Unit 7:

Number Theoretic Algorithms

7.1. Number Theoretic Notations, Euclid’s algorithm and Extended Euclid’s Algorithm and Analysis.

7.2. Solving Modular Linear Equations, Chinese Remainder Theorem, Miller-Rabin Randomized Primality Test Analysis

Prepared by: Shiv Raj Pant

Introduction:

- Number theory was once viewed as a beautiful but largely useless subject in pure mathematics.
- But today algorithms related to number theory are used widely e.g. in cryptography, the theory of prime numbers is used.

Number Theoretic concepts and Notations

Divisibility and Divisors: The notation a|b, read as "*a divides b*", means that $b = ka$ for some integer k. we say that b is multiple of a.

Every integer divides 0. If $a|b$ and $a \geq 0$, we say that a is divisor of b. A divisor of non-zero integer b is at least 1 but not greater than $|a|$.

Prime and composite numbers:

The division theorem : For any integer a and any positive integer n, there exist unique integers q and r such that

$$a = qn + r \ , 0 \leq r < n$$

The value $q = \left\lfloor \frac{a}{n} \right\rfloor$ is the quotient.

The value $r = a \bmod n$ is the remainder.

We have $n|a$ iff $a \bmod n = 0$

Common divisors and GCD:

Relatively prime integers: Two integers a and b are relatively prime if their only common divisor is 1. i.e. if gcd$(a, b) = 1$

e.g. 8 and 15 are relatively prime.

Unique factorization: For all primes p and all integers a and b, if $p|ab$, then $p|a$ or $p|b$ (or both).

Euclid's algorithm

- Believed to be one of the world's oldest algorithms (300 B.C), Euclid's algorithm is an efficient algorithm for computing the GCD of two integers.

Algorithm

```
Euclid(a, b)
 {
  if(b ==0)
    return a
  else
    return Euclid(b, a mod b)
 }
```

Example: compute gcd(30,21)

```
euclid(30,21) = euclid(21,9)
              = euclid(9,3)
              = euclid(3,0)
              = 3
```

Analysis:

- The overall running time of the algorithm is proportional to the recursive calls it makes.
- It has been shown that Euclid's algorithm makes maximum recursive calls when a and b are consecutive Fibonacci numbers.
- Let $a > b \geq 1$ and F_k be the k^{th} Fibonacci number. Then, Euclid(F_{k+1},F_k) makes exactly k-1 recursive calls.

Lame's theorem: *"For any integer $k \geq 1$, if $a > b \geq 1$ and $b < F_{k+1}$, then the Euclid's algorithm makes fewer than k recursive calls."*

- Thus there can be at most k recursive calls when $b \leq F_k$

Now, k^{th} Fibonacci number is approximately

$F_k \approx \frac{\phi^k}{\sqrt{5}}$ Where, ϕ is the golden ratio

$\Rightarrow b \leq \frac{\phi^k}{\sqrt{5}} \Rightarrow k \boxed{= O(\log b)}$

$$\phi = \frac{1+\sqrt{5}}{2}$$

Extended Euclid’s algorithm

- Euclid’s algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid’s algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)   //returns values(d,x,y)
{
 if b=0
  return (a,1,0)
 else
 {
  (d′, x′, y′) = Ext-Euclid(b, a mod b)
  (d, x, y) = (d′,y′, x′-⌊a/b⌋y′)
  return (d, x, y)
 }
}
```

Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d′, x′, y′) = Ext-Euclid(b, a mod b)
  (d, x, y) = (d′, y′, x′ − ⌊a/b⌋ y′)
  return (d, x, y)
 }
}
```

