01073301

Design and Analysis of Algorithm

อ. คณัฐ ตังติสานนท์

ktkanut@kmitl.ac.th

Randomized Algorithm

• A randomized algorithm is one where some of the choices made in execution are determined by a random number generator.

• Initially used primarily ***n*** number theory (eg prime testing) but now more generally used & getting quite popular

•Tend to be much faster and/or simple than traditional deterministic algorithms

•a **deterministic algorithm** is an algorithm which, in informal terms, behaves predictably. Given a particular input, it will always produce the same output, and the underlying machine will always pass through the same sequence of states. Deterministic algorithms are by far the most studied and familiar kind of algorithm, as well as one of the most practical, since they can be run on real machines efficiently.

Randomized Algorithm

- Two types:
  - Las Vegas algorithms: never returns an incorrect answer, but running time for a given input can vary. Expected running time is often fast, but worst case can be slower even non-terminating. (ie. Randomize quick sort)
  - Monte Carlo algorithms: may return an incorrect answer. Often has fixed running time for a given input.

Randomized Algorithm

```
Sorting(A)
    while A is not sorted.
        pick randomly 2 distinct numbers i, j
            between 1 and length(A)
        swap(A[i],A[j])
    return A
Sorting(A,k)
    repeat k times
        pick randomly 2 distinct numbers i, j
            between 1 and length(A)
        swap(A[i],A[j])
        if A is sorted
            return A
    return A
```

Failure, Efficiency

•We say a randomize algorithm is efficient if worst case running time is bounded by some polynomial function of input size / expected running time is bounded by some polynomial function of input size

• We say a randomize algorithm has failed if it does not give a correct answer / it does not terminate in a specified amount of time

• Although randomize algorithms have a non-zero chance of failure their power comes from the fact that we can repeat the algorithm multiple times at low cost (since the algorithm tends to be simple and quick) and rapidly reduce the chance of failure to as small a value as we desire.

Assessing the chance of failure

• Average case analysis for randomized algorithms makes more sense than it does for deterministic algorithms – as the performance of the algorithm tends to depend on the random choices make rather than the particular input values. We know far more about the random number generator than we do about the distribution of inputs. In particular we can make assertions about the random number generator but it's rare we can make assertions about the function inputs.

Prime number (จำนวนเฉพาะ)

จำนวนเฉพาะ คือ จำนวนธรรมชาติ (0,1,2,3,…) ที่มีตัวหารลงตัวที่แตกต่างกันอยู่สองจำนวน คือ หนึ่งและตัวมันเอง (trivial divisors)

ตัวอย่าง

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 ,47, 53, 59, 61, 67, 73, 79, 83, 97,101, 103, 107, 109, 113, 127, 131, 139

Prime Testing (Factoring)

```
Prime-test(n)
1  is_prime = true
2  for a ← 2 to n/2
3     if (n mod a == 0)
4          is_prime = false
5          break
```

Running Time : ?

Let n = 12,345,678,901,234,567 (17 digits)

Assume that 1 uSec per loop

Using time 12,345,678,901.234567 Seconds.

= 3,429,355.25 Hours = 142,889 Days = 391.47 Years

$n = 2^{32582657} - 1$ (9,808,358 digits) Prime = Yes?

Prime Testing

•Factoring numbers is a hard thing to do. Noone has been able to think of efficient way.

•Testing if a number is prime (without actual finding the factors) on the other hand isn't too bad. A fast way is using a randomized algorithm. This was one of the first use of randomized algorithms

•We make use of Fermat's Little Theorem (called FLT here):

If ***n*** is prime then $a^{n-1} \bmod n = 1$ for any integer $a < n$

•Our algorithm is to select an ***a*** at random and compute $a^{n-1} \bmod n$. If the answer is not 1 then n must be composite. Otherwise we are unsure so we repeat. After repeating some large number of times and getting 1 each time we guess n must be a prime.

