# CSE 311 Lecture 14: Euclidean Algorithm and Modular Equations

Emina Torlak and Kevin Zatloukal

# Topics

**Primes and GCD**

A quick review of Lecture 13.

**Euclidean algorithm**

Computing GCDs with the Euclidean algorithm.

**Extended Euclidean algorithm**

Bézout's theorem and the extended Euclidean algorithm.

**Modular equations**

Solving modular equations with the extended Euclidean algorithm.

**Modular exponentiation**

A fast algorithm for computing $a^k \bmod m$.

# Primes and GCD

A quick review of Lecture 13.

# Primes and composites: definitions and theorems

**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.

**Fundamental theorem of arithmetic**

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

**Euclid's theorem**

There are infinitely many primes.

# Greatest common divisor (GCD): definition

**Greatest common divisor (GCD)**

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

We can compute GCDs efficiently using the Euclidean algorithm.

# Greatest common divisor (GCD): definition

**Greatest common divisor (GCD)**

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

We can compute GCDs efficiently using the Euclidean algorithm. Invented in 300 BC!

# 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 $\mathrm{GCD}(a, 0) = a$.

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

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 $\mathrm{GCD}(a, 0) = a$.

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

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

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

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

Now, let $d$ be a common divisor of $a$ and $b$.

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$.
Now, let $d$ be a common divisor of $a$ and $b$. 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 $\mathrm{GCD}(a, 0) = a$.

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$. Now, let $d$ be a common divisor of $a$ and $b$. 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 $\mathrm{GCD}(a, 0) = a$.

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$. Now, let $d$ be a common divisor of $a$ and $b$. 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, $d|(a \bmod b)$.

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$.
Now, let $d$ be a common divisor of $a$ and $b$. 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, $d|(a \bmod b)$.
Next, let $e$ be a common divisor of $b$ and $a \bmod b$.

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$.
Now, let $d$ be a common divisor of $a$ and $b$. 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, $d|(a \bmod b)$.
Next, let $e$ be a common divisor of $b$ and $a \bmod b$. 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 $\mathrm{GCD}(a, 0) = a$.

Proof follows straightforwardly from the definition of GCD and divisibility.

## GCD and modulo

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$.
Now, let $d$ be a common divisor of $a$ and $b$. 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, $d|(a \bmod b)$.
Next, let $e$ be a common divisor of $b$ and $a \bmod b$. 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 $\mathrm{GCD}(a, 0) = a$.

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$. Now, let $d$ be a common divisor of $a$ and $b$. 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, $d|(a \bmod b)$. Next, let $e$ be a common divisor of $b$ and $a \bmod b$. 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, $e|a$.

# Euclidean algorithm is based on two useful facts

**GCD(a, 0)**

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

Proof follows straightforwardly from the definition of GCD and divisibility.

**GCD and modulo**

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

**Proof:**

First note that by definition of mod, $a = qb + a \bmod b$ for some integer $q = a \operatorname{div} b$. Now, let $d$ be a common divisor of $a$ and $b$. 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, $d|(a \bmod b)$. Next, let $e$ be a common divisor of $b$ and $a \bmod b$. 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, $e|a$. This shows that $a, b$ and $b, a \bmod b$ have the same set of common divisors, and must therefore have the same greatest common divisor. □

# Euclidean algorithm

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

# Euclidean algorithm

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

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

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

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)**

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

# Euclidean algorithm

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

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)**

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

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

# Euclidean algorithm

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

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)**

$= \mathrm{GCD}(126, 660 \bmod 126) = \mathrm{GCD}(126, 30)$
$= \mathrm{GCD}(30, 126 \bmod 30) = \mathrm{GCD}(30, 6)$
$= \mathrm{GCD}(6, 30 \bmod 6) = \mathrm{GCD}(6, 0)$

# Euclidean algorithm

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

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)**

$= \mathrm{GCD}(126, 660 \bmod 126) = \mathrm{GCD}(126, 30)$
$= \mathrm{GCD}(30, 126 \bmod 30) = \mathrm{GCD}(30, 6)$
$= \mathrm{GCD}(6, 30 \bmod 6) = \mathrm{GCD}(6, 0)$
$= 6$

# 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)**

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

In *tableau form*:

660 = 5 * 126 + 30
126 = 4 * 30 + 6
30 = 5 * 6 + 0

# 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 $\mathrm{GCD}(a, b) = sa + tb$.

# Bézout's theorem about GCDs

**Bézout's theorem**

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

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

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.

