GB13624 - Maths for Computer Science

Lecture 3 – Number Theory

Claus Aranha
caranha@cs.tsukuba.ac.jp

College of Information Science

2022-10-19

Last updated October 17, 2022

This course is based on Mathematics for Computer Science, Spring 2015, by Albert Meyer and Adam Chlipala, Massachusetts Institute of Technology OpenCourseWare.

Lecture Outline

Number Theory: From division to the RSA algorithm (textbook chapter 8).

- Division and the Greatest Common Divisor (GCD);
- Primality, and simple cryptography;
- Modular Arithmetic, and Euler's theorem;
- The RSA public key algorithm;

Let's get started!

Part 1: Divisibility

Talking about Division

How should we define the *divide* operation on integers?

Consider a/b, for $a, b \in \mathbb{N}$ and $b > 0$. We have:

- q = quotient(a,b)
- r = remainder(a,b)

The **Division Theorem** says that $\exists$ **unique** q and r in $\mathbb{N}$ such as

$$a = bq + r, 0 \leq r < b$$

Example: $16/3 : a = 16, b = 3, q = 5, r = 1;$ $\qquad 16 = 3 \times 5 + 1$

Divisibility

We say that c **divides** $a (c|a)$ **iff**

$$\exists k \in \mathbb{N}, a = k \times c.$$

c divides a if there is a number k where a equals c times k.

- $5|15$ because $15 = 3 \times 5$
- $n|0$ because $0 = 0 \times n$ every number divides 0
- $1|n$ because $n = n \times 1$ 1 divides every number

Implications of Divisibility (1/2)

Lemma 1: $c|a \implies c|(sa)$

If c divides a, then c divides all multiples of a. **Miniproof**: Multiply both sides by s:

- $a = kc$ — Definition of divisibility
- $sa = skc$ — Multiplies both sides by s
- $(sa) = (sk) \times c \implies c|sa$ □

Lemma 2: $c|a \wedge c|b \implies c|(a+b)$

If c divides a and b, then c divides $a+b$. **Miniproof**: Distributive property of multiplication:

- $a = k_1 c, b = k_2 c$ — Definition of divisibility
- $a + b = k_1 c + k_2 c$ — Adding both equations together
- $(a+b) = (k_1 + k_2)c \implies c|(a+b)$ □

Implication of Divisibility (2/2)

$c|a \wedge c|b \implies c|(sa+tb)$

If c divides a and b, then c divides all linear combinations of a and b.
This one is pretty important.

Try to solve it by yourself first:

Implication of Divisibility (2/2)

$c|a \wedge c|b \implies c|(sa+tb)$

If c divides a and b, then c divides all linear combinations of a and b.
This one is pretty important.

Try to solve it by yourself first:

- Lemma 1: $c|a \implies c|ka$ for any k.
- Collorary: $c|a \wedge c|b \implies c|sa \wedge c|tb$
- Lemma 2: $c|a \wedge c|b \implies c|(a+b)$.
- Collorary: $c|sa \wedge c|tb \implies c|(sa+tb)$ □

Common Divisors

If $c|a$ and $c|b$, we say that c is a common divisor of a and b.

As we saw in the last slide, a common divisor of a and b will also divide the linear combinations of a and b:

$$c|a \wedge c|b \implies \forall s, t \in \mathbb{N}, c|(sa + tb)$$

In the next section, let's talk in more details about c, s and t.

Part 2: Greatest Common Divisors

Greatest Common Divisor (GCD)

The Greatest Common Divisor of a and b ($gcd(a, b)$), is the largest c so that $c|a \wedge c|b$.

Examples:

- $gcd(10, 12) = 2$ — $2|10 \wedge 2|12$
- $gcd(13, 12) = 1$ — 13 and 12 have no common factors, and $1|x, \forall x$
- $gcd(17, 17) = 17$
- $gcd(0, n) =$ n — $\forall n \in \mathbb{N}, n > 0, n|0$

Euclidean Algorithm

The Remainder Lemma

The GCD is easy to compute, by using the Remainder Lemma:

$$gcd(a, b) = gcd(b, \text{remainder}(a, b)), \text{ for } b \neq 0$$

Proof Idea

- The division axiom states that: $a = qb + r, 0 \leq r < b$;
- $c = gcd(a, b) \implies c|a \wedge c|qb$;
- $r = a - qb$, and $c|a \wedge c|qb \implies c|(a + (-1) * qb)$
- So $c|r$

Euclidean Algorithm

Calculating with the Remainder Lemma

To calculate the GCD of *a* and *b*, we repeatedly calculate the remainder of *a* and *b*, and replace *a* with the remainder.