•The problem is some non-primes have few values of ***a*** which are witnesses to their primality (eg carmichael numbers)

Primality Test (FLT)

```
Probably_prime(n)
1 repeat k times:
2     pick a randomly in the range [1, n − 1]
3     if a^(n−1) mod n ≠ 1
4         then return composite
5 return probably_prime
```

Composite : จำนวนที่สามารถแยกตัวประกอบได้ เช่น $30 = 2 * 3 * 5$

Greatest common divisor

```
Euclid(a,b)
1  if b = 0
2      then return a
3  else return Euclid(b, a mod b)

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

The extended form of Euclid's algorithm

```
d = gcd(a,b) = ax +by
 x,y = negative, zero or positive
```

EXTENDED-EUCLID(a, b)

1 **if** $b = 0$
2 **then return** $(a, 1, 0)$
3 $(d', x', y') \leftarrow$ EXTENDED-EUCLID$(b, a \bmod b)$
4 $(d, x, y) \leftarrow (d', y', x' - \lfloor a/b \rfloor y')$
5 **return** (d, x, y)

The extended form of Euclid's algorithm

a	b	$\lfloor a/b \rfloor$	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

```
Gcd(99, 78)

        d  = ax + by

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

          = 78*3 + 21*(-11)
```

Solving modular linear equation

$ax \equiv b \pmod{n}$, $n > 0$

a, b, n are given , find x $ax \bmod n = b \bmod n$

There may be zero, one, or more than one such solution.

```
Modular-Linear-Equaltion-Solver(a,b,n)
1  (d,x',y') ← Extended-Euclid(a,n)
2  if d | b        /* d เป็นตัวประกอบของ b */
3       then x0 ← x'(b/d) (mod n)
4            for i ← 0 to d-1
5                 do print(x0 + i(n/d)) (mod n)
6        else print "no solution"
```

Solving modular linear equation

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

$a=14,\ b=30,\ n=100$

Line 1: $(d,x,y) = (2,-7,1)$

Line 2: $2 \mid 30$ is true → line 3-5 are executed.

Line 3: $x_0 = (-7)(15) \pmod{100}$

$= -105-(100*-2)$

$= 95$

Line 5: $x_1 = x_0 + i(n/d) \pmod{n}$

$= 95 + 1(100/2) \pmod{100}$

$= 95 + 50 \pmod{100}$

$= 145 \pmod{100}$

$= 145 - (100*1)$

$= 45$

Solving modular linear equation

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

$a=14,\ b=30,\ n=100$

answer = 95, 45

$14*(95) - (100*13) = 1330 - 1300$

$= 30$

$14*(45) - (100*6) = 630 - 600$

$= 30$

Modular-Exponentiation(a,b,n)

Solving: $a^b \bmod n$

