



# CSE 311 Lecture 13: Primes and GCD

Emina Torlak and Kevin Zatloukal

# Topics

## Modular arithmetic applications

A quick wrap-up of [Lecture 12](#).

## Primes

Fundamental theorem of arithmetic, Euclid's theorem, factoring.

## Greatest Common Divisors (GCD)

GCD definition and properties.

## Euclidean algorithm

Computing GCDs with the Euclidean algorithm.

## Extended Euclidean algorithm

Bézout's theorem and the extended Euclidean algorithm.

# Modular arithmetic applications

A quick wrap-up of [Lecture 12](#).



# Modular arithmetic properties

## Modular addition property

Let  $m$  be a positive integer ( $m \in \mathbb{Z}$  with  $m > 0$ ).

If  $a \equiv b \pmod{m}$  and  $c \equiv d \pmod{m}$ , then  $a + c \equiv b + d \pmod{m}$ .

## Modular multiplication property

Let  $m$  be a positive integer ( $m \in \mathbb{Z}$  with  $m > 0$ ).

If  $a \equiv b \pmod{m}$  and  $c \equiv d \pmod{m}$ , then  $ac \equiv bd \pmod{m}$ .

# Unsigned integer representation

Represent integer  $x$  as a sum of  $n$  powers of 2:

If  $x = \sum_{i=0}^{n-1} b_i 2^i$  where each  $b_i \in \{0, 1\}$ ,  
then the representation is  $b_{n-1} \dots b_2 b_1 b_0$ .

**Examples:**

$$99 = 64 + 32 + 2 + 1$$

$$18 = 16 + 2$$

**So for  $n = 8$ :**

$$99 = 0110\ 0011$$

$$18 = 0001\ 0010$$

# Sign-magnitude integer representation

If  $-2^{n-1} < x < 2^{n-1}$ , represent  $x$  with  $n$  bits as follows:

Use the first bit as the sign (0 for positive and 1 for negative), and the remaining  $n - 1$  bits as the (unsigned) value.

**Examples:**

$$99 = 64 + 32 + 2 + 1$$

$$18 = 16 + 2$$

**So for  $n = 8$ :**

$$99 = 0110\ 0011$$

$$-18 = 1001\ 0010$$

$$81 = 0101\ 0001$$

The problem with this representation is that our standard arithmetic algorithms no longer work, e.g., adding the representation of -18 and 99 doesn't give the representation of 81.

# Two's complement integer representation

Represent  $x$  with  $n$  bits as follows:

If  $0 \leq x < 2^{n-1}$ , use the  $n$ -bit unsigned representation of  $x$ .

If  $-2^{n-1} \leq x < 0$ , use the  $n$ -bit unsigned representation of  $2^n + x$ .

**Key property:**

Two's complement representation of any number  $y$  is equivalent to  $y \bmod 2^n$  so arithmetic works  $\bmod 2^n$ .



# Two's complement integer representation

Represent  $x$  with  $n$  bits as follows:

If  $0 \leq x < 2^{n-1}$ , use the  $n$ -bit unsigned representation of  $x$ .

If  $-2^{n-1} \leq x < 0$ , use the  $n$ -bit unsigned representation of  $2^n + x$ .

**Key property:**

Two's complement representation of any number  $y$  is equivalent to  $y \bmod 2^n$  so arithmetic works  $\bmod 2^n$ .

**Examples:**

$$99 = 64 + 32 + 2 + 1$$

$$18 = 16 + 2$$

$$2^8 - 18 = 256 - 18 = 238 = 128 + 64 + 32 + 8 + 4 + 2$$

$$81 = 64 + 16 + 1$$

**So for  $n = 8$ :**

$$99 = 0110\ 0011$$

$$-18 = 1110\ 1110$$

$$81 = 0101\ 0001$$

# Two's complement integer representation

Represent  $x$  with  $n$  bits as follows:

If  $0 \leq x < 2^{n-1}$ , use the  $n$ -bit unsigned representation of  $x$ .

If  $-2^{n-1} \leq x < 0$ , use the  $n$ -bit unsigned representation of  $2^n + x$ .

**Key property:**

Two's complement representation of any number  $y$  is equivalent to  $y \bmod 2^n$  so arithmetic works  $\bmod 2^n$ .

