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 $a^k \mod m$. 
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 $\text{GCD}(a, b)$, 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 $\text{GCD}(a, b)$, 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 $\text{GCD}(a, 0) = a$. 
Euclidean algorithm is based on two useful facts

\[\text{GCD}(a, 0)\]

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 $\text{GCD}(a, b) = \text{GCD}(b, a \mod b)$.
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 $\text{GCD}(a, b) = \text{GCD}(b, a \mod b)$.

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 $\text{GCD}(a, b) = \text{GCD}(b, a \mod b)$.

Proof:

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

Proof:

First note that by definition of mod, $a = qb + a \mod b$ for some integer $q = a \div b$.
Now, let $d$ be a common divisor of $a$ and $b$. 
Euclidean algorithm is based on two useful facts

\textbf{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.

\textbf{GCD and modulo}

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

**Proof:**

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

Now, let $d$ be a common divisor of $a$ and $b$. Then $d \mid a$ and $d \mid b$, 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 \( \text{GCD}(a, b) = \text{GCD}(b, a \mod b) \).

Proof:

First note that by definition of mod, \( a = qb + a \mod b \) for some integer \( q = a \div b \).

Now, let \( d \) be a common divisor of \( a \) and \( b \). Then \( d \mid a \) and \( d \mid b \), so \( a = kd \) and \( b = jd \) for some \( k, j \in \mathbb{Z} \). Therefore, \( a \mod b = a - qb = kd - qjd = d(k - qj) \).
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 $\text{GCD}(a, b) = \text{GCD}(b, a \mod b)$.

Proof:

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

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, $a \mod b = a - qb = kd - qjd = d(k - qj)$. So, $d|(a \mod b)$. 
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 \( \text{GCD}(a, b) = \text{GCD}(b, a \mod b) \).

Proof:

First note that by definition of mod, \( a = qb + a \mod b \) for some integer \( q = a \div b \).  
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, \( a \mod b = a - qb = kd - qjd = d(k - qj) \). So, \( d|(a \mod b) \). 
Next, let \( e \) be a common divisor of \( b \) and \( a \mod b \).
Euclidean algorithm is based on two useful facts

\textbf{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.

\textbf{GCD and modulo}

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

Proof:
First note that by definition of mod, \(a = qb + a \mod b\) for some integer \(q = a \div b\).
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, \(a \mod b = a - qb = kd - qjd = d(k - qj)\). So, \(d|(a \mod b)\).
Next, let \(e\) be a common divisor of \(b\) and \(a \mod b\). Then \(e|b\) and \(e|(a \mod b)\), so \(b = me\) and \(a \mod b = ne\) 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 \( \text{GCD}(a, b) = \text{GCD}(b, a \mod b) \).

Proof:

First note that by definition of mod, \( a = qb + a \mod b \) for some integer \( q = a \div b \).

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, \( a \mod b = a - qb = kd - qjd = d(k - qj) \). So, \( d|(a \mod b) \).

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

Proof:

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

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, $a \mod b = a - qb = kd - qjd = d(k - qj)$. So, $d | (a \mod b)$.

Next, let $e$ be a common divisor of $b$ and $a \mod b$. Then $e | b$ and $e | (a \mod b)$, so $b = me$ and $a \mod b = ne$ for some $m, n \in \mathbb{Z}$. Therefore, $a = qb + a \mod b = qme + ne$. So, $e | a$. 
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.

**GCD and modulo**

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

Proof:

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

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, $a \mod b = a - qb = kd - qjd = d(k - qj)$. So, $d | (a \mod b)$.

Next, let $e$ be a common divisor of $b$ and $a \mod b$. Then $e | b$ and $e | (a \mod b)$, so $b = me$ and $a \mod b = ne$ for some $m, n \in \mathbb{Z}$. Therefore, $a = qb + a \mod b = qme + ne$. So, $e | a$. This shows that $a, b$ and $b, a \mod b$ have the same set of common divisors, and must therefore have the same greatest common divisor. □
Euclidean algorithm

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

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

Example implementation:

```java
// 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 $\text{GCD}(a, b) = \text{GCD}(b, a \mod b)$ until you get $\text{GCD}(a, 0) = a$.

Example implementation:

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

$\text{GCD}(660, 126)$
Euclidean algorithm

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

Example implementation:

