Algorithms Design II

Algorithms with Numbers I

Guoqiang Li
School of Software, Shanghai Jiao Tong University

SHANGHAI JIAO TONG UNIVERSITY

Two Seemingly Similar Problems

SHANGHAI JIAO TONG UNIVERSITY

在本课件中，默认输入数据都很大，所以其数据位数会对性能产生影响，记其位数n=log_b(N),N为输入的数字，n为输入数字在b进制下的位数。

Factoring: Given a number N, express it as a product of its prime factors.

Primality: Given a number N, determine whether it is a prime.

Two Seemingly Similar Problems

Factoring: Given a number N, express it as a product of its prime factors.

Primality: Given a number N, determine whether it is a prime.

We believe that Factoring is hard and much of the electronic commerce is built on this assumption.

There are efficient algorithms for Primality, e.g., AKS test by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena.

认为因式分解比证明一个数是否是素数要难。

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.

Preliminaries

How to Represent Numbers

SHANGHAI JIAO TONG UNIVERSITY

We are most familiar with decimal representation:

- 1024

How to Represent Numbers

We are most familiar with decimal representation:

- 1024

But computers use binary representation:

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

How to Represent Numbers

We are most familiar with decimal representation:

- 1024

But computers use binary representation:

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

The bigger the base is, the shorter the representation is. But how much do we really gain by choosing large base?

Bases and Logs

Shanghai Jiao Tong University

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

Bases and Logs

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

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

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?

Bases and Logs

Shanghai Jiao Tong University

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?

换底公式 $$\log_b N = \frac{\log_a N}{\log_a b}$$

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 $O(\log N)$

在O()时间表示下，当进制不为1进制时，每次修改只会导致常数项的变化。
NOTE:在进制为1进制时，此时log_1不合法，其就会对时间复杂度产生影响

Roles of $Log\ N$

$log\ N$ is the power to which you need to raise 2 in order to obtain N.

Roles of $Log\ N$

$log\ N$ is the power to which you need to raise 2 in order to obtain N.

It can also be seen as the number of times you must halve N to get down to 1. (More precisely: $\lceil log\ N \rceil$.)

Roles of $Log\ N$

$log\ N$ is the power to which you need to raise 2 in order to obtain N.

It can also be seen as the number of times you must halve N to get down to 1. (More precisely: $\lceil log\ N \rceil$.)

It is the number of bits in the binary representation of N. (More precisely: $\lceil log\ (N+1) \rceil$.)

Roles of $Log\ N$

$log\ N$ is the power to which you need to raise 2 in order to obtain N.

It can also be seen as the number of times you must halve N to get down to 1. (More precisely: $\lceil log\ N \rceil$.)

It is the number of bits in the binary representation of N. (More precisely: $\lceil log\ (N+1) \rceil$.)

It is also the depth of a complete binary tree with N nodes. (More precisely: $\lfloor log\ N \rfloor$.)

Roles of $Log\ N$

此页内容后续在计算时间复杂度时当做已知！

$log\ N$ is the power to which you need to raise 2 in order to obtain N.

It can also be seen as the number of times you must halve N to get down to 1. (More precisely: $\lceil log\ N \rceil$.) 前两种情况等价，实际上就是用二进制表示数字N的bit位数。也可以理解为分治的次数。

It is the number of bits in the binary representation of N. (More precisely: $\lceil log\ (N+1) \rceil$.)

It is also the depth of a complete binary tree with N nodes. (More precisely: $\lfloor log\ N \rfloor$.)

It is even the sum $1 + 1/2 + 1/3 + \ldots + 1/n$, to within a constant factor.

Basics Arithmetic

Addition

Lemma

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

对于lemma的进一步理解：就是说两个b(>=2)进制下的单位数的数字相加之后要么不进位，要么最多进一位。

Addition

Addition

This rule holds not just in decimal but in any $b \geq 2$.

Addition

This rule holds not just in decimal but in any $b \geq 2$.

In binary, the maximum possible sum of three single-bit numbers is 3, which is a 2-bit number.

