ICS141: Discrete Mathematics for Computer Science I

Dept. Information & Computer Sci., University of Hawaii

Jan Stelovsky

based on slides by Dr. Baek and Dr. Still

Originals by Dr. M. P. Frank and Dr. J.L. Gross

Provided by McGraw-Hill

Quiz

1. gcd(84,96) =
2. lcm(84,96) =
3. gcd(84,96) x lcm(84,96)=

- Hints
  - What's the prime factorization of 84?
  - What's the prime factorization of 96?
  - Try the primes 2, 3 and 7

Lecture 18

Chapter 3. The Fundamentals

3.6 Integers and Algorithms

Review: Greatest Common Divisor

- The ***greatest common divisor*** gcd(a,b) of integers a,b (not both 0) is the largest integer d that is a divisor both of a and of b.

  d = gcd(a,b) = max(d: $d|a \land d|b$)
  $\Leftrightarrow d|a \land d|b \land \forall e \in \mathbf{Z}, (e|a \land e|b) \rightarrow (d \geq e)$

- If the prime factorizations are written as $a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$ and $b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$, then the GCD is given by:

$$\boxed{\gcd(a,b) = p_1^{\min(a_1,b_1)} p_2^{\min(a_2,b_2)} \ldots p_n^{\min(a_n,b_n)}.}$$

- Example: $84 = 2^2 \cdot 3^1 \cdot 7^1$ and $96 = 2^5 \cdot 3^1 \cdot 7^0$
  - gcd(84,96) = $2^2 \cdot 3^1 \cdot 7^0 = 2 \cdot 2 \cdot 3 = 12$.

Review: Least Common Multiple

- lcm(a,b) of positive integers a, b, is the smallest positive integer that is a multiple both of a and of b. *E.g.* lcm(6,10) = 30

  $m = \text{lcm}(a,b) = \min(m{:}\ a|m \wedge b|m)$
  $\Leftrightarrow a|m \wedge b|m \wedge \forall n\in\mathbf{Z}{:}\ (a|n \wedge b|n) \rightarrow (m \leq n)$

- If the prime factorizations are written as $a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$ and $b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$, then the LCM is given by:

$$\boxed{\text{lcm}(a,b) = p_1^{\max(a_1,b_1)} p_2^{\max(a_2,b_2)} \ldots p_n^{\max(a_n,b_n)}.}$$

- Example: $84 = 2^2 \cdot 3^1 \cdot 7^1$ and $96 = 2^5 \cdot 3^1 \cdot 7^0$
  - $\text{lcm}(84,96) = 2^5 \cdot 3^1 \cdot 7^1 = 32 \cdot 3 \cdot 7 = 672$.

GCD and LCM

University of Hawaii

- **Theorem**: Let a and b be positive integers. Then

$$ab = \gcd(a,b) \cdot \mathrm{lcm}(a,b)$$

- Example
  - $a = 84 = 2 \cdot 2 \cdot 3 \cdot 7 \quad = 2^2 \cdot 3^1 \cdot 7^1$
  - $b = 96 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 2^5 \cdot 3^1 \cdot 7^0$
  - $ab = (2^2 \cdot 3^1 \cdot 7^1) \cdot (2^5 \cdot 3^1 \cdot 7^0) = 2^2 \cdot 3^1 \cdot 7^0 \cdot 2^5 \cdot 3^1 \cdot 7^1$

$$= \underline{2^{\min(2,5)} \cdot 3^{\min(1,1)} \cdot 7^{\min(1,0)}} \cdot \underline{2^{\max(2,5)} \cdot 3^{\max(1,1)} \cdot 7^{\max(1,0)}}$$

$$= \gcd(a,b) \cdot \mathrm{lcm}(a,b)$$

Integers and Algorithms

University of Hawaii

- Topics:
  - Base-*b* representations of integers.
    - Especially: binary, hexadecimal, octal.
  - Algorithms for computer arithmetic:
    - Binary addition and multiplication.
  - Euclidean algorithm for finding GCD's.