```java
// 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)
}
```

\( \text{GCD}(660, 126) = \text{GCD}(126, 660 \mod 126) = \text{GCD}(126, 30) \)
Euclidean algorithm

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

Example implementation:

```java
// 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)
}
```

\( \text{GCD}(660, 126) \)

\[
= \text{GCD}(126, 660 \mod 126) = \text{GCD}(126, 30)
= \text{GCD}(30, 126 \mod 30) = \text{GCD}(30, 6)
\]
Euclidean algorithm

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

Example implementation:

```java
// 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)
}
```

\[
\text{GCD}(660, 126) = \text{GCD}(126, 660 \mod 126) = \text{GCD}(126, 30) \\
= \text{GCD}(30, 126 \mod 30) = \text{GCD}(30, 6) \\
= \text{GCD}(6, 30 \mod 6) = \text{GCD}(6, 0)
\]
Euclidean algorithm

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

Example implementation:

```java
// Assumes $a \geq b \geq 0$.
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)
    }
}
```

$\text{GCD}(660, 126)$

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

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

$= \text{GCD}(6, 30 \mod 6) = \text{GCD}(6, 0)$

$= 6$
Euclidean algorithm

Apply GCD(a, b) = GCD(b, a mod b) until you get GCD(a, 0) = a.

Example implementation:

```java
// 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)

\[
\begin{align*}
660 &= 5 \times 126 + 30 \\
126 &= 4 \times 30 + 6 \\
30 &= 5 \times 6 + 0
\end{align*}
\]

= GCD(126, 660 mod 126) = GCD(126, 30)
= GCD(30, 126 mod 30) = GCD(30, 6)
= GCD(6, 30 mod 6) = GCD(6, 0)
= 6

In tableau form:
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.

GCD(35, 27) = 35s + 27t.
Extended Euclidean algorithm

1. Compute GCD and keep the tableau.

\[
\begin{array}{cccc}
    a &=& q \times b + r \\
    35 &=& 1 \times 27 + 8 \\
    27 &=& 3 \times 8 + 3 \\
    8 &=& 2 \times 3 + 2 \\
    3 &=& 1 \times 2 + 1 \\
\end{array}
\]

GCD \((a, b)\) \hspace{1cm} GCD \((b, a \mod b)\) \hspace{1cm} \(r = a \mod b\)

\[
\begin{align*}
\text{GCD}(35, 27) &= \text{GCD}(27, 35 \mod 27) = \text{GCD}(27, 8) \\
&= \text{GCD}(8, 27 \mod 8) = \text{GCD}(8, 3) \\
&= \text{GCD}(3, 8 \mod 3) = \text{GCD}(3, 2) \\
&= \text{GCD}(2, 3 \mod 2) = \text{GCD}(2, 1) \\
&= \text{GCD}(1, 2 \mod 1) = \text{GCD}(1, 0)
\end{align*}
\]
Extended Euclidean algorithm

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

\[
\begin{align*}
a &= q \cdot b + r \\
35 &= 1 \cdot 27 + 8 \\
27 &= 3 \cdot 8 + 3 \\
8 &= 2 \cdot 3 + 2 \\
3 &= 1 \cdot 2 + 1 \\
r &= a - q \cdot b \\
8 &= 35 - 1 \cdot 27 \\
3 &= 27 - 3 \cdot 8 \\
2 &= 8 - 2 \cdot 3 \\
1 &= 3 - 1 \cdot 2
\end{align*}
\]

GCD$(35, 27) = 35s + 27t$. 

GCD$(35, 27) = 35s + 27t$. 

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

\[
\begin{align*}
a &= q \cdot b + r \\
35 &= 1 \cdot 27 + 8 \\
27 &= 3 \cdot 8 + 3 \\
8 &= 2 \cdot 3 + 2 \\
3 &= 1 \cdot 2 + 1 \\
r &= a - q \cdot b \\
8 &= 35 - 1 \cdot 27 \\
3 &= 27 - 3 \cdot 8 \\
2 &= 8 - 2 \cdot 3 \\
1 &= 3 - 1 \cdot 2
\end{align*}
\]
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 $r$.

\[
\begin{align*}
    r &= a - q \cdot b \\
    8 &= 35 - 1 \cdot 27 \\
    3 &= 27 - 3 \cdot 8 \\
    2 &= 8 - 2 \cdot 3 \\
    1 &= 3 - 1 \cdot 2 \\
\end{align*}
\]