GCD(899, 493) – a = 899, b = 493

- $899 = 493 \times 1 + 406$ — division axiom
- GCD(899, 493) = GCD(493, 406) — remainder lemma
- GCD(493, 406) = GCD(406, 87) — $493 = 406 \times 1 + 87$
- GCD(406,87) = GCD(87,58) — $406 = 87 \times 4 + 58$
- GCD(87,58) = GCD(58,29) = GCD(29,0) = 29

Euclidean Algorithm

State Machine

Let's use a State Machine to prove the correctness of Euclidean Algorithm:

- States::= $\mathbb{N} \times \mathbb{N}$ (values of a and b)
- Start State::= (a, b)
- State Transitions::= $(x, y) \rightarrow (y, rem(x, y))$ if $y \neq 0$
- End State::= $y = 0$

Remember that to prove correctness, we have to:

- Prove partial correctness
- Prove termination

Proof of GCD

Proof of Partial Correctness

To prove partial correctness, we have to prove the preserved invariant $P((x, y)) ::= [gcd(x, y) = gcd(a, b)]$.

Proof.

We prove the preserved invariant $P((x, y))$ by induction:

- P(start) is trivial: $gcd(a, b) = gcd(a, b)$
- If $P((x, y))$ is true, then it is still true for any transition.
  - There is only one transition: $(x, y) \rightarrow (y, \text{rem}(x, y))$
  - The Remainder Lemma says that: gcd(y,y) = gcd(y,rem(x,y));

By these two items, we proved that $P()$ holds for any state in the machine. □

Proof of GCD

Proof of Termination

- The transition is $(x, y) \rightarrow (y, \text{rem}(x, y))$;
- By the division axiom, $0 \leq \text{rem}(x, y) < y$;
- So y becomes smaller after every transition;
- When $y = 0$, the machine halts;

By the way, when the machine stops, the state is $(x_e, 0)$. Because the preserved invariant state that $gcd(a, b) = gcd(x, y)$, and $gcd(x, 0) = x$, the final result of the algorithm is that $gcd(a, b) = x_e$.

How fast is the GCD?

Analysing the state machine, we can calculate how fast the GCD terminates:

- At each transition, x is replaced by y. There are two cases:
  1. $y \leq x/2$: so x is halved this step.
  2. $y > x/2$: so $\text{rem}(x, y) = x - y$ and x will get halved at the *next* step.

- So every two steps, x gets halved (or even smaller)
- This means that after $\leq 2 \log_2 x$ steps, the algorithm stops.

So GCD(a, b) is calculated after $2 \log_2 a$ steps.

GCD and Linear Combinations

Remember that we showed that a divisor c of a and b is also a divisor of $(sa + tb)$.

Note that c is a divisor of itself too, so we can represent c as $(sa + tb)$ for some s and t.

The Extended Euclid Algorithm (also called "The Pulverizer") can calculate s and t so that $\gcd(a, b) = sa + tb$.

It is useful to note also that the $\gcd(a, b)$ divides **every linear combination** of a and b.

The Pulverizer Algorithm

Calculate Euclid's algorithm as usual:

- GCD(x,y) = GCD(y,rem(x,y) **Start**: GCD(a,b)

As we calculate GCD, keep track of four coefficients: **c,d,e,f**

- $x = ca + db$ and $y = ea + fb$
- **at start:** $x = 1a + 0b$, $y = 0a + 1b$
- **update:** $x_{\text{next}} = y = ea + fb$
- $y_{\text{next}} = \text{rem}(x, y) = x - qy = ca + db - q(ea + fb)$
- $y_{\text{next}} = (c - qe)a + (d - qf)b$

The Pulverizer Algorithm

Example

a = 899, b = 493

(remember: $e_1 = c_0 - q_0 e_0, f_1 = d_0 - q_0 f_0$)

a	b	q	rem(a,b)	c	d	e	f
899	493	1	406	1	0	0	1
493	406	1	87	0	1	1	-1
406	87	4	58	1	-1	-1	2
87	58	1	29	-1	2	5	-9
58	29	2	0	5	-9	-6	11
29	0	-	-	-6	11	-	-

$$\text{GCD}(899,493) = 29 = -6 \times 899 + 11 \times 493$$

The Pulverizer Algorithm

Positive coefficients

So the Pulverizer calculates one linear combination of a and b which is equal to the GCD:

$$\text{GCD}(899, 493) = -6 \times 899 + 11 \times 493$$

It is possible to obtain other linear combinations, by defining s and t as follows:

$$\text{GCD}(899, 493) = (-6 + 493k)899 + (11 - 899k)493, \text{ for any } k$$

For example, if we set $k = 1$, we can find the following coefficients for s and t:

$$\text{GCD}(899, 493) = 487 \times 899 - 888 \times 493$$

The Pulverizer Algorithm

Positive coefficients

Remember the robot from last class? The position of the robot was $x = 5a - 3b$. For any x, we can use the pulverizer to find a and b.