Addition

SHANGHAI JIAO TONG UNIVERSITY

This rule holds not just in decimal but in any $b \geq 2$.

In binary, the maximum possible sum of three single-bit numbers is 3, which is a 2-bit number.

This simple rule gives us a way to add two numbers in any bases.

Addition

对于非1进制都成立

This rule holds not just in decimal but in any $b \geq 2$.

In binary, the maximum possible sum of three single-bit numbers is 3, which is a 2-bit number.

This simple rule gives us a way to add two numbers in any bases.

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

有前述可以知道，相加之后最多进一位，所以直接使用单个进位寄存器就可以存储是否仅为以及进位的结果。

Addition

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

Addition

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

The answer expressed as a function of the size of the input: the number of bits of x and y (suppose they are n bit long).

Addition

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

The answer expressed as a function of the size of the input: the number of bits of x and y (suppose they are n bit long).

The sum of x and y is $n+1$ bits at most. Each individual bit of this sum gets computed in a fixed amount of time.

单独一个bit的加法是O(1)。

Addition

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

The answer expressed as a function of the size of the input: the number of bits of x and y (suppose they are n bit long).

The sum of x and y is $n+1$ bits at most. Each individual bit of this sum gets computed in a fixed amount of time.

The total running time for the addition is of form $c_0 + c_1 n$, where c_0 and c_1 are some constants, i.e., $O(n)$.

长度为n的两个数字(n足够大)相加的时间复杂度为O(N)，由于最多进一位，所以进位运算一定是常数时间。

Addition

Q: Can we do better?

Addition

Q: Can we do better?

In order to add two n-bit numbers, we must read them and write down the answer, and even that requires n operations.

Addition

Q: Can we do better?

In order to add two n-bit numbers, we must read them and write down the answer, and even that requires n operations.

So the addition algorithm is optimal.

因为读入n位数字的时间是不可忽略的。

Perform Addition in One Step?

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

Perform Addition in One Step?

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

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

Perform Addition in One Step?

对于常规的64位机器，因为一般处理的数字都不会超过64位，所以可以近似认为其加法时间为常数时间O(1)。

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

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

To study the basic algorithms encoded 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.

学习与位数有关的位运算的时间复杂度的意义：
对于计算超大的数字的加法的时间复杂度有意义。
有时候还需要考虑硬件的复杂度问题。

Multiplication

Multiplication

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

$$
\begin{array}{cccccccc|l}
 & & & & 1 & 1 & 0 & 1 & \text{a} \\
 & & & \times & 1 & 0 & 1 & 1 & \text{b} \\
\hline
 & & & & 1 & 1 & 0 & 1 & \\
 & & & 1 & 1 & 0 & 1 & & \\
 & & 0 & 0 & 0 & 0 & & & \\
+ & 1 & 1 & 0 & 1 & & & & \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 &
\end{array}
$$

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. (Q: why?)

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

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. (Q: why?)

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

由前述lemma可知，两个数字相加之后的结果最多进1位。
所以结合右图可以知道，最大的位数应该是n+n-1 + 1
第一个n指的是第一个乘数的位数，第二项(n-1)指的是
第一个乘数左移n-1位置后与乘数1不相交的位数，+1指的是
可能存在的一个进位数。所以最长就是2n bits。
所以每一次加法的平均时间为O(2N)，而一共需要相加
N-1次，而每次乘数1左移消耗的时间为O(1),所以总的时间
复杂度为O(N^2)。

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

$$O(n^2)$$

Quiz

What is the complexity of a number times 2?

O(1),因为只需要在计算机中左移1位就行了。

Multiplication by Al Khwarizmi

- write them next to each other.

$$11 \quad 13$$

Multiplication by Al Khwarizmi

- write them next to each other.
- halve the first number by 2, dropping the $.5$, and double the second number.

11	13
5	26

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.

$$\begin{array}{rr} 11 & 13 \\ 5 & 26 \\ 2 & 52 \\ 1 & 104 \\ \hline \end{array}$$

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.

