Number Theoretic Algorithms

Introduction, Strassen's Multiplication

Importance of Number Theory

- Before the dawn of computers, many viewed number theory as last bastion of "pure math" which could not be useful and must be enjoyed only for its aesthetic beauty.
- No longer the case. Number theory is crucial for encryption algorithms.

Divisors

- DEF: Let a, b and c be integers such that
  - $a = b \cdot c$.
- Then b and c are said to ***divide*** (or are ***factors***) of a, while a is said to be a ***multiple*** of b (as well as of c). The pipe symbol "|" denotes "divides" so the situation is summarized by:
  - $b \mid a \wedge c \mid a$.

Divisors. Examples

Q: Which of the following is true?

1. $77 \mid 7$
2. $7 \mid 77$
3. $24 \mid 24$
4. $0 \mid 24$
5. $24 \mid 0$

- If $a>0$ & $d|a$ then $|d|<=|a|$
- a is a multiple of d
- $d|a$ & $d>=0$, d is a divisor of a, d should be atleast 1, but not greater than a
- Trivial divisors of a is 1 & a
- Non trivial divisors of a is/are factors of a

Prime Numbers

DEF: A number $n \geq 2$ ***prime*** if it is only divisible by 1 and itself. A number $n \geq 2$ which isn't prime is called ***composite***.

Q: Which of the following are prime?

0,1,2,3,4,5,6,7,8,9,10

Prime Numbers

- A:
  - 0, and 1 not prime since not positive and greater or equal to 2
  - 2 is prime as 1 and 2 are only factors
  - 3 is prime as 1 and 3 are only factors.
  - 4,6,8,10 not prime as *non-trivially* divisible by 2.
  - 5, 7 prime.
  - $9 = 3 \cdot 3$ not prime.

Last example shows that not all odd numbers are prime.

Fundamental Theorem of Arithmetic

- THEORM: Any number $n \geq 2$ is expressible as as a unique product of 1 or more prime numbers.
- Note: prime numbers are considered to be "products" of 1 prime.

Q: Express each of the following number as a product of primes: 22, 100, 12, 17

Division

Remember long division?

d the ***divisor***

q the ***quotient***

$$\begin{array}{r} 3 \\ 31\overline{)117} \\ \underline{93} \\ 24 \end{array}$$

a the ***dividend***

r the ***remainder***

$117 = 31 \cdot 3 + 24$

$a = dq + r$

Division

- THM: Let a be an integer, and d be a positive integer. There are unique integers q, r with $r \in \{0,1,2,\ldots,d-1\}$ satisfying
  - $a = dq + r$
- The proof is a simple application of long-division. The theorem is called the ***division algorithm***
- a/d ,iff a mod d=0

Greatest Common Divisor
Relatively Prime

- DEF: Let a,b be integers, not both zero. The ***greatest common divisor*** of a and b (or gcd(a,b)) is the biggest number d which divides both a and b.
- DEF: a and b are said to be ***relatively prime*** if gcd(a,b) = 1, so no prime common divisors.

Greatest Common Divisor Relatively Prime

Q: Find the following gcd's:

1. gcd(11,77)
2. gcd(33,77)
3. gcd(24,36)
4. gcd(24,25)

Greatest Common Divisor
Relatively Prime

A:

1. gcd(11,77) = 11
2. gcd(33,77) = 11
3. gcd(24,36) = 12
4. gcd(24,25) = 1. Therefore 24 and 25 are relatively prime.

NOTE: A prime number are relatively prime to all other numbers which it doesn't divide.