- Let's say that we want to find the path of the robot for $x = 8$
- $\gcd(5, 3) = 1 = 2 \times 5 - 3 \times 3$, using the pulverizer.
- $8 = 8 \times 1 = 8 \times (2 \times 5 - 3 \times 3) = (8 \times 2)5 - (8 \times 3)3$
- Result: 16 moves forward, 24 moves back.

This may not be the best solution for $x = 8$, but it is an easy and fast algorithm to calculate a solution.

Part 3: Primality

Prime Numbers

Prime numbers are numbers that have no divisors, others than 1 and themselves.

In many senses, primes numbers are the **building blocks** of arithmetic.

The Fundamental Theorem of Arithmetic

Every positive integer has a **unique** prime factorization.

Proving Prime Number Factorization

Step 1

Lemma 1: If p is prime and $p|ab$, then $p|a$ or $p|b$

Proof.

- Let's assume that p **does not** divide a. So $\gcd(a, p) = 1$
- Consequently, $\exists s, t.sa + tp = 1$
- Multiply everything by b : $sab + tpb = b$
- $p|sab \wedge p|tpb \implies p|(sab + tpb) \implies p|b$

□

Corolary 1: If $p|a_1 a_2 a_3 \ldots a_m$, then $\exists 0 < i \leq m, p|a_i$

(You can prove this by induction on m)

Proving Prime Number Factorization

Step 2

In lecture 1 we showed that every positive integer was a product of prime numbers.
Now we need to show that the prime number factorization is **unique**:

Proof.

Proof by contradiction:

- By WOP, consider n the smallest positive integer with a non-unique factorization;
- $n = p_1 p_2 \dots p_k$ and $n = q_1 q_2 \dots q_k$
- Consider that these factors are **ordered** ($q_1 \geq q_2 \geq \dots$)
- $p_1 \neq q_1$ (else, we could cancel then out, and find a smaller n with 2 factorizations)
- Assume $p_1 > q_1$. $p_1 | n \implies p_1 | q_1 q_2 q_3 \dots q_k$
- However, $p_1 > q_1 \geq q_2 \geq q_3 \dots \geq q_k$, which is a contradiction.

Prime Properties

Because prime numbers are so important, we are interested in knowing how many are there. Consider the $\pi(n)$ set of prime numbers between 2 and n:

$$\pi(n) ::= [\{p \in [2..n] | p \text{ is prime}\}]$$

The Prime Number Theorem states how the size of $\pi(n)$ grows:

$$\lim_{n \to \infty} \frac{\pi(n)}{n/\ln n} = 1$$

As a general idea, the frequency of primes around an integer n is $\frac{1}{\ln n}$. In other words, the quantity of primes gets smaller as n gets bigger.

Prime Cryptography

The Turing Algorithm

Alan Turing, among his many works in computer science, proposed a cryptography method using prime numbers.

1. Transform the message you want to send (m) into a prime number:

```
v  i  c  t  o  r  y       (not prime)          (prime)    **
22 09 03 20 15 18 25 -> 22090320151825 -> 2209032015182513
```

2. Choose a secret key (k) as a large prime, and share it with the receiver;
3. The *sender* calculates the *encrypted message* as follows, and sends it.

$$\hat{m} = m \times k$$

4. The *receiver* recovers the *original message* using the secret key:

$$m = \hat{m}/k$$

Prime Cryptography

How secure is the Turing Algorithm?

Because of the difficulty of finding the prime factors of a large number, it is very hard to find m and k if you only have $\hat{m}$.

So if an adversary only knows $\hat{m}$, it is very hard to break this code. But what happens if the adversary has **two** secret messages, $\hat{m_1}$ and $\hat{m_2}$?

Remember that $\hat{m_1} = k \times m_1$ and $\hat{m_2} = k \times m_2$. k is a common divisor of $\hat{m_1}$ and $\hat{m_2}$, so:

$$k = gcd(\hat{m_1}, \hat{m_2})$$

So it is very easy to find the secret key of this algorithm. Can we improve it?