Example: gcd(99,78)

a	b	$\left\lfloor \frac{a}{b} \right\rfloor$	d	x	y
99	78				
78	21				

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```


Example: gcd(99,78)

a	b	$\left\lfloor \frac{a}{b} \right\rfloor$	d	x	y
99	78				
78	21				
21	15				

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```


Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21				
21	15				
15	6				

Extended Euclid’s algorithm

- Euclid’s algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid’s algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d′, x′, y′) = Ext-Euclid(b, a mod b)
  (d, x, y) = (d′, y′, x′ − ⌊a/b⌋ y′)
  return (d, x, y)
 }
}
```

Example: gcd(99,78)

a	b	$\left\lfloor \frac{a}{b} \right\rfloor$	d	x	y
99	78				
78	21				
21	15				
15	6				
6	3				

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```


Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21				
21	15				
15	6				
6	3				
3	0				

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

Ext-Euclid(a, b)
{
  if b=0
    return (a,1,0)
  else
  {
    $(d', x', y') =$ Ext-Euclid(b, $a \bmod b$)
    $(d, x, y) = (d', y', x' - \lfloor \frac{a}{b} \rfloor y')$
    return (d, x, y)
  }
}

Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21				
21	15				
15	6				
6	3				
3	0	--	3	1	0

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```

Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21				
21	15				
15	6				
6	3	2	3	0	1
3	0	--	3	1	0

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```


Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21				
21	15				
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	--	3	1	0

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d′, x′, y′) = Ext-Euclid(b, a mod b)
  (d, x, y) = (d′,y′, x′-⌊a/b⌋y′)
  return (d, x, y)
 }
}
```


Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21				
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	--	3	1	0

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```

Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78				
78	21	3	3	3	11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	--	3	1	0

Extended Euclid's algorithm

- Euclid's algorithm can be extended to compute additional useful information.
- Specifically, we use an extended form of Euclid's algorithm to compute integer coefficients x and y such that

$$d = \gcd(a, b) = ax + by$$

- The coefficients x and y are very useful in several places (e.g. for computing modular multiplicative inverse).

Algorithm

```
Ext-Euclid(a, b)
{
 if b=0
  return (a,1,0)
 else
 {
  (d', x', y') = Ext-Euclid(b, a mod b)
  (d, x, y) = (d',y', x'-⌊a/b⌋y')
  return (d, x, y)
 }
}
```

Example: gcd(99,78)

a	b	$\lfloor \frac{a}{b} \rfloor$	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	--	3	1	0

Hence, d = 3, x = -11, y = 14

3 = gcd(99,78) = 99*(-11) + (78*14)

Analysis:

The running time is same as Euclid's algorithm

Solving modular linear equation

- The expression,

  $a \equiv b \pmod{n}$

  read as "*a is congruent to b modulo n*",

  means: the a and b have the same remainder when divided by n (or equivalently a – b is divisible by n). Here, $n>0$ and it is called the modulus.

- The equation -

  $ax \equiv b \pmod{n}$

  is called modular linear equation.

- The problem of solving MLE is to find all values of ***x*, *modulo n***, that satisfy the above relation.

Algorithm

```
Modular-Linear-Eqn-Solver(a, b, n)
 {
  (d, x', y') = Ext-Euclid(a, n)
  if d| b
    {
      x_o = x'(b/d) mod n
      for i = 0 to d-1
        print (x_0+i(n/d)) mod n
    }
  else
    print "no solution"
 }
```

Algorithm

Modular-Linear-Eqn-Solver(a, b, n)

{

(d, x', y') = Ext-Euclid(a, n)

if $d|b$

{

$x_o = x'(b/d) \bmod n$

for i = 0 to d-1

print $(x_0+i(n/d)) \bmod n$

}

else

print "no solution"

}

Example:

$14x \equiv 30 \pmod{100}$

Solution:

Here a = 14, b = 30, n = 100

First the algorithm computes (d, x', y')

a	n	$\lfloor \frac{a}{n} \rfloor$	d	x'	y'
14	100				
100	14				
14	2				
2	0	--	2	1	0

So, (d, x', y') = (2,-7,1)

Now 2|30

$x_o = x'(b/d) \bmod n$ = -7*(30/2) mod n = 95

i=0, First solution = $(x_0+i(n/d)) \bmod n$ = 95

i=1, Second sol = $(x_0+i(n/d)) \bmod n$ = 45

Chinese Remainder theorem

- Around 100 A.D., Chinese mathematician Sun-Tsu solved the problem of finding those integers x that leave remainders 2,3 and 2 when divided by 3,5 and 7 respectively.
- One such solution is x = 23. All solutions are of the form 23+105k for any k.
- The CRT provides a correspondence between a system of equations modulo a set of pairwise relatively prime moduli(e.g. 3,5,7) and an equation modulo their product(e.g. 105)

If $m_1, m_2, m_3, .. m_k$ are pairwise relatively prime positive integers and if $a_1, a_2, .. a_k$ are any integers then, the simultaneous congruences-

$x \equiv a_1 \pmod{m_1}$

$x \equiv a_2 \pmod{m_2}$

..