**Examples:**

$$99 = 64 + 32 + 2 + 1$$

$$18 = 16 + 2$$

$$2^8 - 18 = 256 - 18 = 238 = 128 + 64 + 32 + 8 + 4 + 2$$

$$81 = 64 + 16 + 1$$

**So for  $n = 8$ :**

$$99 = 0110\ 0011$$

$$-18 = 1110\ 1110$$

$$81 = 0101\ 0001$$

Arithmetic  $\bmod 2^n$  is easy in hardware: just throw away bits  $n + 1$  and higher.

# Computing the two's complement representation

For  $-2^{n-1} \leq x < 0$ ,  $x$  is represented using the  $n$ -bit unsigned representation of  $2^n + x = 2^n - |x|$ . To compute this value:

- Compute the  $n$ -bit unsigned representation of  $|x|$ .
- Flip the bits of  $|x|$  to get the representation of  $2^n - 1 - |x|$ .
- Add 1 to get  $2^n - |x|$ .

This computation works because  $|x| + \overline{|x|}$  is all 1s, which represents  $2^n - 1$ . So we have  $\overline{|x|} = 2^n - 1 - |x|$  and  $\overline{|x|} + 1 = 2^n - |x|$ .

# Computing the two's complement representation

For  $-2^{n-1} \leq x < 0$ ,  $x$  is represented using the  $n$ -bit unsigned representation of  $2^n + x = 2^n - |x|$ . To compute this value:

- Compute the  $n$ -bit unsigned representation of  $|x|$ .
- Flip the bits of  $|x|$  to get the representation of  $2^n - 1 - |x|$ .
- Add 1 to get  $2^n - |x|$ .

This computation works because  $|x| + \overline{|x|}$  is all 1s, which represents  $2^n - 1$ . So we have  $\overline{|x|} = 2^n - 1 - |x|$  and  $\overline{|x|} + 1 = 2^n - |x|$ .

## Example: -18 in 8-bit two's complement

18 in 8-bit unsigned: 0001 0010

Flip the bits: 1110 1101

Add 1: 1110 1110

# Applications of modular arithmetic

Modular arithmetic is the basis of modern computing, with many applications.

Examples include

- hashing,
- pseudo-random numbers, and
- simple ciphers.

# Hashing

## Problem:

We want to map a small number of data values from a large domain  $\{0, 1, \dots, M - 1\}$  into a small set of locations  $\{0, 1, \dots, n - 1\}$  to be able to quickly check if a value is present.

## Solution:

Compute  $\text{hash}(x) = x \bmod p$  for a prime  $p$  close to  $n$ .

Or, compute  $\text{hash}(x) = ax + b \bmod p$  for a prime  $p$  close to  $n$ .

**This approach depends on all of the bits of data the data.**

Helps avoid collisions due to similar values.

But need to manage them if they occur.

# Pseudo-random number generation

## Linear Congruential method

$$x_{n+1} = (ax_n + c) \bmod m$$

Choose  $x_0$  randomly and  $a, c, m$  carefully to produce a sequence of  $x_n$ 's.

# Pseudo-random number generation

## Linear Congruential method

$$x_{n+1} = (ax_n + c) \bmod m$$

Choose  $x_0$  randomly and  $a, c, m$  carefully to produce a sequence of  $x_n$ 's.

### Example

$$a = 1103515245, c = 12345, m = 2^{31} \text{ from BSD}$$

$$x_0 = 311$$

$$x_1 = 1743353508, x_2 = 1197845517, x_3 = 1069836226, \dots$$



# Simple ciphers

## Cesar or shift cipher

Treat letters as numbers: A = 0, B = 1, ...

$$f(p) = (p + k) \bmod 26$$

$$f^{-1}(p) = (p - k) \bmod 26$$

## More general version

$$f(p) = (ap + b) \bmod 26$$

$$f^{-1}(p) = (a^{-1}(p - b)) \bmod 26$$

# Simple ciphers

## Cesar or shift cipher

Treat letters as numbers: A = 0, B = 1, ...

$$f(p) = (p + k) \bmod 26$$

$$f^{-1}(p) = (p - k) \bmod 26$$

## More general version

$$f(p) = (ap + b) \bmod 26$$

$$f^{-1}(p) = (a^{-1}(p - b)) \bmod 26$$

$a^{-1}$  is the *multiplicative inverse* of  $a$  modulo 26, and we'll soon see how to compute these inverses.

