CS483 Design and Analysis of Algorithms
Lectures 2-3 Algorithms with Numbers

Instructor: Fei Li
lifei@cs.gmu.edu with subject: CS483
Office hours: STII, Room 443, Friday 4:00pm - 6:00pm or by appointments

Course web-site:
http://www.cs.gmu.edu/~lifei/teaching/cs483_fall08/
Figures unclaimed are from books “Algorithms” and “Introduction to Algorithms”.
Chapter 1 of DPV — Algorithms with Numbers

▶ Foundations
  1. Basic Arithmetic
  2. Modular Arithmetic
  3. Primality Testing

▶ Applications
  1. Cryptography
  2. Universal Hashing
Theorem

The sum of any three single-digit numbers is at most two digits long, no matter what the base is.

\[ 9 + 9 + 9 = 27, \quad \text{in decimal} \]
\[ 1 + 1 + 1 = 11, \quad \text{in binary} \]

Proof.

\[ \square \]
Basic Arithmetic — Addition

Remark
Each individual sum is a two-digit number, the carry is always a single digit, and so at any given step, three single-digit numbers are added.

53 + 35 in binary.

\[
\begin{array}{c}
\text{Carry:} & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 & \text{(53)} \\
1 & 0 & 0 & 0 & 1 & 1 & \text{(35)} \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0 & \text{(88)} \\
\end{array}
\]

Addition runs in linear $O(n)$, when two $n$-bits numbers are added.
Basic Arithmetic — Multiplication

$x = 1101$ and $y = 1011$. The multiplication would proceed thus.

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\times & 1 & 0 & 1 \\
\hline
1 & 1 & 0 & 1 & \text{(1101 times 1)} \\
1 & 1 & 0 & 1 & \text{(1101 times 1, shifted once)} \\
0 & 0 & 0 & 0 & \text{(1101 times 0, shifted twice)} \\
\hline
1 & 1 & 0 & 1 & \text{(1101 times 1, shifted thrice)} \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \text{(binary 143)}
\end{array}
\]

1. Is it correct?
2. Running time?

\[
\underbrace{O(n) + O(n) + \cdots + O(n)}_{n-1 \text{ times}},
\]

3. Can we do better?

- Divide-and-Conquer:
  \[
  \approx O(n^{1.59}) \text{ (in Chapter 2)}
  \]
Basic Arithmetic — Division