```
Modular-Exponentiation(a,b,n)
1  c ← 0
2  d ← 1
3  let <b_k,b_k-1,…, b_0> be the binary of b
4  for i← k downto 0
5       do c ← 2c
6          d ← (d*d) mod n
7          if b_i = 1
8             then c ← c+1
9             d ← (d*a) mod n
10 return d
```

Modular-Exponentiation (a,b,n)

$7^{560} \bmod 561$

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
d	7	49	157	526	160	241	298	166	67	1

$a^{2c} \bmod n = (a^c)^2 \bmod n,$

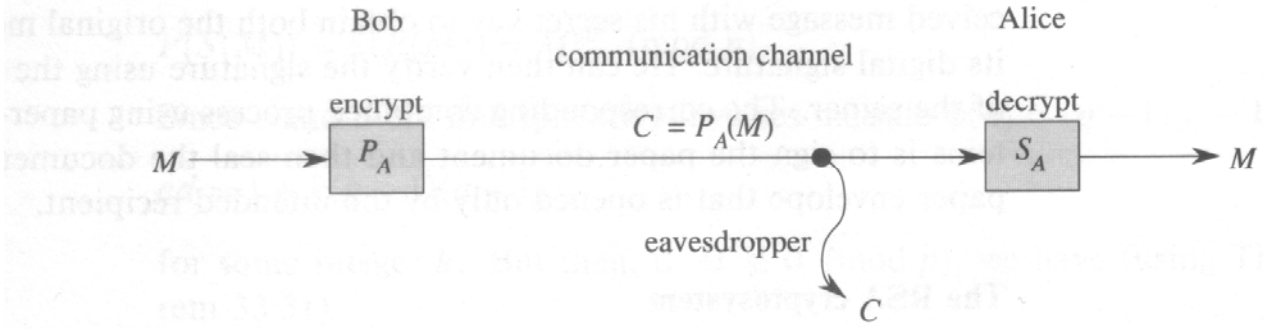
$a^{2c+1} \bmod n = a.(a^c)^2 \bmod n$

$C = (a.b) \bmod n = ((a \bmod n) . (b \bmod n)) \bmod n$

RSA Cryptosystem

•In cryptography, **RSA** is an algorithm for public-key cryptography. It was the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. RSA is widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys and the use of up-to-date implementations.

RSA Cryptosystem



RSA Cryptosystem

RSA involves a public key and a private key. The public key can be known to everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted using the private key. The keys for the RSA algorithm are generated the following way:

1. Choose two distinct large random prime numbers p and q
2. Compute $n = pq$

   n is used as the modulus for both the public and private keys
3. Compute totient $\phi(n) = (p-1)(q-1)$
4. Choose a small odd integer e such that $1 < e < \phi(n)$, $\phi(n)$ and e share no factors other than 1 (coprime)

   e is released as the public key exponent
5. Compute d $de \equiv 1 \pmod{\phi(n)}$

   ie: $de = 1 + k\phi(n)$ for some integer k

   d is kept as the private key exponent

RSA Example

1. Choose two prime numbers

   $p = 61$ and $q = 53$

2. Compute $n = pq$

   $n = 61 * 53 = 3233$

3. Compute the totient $\phi(n) = (p-1)(q-1)$

   $$\phi(n) = (61-1)(53-1) = 3120$$

4. Choose $e > 1$ coprime to 3120

   $e = 17$

5. Compute d such that $de \equiv 1 \pmod{\phi(n)}$

   (d is uniquely determined by e and $\phi(n)$)

   $d = 2753$

   $17 * 2753 = 46801 = 1 + 15 * 3120.$

RSA Example

The **public key** is (n = 3233, e = 17). For a padded message the encryption function is:

$$c = m^e \mod n = m^{17} \pmod{3233}$$

The **private key** is (n = 3233, d = 2753). The decryption function is:

$$m = c^d \mod n = c^{2753} \pmod{3233}$$

For example, to encrypt m = 123, we calculate

$c = 123^{17} \pmod{3233} = 855$

To decrypt c = 855, we calculate

$m = 855^{2753} \pmod{3233} = 123$.

RSA Security

The security of the RSA cryptosystem rests in large part on the difficulty of factoring large integers. If an adversary can factor the modulus n in a public key, then he can derive the secret key from the public key, using the knowledge of the factors p and q in the same way that the creator of the public key used them. So if factoring large integers is easy, then breaking the RSA cryptosystem is easy. The converse statement, that if factoring large integers is hard, then breaking RSA is hard, is unproven.

Integer Factorization

Suppose we have an integer ***n*** that we wish to factor, that is, to decompose into a product of primes. The primality test of the preceding section would tell us that ***n*** is composite, but it usually doesn't tell us the prime factors of ***n*****.** Factoring a large integer ***n*** seems to be much more difficult than simply determining whether ***n*** is prime or composite. It is infeasible with today's supercomputers and the best algorithms to date to factor an arbitrary 200-decimal-digit number.

Cormen : Page 844-845

Integer Factorization

As of 2005, the largest number factored by a general-purpose factoring algorithm was 663 bits long, using a state-of-the-art distributed implementation. RSA keys are typically 1024–2048 bits long. Some experts believe that 1024-bit keys may become breakable in the near term (though this is disputed); few see any way that 4096-bit keys could be broken in the foreseeable future. Therefore, it is generally presumed that RSA is secure if n is sufficiently large. If n is 256 bits or shorter, it can be factored in a few hours on a personal computer, using software already freely available. If n is 512 bits or shorter, it can be factored by several hundred computers as of 1999. A theoretical hardware device named TWIRL and described by Shamir and Tromer in 2003 called into question the security of 1024 bit keys. It is currently recommended that n be at least 2048 bits long.