Greatest Common Divisor
Relatively Prime

- EG: Find gcd(98,420).
- Find prime decomposition of each number and find all the common factors:

  $98 = 2 \cdot 49 = 2 \cdot 7 \cdot 7$

  $420 = 2 \cdot 210 = 2 \cdot 2 \cdot 105 = 2 \cdot 2 \cdot 3 \cdot 35 = 2 \cdot 2 \cdot 3 \cdot 5 \cdot 7$
- Underline common factors: $\underline{2 \cdot 7} \cdot 7$, $2 \cdot \underline{2} \cdot 3 \cdot 5 \cdot \underline{7}$
- Therefore, gcd(98,420) = 14

Euclid's Algorithm for calculating gcd

```
EUCLID(a, b)
if b = 0
    then return a
    else  return EUCLID(b, a mod b)
```

Euclidean Algorithm. Example

gcd(33,77):

Step	$r = x \textbf{ mod } y$	x	y
0	–	33	77

Euclidean Algorithm. Example

gcd(33,77):

Step	$r = x$ **mod** y	x	y
0	–	33	77
1	33 **mod** 77 = 33	77	33

Euclidean Algorithm. Example

gcd(33,77):

Step	$r = x \textbf{ mod } y$	x	y
0	–	33	77
1	33 **mod** 77 = 33	77	33
2	77 **mod** 33 = 11	33	11

Euclidean Algorithm. Example

gcd(33,77):

Step	$r = x$ **mod** y	x	y
0	–	33	77
1	33 **mod** 77 = 33	77	33
2	77 **mod** 33 = 11	33	11
3	33 **mod** 11 = 0	11	0

Euclidean Algorithm. Example

gcd(244,117):

Step	$r = x$ **mod** y	x	y
0	–	244	117

Euclidean Algorithm. Example

gcd(244,117):

Step	$r = x$ **mod** y	x	y
0	–	244	117
1	244 **mod** 117 = 10	117	10

Euclidean Algorithm. Example

gcd(244,117):

Step	$r = x$ **mod** y	x	y
0	–	244	117
1	244 **mod** 117 = 10	117	10
2	117 **mod** 10 = 7	10	7

Euclidean Algorithm. Example

gcd(244,117):

Step	$r = x$ **mod** y	x	y
0	–	244	117
1	244 **mod** 117 = 10	117	10
2	117 **mod** 10 = 7	10	7
3	10 **mod** 7 = 3	7	3

Euclidean Algorithm. Example

gcd(244,117):

Step	$r = x$ **mod** y	x	y
0	–	244	117
1	244 **mod** 117 = 10	117	10
2	117 **mod** 10 = 7	10	7
3	10 **mod** 7 = 3	7	3
4	7 **mod** 3 = 1	3	1

Euclidean Algorithm. Example

Running time is proportional to the no. of recursive calls

gcd(244,117):

Step	$r = x \textbf{ mod } y$	x	y
0	–	244	117
1	244 **mod** 117 = 10	117	10
2	117 **mod** 10 = 7	10	7
3	10 **mod** 7 = 3	7	3
4	7 **mod** 3 = 1	3	1
5	3 **mod** 1=0	1	0

By definition ➔ 244 and 117 are rel. prime.

Least Common Multiple

- DEF: The ***least common multiple*** of a, and b (lcm(a,b)) is the smallest number m which is divisible by both a and b.
- Q: Find the lcm's:

1. lcm(10,100)
2. lcm(7,5)
3. lcm(9,21)

Introduction

- The least common multiple (LCM) of 2 numbers is the smallest number that they both divide evenly into.
- To find the Least Common Multiple of two or more whole numbers, follow this procedure:
  - 1.Make a list of multiples for each whole number.
  - 2.Continue your list until at least two multiples are common to all lists.
  - 3.Identify the common multiples.
  - 4.The Least Common Multiple (LCM) is the smallest of these common multiples.

EXAMPLE

Find the LCM of 12 and 15.

Common multiples of 12 and 15 are 60 and 120

The least common multiple of 12 and 15 is 60.

