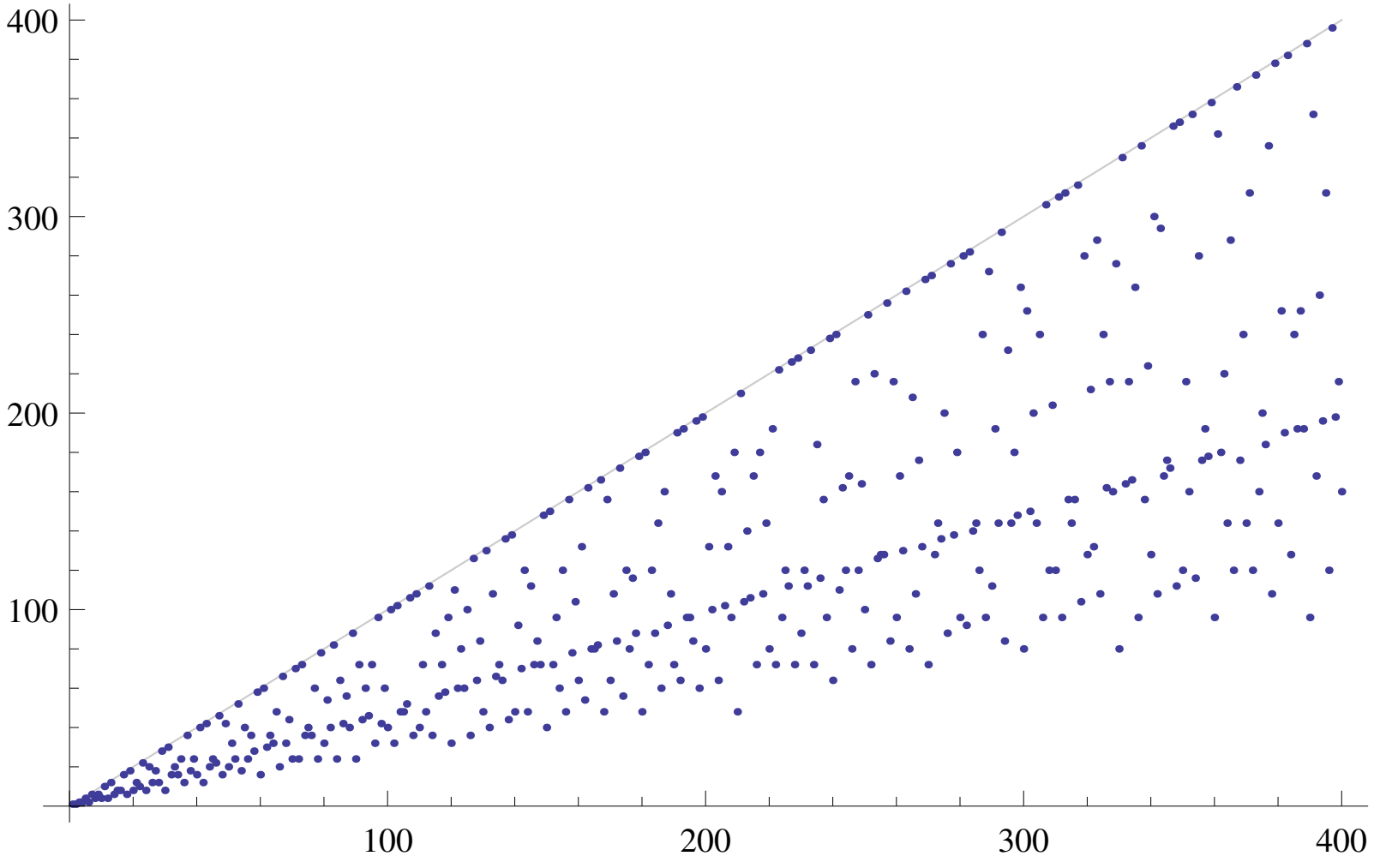
Here are the first few values of φ

$$1, 1, 2, 2, 4, 2, 6, 4, 6, 4, 10, 4, 12, 6, 8, 8, 16, 6, 18, 8, 12, 10, 22, 8, 20, 12, 18,$$
$$12, 28, 8, 30, 16, 20, 16, 24, 12, 36, 18, 24, 16, 40, 12, 42, 20, 24, 22, 46, 16, 42, \ldots$$

Looks complicated.

It is certainly far from clear what the next value would be.

