Theory of Algorithms II

Algorithms with Numbers

Guoqiang Li

School of Software, Shanghai Jiao Tong University

How to Represent Numbers

We are most familiar with decimal representation:

- 1024

But computers use binary representation:

- $1\underbrace{0\ldots0}_{\text{10 times}}$

Does base change the algorithm complexity?

Bases and Logs

Q: How many digits are needed to represent the number $N \geq 0$ in base b?

$$\lceil \log_b(N+1) \rceil$$

Q: How much does the size of a number change when we change bases?

$$\log_b N = \frac{\log_a N}{\log_a b}$$

In O notation, the base is irrelevant, and thus we write the size simply as $O(\log N)$

Thinking Over

Q: Does clouding computing change the algorithm complexity?

Q: Does quantum computing change the algorithm complexity?

Size as Bit

We will consider the algorithm of the basic arithmetics, regarding the length of number bits as the size of the algorithm input.

Bit Complexity

A single instruction we can add integers whose size in bits is within the word length of today's computer - 64 perhaps.

It is useful and necessary to handle numbers much larger than this, perhaps several thousand bits long.

Also in the hardware of today's computers, we shall focus on the bit complexity of the algorithm, the number of elementary operations on individual bits.

Addition

Addition

The sum of any three single-digit number is at most two digits long.

Addition

$$
\begin{array}{ccccccc}
1 & & & 1 & 1 & 1 & \\
 & 1 & 1 & 0 & 1 & 0 & 1 \\
 & 1 & 0 & 0 & 0 & 1 & 1 \\
\hline
1 & 0 & 1 & 1 & 0 & 0 & 0
\end{array}
$$

Addition

Q: Given two binary number x and y, how long does our algorithm take to add them?

A: $O(n)$

Addition

Q: Can we do better?

A: We must read them and write down the answer, and even that requires n operations.

So the addition algorithm is optimal.

Multiplication

The grade-school algorithm for multiplying two number x and y is to create an array of intermediate sums...

If x and y are both n bit, then there are n intermediate rows with length of up to $2n$ bit.

$$
\begin{array}{cccccccc}
 & & & & 1 & 1 & 0 & 1 \\
 & & & \times & 1 & 0 & 1 & 1 \\
\hline
 & & & & 1 & 1 & 0 & 1 \\
 & & & 1 & 1 & 0 & 1 & \\
 & & 0 & 0 & 0 & 0 & & \\
+ & 1 & 1 & 0 & 1 & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

$$\underbrace{O(n) + \ldots + O(n)}_{n-1}$$

$$O(n^2)$$

Multiplication by Al Khwarizmi

- write them next to each other.
- halve the first number by 2, dropping the .5, and double the second number.
- keep going till the first number gets down to 1.
- strike out all the rows where the first number is even.
- add up the remains in the second columns.

11	13
5	26
2	52
1	104
	143

Multiplication by Al Khwarizmi

- write them next to each other.
- halve the first number by 2, dropping the .5, and double the second number.
- keep going till the first number gets down to 1.
- strike out all the rows where the first number is even.
- add up the remains in the second columns.

11	13
5	26
2	52
1	104
	143

- The left is to calculate the binary number.
- The right is to shift the row!

Multiplication á la Françis

```
MULTIPLY(x, y);
Two n-bit integers x and y, where y ≥ 0;
if y = 0 then return 0;
;
z=MULTIPLY(x, ⌊y/2⌋);
if y is even then
    return 2z;
    else return x + 2z;
    ;
end
```

Another formulation:

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

Multiplication á la Françis

- Q: How long does the algorithm take?
- It will terminate after n recursive calls, since at each call y is halved.
- At each call requires these operations:
  - a division by 2 (right shift);
  - a test for odd/even (looking up the last bit);
  - a multiplication by 2 (left shift);
  - and a possibly one addition.
- A total operations are $O(n)$, The total time taken is thus $O(n^2)$.
- Q: Can we do better?
  - Yes!