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

# Extended Euclidean algorithm

1. Compute GCD and keep the tableau.

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

| $a = q * b + r$ |
|---|
| $35 = 1 * 27 + 8$ |
| $27 = 3 * 8 + 3$ |
| $8 = 2 * 3 + 2$ |
| $3 = 1 * 2 + 1$ |

$$
\begin{aligned}
\mathrm{GCD}(a, b) \quad & \mathrm{GCD}(b, a \bmod b) \quad && r = a \bmod b \\
\mathrm{GCD}(35, 27) &= \mathrm{GCD}(27, 35 \bmod 27) && = \mathrm{GCD}(27, 8) \\
&= \mathrm{GCD}(8, 27 \bmod 8) && = \mathrm{GCD}(8, 3) \\
&= \mathrm{GCD}(3, 8 \bmod 3) && = \mathrm{GCD}(3, 2) \\
&= \mathrm{GCD}(2, 3 \bmod 2) && = \mathrm{GCD}(2, 1) \\
&= \mathrm{GCD}(1, 2 \bmod 1) && = \mathrm{GCD}(1, 0)
\end{aligned}
$$

# Extended Euclidean algorithm

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

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

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

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

# 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 $r$.

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

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

# 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 $r$.

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

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

$$\begin{aligned} r_i &= r_{i-2} - q_i * r_{i-1} \\ r_0 &= a = 35 \\ r_1 &= b = 27 \\ r_2 &= r_0 - q_2 * r_1 = 8 \\ r_3 &= r_1 - q_3 * r_2 = 3 \\ r_4 &= r_2 - q_4 * r_3 = 2 \\ r_5 &= r_3 - q_5 * r_4 = 1 \end{aligned}$$

# 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 $r$.

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

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

$$1 = 3 - 1 * 2$$

$$r_5 = r_3 - q_5 * r_4.$$

$$\begin{aligned} r_i &= r_{i-2} - q_i * r_{i-1} \\ r_0 &= a = 35 \\ r_1 &= b = 27 \\ r_2 &= r_0 - q_2 * r_1 = 8 \\ r_3 &= r_1 - q_3 * r_2 = 3 \\ r_4 &= r_2 - q_4 * r_3 = 2 \\ r_5 &= r_3 - q_5 * r_4 = 1 \end{aligned}$$

# 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 $r$.

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

$r = a - q * b$

$8 = 35 - 1 * 27$

$3 = 27 - 3 * 8$

$2 = 8 - 2 * 3$

$1 = 3 - 1 * 2$

$r_i = r_{i-2} - q_i * r_{i-1}$

$r_0 = a = 35$

$r_1 = b = 27$

$r_2 = r_0 - q_2 * r_1 = 8$

$r_3 = r_1 - q_3 * r_2 = 3$

$r_4 = r_2 - q_4 * r_3 = 2$

$r_5 = r_3 - q_5 * r_4 = 1$

$1 = 3 - 1 * 2$

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

$r_5 = r_3 - q_5 * r_4.$

Plug in $r_4 = r_2 - q_4 * r_3.$

# 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 $r$.

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

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

$$
\begin{aligned}
1 &= 3 - 1 * 2 \\
&= 3 - 1 * (8 - 2 * 3) \\
&= (-1) * 8 + 3 * 3
\end{aligned}
$$

$r_5 = r_3 - q_5 * r_4.$

Plug in $r_4 = r_2 - q_4 * r_3.$

Combine $r_2, r_3$ terms.

$$
\begin{aligned}
r_i &= r_{i-2} - q_i * r_{i-1} \\
r_0 &= a = 35 \\
r_1 &= b = 27 \\
r_2 &= r_0 - q_2 * r_1 = 8 \\
r_3 &= r_1 - q_3 * r_2 = 3 \\
r_4 &= r_2 - q_4 * r_3 = 2 \\
r_5 &= r_3 - q_5 * r_4 = 1
\end{aligned}
$$

# 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 $r$.

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

$r = a - q * b$
$8 = 35 - 1 * 27$
$3 = 27 - 3 * 8$
$2 = 8 - 2 * 3$
$1 = 3 - 1 * 2$