Plot



Computing the Totient Function

Obviously we can compute $\varphi(n)$ by brute force, but that's a white lie: what if n has 1000 digits?

Here is a trick: we can compute $\varphi(n)$ if we know the prime factorization of n:

- For p prime $\varphi(p) = p - 1$ and $\varphi(p^k) = (p-1)p^{k-1}$.
- For m and n coprime, $\varphi(mn) = \varphi(m)\varphi(n)$.

Hence, given the prime decomposition

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$$

we can compute the totient function like so:

$$\varphi(n) = (p_1 - 1)p_1^{e_1 - 1}(p_2 - 1)p_2^{e_2 - 1} \dots (p_k - 1)p_k^{e_k - 1}$$

Not so fast . . .

Note the hedge: "given the prime decomposition"

Getting the decomposition requires factorization, and that is a difficult operation and presumably computationally hard in some technical sense.

As a consequence, it is not clear how to compute $\varphi(n)$ efficiently.

In fact, the two problems are computationally closely related. Some cryptographic schemes depend on the totient function being hard to compute.

Lemma

In the general case

$$a \cdot x = c \pmod{m}$$

we have a solution if, and only if, $\gcd(a, m)$ *divides* c*.*
Moreover, the number of solutions is $\gcd(a, m)$*.*

Exercise

Prove the general case.

When p is prime the structure of $\mathbb{Z}_p^\star$ is particularly simple:

$$\mathbb{Z}_p^\star = \{1, 2, \ldots, p-1\}$$

As a consequence, we can solve all equations $ax = b \pmod{p}$ as long as $a \neq 0 \pmod{p}$.

Here are some classical results concerning prime moduli.

Note that $(p-1)! = \prod_{x \in \mathbb{Z}_p^\star} x$ for p prime.

Wilson's Theorem

Theorem (Wilson's Theorem)

p is prime if, and only if, $(p-1)! \equiv -1 \pmod{p}$.

Proof.

First assume p is prime, wlog $p > 2$. We can pair off $a \in \mathbb{Z}_p^\star$ and $a^{-1} \in \mathbb{Z}_p^\star$.

a and a^{-1} are always distinct except in the case $a = \pm 1$: the quadratic equation $x^2 = 1 \pmod{p}$ has at most two solutions since $x^2 - 1 = (x+1)(x-1)$.

For the opposite direction assume p fails to be prime, say, $ab = p$ for $1 < a, b < p$. But then $(p-1)!$ and p are not coprime whereas -1 and p are coprime, contradiction.

□

Food for Thought

You may have heard that primality testing is algorithmically challenging.

There is an amazing polynomial time algorithm, using mostly high school arithmetic, but, sadly, it is useless in practice.

Currently, all usable primality tests involve randomization.

Question:
Why can't we use Wilson's theorem for a fast primality test?

Theorem (Fermat's Little Theorem)

If p is prime and coprime to a, then $a^{p-1} = 1 \pmod{p}$.

Proof.

Consider the map $\widehat{a} : \mathbb{Z}_p^\star \to \mathbb{Z}_p^\star$, $\widehat{a}(x) = ax$.

$\widehat{a}$ is a bijection, so

$$-a^{p-1} = a^{p-1} \prod_{x \in \mathbb{Z}_p^\star} x = \prod_{x \in \mathbb{Z}_p^\star} ax = \prod_{x \in \mathbb{Z}_p^\star} \widehat{a}(x) = \prod_{x \in \mathbb{Z}_p^\star} x = -1 \pmod{p}$$

Here we have used $\varphi(p) = p - 1$. Done.

□

Application: Rotating Arrays

Here is a simple algorithm question: figure out how to rotate an array of length n by s places, say, to the left.

Problem:	**Rotation**
Instance:	An array A, a positive integer s.
Solution:	Rotate A by s places.

This is entirely trivial if we simply copy some of the elements to a second array first.

This approach is linear time, but requires extra memory, up to $n/2$.

Can we do better?

Forcing Constant Space

It is clear that we can get liner time/constant space if we rotate the array by only **one** place: just remember a_0 and move everyone over.

Constant space here means: constant amount of extra space, we do not charge the algorithm for the original array.

Then just repeat s times, done.

No good at all: space is constant, but time is now quadratic.

Reversal

One method that is absolutely not obvious is to exploit reversal.

Clearly, an array can be reversed in linear time/constant space: just move two pointers from both ends towards the center.