# Primes

Fundamental theorem of arithmetic, Euclid's theorem, factoring.

# Primality

## Prime number

An integer  $p > 1$  is called *prime* if its only positive factors are 1 and  $p$ .

## Composite number

An integer  $c > 1$  is called *composite* if it is not prime.

# Primality

## Prime number

An integer  $p > 1$  is called *prime* if its only positive factors are 1 and  $p$ .

## Composite number

An integer  $c > 1$  is called *composite* if it is not prime.

**A prime number is divisible only by itself and 1.**

We say that  $a$  is a *factor* of  $b$  if  $a|b$ .

**Note that 1 is neither prime nor composite.**

The above definitions apply only to integers greater than 1.

# A key theorem about all positive integers

## Fundamental theorem of arithmetic

Every positive integer greater than 1 has a unique prime factorization.

# A key theorem about all positive integers

## Fundamental theorem of arithmetic

Every positive integer greater than 1 has a unique prime factorization.

In other words, every integer  $n > 1$  can be written uniquely as a prime, or the product of two or more primes ordered by size.

### Examples

$$48 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3$$

$$591 = 3 \cdot 197$$

$$45,523 = 45,523$$

$$321,950 = 2 \cdot 5 \cdot 5 \cdot 47 \cdot 137$$

$$1,234,567,890 = 2 \cdot 3 \cdot 3 \cdot 5 \cdot 3,607 \cdot 3,803$$

# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.



# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.

**Proof by contradiction:**

# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.

## Proof by contradiction:

Suppose that there are finitely many primes:  $p_1, \dots, p_n$ .

# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.

### Proof by contradiction:

Suppose that there are finitely many primes:  $p_1, \dots, p_n$ .

Define the number  $P = p_1 \cdot \dots \cdot p_n$ , and let  $Q = P + 1$ .

# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.

### Proof by contradiction:

Suppose that there are finitely many primes:  $p_1, \dots, p_n$ .

Define the number  $P = p_1 \cdot \dots \cdot p_n$ , and let  $Q = P + 1$ .

**Case 1:** If  $Q > 1$  is prime, then  $Q$  is a prime different from all of  $p_1, \dots, p_n$ , since it is bigger than all of them. This contradicts the assumption that the list  $p_1, \dots, p_n$  includes all primes.

# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.

### Proof by contradiction:

Suppose that there are finitely many primes:  $p_1, \dots, p_n$ .

Define the number  $P = p_1 \cdot \dots \cdot p_n$ , and let  $Q = P + 1$ .

**Case 1:** If  $Q > 1$  is prime, then  $Q$  is a prime different from all of  $p_1, \dots, p_n$ , since it is bigger than all of them. This contradicts the assumption that the list  $p_1, \dots, p_n$  includes all primes.

**Case 2:** If  $Q > 1$  is not prime, then  $Q$  has some prime factor  $p$ , which must be in  $p_1, \dots, p_n$ . Therefore  $p|P$  and  $p|Q$  so  $P = jp$  and  $Q = kp$  for some integers  $j, k$ . We then have  $Q - P = (k - j)p = 1$ , which means that  $p|1$ . But no prime divides 1, leading again to a contradiction.

# A key theorem about primes

## Euclid's theorem

There are infinitely many primes.

### Proof by contradiction:

Suppose that there are finitely many primes:  $p_1, \dots, p_n$ .

Define the number  $P = p_1 \cdot \dots \cdot p_n$ , and let  $Q = P + 1$ .

**Case 1:** If  $Q > 1$  is prime, then  $Q$  is a prime different from all of  $p_1, \dots, p_n$ , since it is bigger than all of them. This contradicts the assumption that the list  $p_1, \dots, p_n$  includes all primes.

**Case 2:** If  $Q > 1$  is not prime, then  $Q$  has some prime factor  $p$ , which must be in  $p_1, \dots, p_n$ . Therefore  $p|P$  and  $p|Q$  so  $P = jp$  and  $Q = kp$  for some integers  $j, k$ . We then have  $Q - P = (k - j)p = 1$ , which means that  $p|1$ . But no prime divides 1, leading again to a contradiction.

