Revision Notes
Algorithms Design and Analysis

Mingyu Liu

Nanyang Technological University
University of Waterloo

Revised on February 16, 2018

1 Algorithms with Numbers

- **Factoring**: given N, express as a product of prime numbers.
- **Primality**: given N, is it prime?

Factoring is hard as far as we know. Testing primality is not.

- **N-bit Binary Addition**
  Considering two binary numbers x, y, each with n bits.
  - Sum of x and y is at most $(n+1)$ bits
  - Each bit of sum can be found in constant time
    $\therefore$ Runtime is $O(n)$
- **N-bit Binary Multiplication**
  Consider
  $$x{\cdot}y = \begin{cases} 2(x{\cdot}\lfloor y/2 \rfloor) & \text{if y is even} \\ x + 2(x{\cdot}\lfloor y/2 \rfloor) & \text{if y is odd} \end{cases}$$
  - In binary, multiply by 2 is a "left shift".
  - Floored division by 2 is a "right shift".

Algorithm 1: Multiply two n-bit binary numbers

function multiply(x, y)
Input : x and y, $y \geq 0$
Output: $x{\cdot}y$
if $y = 0$ **then**
    return 0
end
$z =$ multiply $(x, \lfloor y/2 \rfloor)$
if y *is even* **then**
    return $2{\cdot}z$
else
    return $x + 2{\cdot}z$
end

- **N-bit Binary Division**
  - Will cover this later.

1.1 Modular Arithmetic

- x modulo N = the remainder when x is divided by N
- The definition of congruent
- **Substitution Rule**: if $x \equiv x\prime \pmod N$ and $y \equiv y\prime \pmod N$, then:
  $$x + y \equiv x\prime + y\prime \pmod N \text{ and } xy \equiv x\prime y\prime \pmod N$$
  Also, usual properties of addition and multiplication hold.
  Consequence: when simplifying a modulo expression, we can reduce intermediate results to their remainders modulo N at any stage!

- Addition, Multiplication and Division Modulo N
  Consider two modulo N numbers, x and $y \implies$ number of bits $n \leq \log N$.
  - Addition: $O(n)$
  - Multiplication $O(n^2)$
  - Division: we'll come back to this later.

1.2 Modular Exponentiation

We want to compute $x^y \bmod N$.
If x and y are large (say, 500-bit long!), then x^y is huge! It's not possible to explicitly compute it.
But $x^y \bmod N$ is between 0 and $N-1$. One solution:

$E = x \bmod N$
for $i = 1$ *to* $y - 1$ **do**
  $E = E \cdot x \bmod N$
end
return

This is not efficient. Notice that

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

Which gives us the following algorithm:

Algorithm 2: Modular Exponentiation

Input : two n-bit integers x and N, and an integer exponent y
Output: $x^y \bmod N$
function modexp(x, y, N)
if $y = 0$ **then**
  return 1
end
$z = \text{modexp}(x, \lfloor y/2 \rfloor, N)$
if y *is even* **then**
  return $z^2 \bmod N$
else
  return $x \cdot z^2 \bmod N$
end

This algorithm runs in $O(n^3)$ time.

1.3 Finding the Greatest Common Divisor (GCD)

- **Problem**: given two integers a and b, find the largest integer that divides both of them.
- **Euclid's Rule**: if a and b are positive integers, $a \geq b$. then

$$gcd(a, b) = gcd(a \ mod \ b, b)$$

Algorithm 3: Euclid's Algorithm

```
function Euclid(a, b)
  Input  : two integers a and b, a ≥ b ≥ 0
  Output: gcd(a, b)
if b = 0 then
  | return a
end
return Euclid(b, a mod b)
```

This algorithm runs in $O(n^3)$ time.

- **Lemma**: if $a \geq b$, then $a \ mod \ b < a/2$

1.4 An Extension of Euclid's Algorithm

- **Lemma**: if d divides both a and b, and $d = ax + by$ for some integers x and y, then necessarily

$$d = gcd(a, b) \quad (x, y \text{ can be negative})$$

- We can actually compute the coefficients x and y.

