Lecture 12: Modular Exponentiation, Set Theory

“I asked you a question, buddy... What's the square root of 5,248?”
Last class: Euclid’s Algorithm for GCD

Repeatedly use \( \gcd(a, b) = \gcd(b, a \mod b) \) to reduce numbers until you get \( \gcd(a, 0) = a \).

Equations with recursive calls:

\[
\begin{align*}
\gcd(660, 126) &= \gcd(126, 660 \mod 126) = \gcd(126, 30) \\
&= \gcd(30, 126 \mod 30) = \gcd(30, 6) \\
&= \gcd(6, 30 \mod 6) = \gcd(6, 0) \\
&= 6
\end{align*}
\]

Tableau form (which is much easier to work with and will be more useful):

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

Each line computes both quotient and remainder of the shifted numbers.
Last class: Extended Euclidean algorithm

• Can use Euclid’s Algorithm to find \( s, t \) such that

\[
gcd(a, b) = sa + tb
\]

Example: \( a = 35, b = 27 \)

Compute \( \gcd(35, 27) \):

\[
\begin{align*}
35 &= 1 \cdot 27 + 8 & & 8 &= 35 - 1 \cdot 27 \\
27 &= 3 \cdot 8 + 3 & & 3 &= 27 - 3 \cdot 8 \\
8 &= 2 \cdot 3 + 2 & & 2 &= 8 - 2 \cdot 3 \\
3 &= 1 \cdot 2 + 1 & & 1 &= 3 - 1 \cdot 2 \\
2 &= 2 \cdot 1 + 0
\end{align*}
\]
Last class: Extended Euclidean algorithm

- Can use Euclid’s Algorithm to find $s, t$ such that $\gcd(a, b) = sa + tb$

Example: $a = 35, b = 27$

Use equations to substitute back

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

Optional Check:

$(-10) \times 35 = -350$
$13 \times 27 = 351$
Let $0 \leq a, b < m$. Then, $b$ is the multiplicative inverse of $a$ (modulo $m$) iff $ab \equiv 1 \pmod{m}$.
Last class: Multiplicative inverse (mod \( m \))

Let \( 0 \leq a, b < m \). Then, \( b \) is the multiplicative inverse of \( a \) (modulo \( m \)) iff \( ab \equiv 1 \pmod{m} \).

This can’t exist if \( a \) and \( m \) have a common factor >1.

Idea: \( b \) is like \( a^{-1} \pmod{m} \) so multiplying by \( b \) is equivalent to dividing by \( a \).
Finding multiplicative inverse mod $m$

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

Using Extended Euclidean Algorithm, find integers $s$ and $t$ such that $sa + tm = 1$.

Therefore $sa \equiv 1 \pmod{m}$.

The multiplicative inverse $b$ of $a$ modulo $m$ must also satisfy $0 \leq b < m$ so we set $b = s \mod m$.

It works since $ba \equiv sa \equiv 1 \pmod{m}$.
Example

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

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

First compute and check that $\gcd(26, 7) = 1$

\[
\begin{align*}
26 &= 3 \times 7 + 5 \\
7 &= 1 \times 5 + 2 \\
5 &= 2 \times 2 + 1 \\
2 &= 2 \times 1 + 0
\end{align*}
\]
Example

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

Then rewrite equations in form for substitution

\begin{align*}
26 & = 3 \times 7 + 5 & 5 & = 26 - 3 \times 7 \\
7 & = 1 \times 5 + 2 & 2 & = 7 - 1 \times 5 \\
5 & = 2 \times 2 + 1 & 1 & = 5 - 2 \times 2 \\
2 & = 2 \times 1 + 0
\end{align*}
Example

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

Apply substitutions from bottom to top.