Product of Complex Numbers

Carl Friedrich Gauss(1777-1855) noticed that although the product of two complex numbers

$$(a + bi)(c + di) = ac - bd + (bc + ad)i$$

involves four real-number multiplications, it can in fact be done with just three: ac, bd, and $(a + b)(c + d)$, since

$$bc + ad = (a + b)(c + d) - ac - bd$$

Multiplication

Suppose x and y are two n-integers, and assume for convenience that n is a power of 2.

[Hints: For every n there exists an n' with $n \leq n' \leq 2n$ such that n' a power of 2.]

As a first step toward multiplying x and y, we split each of them into their left and right halves, which are $n/2$ bits long

$$x = \boxed{x_L}\ \boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L}\ \boxed{y_R} = 2^{n/2}y_L + y_R$$

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Multiplication

The recurrence relations:

$$T(n) = 4T(n/2) + O(n)$$

Solution: $O(n^2)$

By Gauss's trick, three multiplications $x_L y_L$, $x_R y_R$, and $(x_L + x_R)(y_L + y_R)$ suffice.

Algorithm for Integer Multiplication

`MULTIPLY`(x, y)
Two positive integers x and y, in binary;

n=`max` (size of x, size of y) rounded as a power of 2;
if $n = 1$ **then** `return`(xy);
;
x_L, x_R= leftmost $n/2$, rightmost $n/2$ bits of x;
y_L, y_R= leftmost $n/2$, rightmost $n/2$ bits of y;
$P1$=`MULTIPLY`(x_L, y_L);
$P2$=`MULTIPLY`(x_R, y_R);
$P3$=`MULTIPLY`$(x_L + x_R, y_L + y_R)$;
`return`$(P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2)$

Time Analysis

The recurrence relation:

$$T(n) = 3T(n/2) + O(n)$$

$$O(n^{\log_2 3}) \approx O(n^{1.59})$$

Modular Arithmetic

What Is Modular

Modular arithmetic is a system for dealing with restricted ranges of integers.

x modulo N is the remainder when x is divided by N; that is, if $x = qN + r$ with $0 \leq r < N$, then x modulo N is equal to r.

x and y are congruent modulo N if they differ by a multiple of N, i.e.

$$x \equiv y \pmod{N} \quad \Longleftrightarrow \quad N \texttt{ divides } (x - y)$$

Rules

Substitution rules: if $x \equiv x' \pmod{N}$ and $y \equiv y' \pmod{N}$, then

$$x + y \equiv x' + y' \pmod{N}$$
$$xy \equiv x'y' \pmod{N}$$

$$\begin{array}{rlr} x + (y + z) \equiv (x + y) + z & \pmod{N} & \texttt{Associativity} \\ xy \equiv yx & \pmod{N} & \texttt{Commutativity} \\ x(y + z) \equiv xy + xz & \pmod{N} & \texttt{Distributivity} \end{array}$$

- It is legal to reduce intermediate results to their remainders modulo N at any stage.

$$2^{345} \equiv (2^5)^{69} \equiv 32^{69} \equiv 1^{69} \equiv 1 \pmod{31}$$

Modular Addition

Since x and y are each in the range 0 to $N-1$,their sum is between 0 and $2(N-1)$.

If the sum exceeds $N-1$, subtract off N.

Its running time is $O(n)$.

Modular Multiplication

The product of x and y can be as large as $(N-1)^2$, at most $2n$ bits long since

$$log\,(N-1)^2 = 2log\,(N-1) \leq 2n$$

To reduce the answer modulo N, compute the remainder upon dividing it by N. ($O(n^2)$)

$O(n^2)$.

Modular Exponentiation

In the cyptosystem, it is necessary to compute $x^y \pmod{N}$ for values of x, y, and N that are several hundred bits long.

When x and y are just 20-bit numbers, x^y is at least

$$(2^{19})^{(2^{19})} = 2^{(19)(524288)}$$

about 10 million bits long!

Modular Exponentiation

We need to perform all intermediate computations modulo N.