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

$$r = a - q \cdot b$$

$$8 = 35 - 1 \cdot 27$$
$$3 = 27 - 3 \cdot 8$$
$$2 = 8 - 2 \cdot 3$$
$$1 = 3 - 1 \cdot 2$$

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

$$r_0 = a = 35$$
$$r_1 = b = 27$$
$$r_2 = r_0 - q_2 \cdot r_1 = 8$$
$$r_3 = r_1 - q_3 \cdot r_2 = 3$$
$$r_4 = r_2 - q_4 \cdot r_3 = 2$$
$$r_5 = r_3 - q_5 \cdot r_4 = 1$$

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 \( r \).

\[
gcd(35, 27) = 35s + 27t.
\]

\[
\begin{align*}
1 & = 35 - 1 \times 27 \\
2 & = 8 - 2 \times 3 \\
3 & = 27 - 3 \times 8 \\
8 & = 35 - 1 \times 27 \\
1 & = 3 - 1 \times 2 \\
\end{align*}
\]

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

\[
r_5 = r_3 - q_5 \times r_4.
\]
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 \( r \).

\[
\begin{align*}
\text{GCD}(35, 27) &= 35s + 27t. \\
r &= a - q \times b \\
8 &= 35 - 1 \times 27 \\
3 &= 27 - 3 \times 8 \\
2 &= 8 - 2 \times 3 \\
1 &= 3 - 1 \times 2 \\
\end{align*}
\]

\[
\begin{align*}
1 &= 3 - 1 \times 2 \\
&= 3 - 1 \times (8 - 2 \times 3) \\
1 &= 3 - 1 \times 2 \\
&= 3 - 1 \times (27 - 3 \times 8) \\
&= -13 \\
r_5 &= r_3 - q_5 \times r_4. \\
\text{Plug in } r_4 &= r_2 - q_4 \times r_3.
\end{align*}
\]

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

\[
GCD(35, 27) = 35s + 27t.
\]

\[
\begin{align*}
1 & = 3 - 1 \times 2 \\
& = 3 - 1 \times (8 - 2 \times 3) \\
& = (-1) \times 8 + 3 \times 3
\end{align*}
\]

\[
r_5 = r_3 - q_5 \times r_4.
\] Plug in $r_4 = r_2 - q_4 \times r_3$.

Combine $r_2, r_3$ terms.

\[
\begin{align*}
& 8 = 35 - 1 \times 27 \\
& 3 = 27 - 3 \times 8 \\
& 2 = 8 - 2 \times 3 \\
& 1 = 3 - 1 \times 2
\end{align*}
\]
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 $r$.

\[
\begin{align*}
  r &= a - q \times b \\
  8 &= 35 - 1 \times 27 \\
  3 &= 27 - 3 \times 8 \\
  2 &= 8 - 2 \times 3 \\
  1 &= 3 - 1 \times 2
\end{align*}
\]

\[
\begin{align*}
  1 &= 3 - 1 \times 2 \\
     &= 3 - 1 \times (8 - 2 \times 3) \\
     &= (-1) \times 8 + 3 \times 3 \\
     &= (-1) \times 8 + 3 \times (27 - 3 \times 8)
\end{align*}
\]

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

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

$$r = a - q \times b$$

$r_0 = a = 35$
$r_1 = b = 27$
$r_2 = r_0 - q_2 \times r_1 = 8$
$r_3 = r_1 - q_3 \times r_2 = 3$
$r_4 = r_2 - q_4 \times r_3 = 2$
$r_5 = r_3 - q_5 \times r_4 = 1$

$$1 = 3 - 1 \times 2$$

$$= 3 - 1 \times (8 - 2 \times 3)$$

$$= (-1) \times 8 + 3 \times 3$$

$$= (-1) \times 8 + 3 \times (27 - 3 \times 8)$$

$$= 3 \times 27 + (-10) \times 8$$

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

$GCD(35, 27) = 35s + 27t.$
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 $r$.