By reversing twice (in the right way) we get rotation:

$$\mathsf{rev}(A, 0, s); \mathsf{rev}(A, s, n); \mathsf{rev}(A, 0, n)$$

Here $\mathsf{rev}(A, k, \ell)$ means: reverse that part $a_k, a_{k+1}, \ldots, a_{\ell-1}$, as usual with 0-indexed arrays.

Improvements?

The last method is linear time and constant space. But we can still quibble: it moves every element in the array twice.

Could we move the elements only once? After all, we know exactly where they are supposed to go.

Something like

$$a_0 \leftarrow a_s \leftarrow a_{2s} \leftarrow a_{3s} \leftarrow \ldots$$

a sequence of displacements (we have to remember a_0 which gets clobbered right away).

Could this possibly work? If so, exactly how? What does ... actually mean?

Iterating Addition

Addition on $\mathbb{Z}_m$ is fairly straightforward, and we can easily describe the effect of repeated addition.

Say, we define the additive function

$$\begin{array}{rccl} \alpha: & \mathbb{Z}_m & \longrightarrow & \mathbb{Z}_m \\ & x & \longmapsto & x + s \bmod m \end{array}$$

Question: What would the orbits

$$\mathsf{orb}(a;\alpha) = a, \alpha(a), \alpha^2(a), \ldots, \alpha^n(a), \ldots$$

look like?

Example

Let $m = 20$ and $s = 6$. Then there are essentially two orbits: infinite repetitions of the following basic blocks:

$$0, 6, 12, 18, 4, 10, 16, 2, 8, 14$$

$$1, 7, 13, 19, 5, 11, 17, 3, 9, 15$$

The hedge "essentially" covers the case where we start at different points. As sets, these orbits simply are the even and odd numbers between 0 and 19. By contrast, $s = 11$ produces a single repeating block

$$0, 11, 2, 13, 4, 15, 6, 17, 8, 19, 10, 1, 12, 3, 14, 5, 16, 7, 18, 9$$

General Case

The two examples are no coincidence: α is injective, so all the orbits must be repeating blocks, cycles of elements that repeat forever.

Moreover, since $\alpha(x) + y = \alpha(x + y) \pmod{m}$ all the cycles are just rotations of each other and it suffices to understand the single orbit $\mathrm{orb}(0, \alpha)$.

So we need the least $k > 0$ such that $ks = 0 \pmod{m}$.

We know how to do this: $k = m/\gcd(s, m)$.

Proposition

α has $\gcd(s, m)$ distinct orbits, each of length $m/\gcd(s, m)$.

And Rotation?

This means that we can concoct an algorithm that uses two nested loops:

- The outer loop runs through $\gcd(s, m)$ rounds.
- In round r, $0 \leq r < \gcd(s, m)$, we displace a_r by a_{r+s}, and so on, for $m/\gcd(s, m)$ elements.

Each array element is moved only once, and into its final position.

Exercise

Implement this rotation method. Compare to the reversal method.

Multiple Linear Equations

How about a system of linear equations, with several moduli:

$$a_i x = b_i \pmod{m_i} \qquad \text{where } i = 1, \ldots, n$$

We know how to simplify this system a little bit: for a solution to exist we need that $\gcd(a_i, m_i)$ divides b_i.

So we get equivalent equations $a_i' x = b_i' \pmod{m_i'}$ where a_i' and m_i' are coprime.

But that is equivalent to $x = c_i \pmod{m_i'}$ for some appropriate c_i.

Multiple Linear Equations, II

So we only have to deal with the situation

$$x = a_i \pmod{m_i} \qquad\qquad i = 1, \ldots, n$$

Tricky in general, but for coprime moduli easy. We only consider $n = 2$.

Let $m = m_1 m_2$ and define the double remainder function

$$f : \mathbb{Z}_m \longrightarrow \mathbb{Z}_{m_1} \times \mathbb{Z}_{m_2}$$
$$f(x) = (x \bmod m_1, x \bmod m_2)$$

Claim

f is injective and hence bijective.

Proof

To see this, suppose $f(x) = f(x')$, where $0 \leq x \leq x' < m$. Then

$$x' - x = q_1 m_1 = q_2 m_2.$$

But m_1 and m_2 are coprime, so $m \mid x' - x$ and therefore $x = x'$.

Since domain and codomain of f both have cardinality m, f must be a bijection by General Abstract Nonsense.

□

Hence we can solve $x = a \pmod{m_1}$ and $x = b \pmod{m_2}$: let

$$x = f^{-1}(a, b)$$

Great. But how do we find the x computationally?