First idea: calculate $x^y \mod N$ by repeatedly multiplying by x modulo N.

$$x \mod N \to x^2 \mod N \to x^3 \mod N \to \ldots \to x^y \mod N$$

Still untractable!

Modular Exponentiation

Second idea: starting with x and squaring repeatedly modulo N, we get

$$x \mod N \to x^2 \mod N \to x^4 \mod N \to x^8 \mod N \to \ldots \, x^{2^{\lfloor \log y \rfloor}} \mod N$$

Each takes just $O(\log^2 N)$ time to compute, and only $\log y$ multiplications.

$$x^{25} = x^{11001_2} = x^{10000_2} \cdot x^{1000_2} \cdot x^{1_2} = x^{16} \cdot x^8 \cdot x^1$$

Modular Exponentiation

MODEXP (x, y, N);

Two n-bit integers x and N, and an integer exponent y;

if $y = 0$ **then** return 1;

;

z=MODEXP $(x, \lfloor y/2 \rfloor, \mathrm{N})$;

if *y is even* **then**

  return $z^2 \bmod N$;

  else return $x \cdot z^2 \pmod{N}$;

  ;

end

Another formulation:

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

Modular Exponentiation

`MODEXP` (x, y, N);
Two n-bit integers x and N, and an integer exponent y;

if $y = 0$ **then** `return` 1;
;
z=`MODEXP` $(x, \lfloor y/2 \rfloor, \mathrm{N})$;
if y *is even* **then**
    `return` z^2 `mod` N;
    else `return` $x \cdot z^2$ (`mod` N);
    ;
end

$O(n^3)$

Euclid's Algorithm for Greatest Common Divisor

Q: Given two integers x and y, how to find their greatest common divisor ($gcd(x, y)$)?

Euclid's rule: If x and y are positive integers with $x \geq y$, then $gcd(x, y) = gcd(x \pmod{y}, y)$.

Proof:
It is enough to show the rule $gcd(x, y) = gcd(x - y, y)$. Result can be derived by repeatedly subtracting y from x.

Euclid's Algorithm for Greatest Common Divisor

```
EUCLID(x, y);
```
Two integers x and y with $x \geq y$;

if $y = 0$ **then** `return` x;

;

`return(EUCLID(`$y, x \pmod{y}$`));`

Lemma:

If $a \geq b \geq 0$, then $a \pmod{b} < a/2$

Proof:

- if $b \leq a/2$, $a \bmod b < b \leq a/2$;
- if $b > a/2$, $a \bmod b = a - b < a/2$.

Euclid's Algorithm for Greatest Common Divisor

$\texttt{EUCLID}(x, y)$;
Two integers x and y with $x \geq y$;

if $y = 0$ **then** $\texttt{return}\ x$;
;
$\texttt{return}(\texttt{EUCLID}(y, x \pmod{y}))$;

Lemma:
If $a \geq b \geq 0$, then $a \pmod{b} < a/2$

This means that after any two consecutive rounds, both arguments, x and y are at the very least halved in value, i.e., the length of each decreases at least one bit.

Euclid's Algorithm for Greatest Common Divisor

```
EUCLID(x, y);
```
Two integers x and y with $x \geq y$;

if $y = 0$ **then** `return` x;

;

`return(EUCLID(`$y, x \pmod{y}$`));`

Lemma:

If $a \geq b \geq 0$, then $a \pmod{b} < a/2$

If they are initially n-bit integers, then the base case will be reached within $2n$ recursive calls. Since each call involves a quadratic-time division, the total time is $O(n^3)$.

An Extension of Euclid's Algorithm

Q: Suppose someone claims that d is the greatest common divisor of x and y, how can we check this?

It is not enough to verify that d divides both x and y…

Lemma

If d divides both x and y, and $d = ax + by$ for some integers a and b, then necessarily $d = gcd(x, y)$.

Proof:

- $d \leq gcd(x, y)$, obviously;
- $d \geq gcd(x, y)$, since $gcd(x, y)$ can divide x and y, it must also divide $ax + by = d$.