- **Input:** Two \( n \)-bit integers \( x \) and \( y \), where \( y \geq 1 \)
- **Output:** The quotient and remainder of \( x \) divided by \( y \)

```plaintext
function divide(x, y)

    if (x = 0)
        return (q, r) = (0, 0);

    (q, r) = divide(\lfloor x / 2 \rfloor, y);
    q = 2 \times q; r = 2 \times r;

    if (x is odd)
        r = r + 1;

    if (r \geq y)
        r = r - y; q = q + 1;

    return (q, r);
```

```
Chapter 1 of DPV — Algorithms with Numbers

- **Foundations**
  1. Basic Arithmetic
  2. Modular Arithmetic
  3. Primality Testing

- **Applications**
  1. Cryptography
  2. Universal Hashing
Modular Arithmetic

Definition

Modular arithmetic is a system limiting numbers to a predefined range $[0, 1, \ldots, N - 1]$

$x$ and $y$ are congruent modulo $N \iff N$ divides $(x - y)$

$x$ modulo $N$ is $r \iff x = q \cdot N + r \iff x \equiv r \pmod{N}$, with $0 \leq r < N$

Figure: http://www.mathworks.com
Modular Arithmetic

Modular arithmetic deals with all integers and divide them into $N$ equivalence classes, each of the form \( \{ i + k \cdot N, k \in \mathbb{Z} \} \) for some $i$ between 0 and $N - 1$
For each class, $i$ is the representative

Remark

Substitution rule. If $x \equiv x' \pmod{N}$ and $y \equiv y' \pmod{N}$, then,

\[
\begin{align*}
  x + y & \equiv x' + y' \pmod{N} \\
  x \cdot y & \equiv x' \cdot y' \pmod{N}
\end{align*}
\]

\[
\begin{align*}
  x + (y + z) & \equiv (x + y) + z \pmod{N}, \quad \text{Associativity} \\
  x \cdot y & \equiv y \cdot x \pmod{N}, \quad \text{Commutativity} \\
  x \cdot (y + z) & \equiv x \cdot y + y \cdot z \pmod{N}, \quad \text{Distributivity}
\end{align*}
\]

\(2^{345} \equiv \pmod{31}\)
Modular Addition and Multiplication

1. Modular addition
   ▶ A regular addition \((0 \leq x + y \leq 2 \cdot (N - 1))\) and possibly a subtraction
   ▶ Running time \(O(n)\), where \(n = \lceil \log N \rceil\)

2. Modular multiplication
   ▶ A regular multiplication \((0 \leq x \cdot y \leq (N - 1)^2)\) and divide it by \(N\)
   ▶ Running time \(O(n^3)\)

3. Modular exponentiation
   ▶ Algorithms for \(x^y \pmod{N}\)?
   ▶ Running time?

4. Modular division
   ▶ Algorithms for \(a \cdot b^{-1} \equiv 1 \pmod{N}\)?
   ▶ Running time?
Modular Addition and Multiplication

1. Modular addition
   - A regular addition ($0 \leq x + y \leq 2 \cdot (N - 1)$) and possibly a subtraction
   - Running time $O(n)$, where $n = \lceil \log N \rceil$
Modular Addition and Multiplication

1. Modular addition
   - A regular addition \((0 \leq x + y \leq 2 \cdot (N - 1))\) and possibly a subtraction
   - Running time \(O(n)\), where \(n = \lceil \log N \rceil\)

2. Modular multiplication
   - A regular multiplication \((0 \leq x \cdot y \leq (N - 1)^2)\) and divide it by \(N\)
   - Running time \(O(n^3)\)
Modular Addition and Multiplication

1. Modular addition
   ▶ A regular addition \(0 \leq x + y \leq 2 \cdot (N - 1)\) and possibly a subtraction
   ▶ Running time \(O(n)\), where \(n = \lceil \log N \rceil\)

2. Modular multiplication
   ▶ A regular multiplication \(0 \leq x \cdot y \leq (N - 1)^2\) and divide it by \(N\)
   ▶ Running time \(O(n^3)\)

3. Modular exponentiation
   ▶ Algorithms for \(x^y \mod N\)?
   ▶ Running time?

4. Modular division
   ▶ Algorithms for \(a \cdot \? \equiv 1 \mod N\)?
   ▶ Running time?
Module Exponentiation ($x^y \mod N = ?$)

1. Worst approach
   - Calculate $x^y$, then calculate $x^y \mod N$
   - $(2^{19})^{2^{19}} = 2^{19 \cdot 524288}$
Module Exponentiation \((x^y \mod N = ?)\)

1. **Worst approach**
   - Calculate \(x^y\), then calculate \(x^y \mod N\)
   - \((2^{19})^{2^{19}} = 2^{19 \times 524288}\)

2. **Bad approach**
   - Calculate \(x^y \mod N\) by repeatedly multiplying by \(x\) modulo \(N\)
     
     \[x \mod N \rightarrow x^2 \mod N \rightarrow x^3 \mod N \rightarrow \ldots, \rightarrow x^y \mod N\]
   - \(y - 1 \approx 2^{500}\) multiplications, if \(y\) has 500 bits
Module Exponentiation \((x^y \text{ mod } N = ?)\)

1. Worst approach
   - Calculate \(x^y\), then calculate \(x^y \text{ mod } N\)
   - \((2^{19})^{2^{19}} = 2^{19 \cdot 524288}\)

2. Bad approach
   - Calculate \(x^y \text{ mod } N\) by repeatedly multiplying by \(x\) modulo \(N\)