11	13
5	26
1	104

Multiplication by Al Khwarizmi

SHANGHAI JIAO TONG UNIVERSITY

- 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.

$$\begin{array}{rr} 11 & 13 \\ 5 & 26 \\ \cancel{2} & \cancel{52} \\ 1 & 104 \\ \hline & 143 \end{array}$$

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.

这个算法实际上与39页PPT的附图是做的相同的操作。
记上面的乘数为a，下面的乘数为b，对应于右图，
左面的为数b，右面的为数字a。则有：
右图算法b每除以2，数a每乘以2，实际上就是在做
39页中提到的创建新的一行并将乘数a左移1位作为另外的
加数与上一行做加法的过程。而将乘数b除以2实际上就是
在右移，说明已经处理完一位，因为乘数b没有1位，就会有
一个新的行被创建，所以时间复杂度仍然为O(N^2)。

$$
\begin{array}{rr}
11 & 13 \\
5 & 26 \\
\cancel{2} & \cancel{52} \\
1 & 104 \\
\hline
 & 143
\end{array}
$$

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

而要去除b为偶数的行，实际上就是在对应于乘数b为0的bit位，因为这一步只起到移位作用，而对结果没有影响。

Multiplication á la Françis

MULTIPLY(x, y)
Two n-bit integers x and y, where $y \geq 0$;

if $y = 0$ **then** return 0;
z=MULTIPLY(x, $\lfloor y/2 \rfloor$);
if y *is even* **then**
    return $2z$;
    else return $x + 2z$;
end

TIPS：为何x/2一定是向下取整？因为是直接右移1位，无论最后一位是1还是0.

递归算法时间复杂度分析：
1.递归的深度(次数)
2.每次递归中所作的操作的耗时为多少。

MULTIPLY(x, y)
Two n-bit integers x and y, where $y \geq 0$;

if $y = 0$ **then** return 0;
z=MULTIPLY(x, $\lfloor y/2 \rfloor$);
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?

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.

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:

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.

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 $O(n^2)$.

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. n=log_2(y)

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?

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!

Division

DIVIDE(x, y)

Two n-bit integers x and y, where $y \geq 1$;

if $x = 0$ **then** return($0, 0$);

(q, r)=DIVIDE($\lfloor x/2 \rfloor, y$);

$q = 2 \cdot q$, $r = 2 \cdot r$;

if x *is odd* **then** $r = r + 1$;

if $r \geq y$ **then** $r = r - y$, $q = q + 1$;

return(q, r);

q为商，r为余数

Division


```
DIVIDE(x, y)
Two n-bit integers x and y, where y ≥ 1;
if x = 0 then return(0, 0);
(q, r)=DIVIDE(⌊x/2⌋, y);
q = 2 · q, r = 2 · r;
if x is odd then r = r + 1;
if r ≥ y then r = r − y, q = q + 1;
return(q, r);
```

Q: How long does the algorithm take?

Division

DIVIDE(x, y)
Two n-bit integers x and y, where $y \geq 1$;

if $x = 0$ **then** return($0, 0$);
(q, r)=DIVIDE($\lfloor x/2 \rfloor, y$);
$q = 2 \cdot q, r = 2 \cdot r$;
if x *is odd* **then** $r = r + 1$;
if $r \geq y$ **then** $r = r - y, q = q + 1$;
return(q, r);

Q: How long does the algorithm take?

- Exercise 1.8!

与乘法类似，也为O(N^2)。其单次递归过程中最主要的耗时取决于r+(-y)可能为O(N)的时间。

Modular Arithmetic

取模运算

What Is Modular

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

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.

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. 由于取模也需要除法操作，所以单次取模也为O(N^2)(N为位数)

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

$$x \equiv y \mod N \quad \Leftrightarrow \quad N \texttt{ divides } (x - y)$$

NOTE:X =(三等号)Y modN指的是x mod N == y mod N.后续其余的连等式同理。

Two Interpretations

1. It limits numbers to a predefined range $\{0, 1, \ldots, N\}$ and wraps around whenever you try to leave this range - like the hand of a clock.