Solution:LCM = 60

EUCLIDEAN ALGORITHM

- Much more effective way to get the least common multiple of two numbers
- very fast, because it does not require factorization

$$lcm(n_1, n_2) = \frac{n_1 \cdot n_2}{gcd(n_1, n_2)}$$

$$lcm(140, 72) = \frac{140 \cdot 72}{gcd(140, 72)} = \frac{10080}{4} = 2520$$

Modular Arithmetic

There are two types of "mod" (confusing):

- the **mod** function
  - Inputs a number a and a base b
  - Outputs a **mod** b a number between 0 and $b-1$ inclusive
  - This is the remainder of a÷b.
- the (mod) congruence
  - Relates two numbers a, a' to each other relative some base b
  - $a \equiv a'$ (mod b) means that a and a' have the same remainder when dividing by b

Congruences

- Let a and b be integers and m be a positive integer. We say that **a is congruent to b modulo m** if m divides a – b.

- We use the notation **$a \equiv b \pmod{m}$** to indicate that a is congruent to b modulo m.

- In other words:

$a \equiv b \pmod{m}$ if and only if **$a \bmod m = b \bmod m$**.

Primality Testing

- In this section, we consider the problem of finding large primes.
- For many applications, such as cryptography, we need to find large "random" primes.
- The ***prime distribution function*** p(n) specifies the number of primes that are less than or equal to n.
- For example, p(10)=4, since there are 4 prime numbers less than or equal to 10,namely, 2, 3, 5, and 7.

- There is a procedure called pseudoprime to check primality of a number
- This procedure can make errors, but only of one type. That is, if it says that n is composite, then it is always correct. If it says that n is prime, however, then it makes an error only if n is a base-2 pseudo prime.
- Fermat's Theorem: Fermat's theorem implies that if n is prime, then n satisfies equation, $a^{n-1} \equiv 1 \pmod{n}$ for every a in Z+
- We say that n is a base-a pseudo prime if n is composite and $a^{n-1} \equiv 1 \pmod{n}$

```
PSEUDOPRIME(n)
if MODULAR-EXPONENTIATION(2, n − 1, n) ≢ 1 (mod n)
    then return COMPOSITE        ▷ Definitely.
    else return PRIME            ▷ We hope!
```

➢Modular Exponentiation(a,b,n) gives an efficient way to calculate $a^b \bmod n$

- The **Miller-Rabin** primality test overcomes the problems of the simple test PSEUDOPRIME with two modifications:

  1. It tries several randomly chosen base values a instead of just one base value
  2. While computing each modular exponentiation, it looks for a nontrivial square root of 1modulo n,

during the final set of squaring. If it finds one, it stops and returns COMPOSITE

- Miller-Rabin primality test follows. The input n > 2 is the odd number to be tested for primality, and s is the number of randomly chosen base values from $\mathbb{Z}_n^+$ to be tried

- The code uses an auxiliary procedure WITNESS such that WITNESS(a, n) is TRUE if and only if 'a' is a "witness" to the compositeness of n—that is, if it is possible using "a" to prove (in a manner that we shall see) that n is composite.

WITNESS(a, n)

let t and u be such that $t \geq 1$, u is odd, and $n - 1 = 2^t u$
$x_0 =$ MODULAR-EXPONENTIATION(a, u, n)
for $i = 1$ **to** t
    $x_i = x_{i-1}^2 \bmod n$
    if $x_i == 1$ and $x_{i-1} \neq 1$ and $x_{i-1} \neq n - 1$
        return TRUE
if $x_t \neq 1$
    return TRUE
return FALSE

MILLER-RABIN(n, s)

for $j = 1$ **to** s
    $a =$ RANDOM$(1, n - 1)$
    if WITNESS(a, n)
        return COMPOSITE   // definitely
return PRIME   // almost surely

Example

Consider n= 561 which we want to check whether it is prime or not.

a=2 (choose any number in rage of 1< a < n-1)

n-1=2^t * u

inorder to find t and u, we use (n-1/2^I)

for i=1 to n-1

<u>i=1</u>

560/2^1=280

<u>i=2</u>

560/2^2=140

<u>i=3</u>

560/2^3=70

<u>i=4</u>

560/2^4=35 t=4 & u=35

<u>i=5</u>

560/2^5=17.5

x0= MODULAR-EXPONENTIATION(a, u, n);

x0= a^u mod n;

x0= 2^35 mod 561 = 263

- Main loop

  xi = (xi-1)^2 mod n

  for I=1 to t

  <u>i=1</u>

  x1= (263)^2 mod 561

  = 166

  <u>i=2</u>

  x2= (166)^2 mod 561

  = 67

  i=3

  x3= (67)^2 mod 561

  = 1

- Since x3=1 so a=2 is a witness that n=561 is a composite number

Integer Factorization

- Divide composite to primes
- No well defined algorithm yet
- Brute approach, improvised approach