     \[x \text{ mod } N \to x^2 \text{ mod } N \to x^3 \text{ mod } N \to \ldots, \to x^y \text{ mod } N\]
   - \(y - 1 \approx 2^{500}\) multiplications, if \(y\) has 500 bits

3. Best approach (geometrically calculate the product)

   \[x \text{ mod } N \to x^2 \text{ mod } N \to x^4 \text{ mod } N \to \ldots, \to x^{2^\lfloor \log y \rfloor} \text{ mod } N.\]

   \[x^y = \begin{cases} (x^\lfloor y/2 \rfloor)^2 & \text{if } y \text{ is even} \\ x \cdot (x^\lfloor y/2 \rfloor)^2 & \text{if } y \text{ is odd.} \end{cases}\]

   \[x^{25} = x^{11001_2} = x^{10000_2} \cdot x^{1000_2} \cdot x^{1_2} = x^{16} \cdot x^8 \cdot x^1.\]
Euclid Algorithm \((gcd(a, b))\)

Given two integers \(a\) and \(b\), what is the \textit{largest integer} that divides both — greatest common divisor?

**Theorem**

Let \(a \geq b\). \(gcd(a, b) = gcd(b, a \mod b) = gcd(a - b, b)\).

**Proof.**

\[gcd(25, 11) =?\]
Assume \(d\) is the greatest common divisor of \(a\) and \(b\), how can we check this?

**Lemma**

*If \(d\) divides both \(a\) and \(b\), and \(d = a \cdot x + b \cdot y\) for some integers of \(x\) and \(y\), then necessarily \(d = \gcd(a, b)\).*

**Proof.**

1. \(\gcd(65, 40) = ?\)
2. \(65 \cdot x + 40 \cdot y = \gcd(65, 40)\)
3. \(\gcd(1239, 735) = ?\)
4. \(1239 \cdot x + 735 \cdot y = \gcd(65, 40)\)
Modular Division — $a \cdot x \equiv 1 \pmod{N}$

**Definition**
$x$ is the *multiplicative inverse* of $a$ modulo $N$ if $a \cdot x \equiv 1 \pmod{N}$

**Lemma**
$x$, if it exists, is unique.

**Proof.**
?

**Lemma**
*If $\gcd(a, N) = 1$, $x$ must exist.*

**Proof.**
?
Chapter 1 of DPV — Algorithms with Numbers

▶ Foundations

1. Basic Arithmetic
2. Modular Arithmetic
3. Primality Testing

▶ Applications

1. Cryptography
2. Universal Hashing
Primality Testing

Tell whether a number is a prime without factoring it.

Theorem

Fermat’s little theorem. If \( p \) is prime, then for every \( 1 \leq a < p \),

\[
a^{p-1} \equiv 1 \pmod{p}
\]

Lemma

\[
(S = \{1, 2, \ldots, p - 1\} \cdot a) \mod p = S
\]
\[
(p - 1)! \equiv a^{p-1} \cdot (p - 1)! \pmod{p}
\]
Fermat’s Last Theorem

Figure: http://jeff560.tripod.com

Fermat’s equation:

\[ x^n + y^n = z^n \]

This equation has no solutions in integers for \( n \geq 3 \).

Figure: http://www.fafamonge.com/images
Generate Random Primes

Theorem

Langange’s prime number theorem. Let \( \pi(x) \) be the number of primes \( \leq x \). Then \( \pi(x) \approx \frac{x}{\ln x} \), or more precisely,

\[
\lim_{x \to +\infty} \frac{\pi(x)}{(x/\ln x)} = 1
\]

function random-prime(n)

while()

Pick a random n-bit number N;

Run a primality test on N;

if (test is passed)
    return N;
Chapter 1 of DPV — Algorithms with Numbers

▶ Foundations

1. Basic Arithmetic
2. Modular Arithmetic
3. Primality Testing

▶ Applications

1. Cryptography
2. Universal Hashing
Rivest-Shamir-Adelman (RSA) — Public Key System

1. Anybody can send a message to anybody else using publicly available information
2. Each person has a public key known to the whole world and a secret key known only to him- or herself
3. When Alice wants to send message $x$ to Bob, she encodes it using Bob's public key. Bob decrypts it using his secret key

![Diagram of message flow with Alice, Encoder, Decoder, Bob, Eve]
1. Anybody can send a message to anybody else using publicly available information.
2. Each person has a public key known to the whole world and a secret key known only to him- or herself.
3. When Alice wants to send message $x$ to Bob, she encodes it using Bob's public key. Bob decrypts it using his secret key.
4. Approach
   Think of messages from Alice to Bob as numbers (mod $N$).
Public Key Cryptography

Pick any 2 primes \( p \) and \( q \). Let \( N = p \cdot q \)
For any \( e \equiv 1 \pmod{(p - 1) \cdot (q - 1)} \):

1. The mapping \( x \rightarrow x^e \mod N \) is a bijection on \( \{0, 1, \ldots, N - 1\} \).

2. Let \( d \) be the inverse of \( e \mod (p - 1) \cdot (q - 1) \). Then, for all \( x \in \{0, 1, \ldots, N - 1\} \):
   \[
   (x^e)^d \equiv x \pmod{N}
   \]

A reasonable way to encode \( x \) and decode \( x \) respectively.