$r_i = r_{i-2} - q_i * r_{i-1}$
$r_0 = a = 35$
$r_1 = b = 27$
$r_2 = r_0 - q_2 * r_1 = 8$
$r_3 = r_1 - q_3 * r_2 = 3$
$r_4 = r_2 - q_4 * r_3 = 2$
$r_5 = r_3 - q_5 * r_4 = 1$

$1 = 3 - 1 * 2$
$= 3 - 1 * (8 - 2 * 3)$
$= (-1) * 8 + 3 * 3$
$= (-1) * 8 + 3 * (27 - 3 * 8)$

$r_5 = r_3 - q_5 * r_4$.
Plug in $r_4 = r_2 - q_4 * r_3$.
Combine $r_2$, $r_3$ terms.
Plug in $r_3 = r_1 - q_3 * r_2$.

# 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 $r$.

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

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

$$
\begin{aligned}
1 &= 3 - 1 * 2 \\
&= 3 - 1 * (8 - 2 * 3) \\
&= (-1) * 8 + 3 * 3 \\
&= (-1) * 8 + 3 * (27 - 3 * 8) \\
&= 3 * 27 + (-10) * 8
\end{aligned}
$$

$r_5 = r_3 - q_5 * r_4.$

Plug in $r_4 = r_2 - q_4 * r_3.$

Combine $r_2, r_3$ terms.

Plug in $r_3 = r_1 - q_3 * r_2.$

Combine $r_1, r_2$ terms.

$$
\begin{aligned}
r_i &= r_{i-2} - q_i * r_{i-1} \\
r_0 &= a = 35 \\
r_1 &= b = 27 \\
r_2 &= r_0 - q_2 * r_1 = 8 \\
r_3 &= r_1 - q_3 * r_2 = 3 \\
r_4 &= r_2 - q_4 * r_3 = 2 \\
r_5 &= r_3 - q_5 * r_4 = 1
\end{aligned}
$$

# 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 $r$.

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

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

$r_i = r_{i-2} - q_i * r_{i-1}$

$r_0 = a = 35$

$r_1 = b = 27$

$r_2 = r_0 - q_2 * r_1 = 8$

$r_3 = r_1 - q_3 * r_2 = 3$

$r_4 = r_2 - q_4 * r_3 = 2$

$r_5 = r_3 - q_5 * r_4 = 1$

$$
\begin{aligned}
1 &= 3 - 1 * 2 && r_5 = r_3 - q_5 * r_4. \\
&= 3 - 1 * (8 - 2 * 3) && \text{Plug in } r_4 = r_2 - q_4 * r_3. \\
&= (-1) * 8 + 3 * 3 && \text{Combine } r_2, r_3 \text{ terms.} \\
&= (-1) * 8 + 3 * (27 - 3 * 8) && \text{Plug in } r_3 = r_1 - q_3 * r_2. \\
&= 3 * 27 + (-10) * 8 && \text{Combine } r_1, r_2 \text{ terms.} \\
&= 3 * 27 + (-10) * (35 - 1 * 27) && \text{Plug in } r_2 = r_0 - q_2 * r_1.
\end{aligned}
$$

# 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 $r$.

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

$r = a - q * b$

$8 = 35 - 1 * 27$

$3 = 27 - 3 * 8$

$2 = 8 - 2 * 3$

$1 = 3 - 1 * 2$

$r_i = r_{i-2} - q_i * r_{i-1}$

$r_0 = a = 35$

$r_1 = b = 27$

$r_2 = r_0 - q_2 * r_1 = 8$

$r_3 = r_1 - q_3 * r_2 = 3$

$r_4 = r_2 - q_4 * r_3 = 2$

$r_5 = r_3 - q_5 * r_4 = 1$

$$
\begin{aligned}
1 &= 3 - 1 * 2 && r_5 = r_3 - q_5 * r_4. \\
&= 3 - 1 * (8 - 2 * 3) && \text{Plug in } r_4 = r_2 - q_4 * r_3. \\
&= (-1) * 8 + 3 * 3 && \text{Combine } r_2, r_3 \text{ terms.} \\
&= (-1) * 8 + 3 * (27 - 3 * 8) && \text{Plug in } r_3 = r_1 - q_3 * r_2. \\
&= 3 * 27 + (-10) * 8 && \text{Combine } r_1, r_2 \text{ terms.} \\
&= 3 * 27 + (-10) * (35 - 1 * 27) && \text{Plug in } r_2 = r_0 - q_2 * r_1. \\
&= (-10) * 35 + 13 * 27 && \text{Combine } r_0, r_1 \text{ terms.}
\end{aligned}
$$