POLLARD'S RHO

Concepts used

- Congruent modulo
- Greatest common divisor GCD
- Floyd's cycle detection

ALGORITHM

1. Start with random x and c. Take y equal to x and $f(x)=x^2+c$.
2. While a divisor isn't obtained

   a) Update x to f(x) modulo n// Tortoise

   b) Update y to f(f(y)) modulo n. // Hare

   c) Calculate GCD of |x-y| and n

   d) If GCD is not unity

      1.1) If GCD is n, repeat from step 2 with another set of x, y and c

      1.2) Else GCD is our answer

EXAMPLE

Let us suppose **n = 187** and consider different cases for different random values.

1. An Example of random values such that algorithm **finds result**:

   $y = x = 2$ and $c = 1$, Hence, our $f(x) = x^2 + 1$.

$x_{i+1} = f(x_i)$	$y_{i+1} = f(f(y_i))$	c	d = GCD(|x-y|, n)
5	26	1	1
26	180	1	**11**

2. An Example of random values such that algorithm **finds result faster**:

   $y = x = 110$ and 'c' = 183. Hence, our $f(x) = x^2 + 183$.

$x_{i+1} = f(x_i)$	$y_{i+1} = f(f(y_i))$	c	d = GCD(|x-y|, n)
128	111	183	**17**

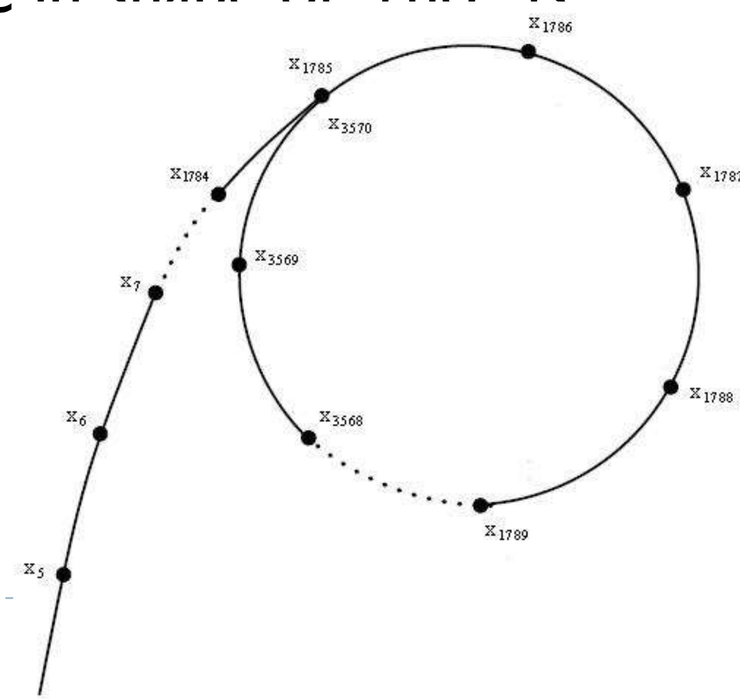
3. An Example of random values such that algorithm **doesn't find result**:

   $x = y = 147$ and $c = 67$. Hence, our $f(x) = x^2 + 67$.

$x_{i+1} = f(x_i)$	$y_{i+1} = f(f(y_i))$	c	d = GCD(|x-y|, n)
32	156	67	1
156	114	67	1
93	48	67	1
114	114	67	**187**

Why called Pollard's "Rho"

- While finding random numbers based on the polynomial $f(x)=x^2+1$ we will get a series of random numbers x1,x2…..At some point a value x_k $f(x_k)$ will be congruent to a previous f(xj). The next set of random numbers will be congruent modulo to numbers starting from xj. So a backward loop is formed , a curve in shape of "rho" is obtained.
- Heuristic



Time Complexity

- The algorithm offers a trade-off between its running time and the probability that it finds a factor. A prime divisor can be achieved with a probability around 0.5, in $O(\sqrt{d}) \le O(n^{1/4})$ iterations.
- This is a heuristic claim.