Base-*b* Number Systems

- Ordinarily, we write *base*-10 representations of numbers, using digits 0-9.
- But, 10 isn't special! Any base $b > 1$ will work.
- For any positive integers n and b, there is a unique sequence $a_k a_{k-1} \ldots a_1 a_0$ of *digits* $a_i < b$ such that:

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

$$= \sum_{i=0}^{k} a_i b^i$$

The "*base-b expansion of n*"

- Notation: $n = (a_k a_{k-1} \ldots a_1 a_0)_b$

Particular Bases of Interest

- Base $b = 10$ (decimal):
  10 digits: 0,1,2,3,4,5,6,7,8,9.

Used only because we have 10 fingers

- Base $b = 2$ (binary):
  2 digits: 0,1. ("Bits"="<u>bi</u>nary dig<u>its</u>.")

Used internally in all modern computers

- Base $b = 8$ (octal):
  8 digits: 0,1,2,3,4,5,6,7.

Octal digits correspond to groups of 3 bits

- Base $b = 16$ (hexadecimal):
  16 digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Hex digits give groups of 4 bits

Examples

- <u>Example 1</u>: Decimal expansion of the integer with binary expansion $(101011111)_2$?
  - $(101011111)_2$

    $= 1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2 + 1$

    $= (351)_{10}$

- <u>Example 2</u>: Decimal expansion of the integer with hexadecimal expansion $(2AE0B)_{16}$?
  - $(2AE0B)_{16} = 2 \cdot 16^4 + 10 \cdot 16^3 + 14 \cdot 16^2 + 0 \cdot 16 + 11$

    $= (175627)_{10}$

Converting to Base *b*

(An algorithm, informally stated.)

- To convert any integer n to any base $b > 1$:
- To find the value of the *rightmost* (lowest-order) digit, simply compute n mod b.
- Now, replace n with the quotient.
- Repeat above two steps to find subsequent digits, until n is gone (= 0).

Exercise: Write this out in pseudocode…

Converting to Base b

$$
\begin{aligned}
n &= bq_0 + a_0 \\
&= b(bq_1 + a_1) + a_0 \\
&= b^2 q_1 + ba_1 + a_0 \\
&= b^2(bq_2 + a_2) + ba_1 + a_0 \\
&= b^3 q_2 + b^2 a_2 + ba_1 + a_0 \\
&= b^3(b \cdot 0 + a_3) + b^2 a_2 + ba_1 + a_0 \\
&= a_3 b^3 + a_2 b^2 + a_1 b + a_0 \\
&= (a_3 a_2 a_1 a_0)_b
\end{aligned}
$$

Examples

- Example 3: Find the base 8, i.e. octal, expansion of $(12345)_{10}$
  - $12345 = 8 \cdot 1543 + 1$
  - $1543 = 8 \cdot 192 + 7$
  - $192 = 8 \cdot 24 + 0$
  - $24 = 8 \cdot 3 + 0$
  - $3 = 8 \cdot 0 + 3$
  - Therefore, $(12345)_{10} = (30071)_8$

Binary ↔ Hexadecimal

TABLE 1 Hexadecimal, Octal, and Binary Representation of the Integers 0 through 15.

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

- Hexadecimal expansion of $(\underline{11}\ \underline{1110}\ \underline{1011}\ \underline{1100})_2$

  $0011 = 3 \quad E \quad B \quad C$

  $\therefore (11\ 1110\ 1011\ 1100)_2 = (3EBC)_{16}$

- Binary expansion of $(A8D)_{16}$

  $(A)_{16} = (1010)_2$, $(8)_{16} = (1000)_2$, $(D)_{16} = (1101)_2$

  $\therefore (A8D)_{16} = (1010\ 1000\ 1101)_2$

Addition of Binary Numbers

Carry: 111000

$$\begin{array}{r} 10111 \\ +\ 11100 \\ \hline 110011 \end{array}$$

- Carry = $\lfloor bitSum / 2 \rfloor$
- $s_{bitIndex} = bitSum \ \mathbf{mod}\ 2 = bitSum - 2 \cdot carry$

Addition of Binary Numbers

University of Hawaii

procedure $add(a_{n-1}\ldots a_0, b_{n-1}\ldots b_0$: binary representations of non-negative integers a, b)