Since both cases are contradictions, the assumption must be false.  $\square$

# Important algorithmic problems

## Primality testing

Given an integer  $n$ , determine if  $n$  is prime.

## Factoring

Given an integer  $n$ , determine the prime factorization of  $n$ .

# Important algorithmic problems

## Primality testing

Given an integer  $n$ , determine if  $n$  is prime.

## Factoring

Given an integer  $n$ , determine the prime factorization of  $n$ .

- We don't know of an efficient algorithm for factoring large numbers.
- The security of commonly used cryptographic protocols (e.g., RSA) hinges on this fact.
- For example, it took two years and thousands of machine-hours to factor a 232-digit (768-bit) number known as [RSA-768](#).
- But factoring is easy for quantum computers!



# Greatest Common Divisors (GCD)

GCD definition and properties.

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

## Examples:

$$\text{GCD}(100, 125) =$$

$$\text{GCD}(17, 49) =$$

$$\text{GCD}(11, 66) =$$

$$\text{GCD}(13, 0) =$$

$$\text{GCD}(180, 252) =$$

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

## Examples:

$$\text{GCD}(100, 125) = 25$$

$$\text{GCD}(17, 49) =$$

$$\text{GCD}(11, 66) =$$

$$\text{GCD}(13, 0) =$$

$$\text{GCD}(180, 252) =$$

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

## Examples:

$$\text{GCD}(100, 125) = 25$$

$$\text{GCD}(17, 49) = 1$$

$$\text{GCD}(11, 66) =$$

$$\text{GCD}(13, 0) =$$

$$\text{GCD}(180, 252) =$$

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

## Examples:

$$\text{GCD}(100, 125) = 25$$

$$\text{GCD}(17, 49) = 1$$

$$\text{GCD}(11, 66) = 11$$

$$\text{GCD}(13, 0) =$$

$$\text{GCD}(180, 252) =$$

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

## Examples:

$$\text{GCD}(100, 125) = 25$$

$$\text{GCD}(17, 49) = 1$$

$$\text{GCD}(11, 66) = 11$$

$$\text{GCD}(13, 0) = 13$$

$$\text{GCD}(180, 252) =$$

# Definition of greatest common divisor (GCD)

## Greatest common divisor (GCD)

The greatest common divisor of integers  $a$  and  $b$ , written as  $\text{GCD}(a, b)$ , is the largest integer  $d$  such that  $d|a$  and  $d|b$ .

## Examples:

$$\text{GCD}(100, 125) = 25$$

$$\text{GCD}(17, 49) = 1$$

$$\text{GCD}(11, 66) = 11$$

$$\text{GCD}(13, 0) = 13$$

$$\text{GCD}(180, 252) = 36$$



# How can we compute $\text{GCD}(a, b)$ ?

A naive approach is to first factor both  $a$  and  $b$ :

$$a = 2^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 = 46,200$$

$$b = 2 \cdot 3^2 \cdot 5^3 \cdot 7 \cdot 13 = 204,750$$

# How can we compute $\text{GCD}(a, b)$ ?

A naive approach is to first factor both  $a$  and  $b$ :

$$a = 2^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 = 46,200$$

$$b = 2 \cdot 3^2 \cdot 5^3 \cdot 7 \cdot 13 = 204,750$$

And then compute  $\text{GCD}(a, b)$  as follows:

$$\text{GCD}(a, b) = 2^{\min(3,1)} \cdot 3^{\min(1,2)} \cdot 5^{\min(2,3)} \cdot 7^{\min(1,1)} \cdot 11^{\min(1,0)} \cdot 13^{\min(0,1)}$$

# How can we compute $\text{GCD}(a, b)$ ?

A naive approach is to first factor both  $a$  and  $b$ :

$$a = 2^3 \cdot 3 \cdot 5^2 \cdot 7 \cdot 11 = 46,200$$

$$b = 2 \cdot 3^2 \cdot 5^3 \cdot 7 \cdot 13 = 204,750$$

And then compute  $\text{GCD}(a, b)$  as follows:

$$\text{GCD}(a, b) = 2^{\min(3,1)} \cdot 3^{\min(1,2)} \cdot 5^{\min(2,3)} \cdot 7^{\min(1,1)} \cdot 11^{\min(1,0)} \cdot 13^{\min(0,1)}$$

But factoring is expensive! Can we compute  $\text{GCD}(a, b)$  without factoring?