# Multiplicative inverse $\mathbf{mod\ m}$

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

# Multiplicative inverse $\mathbf{mod\ m}$

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

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

# Multiplicative inverse **mod m**

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

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

**s mod m is the *multiplicative inverse* of a modulo m**

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

# Multiplicative inverse $\mathbf{mod\ m}$

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

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

**$\mathbf{s\ mod\ m}$ is the *multiplicative inverse* of $\mathbf{a}$ modulo $\mathbf{m}$**

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

In other words, $s \bmod m$ is the multiplicative inverse of $a \bmod m$ iff $sa \equiv 1 \pmod m$.

# Multiplicative inverse mod m

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

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

**s mod m is the *multiplicative inverse* of a modulo m**

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

In other words, $s \bmod m$ is the multiplicative inverse of $a \bmod m$ iff $sa \equiv 1 \pmod m$.

So, we can compute multiplicative inverses with the extended Euclidean algorithm. These inverses let us solve modular equations.

# Modular equations

Solving modular equations with the extended Euclidean algorithm.

# 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.

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

# Using multiplicative inverses to solve modular equations

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

① Compute GCD and keep the tableau.

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

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

$$\begin{aligned} a &= q * b + r \\ 26 &= 3 * 7 + 5 \\ 7 &= 1 * 5 + 2 \\ 5 &= 2 * 2 + 1 \end{aligned}$$

$$\begin{aligned} r &= a - q * b \\ 5 &= 26 - 3 * 7 \\ 2 &= 7 - 1 * 5 \\ 1 &= 5 - 2 * 2 \end{aligned}$$

# Using multiplicative inverses to solve modular equations

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

① Compute GCD and keep the tableau.

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

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

$$\begin{array}{r} a = q * b + r \\ 26 = 3 * 7 + 5 \\ 7 = 1 * 5 + 2 \\ 5 = 2 * 2 + 1 \end{array}$$

$$\begin{array}{r} r = a - q * b \\ 5 = 26 - 3 * 7 \\ 2 = 7 - 1 * 5 \\ 1 = 5 - 2 * 2 \end{array}$$

③ Back substitute the equations for $r$.

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

# Using multiplicative inverses to solve modular equations

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

① Compute GCD and keep the tableau.

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

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

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

③ Back substitute the equations for $r$.

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

④ Solve for $x$.

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

# Solving a more general equation

Solve: $7y \equiv 3 \pmod{26}$

# Solving a more general equation

Solve: $7y \equiv 3 \pmod{26}$

**We computed that 15 is the multiplicative inverse of 7 modulo 26:**

That is, $7 * 15 \equiv 1 \pmod{26}$.

# Solving a more general equation

Solve: $7y \equiv 3 \pmod{26}$

**We computed that 15 is the multiplicative inverse of 7 modulo 26:**

That is, $7 * 15 \equiv 1 \pmod{26}$.

**By the multiplication property of mod, we have**

That is, $7 * 15 * 3 \equiv 1 * 3 \pmod{26}$.

# Solving a more general equation

Solve: $7y \equiv 3 \pmod{26}$

**We computed that 15 is the multiplicative inverse of 7 modulo 26:**

That is, $7 * 15 \equiv 1 \pmod{26}$.

**By the multiplication property of mod, we have**

That is, $7 * 15 * 3 \equiv 1 * 3 \pmod{26}$.

**So, any $\mathbf{y \equiv 15 * 3 \pmod{26}}$ is a solution.**

That is, $y = 19 + 26k$ for any $k \in \mathbb{Z}$ is a solution.

# Solving equations modulo a prime number

$\mathrm{GCD}(a, m) = 1$ if $m$ is prime and $0 < a < m$, so we can always solve modular equations for prime $m$.