$r = a - q \cdot b$

\[
\begin{align*}
8 & = 35 - 1 \cdot 27 \\
3 & = 27 - 3 \cdot 8 \\
2 & = 8 - 2 \cdot 3 \\
1 & = 3 - 1 \cdot 2 \\
\end{align*}
\]

$1 = 3 - 1 \cdot 2$

\[
\begin{align*}
&= 3 - 1 \cdot (8 - 2 \cdot 3) \\
&= (-1) \cdot 8 + 3 \cdot 3 \\
&= (-1) \cdot 8 + 3 \cdot (27 - 3 \cdot 8) \\
&= 3 \cdot 27 + (-10) \cdot 8 \\
&= 3 \cdot 27 + (-10) \cdot (35 - 1 \cdot 27) \\
\end{align*}
\]

$GCD(35, 27) = 35s + 27t.$

\[
\begin{align*}
r_5 & = r_3 - q_5 \cdot r_4. \\
&= r_3 - q_5 \cdot r_4. \\
&= r_3 - q_5 \cdot r_4. \\
\end{align*}
\]

Plug in $r_4 = r_2 - q_4 \cdot r_3$. 

Combine $r_2, r_3$ terms.

Plug in $r_3 = r_1 - q_3 \cdot r_2$. 

Combine $r_1, r_2$ terms.

Plug in $r_2 = r_0 - q_2 \cdot r_1$. 

\[
\begin{align*}
r_0 & = a = 35 \\
r_1 & = b = 27 \\
r_2 & = r_0 - q_2 \cdot r_1 = 8 \\
r_3 & = r_1 - q_3 \cdot r_2 = 3 \\
r_4 & = r_2 - q_4 \cdot r_3 = 2 \\
r_5 & = r_3 - q_5 \cdot r_4 = 1 \\
\end{align*}
\]
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 $r$.

$$ r = a - q \times b $$

$$
8 = 35 - 1 \times 27 \\
3 = 27 - 3 \times 8 \\
2 = 8 - 2 \times 3 \\
1 = 3 - 1 \times 2
$$

$$
1 = 3 - 1 \times 2 \\
= 3 - 1 \times (8 - 2 \times 3) \\
= (-1) \times 8 + 3 \times 3 \\
= (-1) \times 8 + 3 \times (27 - 3 \times 8) \\
= 3 \times 27 + (-10) \times 8 \\
= 3 \times 27 + (-10) \times (35 - 1 \times 27) \\
= (-10) \times 35 + 13 \times 27
$$

$$
r_5 = r_3 - q_5 \times r_4. \\
\text{Plug in } r_4 = r_2 - q_4 \times r_3. \\
\text{Combine } r_2, r_3 \text{ terms.} \\
\text{Plug in } r_3 = r_1 - q_3 \times r_2. \\
\text{Combine } r_1, r_2 \text{ terms.} \\
\text{Plug in } r_2 = r_0 - q_2 \times r_1. \\
\text{Combine } r_0, r_1 \text{ terms.}
$$

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

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

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

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

Multiplicative inverse \( \text{mod } m \)

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

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 \( m \)

\[ 1 = (sa + tm) \mod m = sa \mod m \]
Multiplicative inverse \( \mod m \)

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

By Bézout’s theorem, there exist integers \( s \) and \( t \) such that \( sa + tm = 1 \).

\( s \mod m \) is the multiplicative inverse of \( a \) modulo \( m \)

\[
1 = (sa + tm) \mod m = sa \mod m
\]

In other words, \( s \mod m \) is the multiplicative inverse of \( a \mod m \) iff \( sa \equiv 1 \pmod{m} \).
Multiplicative inverse \( \mod m \)

Suppose \( \gcd(a, m) = 1 \).

By Bézout’s theorem, there exist integers \( s \) and \( t \) such that \( sa + tm = 1 \).

\( s \mod m \) is the \textit{multiplicative inverse of } \( a \) \text{ modulo } \( m \)

\[
1 = (sa + tm) \mod m = sa \mod m
\]

In other words, \( s \mod m \) is the multiplicative inverse of \( a \mod m \) iff \( sa \equiv 1 \) (mod \( m \)).