Part 4: Modular Arithmetic

Modular Arithmetic and Congruence

Using modular arithmetic, we can make the Turing Algorithm stronger.

Modular arithmetic is centred around the concept of **Congruence**:

$$a \equiv b(\text{modulo } n) \iff n|(a-b)$$

Examples:

- $30 \equiv 12(\text{modulo } 9)$, because $9|(30-12)$
- $66666663 \equiv 788253(\text{modulo } 10)$, because $10|66666663-788253$

Modular Arithmetic and Congruence

Remainder Theorem

One important implication of the definition of equivalence is the remainder theorem:

$$a \equiv b (\text{mod } n) \iff \text{rem}(a, n) = \text{rem}(b, n)$$

Proof.

Proof by two way implication. Let the remainder of a and b be $r_{a,b}$:

- Let $a = q_a n + r_{a,n}$, and $b = q_b n + r_{b,n}$
- (right side) **if** $r_{a,n} = r_{b,n}$ then $a - b = (q_a - q_b)n \implies n|(a - b)$
- (left side) **if** $n|(a - b)$ then $n|((q_a - q_b)n + (r_{a,n} - r_{b,n}))$
  - but $0 \leq r_{*,n} < n$ so $r_{a,n} - r_{b,n}$ must be 0

□

Modular Arithmetic and Congruence

Consequences of the Remainder Theorem

Remainder Theorem:

$$a \equiv b (\text{mod } n) \iff \text{rem}(a, n) = \text{rem}(b, n)$$

- $a \equiv b$ (mod n) **and** $b \equiv c$ (mod n) **implies** $a \equiv c$ (mod n)
- $a \equiv$ rem(a,n) (mod n) (important!)

- $a \equiv b$ (mod n) **implies** $a + c \equiv b + c$ (mod n)
- $a \equiv b$ (mod n) **implies** $ac \equiv bc$ (mod n)
- $a \equiv b$ (mod n) **and** $c \equiv d$ (mod n)
  implies $a + c \equiv b + d$ (mod n) **and** $ac \equiv bd$ (mod n)

The last three consequences show that we can freely use addition and multiplication in modular arithmetic. Try to prove some of these consequences!

Using Modular Arithmetic

General Principle of Modular Arithmetic

You can simplify modulo operations composed of additions and multiplications by replacing integer operands by their remaiders.

Example: What is $287^9 \equiv ?\ (\text{mod } 4)$

- Simplify $287^9\ (\text{mod } 4)$ to $3^9\ (\text{mod } 4)$, becase $r_{287,4} = 3$
- $3^9 \rightarrow 3^8 \times 3 \rightarrow 9^4 \times 3$
- Simplify $9^4 \times 3\ (\text{mod } 4)$ to $1^4 \times 3$, because $r_{9,4} = 1$
- $289^9 \equiv 1^4 \times 3 \equiv 3\ (\text{mod } 4)$

We calculated a large exponent without actually calculating the exponentiation!

Modular Arithmetic and Division

We so that modular arithmetic works for addition and multiplication, so what about division?

- $8 \times 2 \equiv 3 \times 2 \pmod{10}$
- $8 \times \not{2} \equiv 3 \times \not{2} \pmod{10}$
- $8 \equiv 3 \pmod{10}$ **FALSE!**

We cannot cancel out multiplications arbitrarily!

Modular Arithmetic and Division

Modular Inverses

When we cancel the multiplication of a real number, it is equivalent of multiplying it by its **multiplicative inverse**:

$$an = bn \rightarrow a \not{n} = b \not{n} \rightarrow a(n \times \frac{1}{n}) = b(n \times \frac{1}{n})$$

So a multiplicative inverse of n is the number k so that $n \times k = 1$.

In modulo arithmetic, we have: $n \times k \equiv 1$ (mod m). But, it turns out that it is not possible to find a k that solves this equation for every n and m!

Modular Inverses

When does n have an inverse, and how can we find it?

It turns out that n only has a modular inverse (module m) if $gcd(n, m) = 1$.

It is relatively easy to find the modular inverse:

- First we calculate gcd(n,m) using the pulverizer, and obtaining s and t so that $sm + tn = 1$
- The multiplicative inverse of n modulo m is given by: $r_{m,t}$
- So $n \times r_{m,t} \equiv 1 \pmod{m}$

Turing Code 2.0

Using Modular Arithmetic, we can solve some of the problems of the cryptography algorithm described in the last section:

1. The sender and the receiver agree on a large prime number n, which will be the module, as well as the secret key $k < n$
2. **Encription:** The message m is a prime number ($0 < m < n$). The sender calculates the secret message $\hat{m}$ using modulo multiplication:

$$\hat{m} = m \times k \bmod n$$

3. **Decription:** The receiver multiplies the encrypted message $\hat{m}$ by the inverse of the key: k^{-1} (mod n):

$$m = \hat{m}0 \times k^{-1} \bmod n$$

We cannot find k anymore by having two encrypted messages, so is this algorithm safe?