# Euclidean algorithm

Computing GCDs with the Euclidean algorithm.

# Euclidean algorithm is based on two useful facts

## **GCD(a, 0)**

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

# Euclidean algorithm is based on two useful facts

## **GCD(a, 0)**

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

# Euclidean algorithm is based on two useful facts

## **GCD(a, 0)**

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## **GCD and modulo**

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

# Euclidean algorithm is based on two useful facts

## **GCD(a, 0)**

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## **GCD and modulo**

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

**Proof:**



# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

**Proof:**

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ . So,  $d|(a \bmod b)$  and since  $d|b$ , we have that  $d = \text{GCD}(a, b) \leq \text{GCD}(b, a \bmod b)$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ . So,  $d|(a \bmod b)$  and since  $d|b$ , we have that  $d = \text{GCD}(a, b) \leq \text{GCD}(b, a \bmod b)$ .

Next, let  $e = \text{GCD}(b, a \bmod b)$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ . So,  $d|(a \bmod b)$  and since  $d|b$ , we have that  $d = \text{GCD}(a, b) \leq \text{GCD}(b, a \bmod b)$ .

Next, let  $e = \text{GCD}(b, a \bmod b)$ . Then  $e|b$  and  $e|(a \bmod b)$ , so  $b = me$  and  $a \bmod b = ne$  for some  $m, n \in \mathbb{Z}$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

### Proof:

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ . So,  $d|(a \bmod b)$  and since  $d|b$ , we have that  $d = \text{GCD}(a, b) \leq \text{GCD}(b, a \bmod b)$ .

Next, let  $e = \text{GCD}(b, a \bmod b)$ . Then  $e|b$  and  $e|(a \bmod b)$ , so  $b = me$  and  $a \bmod b = ne$  for some  $m, n \in \mathbb{Z}$ . Therefore,  $a = qb + a \bmod b = qme + ne$ .



# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

**Proof:**

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ . So,  $d|(a \bmod b)$  and since  $d|b$ , we have that  $d = \text{GCD}(a, b) \leq \text{GCD}(b, a \bmod b)$ .

Next, let  $e = \text{GCD}(b, a \bmod b)$ . Then  $e|b$  and  $e|(a \bmod b)$ , so  $b = me$  and  $a \bmod b = ne$  for some  $m, n \in \mathbb{Z}$ . Therefore,  $a = qb + a \bmod b = qme + ne$ . So,  $e|a$  and  $e|b$ , we have that  $e = \text{GCD}(b, a \bmod b) \leq \text{GCD}(a, b)$ .

# Euclidean algorithm is based on two useful facts

## GCD(a, 0)

If  $a$  is a positive integer, then  $\text{GCD}(a, 0) = a$ .

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

If  $a$  and  $b$  are positive integers, then  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$ .

**Proof:**

First note that by definition of mod,  $a = qb + a \bmod b$  for some integer  $q = a \text{ div } b$ .

Now, let  $d = \text{GCD}(a, b)$ . Then  $d|a$  and  $d|b$ , so  $a = kd$  and  $b = jd$  for some  $k, j \in \mathbb{Z}$ .

Therefore,  $a \bmod b = a - qb = kd - qjd = d(k - qj)$ . So,  $d|(a \bmod b)$  and since  $d|b$ , we have that  $d = \text{GCD}(a, b) \leq \text{GCD}(b, a \bmod b)$ .

Next, let  $e = \text{GCD}(b, a \bmod b)$ . Then  $e|b$  and  $e|(a \bmod b)$ , so  $b = me$  and  $a \bmod b = ne$  for some  $m, n \in \mathbb{Z}$ . Therefore,  $a = qb + a \bmod b = qme + ne$ . So,  $e|a$  and  $e|b$ , we have that  $e = \text{GCD}(b, a \bmod b) \leq \text{GCD}(a, b)$ . The result follows from these cases.  $\square$

# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.
public static int gcd(int a, int b) {
    if (b == 0)
        return a;           // GCD(a, 0) = a
    else
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)
}
```

# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.
public static int gcd(int a, int b) {
    if (b == 0)
        return a;           // GCD(a, 0) = a
    else
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)
}
```