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.

\[
\begin{align*}
\text{GCD}(26, 7) &= \text{GCD}(7, 5) = \text{GCD}(5, 2) \\
&= \text{GCD}(2, 1) = \text{GCD}(1, 0) \\
&= 1
\end{align*}
\]
Using multiplicative inverses to solve modular equations

Solve: \( 7x \equiv 1 \pmod{26} \)

① Compute GCD and keep the tableau.

\[
\text{GCD}(26, 7) = \text{GCD}(7, 5) = \text{GCD}(5, 2) = \text{GCD}(2, 1) = \text{GCD}(1, 0) = 1
\]

② Solve the equations for \( r \) in the tableau.

\[
\begin{align*}
a &= q \cdot b + r & r &= a - q \cdot b \\
26 &= 3 \cdot 7 + 5 & 5 &= 26 - 3 \cdot 7 \\
7 &= 1 \cdot 5 + 2 & 2 &= 7 - 1 \cdot 5 \\
5 &= 2 \cdot 2 + 1 & 1 &= 5 - 2 \cdot 2
\end{align*}
\]
Using multiplicative inverses to solve modular equations

Solve: \( 7x \equiv 1 \pmod{26} \)

① Compute GCD and keep the tableau.

\[
\text{GCD}(26, 7) = \text{GCD}(7, 5) = \text{GCD}(5, 2) = \text{GCD}(2, 1) = \text{GCD}(1, 0) = 1
\]

② Solve the equations for \( r \) in the tableau.

\[
\begin{align*}
a &= q \cdot b + r \\
26 &= 3 \cdot 7 + 5 \\
7 &= 1 \cdot 5 + 2 \\
5 &= 2 \cdot 2 + 1
\end{align*}
\]

\[
\begin{align*}
r &= a - q \cdot b \\
5 &= 26 - 3 \cdot 7 \\
2 &= 7 - 1 \cdot 5 \\
1 &= 5 - 2 \cdot 2
\end{align*}
\]

③ Back substitute the equations for \( r \).

\[
\begin{align*}
1 &= 5 - 2 \cdot (7 - 1 \cdot 5) \\
&= (\text{not simplified}) \\
&= (-2) \cdot 7 + 3 \cdot 5 \\
&= (-2) \cdot 7 + 3 \cdot (26 - 3 \cdot 7) \\
&= 3 \cdot 26 + (-11) \cdot 7
\end{align*}
\]
Using multiplicative inverses to solve modular equations

Solve: \( 7x \equiv 1 \pmod{26} \)

1. Compute GCD and keep the tableau.

\[
\begin{align*}
\text{GCD}(26, 7) &= \text{GCD}(7, 5) = \text{GCD}(5, 2) \\
&= \text{GCD}(2, 1) = \text{GCD}(1, 0) \\
&= 1
\end{align*}
\]

2. Solve the equations for \( r \) in the tableau.

\[
\begin{align*}
a &= q \cdot b + r \\
26 &= 3 \cdot 7 + 5 \\
7 &= 1 \cdot 5 + 2 \\
5 &= 2 \cdot 2 + 1
\end{align*}
\]

3. Back substitute the equations for \( r \).

\[
\begin{align*}
1 &= 5 - 2 \cdot (7 - 1 \cdot 5) \\
&= (-2) \cdot 7 + 3 \cdot 5 \\
&= (-2) \cdot 7 + 3 \cdot (26 - 3 \cdot 7) \\
&= 3 \cdot 26 + (-11) \cdot 7
\end{align*}
\]

4. Solve for \( x \).

- Multiplicative inverse of 7 mod 26
  - \((-11) \mod 26 = 15\)
- 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, \( 7 \times 15 \equiv 1 \pmod{26} \).
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, \( 7 \cdot 15 \equiv 1 \pmod{26} \).

By the multiplication property of mod, we have

That is, \( 7 \cdot 15 \cdot 3 \equiv 1 \cdot 3 \pmod{26} \).
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, $7 \times 15 \equiv 1 \pmod{26}$.