An Extension of Euclid's Algorithm

`EXTENDED-EUCLID`(a, b);
Two integers a and b with $a \geq b \geq 0$;

if $b = 0$ **then** `return`$(1, 0, a)$;
;
(x', y', d)=`EXTENDED-EUCLID`$(b, a \pmod{b})$;
`return`$(y', x' - \lfloor a/b \rfloor y', d)$;

Correctness of the algorithm?
DIY!

Modular Inverse

We say x is the multiplicative inverse of a modulo N if

$$ax \equiv 1 \pmod{N}$$

There can be at most one such x modulo N, denoted a^{-1}.

Remark:
The inverse does not always exists! for instance, 2 is not invertible modulo 6.

Modular Inverse

If $gcd(a, N) > 1$, then $ax \not\equiv 1 \mod N$.

Proof: $ax \mod N = ax + kN$, then $gcd(a, N)$ divides $ax \mod N$

If $gcd(a, N) = 1$, then extended Euclid algorithm gives us integers x and y such that $ax + Ny = 1$, which means $ax \equiv 1 \mod N$. Thus x is a's sought inverse.

Modular Division

Modular division theorem:

- For any $a \pmod{N}$, a has a multiplicative inverse modulo N if and only if it is relatively prime to N.
- When this inverse exists, it can be found in time $O(n^3)$ by running the extended Euclid algorithm.

This resolves the issues of modular division: when working modulo N, we can divide by numbers relatively prime to N. And to actually carry out the division, we multiply by the inverse.

Primality

A Notable Result

The AKS primality test is a deterministic primality-proving algorithm created and published by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, computer scientists at the Indian Institute of Technology Kanpur, on August 6, 2002, The algorithm was the first to determine whether any given number is prime or composite within polynomial time. The authors received the 2006 Gödel Prize and the 2006 Fulkerson Prize for this work.

Fermat's Little Theorem

If p is a prime, then for every $1 \leq a < p$,

$$a^{p-1} \equiv 1 \pmod{p}$$

Proof:

- Let $S = \{1, 2, \ldots, p-1\}$, then multiplying these numbers by $a \pmod{p}$ is to permute them.
- It is enough to prove that $a.i \pmod{p}$ are distinct for $i \in S$, and all the values are nonzero.
- multiplying all numbers in each representation, then gives $(p-1)! \equiv a^{(p-1)}.(p-1)! \pmod{p}$, and thus

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

(Problematic) Testing Primality

```
PRIMALITY(N);
```
Positive integer N;

Pick a positive integer $a < N$ at random;
if $a^{N-1} \equiv 1 \pmod{N}$ **then**
    `return` yes;
    else `return` no;
end

(Problematic) Testing Primality

The problem is that Fermat's theorem is not an if-and-only-if condition.

- e.g. $341 = 11 \cdot 31$, and $2^{340} \equiv 1 \pmod{341}$

Our best hope: for composite N, most values of a will fail the test.

Rather than fixing an arbitrary value of a, we should choose it randomly from $\{1, \ldots, N-1\}$.

Carmichael Number

Theorem: There are composite numbers N such that for every $a < N$ relatively prime to N,

$$a^{N-1} \equiv 1 \pmod{N}$$

Example:

$$561 = 3 \cdot 11 \cdot 17$$

Non-Carmichael Number

Lemma:

If $a^{N-1} \not\equiv 1 \pmod{N}$ for some a relatively prime to N, then it must hold for at least half the choices of $a < N$.

Proof:

- Fix some value of a for which $a^{N-1} \not\equiv 1 \pmod{N}$.
- Assume some $b < N$ satisfies $b^{N-1} \equiv 1 \pmod{N}$, then

$$(a \cdot b)^{N-1} \equiv a^{N-1} \cdot b^{N-1} \equiv a^{N-1} \not\equiv 1 \pmod{N}$$

- For $b \neq b'$, we have

$$a \cdot b \not\equiv a \cdot b' \pmod{N}$$

- The one-to-one function $b \mapsto a \cdot b \pmod{N}$ shows that at least as many elements fail the test as pass it.