Two Interpretations

SHANGHAI JIAO TONG UNIVERSITY

取模运算再将一个数字约束在一定范围内的方面起到很大作用。

1. It limits numbers to a predefined range $\{0, 1, \dots, N\}$ and wraps around whenever you try to leave this range - like the hand of a clock.
2. Modular arithmetic deals with all the integers, but divides them into N equivalence classes, each of the form $\{i + k \cdot N \mid k \in \mathbb{Z}\}$ for some i between 0 and $N - 1$.

Two's Complement 二进制补码

Modular arithmetic is nicely illustrated in two's complement, the most common format for storing signed integers.

Two's Complement

Modular arithmetic is nicely illustrated in two's complement, the most common format for storing signed integers.

It uses n bits to represent numbers in the range

$$-2^{n-1}, 2^{n-1} - 1$$

and is usually described as follows:

Two's Complement

Modular arithmetic is nicely illustrated in two's complement, the most common format for storing signed integers.

It uses n bits to represent numbers in the range

$$-2^{n-1}, 2^{n-1} - 1$$

and is usually described as follows:

- Positive integers, in the range 0 to $2^{n-1} - 1$, are stored in regular binary and have a leading bit of 0.

Two's Complement

Modular arithmetic is nicely illustrated in two's complement, the most common format for storing signed integers.

It uses n bits to represent numbers in the range

$$-2^{n-1}, 2^{n-1} - 1$$

and is usually described as follows:

- Positive integers, in the range 0 to $2^{n-1} - 1$, are stored in regular binary and have a leading bit of 0.
- Negative integers $-x$, with $1 \leq x \leq 2^{n-1}$, are stored by first constructing x in binary, then flipping all the bits, and finally adding 1. The leading bit in this case is 1.

Two's Complement

0	1	1	1	1	1	1	1	=	127
0	0	0	0	0	0	1	0	=	2
0	0	0	0	0	0	0	1	=	1
0	0	0	0	0	0	0	0	=	0
1	1	1	1	1	1	1	1	=	−1
1	1	1	1	1	1	1	0	=	−2
1	0	0	0	0	0	0	1	=	−127
1	0	0	0	0	0	0	0	=	−128

(from wiki)

Rules

Substitution rules: if $x \equiv x' \mod N$ and $y \equiv y' \mod N$, then

$$x + y \equiv x' + y' \mod N$$
$$xy \equiv x'y' \mod N$$

Rules

Substitution rules: if $x \equiv x' \mod N$ and $y \equiv y' \mod N$, then

$$\begin{aligned} x + y &\equiv x' + y' \mod N \\ xy &\equiv x'y' \mod N \end{aligned}$$

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

Rules

Substitution rules: if $x \equiv x' \mod N$ and $y \equiv y' \mod N$, then

$$\begin{aligned} x + y &\equiv x' + y' \mod N \\ xy &\equiv x'y' \mod N \end{aligned}$$

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

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

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

取x=x'

Modular Addition

Shanghai Jiao Tong University

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

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 $N-1$, we merely need to subtract off N to bring it back into the required range.

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 $N-1$, we merely need to subtract off N to bring it back into the required range.

The overall computation therefore consists of an addition, and possibly a subtraction, of numbers that never exceed $2N$.

Modular Addition 模加

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 $N-1$, we merely need to subtract off N to bring it back into the required range.

The overall computation therefore consists of an addition, and possibly a subtraction, of numbers that never exceed $2N$.

模加越界-->减去一个N

Its running time is $O(n)$, where $n = \lceil \log N \rceil$.

Modular Multiplication

Modular Multiplication

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

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

Modular Multiplication

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

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

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

Modular Multiplication

模乘

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

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

模乘越界，直接除以一个N！

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

Multiplication thus remains a quadratic operation.

Modular Division

Not quite so easy!

Modular Division

Not quite so easy!

In ordinary arithmetic there is just one tricky case - division by zero.

Modular Division

Not quite so easy!

In ordinary arithmetic there is just one tricky case - division by zero.

It turns out that in modular arithmetic there are potentially other such cases as well.

Modular Division

Not quite so easy!

In ordinary arithmetic there is just one tricky case - division by zero.

