ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

4 Randomized Algorithms: Test of Primalty

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik
Wintersemester 2007/08

CoNe Freiburg

IIF
INSTITUT FÜR INFORMATIK FREIBURG

Randomized algorithms

- Classes of randomized algorithms
- Randomized Quicksort
- Randomized primality test
- Cryptography

Classes of randomized algorithms

- **Las Vegas algorithms**
  - always correct; expected running time (“probably fast”)
  - Examples:
    - randomized Quicksort,
    - randomized algorithm for closest pair
- **Monte Carlo algorithms (mostly correct):**
  - probably correct; guaranteed running time
  - Example: randomized primality test

Primality test

Definition: An integer $p \geq 2$ is prime iff ($a \mid p \rightarrow a = 1$ or $a = p$).

Algorithm: deterministic primality test (naive)

Input: integer $n \geq 2$

Output: answer to the question: Is n prime?

```
if n = 2 then return true
if n even then return false
for i = 1 to √n/2 do
        if 2i + 1 divides n
        then return false
return true
```

Complexity: $\Theta(\sqrt{n})$

Primality test

Goal:

Randomized method

- **Polynomial time complexity (in the length of the input)**
- **If answer is "not prime", then n is not prime**
- **If answer is "prime", then the probability that n is not prime is at most $p>0$**

k **iterations: probability that n is not prime is at most p^k**

Primality test

Observation:

Each odd prime number p divides $2^{p-1} - 1$.

Examples: $p = 17$, $2^{16} - 1 = 65535 = 17 * 3855$

$p = 23$, $2^{22} - 1 = 4194303 = 23 * 182361$

Simple primality test:

```
Calculate z = 2^(n-1) mod n
if z = 1
   then n is possibly prime
   else n is definitely not prime
```

Advantage: This only takes polynomial time

Simple primality test

Definition:

n **is called pseudoprime to base 2, if *n* is not prime and**

$$2^{n-1} \bmod n = 1.$$

Example: $n = 11 * 31 = 341$

$$2^{340} \bmod 341 = 1$$

Randomized primality test

Theorem: **(Fermat's little theorem)**

If p prime and $0 < a < p$, then

$$a^{p-1} \bmod p = 1.$$

Definition:

n **is pseudoprime to base** a**, if** n **not prime and**

$$a^{n-1} \bmod n = 1.$$

Example: $n = 341$, $a = 3$

$$3^{340} \bmod 341 = 56 \neq 1$$

Randomized primality test

Algorithm: Randomized primality test 1

Randomly choose $a \in [2, n-1]$

Calculate $a^{n-1} \bmod n$

if $a^{n-1} \bmod n = 1$

  then n **is possibly prime**

  else n **is definitely not prime**

$Prob$**(**n **is not prim, but** $a^{n-1} \bmod n = 1$ **) ?**

Carmichael numbers

Problem: Carmichael numbers

Definition: An integer *n* is called Carmichael number if

$$a^{n-1} \bmod n = 1$$

for all *a* with GCD(*a*, *n*) = 1. **(GCD = greatest common divisor)**

Example:

Smallest Carmichael number: 561 = 3 * 11 * 17

Randomized primality test 2

Theorem:

If p prime and $0 < a < p$, then the only solutions to the equation

$$a^2 \bmod p = 1$$

are $a = 1$ and $a = p - 1$.

Definition:

a is called non-trivial square root of 1 mod n, if

$$a^2 \bmod n = 1 \quad \text{and} \quad a \neq 1, n - 1.$$

Example: $n = 35$

$$6^2 \bmod 35 = 1$$

Fast exponentiation

Idea:

During the computation of a^{n-1} (0 < a < n randomly chosen), test whether there is a non-trivial square root mod n.

Method for the computation of a^n:

Case 1: [n is even]

$$a^n = a^{n/2} * a^{n/2}$$

Case 2: [n is odd]

$$a^n = a^{(n-1)/2} * a^{(n-1)/2} * a$$

Fast exponentiation

Example:

$a^{62} = (a^{31})^2$

$a^{31} = (a^{15})^2 * a$

$a^{15} = (a^{7})^2 * a$

$a^{7} = (a^{3})^2 * a$

$a^{3} = (a)^2 * a$

Complexity: $O(\log^2 a^n \log n)$

Fast exponentiation

```
boolean isProbablyPrime;

power(int a, int p, int n) {
    /* computes a^p mod n and checks during the
       computation whether there is an x with
       x^2 mod n = 1 and x ≠ 1, n-1 */

    if (p == 0) return 1;
    x = power(a, p/2, n)
    result = (x * x) % n;
```

Fast exponentiation

```
    /* check whether x² mod n = 1 and x ≠ 1, n-1 */
    if (result == 1 && x != 1 && x != n -1 )
        isProbablyPrime = false;

    if (p % 2 == 1)
        result = (a * result) % n;

    return result;
}
```

Complexity: O(log^2*n* log *p*)

Randomized primality test 2

```
primalityTest(int n) {
    /* carries out the randomized primality test for
       a randomly selected a */

    a = random(2, n-1);

    isProbablyPrime = true;

    result = power(a, n-1, n);

    if (result != 1 || !isProbablyPrime)
        return false;
    else
        return true;
}
```

Randomized primality test 2

Theorem:

If n is not prime, there are at most

$$\frac{n-9}{4}$$

integers $0 < a < n$, for which the algorithm primalityTest fails (for non Carmichael Numbers)

ALBERT-LUDWIGS-
UNIVERSITÄT FREIBURG

Algorithm Theory

4 Randomized Algorithms: Test of Primalty

Christian Schindelhauer

Albert-Ludwigs-Universität Freiburg
Institut für Informatik
Rechnernetze und Telematik

Wintersemester 2007/08

CoNe Freiburg

IIF
INSTITUT FÜR
INFORMATIK
FREIBURG