CRT Example

Let $m_1 = 3$ and $m_2 = 5$, so $m = 15$.

Here is the canonical map $f : \mathbb{Z}_{15} \to \mathbb{Z}_3 \times \mathbb{Z}_5$,
$f(x) = (x \bmod 3, x \bmod 5)$.

0	$(0,0)$	8	$(2,3)$
1	$(1,1)$	9	$(0,4)$
2	$(2,2)$	10	$(1,0)$
3	$(0,3)$	11	$(2,1)$
4	$(1,4)$	12	$(0,2)$
5	$(2,0)$	13	$(1,3)$
6	$(0,1)$	14	$(2,4)$
7	$(1,2)$		

By table lookup, the solution to $x = 2 \bmod 3$, $x = 1 \bmod 5$ is
$x = f^{-1}(2,1) = 11$.

EEA to the Rescue

A better method is to use the EEA. Compute the cofactors:

$$\alpha m_1 + \beta m_2 = 1$$

Then

$$f(\alpha m_1) = (0, 1)$$
$$f(\beta m_2) = (1, 0)$$

whence

$$f(b\alpha m_1 + a\beta m_2) = (a, b)$$

So the solution is $x = b \cdot \alpha m_1 + a \cdot \beta m_2$.

CRT Example II

As we have seen, the solution to

$$x = 2 \bmod 3 \qquad\qquad x = 1 \bmod 5$$

is $x = 11$.

Here is the computationally superior solution: determine cofactors

$$(-3) \cdot 3 + 2 \cdot 5 = 1$$

which produce a solution

$$x = 1 \cdot (-3) \cdot 3 + 2 \cdot 2 \cdot 5 = 11$$

Chinese Remainder Theorem

Our result also holds for more than 2 equations (and is very old).

Theorem (CRT)

Let m_i, $i = 1, \ldots, n$ be pairwise coprime. Then the equations

$$x = a_i \pmod{m_i} \qquad i = 1, \ldots, n$$

have a unique solution in $\mathbb{Z}_m$, $m = m_1 m_2 \ldots m_n$.

This follows from repeated application of the solution for $n = 2$ since m_1 and $m_2 \ldots m_2$ are also coprime.

Computation

How do we compute the solution for $n > 2$? We could use the method for $n = 2$ recursively, but that is a bit tedious. Here is a better way.

Define

$$c_i = m/m_i$$

so that $c_i = 0 \pmod{m_j}$, $i \neq j$, but c_i and m_i are coprime. Use EEA to find inverses

$$\alpha_i c_i = 1 \pmod{m_i}$$

Then

$$x = a_1\alpha_1 c_1 + a_2\alpha_2 c_2 + \ldots a_n\alpha_n c_n \pmod{m}$$

In general, a solution may exist even if some of the moduli are not coprime. This is expressed in the following generalization[†].

Theorem (Generalized CRT)

The equations

$$x = a_i \pmod{m_i} \qquad i = 1, \ldots, n$$

have a solution if, and only if, for all $i \neq j$:

$$a_i = a_j \pmod{\gcd(m_i, m_j)}$$

The solution is unique modulo $m = \operatorname{lcm}(m_1, m_2, \ldots, m_n)$.

[†]This is the kind of result that you might want to be aware of, but there is no need memorizing it in detail.

Application: Large Numbers

Suppose you have a 64-bit architecture, but you need to compute with 100-bit numbers.

A bignum library is overkill, but we can fake 100-bit numbers relatively cheaply:

- find two primes p and q, each 60 bits long,
- compute in $\mathbb{Z}_p \times \mathbb{Z}_q$, using built-in arithmetic for each component.

The computation can involve many steps, we always keep our numbers in two-component form, each using 64 bits. Only in the end will we convert back to a single 100-bit number.

And the Primes?

There is an old result by Chebyshev that provides a lower bound for the number of k-digit primes (primes in $[2^{k-1}, 2^k - 1]$).

$$\frac{7\left(2^k - 1\right)}{8\ln\left(2^k - 1\right)} - \frac{9\left(2^{k-1} - 1\right)}{8\ln\left(2^{k-1} - 1\right)}$$

For example, for $k = 100$ we get at least 5.61×10^{27} primes. The length of the interval is about 6.34×10^{29}.

So we can pick a number at random and check for primality. If it fails, pick another one or start searching starting at the first number.