$26 = 3 \times 7 + 5$ \hspace{1cm} $5 = 26 - 3 \times 7$
$7 = 1 \times 5 + 2$ \hspace{1cm} $2 = 7 - 1 \times 5$
$5 = 2 \times 2 + 1$ \hspace{1cm} $1 = 5 - 2 \times 2$
$2 = 2 \times 1 + 0$

$1 = 5 - 2 \times 2$

$= 5 - 2 \times (7 - 1 \times 5)$

$= (-2) \times 7 + 3 \times 5$

$= (-2) \times 7 + 3 \times (26 - 3 \times 7)$

$= (-11) \times 7 + 3 \times 26$
Example

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

Read off coefficient and reduce modulo 26.

\[
26 = 3 \times 7 + 5 \quad 5 = 26 - 3 \times 7 \\
7 = 1 \times 5 + 2 \quad 2 = 7 - 1 \times 5 \\
5 = 2 \times 2 + 1 \quad 1 = 5 - 2 \times 2 \\
2 = 2 \times 1 + 0
\]

\[
1 = 5 - 2 \times 2 \\
= 5 - 2 \times (7 - 1 \times 5) \\
= (-2) \times 7 + 3 \times 5 \\
= (-2) \times 7 + 3 \times (26 - 3 \times 7) \\
= (-11) \times 7 + 3 \times 26
\]

Now \((-11) \mod 26 = 15\). So, \(x = 15 + 26k\) for integer \(k\).
Example of a more general equation

Now solve: \( 7y \equiv 3 \pmod{26} \)

We already computed that 15 is the multiplicative inverse of 7 modulo 26. That is, \( 7 \cdot 15 \equiv 1 \pmod{26} \).

If \( y \) is a solution, then multiplying by 15 we have

\[
15 \cdot 7 \cdot y \equiv 15 \cdot 3 \pmod{26}
\]

Substituting \( 15 \cdot 7 \equiv 1 \pmod{26} \) on the left gives

\[
y = 1 \cdot y \equiv 15 \cdot 3 \equiv 19 \pmod{26}
\]

This shows that every solution \( y \) is congruent to 19.
Example of a more general equation

Now solve: \(7y \equiv 3 \pmod{26}\)

Multiplying both sides of \(y \equiv 19 \pmod{26}\) by 7 gives
\[
7y \equiv 7 \cdot 19 \equiv 3 \pmod{26}
\]

So, any \(y \equiv 19 \pmod{26}\) is a solution.

Thus, the set of numbers of the form \(y = 19 + 26k\), for any integer \(k\), are exactly solutions of this equation.
Math mod a prime is especially nice

gcd(a, m) = 1 if m is prime and 0 < a < m so can always solve these equations mod a prime.

\[
\begin{array}{cccccccc}
+ & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 & 0 \\
2 & 2 & 3 & 4 & 5 & 6 & 0 & 1 \\
3 & 3 & 4 & 5 & 6 & 0 & 1 & 2 \\
4 & 4 & 5 & 6 & 0 & 1 & 2 & 3 \\
5 & 5 & 6 & 0 & 1 & 2 & 3 & 4 \\
6 & 6 & 0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}
\] mod 7

\[
\begin{array}{cccccccc}
x & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 0 & 2 & 4 & 6 & 1 & 3 & 5 \\
3 & 0 & 3 & 6 & 2 & 5 & 1 & 4 \\
4 & 0 & 4 & 1 & 5 & 2 & 6 & 3 \\
5 & 0 & 5 & 3 & 1 & 6 & 4 & 2 \\
6 & 0 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]
Scenario:

Map a small number of data values from a large domain \( \{0, 1, \ldots, M - 1\} \) ...
...into a small set of locations \( \{0, 1, \ldots, n - 1\} \) so one can quickly check if some value is present

- \( \text{hash}(x) = (ax + b) \mod p \) for \( p \) a prime close to \( n \)
  - Relies on \( \gcd(a, p) = 1 \) to avoid many collisions

- Depends on all of the bits of the data
  - helps avoid collisions due to similar values
  - need to manage them if they occur
Hashing

- hash(x) = (ax + b) mod p for p a prime close to n

- Applications
  - map integer to location in array (hash tables)
  - map user ID or IP address to machine
    requests from the same user / IP address go to the same machine
    requests from different users / IP addresses spread randomly
Attack on RSA security with GCD