Algorithm 4: Extended-Euclid's Algorithm

```
  Input  : two integers a and b, a ≥ b ≥ 0
  Output: integers (x, y, d) s.t. d = gcd(a, b) and ax + by = d
function extended-Euclid(a, b)
if b = 0 then
  | return (1, 0, a)
end
(x', y', d) =extended-Euclid(b, a mod b)
return (y', x' − ⌊a/b⌋y', d)
```

1.5 Induction and Strong Induction

- **Induction**: to show that a statement $S(n)$ holds for all $k \geq 0$ (positive integers):

  Step 1 (base case): show $S(0)$ holds.

  Step 2 (inductive step): show that if $S(k)$ holds, then $S(k+1)$ holds.

  By doing this, we can show $S(k)$ holds for all $k \geq 0$.

- **Strong Induction**: in some cases, the weight of the k^{th} domino is not strong enough to knowck down the $(k+1)^{th}$ domino. Knocking down the $(k+1)^{th}$ domino requires the weight of all the dominoes before it.

  Step 1 (base case): same.

  Step 2 (inductive step): show that if $S(k)$ holds for all $k \leq \bar{k}$, then $S(\bar{k}+1)$ holds.

  Sometimes we may have to verify multiple base cases. E.g. you may have to show it holds for both $S(0)$ and $S(1)$ to make induction on the rule $S(k) = S(k-1) - S(k-2)$.

1.6 Modular Division

- **Multiplicative Inverse**: an integer x is the multiplicative inverse of a modulo N if

$$ax \equiv 1 \ (mod\ N)$$

For given a, there can be at most one inverse x $(mod\ N)$.

Multiplicative inverse does not always exist.

- If $gcd(a, N) = 1$, then multiplicative inverse of a is given by x from Extended-Euclid's algorithm. We should call extended-Euclid(N, a) so that the first argument is larger. In this case, multiplicative inverse would be y in (x, y, d) where $d = 1$ and $Nx + ay = d$.

- **Modular Division Theorem**: for any integer a, modulo N, a has multiplicative inverse if and only if $gcd(a, N) = 1$ (i.e. a and N are relatively prime.)
When multiplicative invere exists, it can be found in $O(n^3)$ time as $y\ mod\ N$ in

$$(x, y, d) = \text{extended-Euclid}(N, a)$$

1.7 Primality Testing

- **Fermat's Little Theorem**: if p is prime, then for every integer $1 \leq a < p$,

$$a^{p-1} \equiv 1 \ (mod\ p)$$

Note that this is not an if-and-only-if condition. Suggest a test for primeness of N, pick some $a \in \{1, 2, \cdots, N-1\}$ for Fermat's test:

- Pass: likely prime
- Fail: composite (not prime)

Algorithm 5: Primality Testing

Input : positive integer N
Output: yes/no

function primality(N)
Pick a positive integer $a < N$ at random
if $a^{N-1} \equiv 1 \ (mod\ N)$ **then**
  return 'yes' // *might be prime*
else
  return 'no' // *composite*
end

The algorithm has the following behavior:

- P(Algorithm return 'yes' when N is prime) $= 1$
- P(Algorithm return 'yes' when N is composite) $\leq 1/2$

Introducing the following randomized version of primality testing s.t.

$$P(\text{Algorithm return 'yes' when N is composite}) \leq 1/2^k$$

Algorithm 6: Primality Testing (Randomized)

Input : positive integer N
Output: yes/no

function primality2(N)
Pick k positive integers randomly from $a < N$ at random from $\{0, 1, 2, \cdots, N-1\}$
if ${a_i}^{N-1} \equiv 1$ $(mod\ N)$ *for all* $i = 1, 2, \cdots, k$ **then**
 return 'yes' // *might be prime*
else
 return 'no' // *composite*
end

Algorithm 7: Generate Random Prime Numbers

Pick a random n-bit number N
Run a primality test on N
if N *passes the test* **then**
 Output N
else
 Repeat the process
end

1.8 Generate Random Prime Numbers

- **Lagrange's Prime Number Theorem:** let $\Pi(x)$ be the number of primes $\leq x$, then $\Pi(x) \approx x/lnx$. Precisely, we have

$$\lim_{x\to\infty} \frac{\Pi(x)}{x/\ln x} = 1$$

Which implies that prime numbers are abundant!