..

$x \equiv a_k \pmod{m_k}$

have a solution and the solution is unique, modulo m, where $m = m_1 m_2 m_3 .. m_k$

Example:

$x \equiv 2 \pmod 5$

$x \equiv 3 \pmod{13}$

Here, $a_1 = 2$, $m_1 = 5$, $a_2 = 3$, $m_2 = 13$, m =65

Step 1: Compute:

$z_1 = m/m_1 = m_2 = 13$

$z_2 = m/m_2 = m_1 = 5$

Step 2: compute:

$y_1 = z_1^{-1} \pmod{m_1}$

$= 13^{-1} \pmod 5$

$= 2$

$y_2 = z_2^{-1} \pmod{m_2}$

$= 5^{-1} \pmod{13}$

$= 8$

$a^{-1}=b$ means: $ab \equiv 1 \pmod n$

Step 3:

$w_1 = y_1 z_1 \pmod m = 26 \pmod{65}$

$w_2 = y_2 z_2 \pmod m = 40 \pmod{65}$

Step 4:

The solution is

$x = a_1 w_1 + a_2 w_2$

$= (2*26 + 3*40) \pmod{65}$

$= 42 \pmod{65}$

Algorithm

```
chineseRemainder(a, m) // a, m are arrays
 {
  int z[],y[],w[]
  m = 1
  x = 0

  for(i=0; i<n; i++)
    M = m * m_i

  for(i=0; i<n; i++)
    z_i = M/m_i

  for(i=0; i<n; i++)
    y_i = z_i^{-1} (mod m_i)

  for(i=0; i<n; i++)
    w_i = y_i z_i (mod M)

  for(i=0; i<n; i++)
    x = x + a_i w_i

  return x
 }
```

Primality testing

- Primality testing is the problem of finding whether a given number is prime or not.
- This problem arises in several applications e.g. In RSA cryptography, we need to find two large prime numbers during the key generation.

Naïve approach: Trial division

- Try dividing n by each integer $2, 3, .. \sqrt{n}$
- n is prime if none of above numbers divide n.

```
testPrime(n)
 {
  for i = 2 to √n
    if(i divides n)
      return "composite"
  return "prime"
 }
```

- This approach is accurate but inefficient.

Miller-Rabin approach

- Gary L. Miller discovered in 1976. Later, Michael O. Rabin modified it in 1980.
- It is a randomized algorithm.
- The basic idea behind MR approach is:

1. *Choose some random numbers between 1 and n-1*
2. *Perform the "witness" check to decide the primality of n with each of the random numbers chosen in step 1.*

- Since the algorithm is randomized and chooses only a few numbers to check the primality, it does not guarantee correctness.

```
Miller-Rabin(n,s)
 {
   for j=1 to s
     {
       a = RANDOM(1, n-1)
       if(witness(a, n))
          return "composite"       //guaranteed
     }
   return "prime" //not 100% sure, but almost sure
 }

witness(a, n)
{
  Let t≥1, u is odd and n-1 = 2^t u
  x_0 = a^u mod n
  for i = 1 to t
   {
    x_i = x_{i-1}^2 mod n
    if(x_i = 1, and x_{i-1} ≠ 1 and  x_{i-1} ≠ n-1)
      return TRUE
   }
  if(x_t ≠ 1)
    return TRUE
  return FALSE
}
```

Example: Determine whether n = 561 is prime.

Here, n = 561

$n-1 = 560 = 2^4 * 35$

So $t = 4$ and $u = 35$

Let us choose a = 7 (randomly)

Then $x_0 \equiv a^u \bmod n \equiv 241 \pmod{561}$

$x_1 = 298$
$x_2 = 166$
$x_3 = 67$
$x_4 = 1$

The "witness" returns TRUE.

Finally, the algorithm returns "Composite".

End of unit 7

Thank You!