Strassen’s Multiplication

Basic Matrix Multiplication

Suppose we want to multiply two matrices of size N x N: for example A x B = C.

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$\mathbf{C_{11} = a_{11}b_{11} + a_{12}b_{21}}$

$\mathbf{C_{12} = a_{11}b_{12} + a_{12}b_{22}}$

$\mathbf{C_{21} = a_{21}b_{11} + a_{22}b_{21}}$

$\mathbf{C_{22} = a_{21}b_{12} + a_{22}b_{22}}$

2x2 matrix multiplication can be accomplished in 8 multiplication.$\mathbf{(2^{\log_2 8} = 2^3)}$

Basic Matrix Multiplication

```
void matrix_mult (){
  for (i = 1; i <= N; i++) {
        for (j = 1; j <= N; j++) {
                compute C_{i,j};
                                }
  }}
```

algorithm

Time analysis

$$C_{i,j} = \sum_{k=1}^{N} a_{i,k} b_{k,j}$$

$$\text{Thus } T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} c = cN^3 = O(N^3)$$

STRASSEN'S MATRIX MULTIPLICATION

IN LINEAR ALGEBRA STRESSEN ALGORITHM IS AN ALGORITHM FOR MATRIX MULTIPLI CATION.

ITS NAMED AFTER VOLKER STRASSEN

ITS FASTER THAN STANDARD MATRIX MULTIPLICATION AND USEFUL IN PRACTISE FOR LARGE MATRICES.