$a +_7 b = (a + b) \bmod 7$

| + | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| **1** | 1 | 2 | 3 | 4 | 5 | 6 | 0 |
| **2** | 2 | 3 | 4 | 5 | 6 | 0 | 1 |
| **3** | 3 | 4 | 5 | 6 | 0 | 1 | 2 |
| **4** | 4 | 5 | 6 | 0 | 1 | 2 | 3 |
| **5** | 5 | 6 | 0 | 1 | 2 | 3 | 4 |
| **6** | 6 | 0 | 1 | 2 | 3 | 4 | 5 |

$a *_7 b = (a * b) \bmod 7$

| * | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| **2** | 0 | 2 | 4 | 6 | 1 | 3 | 5 |
| **3** | 0 | 3 | 6 | 2 | 5 | 1 | 4 |
| **4** | 0 | 4 | 1 | 5 | 2 | 6 | 3 |
| **5** | 0 | 5 | 3 | 1 | 6 | 4 | 2 |
| **6** | 0 | 6 | 5 | 4 | 3 | 2 | 1 |

# Modular exponentiation

A fast algorithm for computing $a^k \mod m$.

# The modular exponentiation problem: $\mathbf{a^k \bmod m}$

How would you compute $78365^{81453} \bmod 104729$?

# The modular exponentiation problem: $\mathbf{a^k \bmod m}$

How would you compute $78365^{81453} \bmod 104729$?

**Naive approach**

First compute $78365^{81453}$.
Then take the result modulo $104729$.

# The modular exponentiation problem: $a^k \bmod m$

How would you compute $78365^{81453} \bmod 104729$?

**Naive approach**

- First compute $78365^{81453}$.
- Then take the result modulo $104729$.

**This works but is very inefficient …**

- The intermediate result $78365^{81453}$ is a 1,324,257-bit number!
- But we only need the remainder mod 104,729, which is 17 bits.

# The modular exponentiation problem: $a^k \bmod m$

How would you compute $78365^{81453} \bmod 104729$?

**Naive approach**

First compute $78365^{81453}$.

Then take the result modulo $104729$.

**This works but is very inefficient …**

The intermediate result $78365^{81453}$ is a 1,324,257-bit number!

But we only need the remainder mod 104,729, which is 17 bits.

To keep the intermediate results small, we use *fast modular exponentiation*.

# Repeated squaring: $\mathbf{a^k \bmod m}$ for $\mathbf{k = 2^i}$

If $k = 2^i$, we can compute $a^k \bmod m$ in just $i$ steps.

Note that $a \bmod m \equiv a \pmod m$ and $b \bmod m \equiv b \pmod m$. So, we have $ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$.

# Repeated squaring: $\mathbf{a^k \bmod m}$ for $\mathbf{k = 2^i}$

If $k = 2^i$, we can compute $a^k \bmod m$ in just $i$ steps.

Note that $a \bmod m \equiv a \pmod m$ and $b \bmod m \equiv b \pmod m$. So, we have $ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$.

For example:

$$a^2 \bmod m = (a \bmod m)^2 \bmod m$$
$$a^4 \bmod m = (a^2 \bmod m)^2 \bmod m$$
$$a^8 \bmod m = (a^4 \bmod m)^2 \bmod m$$
$$a^{16} \bmod m = (a^8 \bmod m)^2 \bmod m$$
$$a^{32} \bmod m = (a^{16} \bmod m)^2 \bmod m$$

# Repeated squaring: $\mathbf{a^k \bmod m}$ for $\mathbf{k = 2^i}$

If $k = 2^i$, we can compute $a^k \bmod m$ in just $i$ steps.

Note that $a \bmod m \equiv a \pmod m$ and $b \bmod m \equiv b \pmod m$. So, we have $ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$.

For example:

$$a^2 \bmod m = (a \bmod m)^2 \bmod m$$
$$a^4 \bmod m = (a^2 \bmod m)^2 \bmod m$$
$$a^8 \bmod m = (a^4 \bmod m)^2 \bmod m$$
$$a^{16} \bmod m = (a^8 \bmod m)^2 \bmod m$$
$$a^{32} \bmod m = (a^{16} \bmod m)^2 \bmod m$$

What if $k$ is not a power of 2?