- RSA public key includes $m$ that is the product of two large randomly chosen primes $p, q$
  - Everyone can see all the public keys (millions)
  - Security depends on keeping $p$ and $q$ secret
  - OK since factoring $m$ seems very hard

- In 2012 a new attack using GCD broke a huge number of RSA public keys!
  - Weak keys: Algorithms/devices cut corners:
    Skimped on random bits or size of primes
Attack on RSA security with GCD

Weak keys: few random bits

– Few enough that some public keys $m_1$ and $m_2$ happen to share just one of their two factors:

\[
m_1 = pq \quad \text{and} \quad m_2 = pr
\]

– Then can break both since $p = \gcd(m_1, m_2)$

2012: 11 million RSA keys, 23,500 broken
2016: 1024-bit RSA keys available from Internet
   – 26 million keys, 63,500 broken
2019: 750 million RSA keys, 250,000 broken
   – IoT (Internet of Things) devices often the culprit
RSA Relies on Modular Exponentiation

\[
\begin{array}{cccccc}
\times & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & 1 & 2 & 3 & 4 & 5 & 6 \\
2 & 2 & 4 & 6 & 1 & 3 & 5 \\
3 & 3 & 6 & 2 & 5 & 1 & 4 \\
4 & 4 & 1 & 5 & 2 & 6 & 3 \\
5 & 5 & 3 & 1 & 6 & 4 & 2 \\
6 & 6 & 5 & 4 & 3 & 2 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
a & a^1 & a^2 & a^3 & a^4 & a^5 & a^6 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 \\
2 & 2 & 4 & 1 & 2 & 4 & 1 \\
3 & 3 & 2 & 6 & 4 & 5 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 1 \\
5 & 5 & 4 & 6 & 2 & 3 & 1 \\
6 & 6 & 1 & 6 & 1 & 6 & 1 \\
\end{array}
\]

mod 7
Modular Exponentiation: (Essential for RSA)

- Compute $78365^{81453}$
  - How many digits? = $5 \times 81453$ digits

- Compute $78365^{81453} \mod 104729$

- Output is small
  - need to keep intermediate results small
Small Multiplications

By the multiplicative property modulo $m$, if you want to compute $ab \mod m$ then you can do the following:

1. Reduce $a$ and $b$ modulo $m$ to get $a \mod m$ and $b \mod m$
2. Multiply to produce $c = (a \mod m)(b \mod m)$
3. Output $c \mod m$

Claim: $c \mod m = ab \mod m$

Proof: Just need to show that $c \equiv ab \pmod{m}$.

That follows from $(a \mod m) \equiv a \pmod{m}$

$(b \mod m) \equiv b \pmod{m}$

and the multiplicative property since $c$ is the product of the left sides and $ab$ is the product of the right sides. □
Repeated Squaring – small and fast

Then we have $ab \mod m = ((a \mod m)(b \mod m)) \mod m$

So $a^2 \mod m = (a \mod m)^2 \mod m$
and $a^4 \mod m = (a^2 \mod m)^2 \mod m$
and $a^8 \mod m = (a^4 \mod m)^2 \mod m$
and $a^{16} \mod m = (a^8 \mod m)^2 \mod m$
and $a^{32} \mod m = (a^{16} \mod m)^2 \mod m$

Can compute $a^k \mod m$ for $k = 2^i$ in only $i$ steps

What if $k$ is not a power of 2?
Fast Exponentiation Algorithm

81453 in binary is 10011101000101101

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

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

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

The fast exponentiation algorithm computes \( a^k \mod m \) using \( \leq 2\log k \) multiplications \( \mod m \)
Fast Exponentiation: $a^k \mod m$ for all $k$

Another way....

$$a^{2j} \mod m = (a^j \mod m)^2 \mod m$$

$$a^{2j+1} \mod m = ((a \mod m) \cdot (a^{2j} \mod m)) \mod m$$
Recursive Fast Exponentiation

```java
public static int FastModExp(int a, int k, int modulus) {
    if (k == 0) {
        return 1;
    } else if ((k % 2) == 0) {
        long temp = FastModExp(a, k/2, modulus);
        return (temp * temp) % modulus;
    } else {
        long temp = FastModExp(a, k-1, modulus);
        return (a * temp) % modulus;
    }
}
```