**GCD(660, 126)**

# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.
public static int gcd(int a, int b) {
    if (b == 0)
        return a;           // GCD(a, 0) = a
    else
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)
}
```

**GCD(660, 126)**

=  $\text{GCD}(126, 660 \bmod 126) = \text{GCD}(126, 30)$

# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.  
public static int gcd(int a, int b) {  
    if (b == 0)  
        return a;           // GCD(a, 0) = a  
    else  
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)  
}
```

**GCD(660, 126)**

=  $\text{GCD}(126, 660 \bmod 126) = \text{GCD}(126, 30)$

=  $\text{GCD}(30, 126 \bmod 30) = \text{GCD}(30, 6)$

# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.  
public static int gcd(int a, int b) {  
    if (b == 0)  
        return a;           // GCD(a, 0) = a  
    else  
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)  
}
```

**GCD(660, 126)**

$$\begin{aligned} &= \text{GCD}(126, 660 \bmod 126) = \text{GCD}(126, 30) \\ &= \text{GCD}(30, 126 \bmod 30) = \text{GCD}(30, 6) \\ &= \text{GCD}(6, 30 \bmod 6) = \text{GCD}(6, 0) \end{aligned}$$

# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.  
public static int gcd(int a, int b) {  
    if (b == 0)  
        return a;           // GCD(a, 0) = a  
    else  
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)  
}
```

**GCD(660, 126)**

$$\begin{aligned} &= \text{GCD}(126, 660 \bmod 126) = \text{GCD}(126, 30) \\ &= \text{GCD}(30, 126 \bmod 30) = \text{GCD}(30, 6) \\ &= \text{GCD}(6, 30 \bmod 6) = \text{GCD}(6, 0) \\ &= 6 \end{aligned}$$



# Euclidean algorithm

Apply  $\text{GCD}(a, b) = \text{GCD}(b, a \bmod b)$  until you get  $\text{GCD}(a, 0) = a$ .

Example **implementation**:

```
// Assumes a >= b >= 0.
public static int gcd(int a, int b) {
    if (b == 0)
        return a;           // GCD(a, 0) = a
    else
        return gcd(b, a % b); // GCD(a, b) = GCD(b, a mod b)
}
```

**GCD(660, 126)**

$$\begin{aligned} &= \text{GCD}(126, 660 \bmod 126) = \text{GCD}(126, 30) \\ &= \text{GCD}(30, 126 \bmod 30) = \text{GCD}(30, 6) \\ &= \text{GCD}(6, 30 \bmod 6) = \text{GCD}(6, 0) \\ &= 6 \end{aligned}$$

**In tableau form:**

$$\begin{aligned} 660 &= 5 * 126 + 30 \\ 126 &= 4 * 30 + 6 \\ 30 &= 5 * 6 + 0 \end{aligned}$$

# Extended Euclidean algorithm

Bézout's theorem and the extended Euclidean algorithm.

# Bézout's theorem about GCDs

## Bézout's theorem

If  $a$  and  $b$  are positive integers, then there exist integers  $s$  and  $t$  such that  $\text{GCD}(a, b) = sa + tb$ .

# Bézout's theorem about GCDs

## Bézout's theorem

If  $a$  and  $b$  are positive integers, then there exist integers  $s$  and  $t$  such that  $\text{GCD}(a, b) = sa + tb$ .

We can extend Euclidean algorithm to find  $s$  and  $t$  in addition to computing  $\text{GCD}(a, b)$ .

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.

$$\text{GCD}(35, 27) = 35s + 27t.$$

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.

$$\text{GCD}(35, 27) = 35s + 27t.$$

$$\begin{array}{l} a = q * b + r \\ 35 = 1 * 27 + 8 \\ 27 = 3 * 8 + 3 \\ 8 = 2 * 3 + 2 \\ 3 = 1 * 2 + 1 \end{array}$$

$$\begin{array}{lll} \text{GCD}(a, b) & \text{GCD}(b, a \bmod b) & r = a \bmod b \\ \text{GCD}(35, 27) = \text{GCD}(27, 35 \bmod 27) = \text{GCD}(27, 8) & & \\ = \text{GCD}(8, 27 \bmod 8) & = \text{GCD}(8, 3) & \\ = \text{GCD}(3, 8 \bmod 3) & = \text{GCD}(3, 2) & \\ = \text{GCD}(2, 3 \bmod 2) & = \text{GCD}(2, 1) & \\ = \text{GCD}(1, 2 \bmod 1) & = \text{GCD}(1, 0) & \end{array}$$