$carry := 0$

for $bitIndex := 0$ **to** $n-1$ **begin** {go through bits}

$bitSum := a_{bitIndex}+b_{bitIndex}+carry$ {2-bit sum}

$carry := \lfloor bitSum / 2 \rfloor$ {high bit of sum}

$s_{bitIndex} := bitSum - 2\cdot carry$ {low bit of sum}

end

$s_n := carry$

return $s_n\ldots s_0$: binary representation of integer s

Multiplication of Binary Numbers

University of Hawaii

- $ab = a(b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_{n-1} \cdot 2^{n-1})$
  $= a(b_0 \cdot 2^0) + a(b_1 \cdot 2^1) + \cdots + a(b_{n-1} \cdot 2^{n-1})$

$$
\begin{array}{rl}
110 & a \\
\times\ 101 & b \\
\hline
110 & \\
0000 & \text{shift 1 bit to the left, i.e. append 1 extra 0-bit} \\
11000 & \text{shift 2 bit to the left, i.e. append 2 extra 0-bits} \\
\hline
11110 &
\end{array}
$$

Multiplication of Binary Numbers

University of Hawaii

- $ab = a(b_0 \cdot 2^0 + b_1 \cdot 2^1 + \cdots + b_{n\text{-}1} \cdot 2^{n\text{-}1})$
  $= a(b_0 \cdot 2^0\,) + a(b_1 \cdot 2^1\,) + \cdots + a(b_{n\text{-}1} \cdot 2^{n\text{-}1})$

procedure *multiply*($a_{n-1} \ldots a_0$, $b_{n-1} \ldots b_0$: binary representations of positive integers a,b)

product := 0

for i := 0 to $n-1$

if $b_i = 1$ **then**

product := *add*($a_{n-1} \ldots a_0 \underbrace{0 \cdots 0}_{i \text{ times}}$, *product*)

return *product*

i extra 0-bits appended after the digits of *a*

Division with Remainder

procedure *div-mod*($a \in \mathbf{Z}, d \in \mathbf{Z}^+$)
{quotient & remainder of a/d}
$q := 0$
$r := |a|$
while $r \geq d$ **begin**
$r := r - d$
$q := q + 1$
end
if $a < 0$ *and* $r > 0$ **then begin** {a is a negative}
$r := d - r$
$q := -(q + 1)$
end
{$q = a$ **div** d (quotient), $r = a$ **mod** d (remainder)}

Euclid's Algorithm for GCD

University of Hawaii

- Finding GCDs by comparing prime factorizations can be difficult when the prime factors are not known!
- **Euclid discovered:** Let $a = bq + r$, where a, b, q, and r are integers. Then $\gcd(a,b) = \gcd(b,r)$ (i.e. $\gcd(a,b) = \gcd(b, (a \bmod b))$)
  - Example: gcd(36, 24) = gcd(24, 12)
- Sort a, b so that $a > b$, and then (given $b > 1$) $(a \bmod b) < b$, so problem is simplified.

Euclid's Algorithm Example

- $\gcd(372, 164) = \gcd(164, 372 \bmod 164)$
  - $372 \bmod 164 = 372 - 164\lfloor 372/164 \rfloor$
    $= 372 - 164 \cdot 2 = 372 - 328 = 44$
- $\gcd(164, 44) = \gcd(44, 164 \bmod 44)$
  - $164 \bmod 44 = 164 - 44\lfloor 164/44 \rfloor$
    $= 164 - 44 \cdot 3 = 164 - 132 = 32$
- $\gcd(44, 32) = \gcd(32, 44 \bmod 32) = \gcd(32, 12)$
  $= \gcd(12, 32 \bmod 12) = \gcd(12, 8)$
  $= \gcd(8, 12 \bmod 8) = \gcd(8, 4)$
  $= \gcd(4, 8 \bmod 4) = \gcd(4, 0) = 4$

Euclid's Algorithm Pseudocode

University of Hawaii

procedure *gcd*(*a*, *b*: positive integers)

$x := a$

$y := b$

while $y \neq 0$ **begin**

$r := x$ **mod** y;

$x := y$;

$y := r$;

end

return x $\{x = \gcd(a, b)\}$

Proof That Euclid's Algorithm Works

- **Theorem 0:** $\gcd(a,b) = \gcd(b,c)$ if $c = a \bmod b$.