Primality Testing

We are ignoring Carmichael numbers, so we can assert,

- If N is prime, then $a^{N-1} \equiv 1 \pmod N$ for all $a < N$
- If N is not prime, then $a^{N-1} \equiv 1 \pmod N$ for at most half the values of $a < N$.

Therefore, (for non-Carmichael numbers)

- $Pr(\texttt{PRIMALITY}$ returns yes when N is prime$) = 1$
- $Pr(\texttt{PRIMALITY}$ returns yes when N is not prime$) \leq 1/2$

Low Error Probability

```
PRIMALITY2(N);
```
Positive integer N;

Pick positive integers $a_1, \ldots, a_k < N$ at random;
if $a_i^{N-1} \equiv 1 \pmod{N}$ *for all* $1 \leq i \leq k$ **then**
  `return` yes;
  else `return` no;
end

- *Pr*(`PRIMALITY2` returns yes when N is prime)$= 1$
- *Pr*(`PRIMALITY2` returns yes when N is not prime)$\leq 1/2^k$

Generating Random Primes

Lagrange's Prime Number Theorem

Let $\pi(x)$ be the number of primes $\leq x$. Then $\pi(x) \approx x/ln(x)$, or more precisely,

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

Such abundance makes it simple to generate a random n-bit prime:

- Pick a random n-bit number N.
- Run a primarily test on N.
- If it passes the test, output N; else repeat the process.

Generating Random Primes

Q: How fast is this algorithm?

If the randomly chosen N is truly prime, which happens with probability at least $1/n$, then it will certainly pass the test.

On each iteration, this procedure has at least a $1/n$ chance of halting.

Therefore on average it will halt within $O(n)$ rounds.

- Exercise 1.34!

Cryptography

The Typical Setting

Alice and Bob, who wish to communicate in private.

Eve, an eavesdropper, will go to great lengths to find out what Alice and Bob are saying.

Even Ida, an intruder, will break the rules of communications positively.

The Typical Setting

Alice wants to send a specific message x, written in binary, to her friend Bob.

- Alice encodes it as $e(x)$, sends it over.
- Bob applies his decryption function $d(.)$ to decode it: $d(e(x)) = x$.
- Eve, will intercept $e(x)$: for instance, she might be a sniffer on the network.
- Ida, can do anything Eve does, he may also be able to pretend to be Alice or Bob.

Ideally, $e(x)$ is chosen that without knowing $d(.)$, Eve cannot do anything with the information she has picked up.

iow, knowing $e(x)$ tells her little or nothing about what x might be.

Private VS. Public Schemes

For centuries, cryptography was based on what we now call private-key protocols. In such a scheme, Alice and Bob meet beforehand and together choose a secret codebook.

Public-key schemes allow Alice to send Bob a message without ever having met him before.

Bob is able to implement a digital lock, to which only he has the key. Now by making this digital lock public, he gives Alice a way to send him a secure message.

Public-Key Schemes

Anybody can send a message to anybody else using publicly available information, rather like addresses or phone numbers.

Each person has a public key known to the whole world and a secret key known only to himself.

When Alice wants to send message x to Bob, she encodes it using his public key.

Bob decrypts it using his secret key, to retrieve x.

Eve is welcome to see as many encrypted messages, but she will not be able to decode them, under certain simple assumptions.

The RSA Cryptosystem

Pick up two primes p and q and let $N = pq$.

For any e relatively prime to $(p-1)(q-1)$:

- The mapping $x \mapsto x^e \pmod N$ is a bijection on $\{0, 1, \ldots N-1\}$.
- The inverse mapping is easily realized: let d be the inverse of e modulo $(p-1)(q-1)$. Then for all $x \in \{0, 1, \ldots, N-1\}$,

$$(x^e)^d \equiv x \pmod N$$

The mapping $x \mapsto x^e \mod N$ is a reasonable way to encode messages x. If Bob publishes (N, e) as his public key, everyone else can use it to send him encrypted messages.

Bob retains the value d as his secret key. He decodes all messages that come to him by the d-th power modulo N.