By the multiplication property of mod, we have

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

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

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

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

\[
a +_7 b = (a + b) \mod 7
\]

\[
a *_7 b = (a * b) \mod 7
\]

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>*</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Modular exponentiation

A fast algorithm for computing $a^k \mod m$. 
The modular exponentiation problem: $a^k \mod m$

How would you compute $78365^{81453} \mod 104729$?
The modular exponentiation problem: $a^k \mod m$

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

Naive approach

First compute $78365^{81453}$.
Then take the result modulo 104729.
The modular exponentiation problem: $a^k \mod m$

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

**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: $a^k \mod m$

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

**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: $a^k \mod m$ for $k = 2^i$

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

Note that $a \mod m \equiv a \pmod{m}$ and $b \mod m \equiv b \pmod{m}$. So, we have $ab \mod m = ((a \mod m)(b \mod m)) \mod m$. 
Repeated squaring: $a^k \mod m$ for $k = 2^i$

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

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

For example:

\[
\begin{align*}
a^2 \mod m &= (a \mod m)^2 \mod m \\
a^4 \mod m &= (a^2 \mod m)^2 \mod m \\
a^8 \mod m &= (a^4 \mod m)^2 \mod m \\
a^{16} \mod m &= (a^8 \mod m)^2 \mod m \\
a^{32} \mod m &= (a^{16} \mod m)^2 \mod m
\end{align*}
\]
Repeated squaring: $a^k \mod m$ for $k = 2^i$

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

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

For example:

\[
\begin{align*}
a^2 \mod m &= (a \mod m)^2 \mod m \\
a^4 \mod m &= (a^2 \mod m)^2 \mod m \\
a^8 \mod m &= (a^4 \mod m)^2 \mod m \\
a^{16} \mod m &= (a^8 \mod m)^2 \mod m \\
a^{32} \mod m &= (a^{16} \mod m)^2 \mod m
\end{align*}
\]
Fast exponentiation: $a^k \mod m$ for all $k$

\[
\begin{align*}
a^{2j} \mod m &= (a^j \mod m)^2 \mod m \\
a^{2j+1} \mod m &= ((a \mod m) \times (a^{2j} \mod m)) \mod m
\end{align*}
\]
Fast exponentiation: $a^k \mod m$ for all $k$

\[
a^{2j} \mod m = (a^j \mod m)^2 \mod m
\]
\[
a^{2j+1} \mod m = ((a \mod m) \ast (a^{2j} \mod m)) \mod m
\]

Example implementation:

```java
// 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: $a^k \mod m$ for all $k$

\[
a^{2j} \mod m = (a^j \mod m)^2 \mod m \\
(a^{2j+1} \mod m = ((a \mod m) \ast (a^{2j} \mod m)) \mod m
\]

Example implementation:

```java
// 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;
    }
}
```

$78365^{81453} \mod 104729 = 45235$
Fast exponentiation: how fast is it?

Note that 81453 is 10011111000101101 in binary.

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

\[ a^{81453} = a^{2^{16}} \ast a^{2^{13}} \ast a^{2^{12}} \ast a^{2^{11}} \ast a^{2^{10}} \ast a^9 \ast a^5 \ast a^3 \ast a^2 \ast a^0 \]

\[ a^{81453} \mod m = ((((((((((a^{2^{16}} \mod m) \mod m \ast a^{2^{13}} \mod m) \mod m \ast a^{2^{12}} \mod m) \mod m \ast a^{2^{11}} \mod m) \mod m \ast a^{2^{10}} \mod m) \mod m \ast a^9 \mod m) \mod m \ast a^5 \mod m) \mod m \ast a^3 \mod m) \mod m \ast a^2 \mod m) \mod m \ast a^0 \mod m) \mod m) \]

The fast exponentiation algorithm computes \( a^k \mod m \) using \( \leq 2 \log k \) multiplications \( \mod m \).
Using fast modular exponentiation: RSA encryption

Alice chooses random 512-bit (or 1024-bit) primes $p$, $q$ and exponent $e$. Alice computes $m = pq$ and broadcasts $(m, e)$, which is her public key. She also computes the multiplicative inverse $d$ of $e$ mod $(p - 1)(q - 1)$, which serves as her private key.

To encrypt a message $a$ with Alice’s public key, Bob computes $C = a^e \mod m$. This computation uses fast modular exponentiation. Bob sends the ciphertext $C$ to Alice.

To decrypt $C$, Alice computes $C^d \mod m$. This computation also uses fast modular exponentiation. It works because $C^d \mod m = a$ for $0 < a < m$ unless $p | a$ or $q | a$. 
Summary

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 GCD(a, b) = 1, s mod b is the multiplicative inverse of a modulo b.
   Multiplicative inverses can be used to solve modular equations.

Fast modular exponentiation efficiently computes a^k mod m.
   Important practical applications include public-key cryptography (RSA).