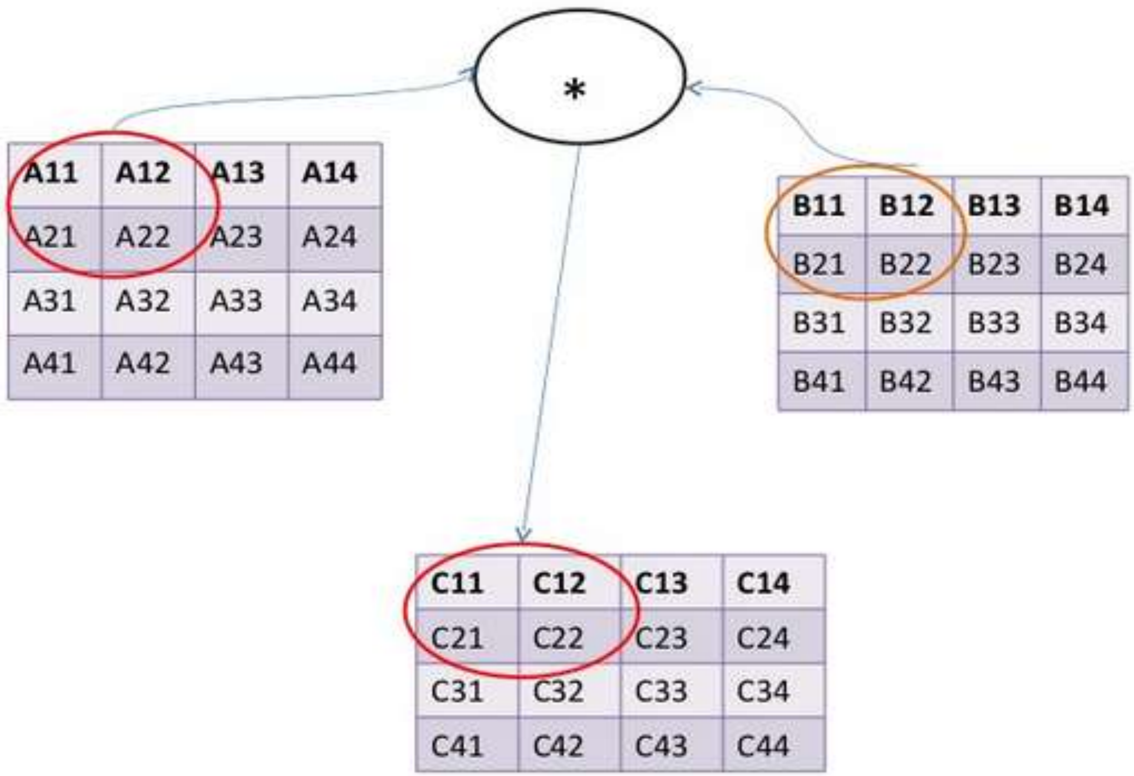
Strassens’s Matrix Multiplication

- Strassen showed that 2x2 matrix multiplication can be accomplished in 7 multiplication and 18 additions or subtractions. **.($2^{\log_2 7} = 2^{2.807}$)**

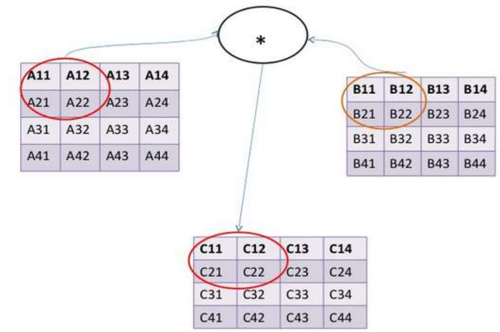
- This reduce can be done by Divide and Conquer Approach.

Using Divide And Conquer Approach

Consider Multiplication of 4x4 matrix:



Consider Multiplication of 4x4 matrix:



Strassens's Matrix Multiplication

$$\begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix}$$

$P_1 = (A_{11}+ A_{22})(B_{11}+B_{22})$
$P_2 = (A_{21} + A_{22}) * B_{11}$
$P_3 = A_{11} * (B_{12} - B_{22})$
$P_4 = A_{22} * (B_{21} - B_{11})$
$P_5 = (A_{11} + A_{12}) * B_{22}$
$P_6 = (A_{21} - A_{11}) * (B_{11} + B_{12})$
$P_7 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$C_{11} = P_1 + P_4 - P_5 + P_7$
$C_{12} = P_3 + P_5$
$C_{21} = P_2 + P_4$
$C_{22} = P_1 + P_3 - P_2 + P_6$

Comparison

$$
\begin{aligned}
C_{11} &= P_1 + P_4 - P_5 + P_7 \\
&= (A_{11}+ A_{22})(B_{11}+B_{22}) + A_{22} * (B_{21} - B_{11}) - (A_{11} + A_{12}) * B_{22}+ \\
&\quad (A_{12} - A_{22}) * (B_{21} + B_{22}) \\
&= A_{11} B_{11} + A_{11} B_{22} + A_{22} B_{11} + A_{22} B_{22} + A_{22} B_{21} – A_{22} B_{11} - \\
&\quad A_{11} B_{22} -A_{12} B_{22} + A_{12} B_{21} + A_{12} B_{22} – A_{22} B_{21} – A_{22} B_{22} \\
&= A_{11} B_{11} + A_{12} B_{21}
\end{aligned}
$$

Time Analysis

$$T(1) = 1 \qquad\qquad (\text{assume } N = 2^k)$$

$$T(N) = 7T(N/2)$$

$$T(N) = 7^k T(N/2^k) = 7^k$$

$$T(N) = 7^{\log N} = N^{\log 7} = N^{2.81}$$

In similar way we can calculate all the multiplications.

And same steps we can perform on any dimension of matrix.

Results :

Input Size(n)	No. of multiplication In sequential algo.	No. of multiplication In the implemented algo.	No. of multiplication In Strassen's
2	8	7	7
4	64	56	49
8	512	448	343
16	4096	3584	2401
. .	. .	. .	. .
p	p^3	$(7/8)*p^3$	$n^{2.807}$