Number Theory and Cryptography

Chapter 4

With Question/Answer Animations

Chapter Summary

4.1 Divisibility and Modular Arithmetic

4.2 Integer Representations and Algorithms

4.3 Primes and Greatest Common Divisors

4.4 Solving Congruences

4.5 Applications of Congruences

4.6 Cryptography

Integer Representations and Algorithms

Section 4.2

Section Summary

- Integer Representations
  - Base *b* Expansions
  - Binary Expansions
  - Octal Expansions
  - Hexadecimal Expansions
- Base Conversion Algorithm
- Algorithms for Integer Operations

Representations of Integers

- The modern world uses *decimal*, or *base* 10, *notation*:
  - 965 means $9 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0$.
- We can represent #'s using any base $b > 1$, $b \in \mathbb{Z}^+$
- For computing and communications, bases b =
  - 2 (*binary*)
  - 8 (*octal*)
  - 16 (*hexadecimal*)

  are important
- The ancient
  - Mayans used base 20
  - Babylonians used base 60.

Base *b* Representations

- We can use any base $b > 1$, $b \in Z^+$ because of this theorem:

  Theorem 1: Let $b > 1$, $b \in Z^+$. Then if $n \in Z^+$, it can be expressed uniquely in the form:

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b + a_0$$

  where $k \in N$, $a_0, a_1, ... a_k \in N$, $< b$, and $a_k \neq 0$.

  (We will prove this using math induction in Section 5.1.)

- a_j are *digits* (or bits in case b = 2).
- The *base b representation or expansion* is denoted

$$(a_k a_{k-1} .... a_1 a_0)_b.$$

(We usually omit the subscript for base 10 expansions.)

Binary Representations

Most computers represent integers and do arithmetic with binary (base 2), using digits (bits) 0 and 1.

Example: What are the decimals for the following binary representations?

a. $(11011)_2$

b. $(1\ 0101\ 1111)_2$

Solution:

a. $(11011)_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 27.$

b. $(1\ 0101\ 1111)_2 = 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 351.$

Octal Expansions

The octal expansion (base 8) uses the digits {0,1,…7}.

Example: Find decimal expansions for

a. $(111)_8$

b. $(7016)_8$

Solution:

a. $1{\cdot}8^2 + 1{\cdot}8^1 + 1{\cdot}8^0 = 64 + 8 + 1 = 73$

b. $7{\cdot}8^3 + 0{\cdot}8^2 + 1{\cdot}8^1 + 6{\cdot}8^0 = 3598$

Hexadecimal Expansions

Hexadecimal expansion needs 16 digits, but decimals provide only 10. So 6 letters are used:

{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

dec	hex
10	A
11	B
12	C
13	D
14	E
15	F

Example: Find decimal expansions for

a. $(E5)_{16}$

b. $(2AE0B)_{16}$

Solution:

a. $(E5)_{16} = 14 \cdot 16^1 + 5 \cdot 16^0 = 224 + 5 = 229$

b. $(2AE0B)_{16} = 2 \cdot 16^4 + 10 \cdot 16^3 + 14 \cdot 16^2$

$+ 0 \cdot 16^1 + 11 \cdot 16^0 = 175627$

Decimal to Base *b* Conversion

To construct base b expansion of $n \in Z^+$:

- Divide n by b

$$n = bq_0 + a_0 \quad 0 \le a_0 \le b$$

  - The remainder, a_0, is rightmost digit.
- Next, divide q_0 by b (previous quotient is new dividend)

$$q_0 = bq_1 + a_1 \quad 0 \le a_1 \le b$$

  - The remainder, a_1, is 2nd digit from right.
- Continue by successively dividing the quotients by b,
  - obtaining additional base b digits as the remainder.
- The process terminates when the quotient is 0.

continued →

Algorithm: Constructing Base *b* Expansions

procedure *expansion*($n, b \in Z^+, b > 1$)
$q := n$
$k := 0$
while ($q \neq 0$)
    $a_k := q$ **mod** b
    $q := q$ **div** b
    $k := k + 1$
return($a_{k-1}, \ldots, a_1, a_0$){($a_{k-1} \ldots a_1 a_0)_b$ is base b expansion of n}

- q represents the quotient obtained by successive divisions by b, starting with $q = n$.
- The digits in the expansion are the remainders of the division given by q **mod** b.
- The algorithm terminates when $q = 0$ is reached.

Conversion to octal

Example: Find octal expansion of 12345

Solution: Successively divide by 8:

- $12345 = 8 \cdot 1543 + \mathbf{1}$
- $1543 = 8 \cdot 192 + \mathbf{7}$
- $192 = 8 \cdot 24 + \mathbf{0}$
- $24 = 8 \cdot 3 + \mathbf{0}$
- $3 = 8 \cdot \mathbf{0} + \mathbf{3}$ (stop when the quotient is 0)

The digits are the remainders read backwards:

$$(30071)_8$$

Hex, Octal and Binary Chart

Iexadecimal, Octal, and Binary Representation of the Integers 0 through 15.