Turing Code 2.0

Breaking the Turing Code with a Plaintext Attack

The new algorithm can be broken by what is called a **Plaintext Attack**.

Imagine that an attacker is able to acquire an encrypted message $\hat{m}$ and its respective clear message m. To break this code:

- Calculate the multiplicative inverse of m modulo n: m^{-1}, using the Pulverizer.
- Multiply the result by the encrypted message:
  - $m^{-1} \times \hat{m} = m^{-1} \times (m \times k) = (m^{-1} \times m) \times k = 1 \times k \pmod{n}$
- This allows us to recover the secret key k.

So this algorithm is not so good after all! :-(

Part 5: RSA Algorithm

The RSA Cryptosystem

Turing's cryptography algorithm that we described in the previous sections was flawed. But there is a cryptographic algorithm based on modular arithmetic which is the base of much of the security of the internet.

The RSA algorithm involves computing the remainders of numbers raised to large powers.

Euler's Function

The number of integers between 0 and n that are relatively prime to n is represented by Euler's Function:

$$\Phi(n) ::= \#k \in [0, n), GCD(k, n) = 1$$

For example:

- $\Phi(7) = 6$, because $\{1, 2, 3, 4, 5, 6\}$ are relatively prime to 7;
- $\Phi(12) = 4$, because $\{1, 5, 7, 11\}$ are relatively prime to 12;

Euler's Theorem

The value of Φ is important because of Euler's Theorem: If n and k are relatively prime, then

$$k^{\Phi(n)} \equiv 1 (\text{mod } n)$$

Additionally, Euler's function and Euler's theorem gives us another way to calculate modular inverses:

- $k^{\Phi(n)} \equiv 1 \text{ (mod } n)$
- $k^{\Phi(n)-1} \times k \equiv 1 \text{ (mod } n)$

So k and $k^{\Phi(n)-1}$ are inverses, modulo n.

Calculating $\Phi(n)$

We can calculate $\Phi(n)$ quickly with some simple rules:

- If n is prime, $\Phi(n) = n - 1$
- If n is a power of a prime, $\Phi(p^k) = p^k - p^{k-1}$
  - Ex: $\Phi(9) = 3^2 - 3 = 6 \qquad \{1, 2, 4, 5, 7, 8\}$
- If n is ab where GCD(a,b)=1, $\Phi(ab) = \Phi(a)\Phi(b)$
  - Ex: $\Phi(12) = \Phi(3) \times \Phi(4) = (3 - 1) \times (2^2 - 2) = 4$

The RSA Encryption System

The RSA, created in 1977, is a Public Key Cyrptosystem. This means that, unlike the Turing algorithm, it is not necessary to exchange a secret key between the *sender* and the *receiver*.

The RSA uses Euler's theorem for encrypting and decrypting messages, and its security is based on the difficulty of factoring large numbers.

The RSA System

Algorithm

Preparation

- Generate two distinct large primes, p and q, which are secret
- Let $n = pq$
- Create a public key $e \in [0..n)$ such that $\gcd(e, (p-1)(q-1)) = 1$;
- Create a private key $d \in [0..n)$ be the modular inverse of $e \pmod{(p-1)(q-1)}$;

Encoding a Secret Message

- Given a message $m \in [0..n)$, construct the encrypted message as $\hat{m} = m^e \pmod{n}$

Decoding a Secret Message

- Given an encrypted message $\hat{m}$, it is decrypted as $m = \hat{m}^d \pmod{n}$

The RSA System

Assumptions

- Basic Assumption: **One Way Functions** that are easy to compute but hard to invert
- It is easy to compute the product n of two large primes p and q ($n = pq$)
- It is very hard to factor n into p and q.

Slide Credits

These slides were made by Claus Aranha, 2020. You are welcome to copy, re-use and modify this material, following the CC-SA-NC license.

These slides are based on "Mathematics For Computer Science (Spring 2015)", by Albert Meyer and Adam Chlipala, MIT OpenCourseWare. `https://ocw.mit.edu`.

Individual images in some slides might have been made by other authors. Please see the following slides for information about these cases.

Image Credits I