1.9 Basic Cryptography

- Alice wants to send message x to Bob
- Alice encodes it as $e(x)$
- Bob decodes it using his decoding function $d(\cdot)$, $d(e(x)) = x$
- In **private-key schemes**, Alice and Bob meet beforehand to choose $d(\cdot)$
- An example of private-key schemes: AES (Advanced-Encryption Standard)

1.10 RSA (Public-Key Encryprography)

- **Public Key**: e and N for encryption
- **Private Key**: d for decryption
- To send an encrypted message x:

$$encryption = x^e\ mod\ N$$

- To decrypt an encrypted message received:

$$decrypted = (encrypted)^d\ mod\ N$$

- How to Choose e, N, d?

  $N = p \cdot q$, where $\{p, q\}$ are two prime numbers roughly of the same size.
  $e =$ some number co-prime with $(p-1)(q-1)$, can be computed with extended-Euclid$(\cdot)$.
  $d =$ multiplicative inverse of e $modulo(p-1)(q-1)$, i.e. $ed \equiv 1 \ mod \ (p-1)(q-1)$

2 Divide and Conquer

- Break into independent problems.
- Recursively solve subproblems.
- Combine the answers.

2.1 Multiplication

Multiply two n-bit binary numbers x and y (assume both are in power of 2 for simplicity)

- **Idea**: break x and y into left and right halves.

  $x = \boxed{1010}\boxed{0111}$ into $x_L = 1010$, $x_R = 0111 \implies x = \underbrace{2^4 x_L}_{\text{left shift}} + x_R$

  (x_L and x_R are $n/2$-bit numbers)

  $\therefore x \cdot y = (2^{n/2}x_L + x_R) \cdot (2^{n/2}y_L + y_R) = 2^n \underbrace{x_L \cdot y_L} + 2^{n/2}(\underbrace{x_L \cdot y_R} + \underbrace{x_R \cdot y_L}) + \underbrace{x_R \cdot y_R}$

  Compute recursively, runtime $T(n) = 4T(\frac{n}{2}) + O(n) \implies O(n^2)$

- **A Better Idea**:

  In above equation, notice that $\underbrace{x_L \cdot y_R + x_R \cdot y_L}_{\text{what we want}} = \underbrace{(x_L + x_R) \cdot (y_L + y_R)}_{T(\frac{n}{2})} - \underbrace{x_L \cdot y_R}_{T(\frac{n}{2})} - \underbrace{x_R \cdot y_R}_{T(\frac{n}{2})}$

  New runtime $T(n) = 3T(\frac{n}{2}) + O(n) \implies O(n^{1.59})$

Algorithm 8: Multiply two n-bit binary numbers

Input : x and y, $y \geq 0$
Output: $x \cdot y$

function multiplyDC(x, y)
if $n = 1$ **then**
  return $x \cdot y$
end
$x_L, y_L =$ leftmost $\lceil n/2 \rceil$-bits of x, y
$x_R, y_R =$ rightmost $\lceil n/2 \rceil$-bits of x, y
$P_1 =$ multiplyDC(x_L, y_L)
$P_2 =$ multiplyDC(x_R, y_R)
$P_3 =$ multiplyDC$(x_L + x_R, y_L + y_R)$
return $2^{2 \cdot \lfloor n/2 \rfloor} \cdot P_1 + (P_3 - P_1 - P_2) \cdot 2^{\lfloor n/2 \rfloor} + P_2$

2.2 Master Theorem

2.3 Mergesort

2.4 Medians

2.5 Selection

2.6 Matrix Multiplication

2.7 Fast Fourier Transform

2.8 Polynomial Multiplication

2.9 Multiply n-bit numbers with FFT

3 Graph

4 Greedy Algorithms