It turns out that in modular arithmetic there are potentially other such cases as well.

Whenever division is legal, however, it can be managed in cubic time, $O(n^3)$.

Modular Exponentiation

取模的幂运算

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.

Modular Exponentiation

SHANGHAI JIAO TONG UNIVERSITY

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.

The result is some number $\mod N$ and is therefore a few hundred bits long. However, the raw value x^y could be much, much longer.

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.

The result is some number $\mod N$ and is therefore a few hundred bits long. However, the raw value x^y could be much, much longer.

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

To make sure the numbers never grow too large, we need to perform all intermediate computations modulo N.

Modular Exponentiation

To make sure the numbers never grow too large, we need to perform all intermediate computations modulo N.

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

Modular Exponentiation

To make sure the numbers never grow too large, we need to perform all intermediate computations modulo N.

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

The resulting sequence of intermediate products,

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

consists of numbers that are smaller than N, and so the individual multiplications do not take too long.

Modular Exponentiation

SHANGHAI JIAO TONG UNIVERSITY

To make sure the numbers never grow too large, we need to perform all intermediate computations modulo N.

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

The resulting sequence of intermediate products,

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

consists of numbers that are smaller than N, and so the individual multiplications do not take too long.

But imagine if y is 500 bits long . . .

第一种做法的时间复杂度：(即每次取模之后再乘1个新的x)：模乘的时间复杂度为O(N^2)，共要乘y次，而y=2^n,所以总的时间复杂度为O(N^2*2^N)，为一个指数级的时间复杂度，当y的位数过大时，无法接受。

Modular Exponentiation

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

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

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 in this case there are only $\log y$ multiplications.

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 in this case there are only $\log y$ multiplications.

To determine $x^y \mod N$, multiply together an appropriate subset of these powers, those corresponding to 1's in the binary representation of y.

Modular Exponentiation

方法2是每次取模之后去平方再次取模。时间复杂度分析：单次取模平方相当于一次模乘，所以耗时为O(N^2)，N为位数，而此时采用了分治策略，所以只需要计算logY = log(2^n) =n(n为位数)就可以得出最终结果，所以此时的时间复杂度为O(n^3).n为输入y的位数。(假定x与y位数相等。)

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

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

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

To determine $x^y \mod N$, multiply together an appropriate subset of these powers, those corresponding to 1's in the binary representation of y.

For instance,

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

对于上述的分析，只适用于y正好可以写成y=2^n的形式，所以对于不满足此形式的输入y，应将其改写为对应形式的数字的组合(根据其二进制表示),如25 = 16 + 8 + 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, $\lfloor y/2 \rfloor$, N);
if y *is even* **then**
  return $z^2 \mod N$;
  else return $x \cdot z^2 \mod N$;
end

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, $\lfloor y/2 \rfloor$, N);
if y *is even* **then**
  `return` $z^2 \mod N$;
  else `return` $x \cdot z^2 \mod N$;
end

Another formulation:

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

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, ⌊y/2⌋, N);
if y is even then
    return z^2 mod N;
    else return x · z^2 mod N;
end
```

The algorithm will halt after at most n recursive calls, and during each call it multiplies n-bit numbers. for a total running time of $O(n^3)$

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 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 $gcd(x, y) = gcd(x \pmod y, y)$.

欧几里得算法

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:

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:　　根据x mod y的定义，若x mod y = r，则x = N * y + r.所以与证明gcd(x-y, y)等价.

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

证明：t = gcd(x, y) <= gcd(x - y, y)
t1 = gcd(x - y, y) <= gcd(x, y),
所以有gcd(x, y) = gcd(x - y, y)

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(`$y, x \mod y$`));`

EUCLID(x, y)

Two integers x and y with $x \geq y$;

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

return(EUCLID($y, x \mod y$));

Lemma

If $a \geq b \geq 0$, then $a \mod b < a/2$

这个lemma给出了上述递归函数的最大递归深度为2*log(n)(n为输入的x的bit位数).

Euclid's Algorithm for Greatest Common Divisor