# Fast exponentiation: $\mathbf{a^k \bmod m}$ for all $\mathbf{k}$

$$a^{2j} \bmod m = (a^j \bmod m)^2 \bmod m$$
$$a^{2j+1} \bmod m = ((a \bmod m) * (a^{2j} \bmod m)) \bmod m$$

# Fast exponentiation: $a^k \bmod m$ for all $k$

$$a^{2j} \bmod m = (a^j \bmod m)^2 \bmod m$$
$$a^{2j+1} \bmod m = ((a \bmod m) * (a^{2j} \bmod m)) \bmod m$$

Example implementation:

```
// Assumes a > 0, k >= 0, m > 0.
public static long fastModExp(long a, long k, long m) {
    if (k == 0) {            // k = 0
        return 1;
    } else if (k % 2 == 0) { // k is even
        long tmp = fastModExp(a, k/2, m);
        return (tmp * tmp) % m;
    } else {                 // k is odd
        long tmp = fastModExp(a, k-1, m);
        return (a * tmp) % m;
    }
}
```

# Fast exponentiation: $a^k \bmod m$ for all $k$

$$a^{2j} \bmod m = (a^j \bmod m)^2 \bmod m$$
$$a^{2j+1} \bmod m = ((a \bmod m) * (a^{2j} \bmod m)) \bmod m$$

Example implementation:

```
// Assumes a > 0, k >= 0, m > 0.
public static long fastModExp(long a, long k, long m) {
    if (k == 0) {              // k = 0
        return 1;
    } else if (k % 2 == 0) { // k is even
        long tmp = fastModExp(a, k/2, m);
        return (tmp * tmp) % m;
    } else {                  // k is odd
        long tmp = fastModExp(a, k-1, m);
        return (a * tmp) % m;
    }
}
```

$$78365^{81453} \bmod 104729 = 45235$$

# Fast exponentiation: how fast is it?

**Note that 81453 is 1001111000101101 in binary.**

$$81453 = 2^{16} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^5 + 2^3 + 2^2 + 2^0$$

$$a^{81453} = a^{2^{16}} * a^{2^{13}} * a^{2^{12}} * a^{2^{11}} * a^{2^{10}} * a^{2^9} * a^{2^5} * a^{2^3} * a^{2^2} * a^{2^0}$$

$$\begin{aligned} a^{81453} \bmod m = ((((((((((a^{2^{16}} &\bmod m * \\ a^{2^{13}} &\bmod m) \bmod m * \\ a^{2^{12}} &\bmod m) \bmod m * \\ a^{2^{11}} &\bmod m) \bmod m * \\ a^{2^{10}} &\bmod m) \bmod m * \\ a^{2^9} &\bmod m) \bmod m * \\ a^{2^5} &\bmod m) \bmod m * \\ a^{2^3} &\bmod m) \bmod m * \\ a^{2^2} &\bmod m) \bmod m * \\ a^{2^0} &\bmod m) \bmod m) \end{aligned}$$

The fast exponentiation algorithm computes $a^k \bmod m$ using $\leq 2 \log k$ multiplications mod $m$.

# Using fast modular exponentiation: RSA encryption

**Alice chooses random 512-bit (or 1024-bit) primes $\mathbf{p}$, $\mathbf{q}$ and exponent $\mathbf{e}$.**

Alice computes $m = pq$ and broadcasts $(m, e)$, which is her public key.
She also computes the multiplicative inverse $d$ of $e \bmod (p-1)(q-1)$,
which serves as her private key.

**To encrypt a message $\mathbf{a}$ with Alice's public key, Bob computes $\mathbf{C = a^e \bmod m}$.**

This computation uses fast modular exponentiation.
Bob sends the ciphertext $C$ to Alice.

**To decrypt $\mathbf{C}$, Alice computes $\mathbf{C^d \bmod m}$.**

This computation also uses fast modular exponentiation.
It works because $C^d \bmod m = a$ for $0 < a < m$ unless $p|a$ or $q|a$.

# Summary

**$\mathbf{GCD(a, b)}$ is the greatest integer that divides both $\mathbf{a}$ and $\mathbf{b}$.**

It can be computed efficiently using the Euclidean algorithm.

**By Bézout's theorem, $\mathbf{GCD(a, b) = sa + tb}$ for some integers $\mathbf{s, t}$.**

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

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

Multiplicative inverses can be used to solve modular equations.

**Fast modular exponentiation efficiently computes $\mathbf{a^k \bmod m}$.**

Important practical applications include public-key cryptography (RSA).