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

\[
a^{2j+1} \mod m = ((a \mod m) \cdot (a^{2j} \mod m)) \mod m
\]
Using Fast Modular Exponentiation

• Your e-commerce web transactions use SSL (Secure Socket Layer) based on RSA encryption

• RSA …as of 2023
  – Vendor chooses random 1024-bit or 2048-bit primes \( p, q \) and 1024/2048-bit exponent \( e \). Computes \( m = p \cdot q \)
  – Vendor broadcasts \( (m, e) \)
  – To send \( a \) to vendor, you compute \( C = a^e \mod m \) using fast modular exponentiation and send \( C \) to the vendor.
  – Using secret \( p, q \) the vendor computes \( d \) that is the multiplicative inverse of \( e \mod (p - 1)(q - 1) \).
  – Vendor computes \( C^d \mod m \) using fast modular exponentiation.
  – Fact: \( a = C^d \mod m \) for \( 0 < a < m \) unless \( p|a \) or \( q|a \)
Sets
Sets

Sets are collections of objects called elements.

Write \( a \in B \) to say that \( a \) is an element of set \( B \), and \( a \notin B \) to say that it is not.

Some simple examples
A = \{1\}
B = \{1, 3, 2\}
C = \{☐, 1\}
D = \{\{17\}, 17\}
E = \{1, 2, 7, \text{cat}, \text{dog}, \emptyset, \alpha\}
**Some Common Sets**

- **ℕ** is the set of **Natural Numbers**; \( \mathbb{N} = \{0, 1, 2, \ldots\} \)
- **ℤ** is the set of **Integers**; \( \mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\} \)
- **ℚ** is the set of **Rational Numbers**; e.g. \( \frac{1}{2}, -17, 32/48 \)
- **ℝ** is the set of **Real Numbers**; e.g. 1, -17, 32/48, \( \pi, \sqrt{2} \)
- \([n]\) is the set \( \{1, 2, \ldots, n\} \) when \( n \) is a natural number
- \( \emptyset = \{\} \) is the **empty set**; the *only* set with no elements
Sets can be elements of other sets

For example
A = \{\{1\}, \{2\}, \{1,2\}, \emptyset\}
B = \{1,2\}

Then B ∈ A.
Definitions

• A and B are *equal* if they have the same elements

\[ A = B := \forall x (x \in A \iff x \in B) \]

• A is a *subset* of B if every element of A is also in B

\[ A \subseteq B := \forall x (x \in A \rightarrow x \in B) \]

• Notes:

\[ (A = B) \equiv (A \subseteq B) \land (B \subseteq A) \]

\[ A \supseteq B \text{ means } B \subseteq A \]

\[ A \subset B \text{ means } A \subseteq B \text{ but } A \neq B \]
Definition: Equality

A and B are *equal* if they have the same elements

\[ A = B \iff \forall x (x \in A \iff x \in B) \]

Which sets are equal to each other?

\[
\begin{align*}
A &= \{1, 2, 3\} \\
B &= \{3, 4, 5\} \\
C &= \{3, 4\} \\
D &= \{4, 3, 3\} \\
E &= \{3, 4, 3\} \\
F &= \{4, \{3\}\}
\end{align*}
\]
Definition: Subset

A is a *subset* of B if every element of A is also in B

\[
A \subseteq B := \forall x \ (x \in A \rightarrow x \in B)
\]

A = \{1, 2, 3\}
B = \{3, 4, 5\}
C = \{3, 4\}

Questions:

- \(\emptyset \subseteq A\)? ✓
- A \(\subseteq B\)? x
- C \(\subseteq B\)? ✓
Definition: Subset

A is a *subset* of B if every element of A is also in B

\[ A \subseteq B := \forall x (x \in A \rightarrow x \in B) \]

Note the domain restriction.

We will use a shorthand restriction to a set

\[ \forall x \in A, \ P(x) := \forall x \ (x \in A \rightarrow P(x)) \]

Restricting all quantified variables improves *clarity*