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.
2. Solve the equations for  $r$  in the tableau.

$$\text{GCD}(35, 27) = 35s + 27t.$$

$$\begin{array}{l} a = q * b + r \\ 35 = 1 * 27 + 8 \\ 27 = 3 * 8 + 3 \\ 8 = 2 * 3 + 2 \\ 3 = 1 * 2 + 1 \end{array}$$

$$\begin{array}{l} r = a - q * b \\ 8 = 35 - 1 * 27 \\ 3 = 27 - 3 * 8 \\ 2 = 8 - 2 * 3 \\ 1 = 3 - 1 * 2 \end{array}$$

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.
2. Solve the equations for  $r$  in the tableau.
3. Back substitute the equations for  $r$ .

$$r = a - q * b$$

$$8 = 35 - 1 * 27$$

$$3 = 27 - 3 * 8$$

$$2 = 8 - 2 * 3$$

$$1 = 3 - 1 * 2$$

$$\text{GCD}(35, 27) = 35s + 27t.$$



# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.
2. Solve the equations for  $r$  in the tableau.
3. Back substitute the equations for  $r$ .

$$r = a - q * b$$

$$8 = 35 - 1 * 27$$

$$3 = 27 - 3 * 8$$

$$2 = 8 - 2 * 3$$

$$1 = 3 - 1 * 2$$

$$1 = 3 - 1 * (8 - 2 * 3)$$

$$= 3 - 8 + 2 * 3$$

$$= (-1) * 8 + 3 * 3$$

$$\text{GCD}(35, 27) = 35s + 27t.$$

Plug in the def of 2.

Group 8's and 3's.

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.
2. Solve the equations for  $r$  in the tableau.
3. Back substitute the equations for  $r$ .

$$\text{GCD}(35, 27) = 35s + 27t.$$

$$r = a - q * b$$

$$8 = 35 - 1 * 27$$

$$3 = 27 - 3 * 8$$

$$2 = 8 - 2 * 3$$

$$1 = 3 - 1 * 2$$

$$1 = 3 - 1 * (8 - 2 * 3)$$

$$= 3 - 8 + 2 * 3$$

$$= (-1) * 8 + 3 * 3$$

$$= (-1) * 8 + 3 * (27 - 3 * 8)$$

$$= (-1) * 8 + 3 * 27 + (-9) * 8$$

$$= 3 * 27 + (-10) * 8$$

Plug in the def of 2.

Group 8's and 3's.

Plug in the def of 3.

Group 8's and 27's.

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.
2. Solve the equations for  $r$  in the tableau.
3. Back substitute the equations for  $r$ .

$$\text{GCD}(35, 27) = 35s + 27t.$$

$$r = a - q * b$$

$$8 = 35 - 1 * 27$$

$$3 = 27 - 3 * 8$$

$$2 = 8 - 2 * 3$$

$$1 = 3 - 1 * 2$$

$$1 = 3 - 1 * (8 - 2 * 3)$$

$$= 3 - 8 + 2 * 3$$

$$= (-1) * 8 + 3 * 3$$

$$= (-1) * 8 + 3 * (27 - 3 * 8)$$

$$= (-1) * 8 + 3 * 27 + (-9) * 8$$

$$= 3 * 27 + (-10) * 8$$

$$= 3 * 27 + (-10) * (35 - 1 * 27)$$

$$= 3 * 27 + (-10) * 35 + 10 * 27$$

$$= 13 * 27 + (-10) * 35$$

Plug in the def of 2.

Group 8's and 3's.

Plug in the def of 3.

Group 8's and 27's.

Plug in the def of 8.

Group 27's and 35's.

# Multiplicative inverse mod $m$

Suppose  $\text{GCD}(a, m) = 1$ .

By Bézout's Theorem, there exist integers  $s$  and  $t$  such that  $sa + tm = 1$ .