```
EUCLID(x, y)
Two integers x and y with x ≥ y;
if y = 0 then return x;
return(EUCLID(y, x mod y));
```

Lemma

If $a \geq b \geq 0$, *then* $a \mod b < a/2$

Proof:

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(`$y, x \mod y$`))`;

Lemma

If $a \geq b \geq 0$, *then* $a \mod b < a/2$

Proof:

- if $b \leq a/2$, $a \mod b < b \leq a/2$;

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$(y, x \mod y)$); 注意顺序!

Lemma

If $a \geq b \geq 0$, then $a \mod b < a/2$

Proof:

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

  当b>a/2时，a最多只能装下一个b了，所以a mod b = a - b < a / 2.

Euclid's Algorithm for Greatest Common Divisor

SHANGHAI JIAO TONG UNIVERSITY

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

if $y = 0$ **then** return x;
return(EUCLID(y, $x \mod y$));

Lemma

If $a \geq b \geq 0$, then $a \mod b < a/2$

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.

因为连续两轮之后x和y都被去了一次模，所以最大变为原来的一半，即右移1位。

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(`$y, x \mod y$`));`

Lemma

If $a \geq b \geq 0$, then $a \mod b < a/2$

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 $O(n^3)$.

需要做一次除法来获得余数。

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?

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...

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 $d = gcd(x, y)$.

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 $d = gcd(x, y)$.

由于d一定同时小于x和y，所以a和b一定一正一负。

Proof:

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;

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; 因为d是一个公约数，其一定小于最大公约数

$d \geq gcd(x, y)$, 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 ≥ b ≥ 0;
if b = 0 then return(1, 0, a);
(x′, y′, d)=EXTENDED-EUCLID(b, a ( mod b));
return(y′, x′ − ⌊a/b⌋y′, d);
```

An Extension of Euclid's Algorithm

return的三个数值分别为d=a*x+b*y中的a，b和d。

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$(y', x' - \lfloor a/b \rfloor y', d)$;

如此返回的原因：a mod b<=> a - (a / b) * b,之后按照d=a*x+b*y展开即可得到

Correctness of the algorithm?
DIY!

时间复杂度仍然为O(n^2),因为只是额外做了减法操作，而减法
为O(n)，在O条件下舍弃。

Modular Inverse（逆元）

We say x is the multiplicative inverse of $a \mod N$ if

$$ax \equiv 1 \mod N$$

Modular Inverse

We say x is the multiplicative inverse of $a \mod N$ if

$$ax \equiv 1 \mod N$$

此处的**逆元**概念与数学上面的倒数不同，要求这里的逆元x一定要是整数！

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

Modular Inverse

We say x is the multiplicative inverse of $a \mod N$ if

$$ax \equiv 1 \mod N$$

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

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

Modular Inverse

Lemma

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

Lemma

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

Proof:

Modular Inverse

Lemma

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

Proof:

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

此处是在证明lemma的逆否命题:若 ax=1 mod N,则有gcd(a, N) = 1.
(gcd(a, b)=1说明整数a与b互质/互素)

Lemma

If $gcd(a, N) > 1$*, then* $ax \neq 1 \mod N$*.*

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.

这里要注意，在获取ax+Ny=1的同时，通过扩展的欧几里得算法，我们就可以拿到对应的x，也就是a mod N的逆元了

Modular Division

SHANGHAI JIAO TONG UNIVERSITY

Theorem (Modular Division Theorem)

- *For any $a \mod N$, a has a multiplicative inverse modulo N if and only if it is relatively prime to N.* （互素，互质）

Modular Division

Theorem (Modular Division Theorem)

- *For any $a \mod N$, a has a multiplicative inverse modulo N if and only if it is relatively prime to N.*
- *When the 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, can divide by numbers relatively prime to N. And to actually carry out the division, multiply by the inverse.

a / b mod N <==> a * b ^-1 mod N,前提是b与N互质，才能保证一定有结果

b*b_-1 + Ny = 1,ab_-1 mod N = a / b * (1 - Ny) mod N = (a/b) mod N(在a/b为整数情况下)