D	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
O	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17
B	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111

Initial 0s are not shown

Each octal digit corresponds to a block of 3 binary digits.
Each hexadecimal digit corresponds to a block of 4 binary digits.
So, conversion between binary, octal, and hexadecimal is easy.

Conversion within Hex, Octal & Binary

Example: Find octal and hex expansions of

$$(11\ 1110\ 1011\ 1100)_2.$$

Solution:

- Octal: group into blocks of **three** adding initial 0s as needed

$$(011\ 111\ 010\ 111\ 100)_2.$$

Blocks correspond to 3 7 2 7 4. Hence, solution is

$$(37274)_8.$$

- Hex: group into blocks of **four** adding initial 0s as needed

$$(0011\ 1110\ 1011\ 1100)_2.$$

Blocks correspond to 3 E B C. Hence, solution is

$$(3EBC)_{16}.$$

Binary Addition of Integers

- Since computer chips work with binary numbers, algorithms for performing operations are important.

procedure *add*($a = (a_{n-1}, a_{n-2}, \ldots, a_0)_2$,($b = b_{n-1}, b_{n-2}, \ldots, b_0)_2$){binary expansions for a,b}
c := 0 (carry from previous addition)
for j := 0 to $n - 1$
    $t := a_j + b_j + c$
    $c := t$ **div** 2
    $s_j := t$ **mod** 2
$s_n := c$
return($s = (s_n, s_{n-1}, \ldots, s_0)_2$){$s$, the binary expansion of $a + b$.}

- #operations is 4*n* (2*n* bit adds, *n* **div**’s, *n* **mod**’s).
- So in particular, #bit additions is $O(n)$.

Binary Multiplication of Integers

procedure $mult(a = (a_{n-1}, a_{n-2}, \ldots, a_0)_2, (b = b_{n-1}, b_{n-2}, \ldots, b_0)_2)$
for $j := 0$ to $n - 1$
    if $b_j = 1$ **then** $c_j = a$ shifted j places
    else $c_j := 0$ {$c_o, c_1, \ldots, c_{n-1}$ are the partial products}
$p := 0$
for $j := 0$ to $n - 1$
    $p := p + c_j$
return p {p is the value of ab}

- Output will be of length $2n$
  - $7\ (1, 1, 1)_2 \times 7\ (1, 1, 1)_2 = 49\ (1, 1, 0, 0, 0, 1)_2$
- #additions of bits is $O(n^2)$.
- Could easily modify so that inputs are of lengths m, n.

Binary Modular Exponentiation

In cryptography, it is important to be able to find b^n **mod** m efficiently, where b, n, and m are large integers.

- Use the binary expansion of n $(a_{k-1},\dots,a_1,a_0)_2$, to compute b^n .

  Note that:

$$b^n = b^{a_{k-1}\cdot 2^{k-1}+\cdots+a_1\cdot 2+a_0} = b^{a_{k-1}\cdot 2^{k-1}}\cdots b^{a_1\cdot 2}\cdot b^{a_0}.$$

- $\therefore$ to compute b^n, compute b, b^2, $(b^2)^2 = b^4$, $(b^4)^2 = b^8$, …, b^{2^k} and then multiply the terms b^{2^j} in this list, where $a_j = 1$.

Example: Compute 3^{11} using this method.

Solution: Note that $11 = (1011)_2$ so $3^{11} = 3^8\, 3^2\, 3^1 = ((3^2)^2)^2\, 3^2\, 3^1$

$= (9^2)^2 \cdot 9 \cdot 3 = (81)^2 \cdot 9 \cdot 3 = 6561 \cdot 9 \cdot 3 = 117{,}147.$

continued →

Binary Modular Exponentiation Algorithm

procedure *modular exponentiation* (b: integer, $n = (a_{k-1}a_{k-2}\ldots a_1a_0)_2$, $m \in \mathbf{Z}^+$)
$x := 1$
$power := b$ **mod** m
for $i := 0$ to $k - 1$
  if $a_i = 1$ **then** $x := (x \cdot power)$ **mod** m
  $power := (power \cdot power)$ **mod** m
return x {x equals b^n **mod** m }

- Algorithm successively finds
  b **mod** m, b^2 **mod** m, b^4 **mod** m, …, $b^{2^{k-1}}$ **mod** m,
- And multiplies together the terms b^{2^j} where $a_j = 1$.
- $O((\log m)^2 \log n)$ bit operations used.