**$s \bmod m$  is the *multiplicative inverse* of  $a$**

$$1 = (sa + tm) \bmod m = sa \bmod m$$

# Using multiplicative inverses to solve modular equations

Solve:  $7x \equiv 1 \pmod{26}$

# Using multiplicative inverses to solve modular equations

Solve:  $7x \equiv 1 \pmod{26}$

① Compute GCD and keep the tableau.

$$\begin{aligned} \text{GCD}(26, 7) &= \text{GCD}(7, 5) = \text{GCD}(5, 2) \\ &= \text{GCD}(2, 1) = \text{GCD}(1, 0) \\ &= 1 \end{aligned}$$

# Using multiplicative inverses to solve modular equations

Solve:  $7x \equiv 1 \pmod{26}$

① Compute GCD and keep the tableau.

$$\begin{aligned}\text{GCD}(26, 7) &= \text{GCD}(7, 5) = \text{GCD}(5, 2) \\ &= \text{GCD}(2, 1) = \text{GCD}(1, 0) \\ &= 1\end{aligned}$$

② Solve the equations for  $r$  in the tableau.

$a = b * q + r$	$r = a - b * q$
$26 = 7 * 3 + 5$	$5 = 26 - 7 * 3$
$7 = 5 * 1 + 2$	$2 = 7 - 5 * 1$
$5 = 2 * 2 + 1$	$1 = 5 - 2 * 2$

# Using multiplicative inverses to solve modular equations

Solve:  $7x \equiv 1 \pmod{26}$

① Compute GCD and keep the tableau.

$$\begin{aligned}\text{GCD}(26, 7) &= \text{GCD}(7, 5) = \text{GCD}(5, 2) \\ &= \text{GCD}(2, 1) = \text{GCD}(1, 0) \\ &= 1\end{aligned}$$

② Solve the equations for  $r$  in the tableau.

$a = b * q + r$	$r = a - b * q$
$26 = 7 * 3 + 5$	$5 = 26 - 7 * 3$
$7 = 5 * 1 + 2$	$2 = 7 - 5 * 1$
$5 = 2 * 2 + 1$	$1 = 5 - 2 * 2$

③ Back substitute the equations for  $r$ .

$$\begin{aligned}1 &= 5 - 2 * (7 - 5 * 1) \\ &= (-2) * 7 + 3 * 5 \\ &= (-2) * 7 + 3 * (26 - 7 * 3) \\ &= (-11) * 7 + 3 * 26\end{aligned}$$



# Using multiplicative inverses to solve modular equations

Solve:  $7x \equiv 1 \pmod{26}$

① Compute GCD and keep the tableau.

$$\begin{aligned}\text{GCD}(26, 7) &= \text{GCD}(7, 5) = \text{GCD}(5, 2) \\ &= \text{GCD}(2, 1) = \text{GCD}(1, 0) \\ &= 1\end{aligned}$$

② Solve the equations for  $r$  in the tableau.

$a = b * q + r$	$r = a - b * q$
$26 = 7 * 3 + 5$	$5 = 26 - 7 * 3$
$7 = 5 * 1 + 2$	$2 = 7 - 5 * 1$
$5 = 2 * 2 + 1$	$1 = 5 - 2 * 2$

③ Back substitute the equations for  $r$ .

$$\begin{aligned}1 &= 5 - 2 * (7 - 5 * 1) \\ &= (-2) * 7 + 3 * 5 \\ &= (-2) * 7 + 3 * (26 - 7 * 3) \\ &= (-11) * 7 + 3 * 26\end{aligned}$$

④ Solve for  $x$ .

- Multiplicative inverse of 7 mod 26
  - $(-11) \pmod{26} = 15$
- So,  $x = 26k + 15$  for  $k \in \mathbb{Z}$ .

# Summary

Every positive integer  $p > 1$  is either prime or composite.

$p$  is *prime* if its only factors are  $p$  and 1.

Otherwise,  $p$  is composite.

**GCD(a, b)** is the greatest integer that divides both **a** and **b**.

It can be computed efficiently using the Euclidean algorithm.

By Bézout's Theorem, **GCD(a, b) = sa + tb** for some integers **s, t**.

$s, t$  can be computed using the extended Euclidean algorithm.

If  $\text{GCD}(a, b) = 1$ ,  $s \bmod b$  is the multiplicative inverse of  $a$  modulo  $b$ .