Proof of the Property

Proof:
If the mapping $x \to x^e \mod N$ is invertible, it must be a bijection; hence statement 2 implies statement 1.

To prove statement 2, e is invertible modulo $(p-1)(q-1)$ because they are relatively prime.

To show $(x^e)^d \equiv x \mod N$: Since $ed \equiv 1 \mod (p-1)(q-1)$, $ed = 1 + k(p-1)(q-1)$ for some k.

Then

$$(x^e)^d - x = x^{ed} - x = x^{1+k(p-1)(q-1)} - x$$

$x^{1+k(p-1)(q-1)} - x$ is divisible by p (since $x^{p-1} \equiv 1 \mod p$) and likewise by q. Since p and q are primes, it is divisible by $N = pq$.

RSA protocols

Bob chooses his public and secret keys:

- He starts by picking two large (n-bit) random primes p and q.
- His public key is (N, e) where $N = pq$ and e is a $2n$-bit number relatively prime to $(p-1)(q-1)$.
- his secret key is d, the inverse of e modulo $(p-1)(q-1)$.

Alice wishes to send message x to Bob:

- She looks up his public key (N, e) and sends him $y = (x^e \mod N)$.
- He decodes the message by computing $y^d \mod N$.

Security Assumption of RSA

The security of RSA hinges upon a simple assumption:
Given N, e and $y = x^e \pmod{N}$, it is computationally intractable to determine x.

How might Eve try to guess x?
She could experiment with all possible values of x, each time checking whether $x^e \equiv y \pmod{N}$, but this would take exponential time.

How might Eve try to guess x?
she could try to factor N to retrieve p and q, and then figure out d by inverting e modulo $(p-1)(q-1)$, but we believe factoring to be hard.

Digital Signature

A digital signature scheme is a mathematical scheme for demonstrating the authenticity of a digital message or document.

In a digital signature scheme, there are two algorithms, signing and verifying.

A signing algorithm that, given a message and a private key, produces a signature.

A signature verifying algorithm that, given a message, public key and a signature, either accepts or rejects the message's claim to authenticity.

Is Communication Safe?

Is a communication safe in the internet when cryptography is unbreakable?

- No!

The NSPK Protocol

$$
\begin{aligned}
A \longrightarrow B: &\quad \{A, N_A\}_{+K_B} \\
B \longrightarrow A: &\quad \{N_A, \mathbb{S}\}_{+K_A} \\
A \longrightarrow B: &\quad \{\mathbb{S}\}_{+K_B}
\end{aligned}
$$

An Attack

$$
\begin{array}{rcll}
A & \longrightarrow & I: & \{A, N_A\}_{+K_I} \\
I(A) & \longrightarrow & B: & \{A, N_A\}_{+K_B} \\
B & \longrightarrow & I(A): & \{N_A, \mathbb{S}\}_{+K_A} \\
I & \longrightarrow & A: & \{N_A, \mathbb{S}\}_{+K_A} \\
A & \longrightarrow & I: & \{\mathbb{S}\}_{+K_I} \\
I(A) & \longrightarrow & B: & \{\mathbb{S}\}_{+K_B}
\end{array}
$$

The Fixed NSPK Protocol

$$\begin{aligned} A \longrightarrow B: &\quad \{A, N_A\}_{+K_B} \\ B \longrightarrow A: &\quad \{B, N_A, \mathbb{S}\}_{+K_A} \\ A \longrightarrow B: &\quad \{\mathbb{S}\}_{+K_B} \end{aligned}$$

$$\begin{array}{rcll} A & \longrightarrow & I: & \{A, N_A\}_{+K_I} \\ I(A) & \longrightarrow & B: & \{A, N_A\}_{+K_B} \\ B & \longrightarrow & I(A): & \{B, N_A, \mathbb{S}\}_{+K_A} \\ I & \not\longrightarrow & A: & \{I, N_A, \mathbb{S}\}_{+K_A} \end{array}$$

Exercise

[DPV07] 1.8, 1.20, 1.22, 1.31, 1.34 and 1.35