  Proof:
  - First, $c = a \bmod b$ implies $\exists t$: $a = bt + c$.
  - Let $g = \gcd(a,b)$, and $g' = \gcd(b,c)$.
  - Since $g|a$ and $g|b$ (thus $g|bt$) we know $g|(a-bt)$, *i.e.* $g|c$. Since $g|b \wedge g|c$, it follows that $g \leq \gcd(b,c) = g'$.
  - Now, since $g'|b$ (thus $g'|bt$) and $g'|c$, we know $g'|(bt+c)$, *i.e.*, $g'|a$. Since $g'|a \wedge g'|b$, it follows that $g' \leq \gcd(a,b) = g$.
  - Since we have shown that both $g \leq g'$ and $g' \leq g$, it must be the case that $g = g'$. ■

Two's Complement

- In binary, negative numbers can be conveniently represented using ***two's complement notation***.
- In this scheme, a string of n bits can represent any integer i such that $-2^{n-1} \le i < 2^{n-1}$.
- The leftmost bit is used to represent the sign (0:positive, 1:negative integer)
- The negation of any n-bit two's complement number $a = a_{n-1}\ldots a_0$ is given by $\overline{a_{n-1}\ldots a_0} + 1$.

The bitwise logical complement of the n-bit string $a_{n-1}\ldots a_0$.

Two's Complement Example

$$-2^2 \le i < 2^2$$
$$(-2^{n-1} \le i < 2^{n-1})$$

value	3-bit pattern
3	0 1 1
2	0 1 0
1	0 0 1
0	0 0 0
–1	1 1 1
–2	1 1 0
–3	1 0 1
–4	1 0 0

- To obtain the results for $-4 \le n \le -1$, consider $|n|$, then
  - In the binary representation of $|n|$, replace each 0 by 1, and each 1 by 0,
    This is the one's complement of *n*.
  - Add 1 (i.e. 001) to the result from the previous step.
    This is the two's complement of *n*.
- Example
  - –3: 011 -> 100
    + 001 = 101

Subtraction of Binary Numbers

University of Hawaii

procedure *subtract*($a_{n-1}\ldots a_0$, $b_{n-1}\ldots b_0$: binary two's complement reps. of integers a, b)

return *add*(a, *add*($\overline{b}$,1)) { $a + (-b)$ }

- Note that this fails if either of the adds causes a carry into or out of the $n-1$ position, since $2^{n-2}+2^{n-2} \neq -2^{n-1}$, and $-2^{n-1} + (-2^{n-1}) = -2^n$ isn't representable!
  We call this an *overflow*.

Modular Exponentiation

- **Problem:** Given large integers *b* (base), *n* (exponent), and *m* (modulus), efficiently compute $b^n \bmod m$.
  - Note that b^n itself may be completely infeasible to compute and store directly.
  - *E.g.* if *n* is a 1,000-bit number, then b^n itself will have far more digits than there are atoms in the universe!
- Yet, this is a type of calculation that is commonly required in modern cryptographic algorithms!

Algorithm Concept

University of Hawaii

- Note that:

$$b^n = b^{\overbrace{n_{k-1}\cdot 2^{k-1} + n_{k-2}\cdot 2^{k-2} + \ldots + n_0 \cdot 2^0}^{\text{The binary expansion of } n}}$$

$$= (b^{2^{k-1}})^{n_{k-1}} \times (b^{2^{k-2}})^{n_{k-2}} \times \cdots \times (\overbrace{b^{2^0}}^{= b^1 = b})^{n_0}$$

- We can compute *b* to various powers of 2 by repeated squaring.
  - Then multiply them into the partial product, or not, depending on whether the corresponding n_i bit is 1.
- Crucially, we can do the **mod** *m* operations <u>as we go along</u>, because of the various identity laws of modular arithmetic.
- All the numbers stay small.

Modular Exponentiation

University of Hawaii

procedure *modularExponentiation*(b: integer,

$n = (n_{k-1} \ldots n_0)_2$, m: positive integers)

$x := 1$ {accumulates the result}

$b2i := b$ **mod** m { $\boxed{b^{2^i}}$ **mod** m; i=0 initially}

for $i := 0$ to $k-1$ **begin** {go thru all k bits of n}

if $n_i = 1$ **then** $x := (x \cdot b2i)$ **mod** m

$b2i := (b2i \cdot b2i)$ **mod** m

end

return x

{x equals b^n **mod** m}

$$\boxed{b^{2^{i+1}} = b^{2 \cdot 2^i} = (b^{2^i}) \cdot (b^{2^i})}$$