Pick any 2 primes \( p \) and \( q \). Let \( N = p \cdot q \)
For any \( e \equiv 1 \pmod{(p-1) \cdot (q-1)} \):

1. The mapping \( x \rightarrow x^e \pmod{N} \) is a bijection on \( \{0, 1, \ldots, N - 1\} \).
   
   ▶ A reasonable way to encode \( x \)
Pick any 2 primes \( p \) and \( q \). Let \( N = p \cdot q \)

For any \( e \equiv 1 \pmod{(p - 1) \cdot (q - 1)} \): 

1. The mapping \( x \mapsto x^e \mod N \) is a bijection on \( \{0, 1, \ldots, N - 1\} \).
   - A reasonable way to encode \( x \)

2. Let \( d \) be the inverse of \( e \mod (p - 1) \cdot (q - 1) \). Then, \( \forall x \in \{0, 1, \ldots, N - 1\} \):
   
   \[ (x^e)^d \equiv x \pmod{N} \]
   - A reasonable way to decode \( x \)
Proof of RSA

1. (2.) implies (1.) since the mapping is invertible
2. $e$ is invertible module $(p - 1) \cdot (q - 1)$ because $e$ is relatively prime to this number
3. $e \cdot d \equiv 1 \mod (p - 1) \cdot (q - 1)$, then,
   
   $e \cdot d = 1 + k \cdot (p - 1) \cdot (q - 1)$ for some $k$. Show
   
   $x^{e \cdot d} - x \equiv x^{1 + k \cdot (p - 1) \cdot (q - 1)} - x$

   is always 0 (mod $N$)
RSA: R Rivest, A. Shamir and L. Adleman (MIT)

1. Bob picks up 2 large prime numbers $p$ and $q$. His public key is $(N = p \cdot q, e)$. 
   $e \equiv 1 \pmod{(p - 1) \cdot (q - 1)}$
   Bob’s secret key is $d$, 
   $d \cdot e \equiv 1 \pmod{(p - 1) \cdot (q - 1)}$

2. Alice sends Bob $y = x^e \mod N$

3. Bob decodes $x$ by computing $y^d \mod N$

1. Given $N$, $e$, and $y = x^e \mod N$, it is computational intractable to determine $x$

2. FACTORING is HARD
Chapter 1 of DPV — Algorithms with Numbers

▶ Foundations

1. Basic Arithmetic
2. Modular Arithmetic
3. Primality Testing

▶ Applications

1. Cryptography
2. Universal Hashing
Hashing Table

- **Dictionary**
  Given a universe $U$ of possible elements, maintain a subset $S \subseteq U$ so that inserting, deleting, and searching in $S$ is efficient.

- **Challenge**
  Universe $U$ can be extremely large so defining an array of size $|U|$ is infeasible.

- **Applications**
  File systems, databases, Google, compilers, checksums P2P networks, associative arrays, cryptography, web caching, etc.
Hashing Table

- fast access + efficient storage
- random function + consistent function
- distribution is unknown

![Hashing Table Diagram](image)
1. **Hash function**

   \[ h : U \rightarrow \{0, 1, \ldots, n - 1\} \]

2. **Hashing**

   Create an array \( H \) of size \( n \). When processing element \( u \in U \), access array element \( H[h(u)] \)

3. **Collision**

   When \( h(u) = h(v) \) but \( u \neq v \)
   - A collision is expected after \( \Omega(\sqrt{n}) \) random insertions
     Why? *birthday paradox* — next Lecture
   - *Separate chaining*
     \( H[i] \) stores linked list of elements \( u \) with \( h(u) = i \)
1. **Idealistic hash function**
   Maps \( m \) elements uniformly at random to \( n \) hash slots
   
   a. Running time depends on length of chains
   b. Average length of chain = \( m/n \)
   c. Choose \( n \approx m \) \( \Rightarrow \) on average \( O(1) \) per insert, lookup, or delete

2. **Universal hashing**
   
   a. For any pair of elements \( u, v \in U \)
      
      \[
      \Pr_{h \in H} [h(u) = h(v)] \leq \frac{1}{n}
      \]
   b. Can select random \( h \) efficiently
   c. Can compute \( h(u) \) efficiently
Theorem

Universal hashing property. Assume $H$ be a universal class of hash functions. Let $h \in H$ be chosen uniformly at random from $H$; and let $u \in U$.
Then, for any subset $S \subseteq U$ of size at most $n$, the expected number of items in $S$ that collide with $u$ is at most 1.

Proof.

?
A Universal Hashing

For any 4 coefficients \( a_1, a_2, a_3, a_4 \in \{0, 1, \ldots, n - 1\} \), write \( a = (a_1, a_2, a_3, a_4) \) and define

\[
h_a(x_1, x_2, x_3, x_4) = \sum_{i=1}^{4} (a_i \cdot x_i \mod n)
\]

Theorem

Consider any pair of distinct IP addresses \( x = (x_1, x_2, x_3, x_4) \) and \( y = (y_1, y_2, y_3, y_4) \). If the coefficients \( a = (a_1, a_2, a_3, a_4) \) are chosen uniformly at random from \( \{0, 1, \ldots, n - 1\} \), then

\[
\Pr\{ h_a(x_1, x_2, x_3, x_4) = h_a(y_1, y_2, y_3, y_4) \} = \frac{1}{n}
\]

Proof. 

\[
\square
\]