Primes and How to Reconginze them

Primality Testing Algorithms

Satwant Rana[1]

Advised by, Amitabha Tripathi[2]

[1]2012 MT 50618
Mathematics Department
IIT Delhi

[2]Professor
Mathematics Department
IIT Delhi

Mid Semester MTP Presentation, 2016

Motivation

Prime numbers are central to Number Theory, acting as the atomic units around which all numbers are built. Therefore it is barely a surprise that *Primality Testing* is a problem with a rich history in Number Theory.

The motivation behind this project is to rediscover and implement the greatest and the latest in primality testing algorithms.

Primes and Composites

Primes are defined as natural numbers which are only divisible by 1 and themselves. Formally, given $p \in \mathbb{N}$ is a prime, if whenever $q \mid p$, then $q \in \{1, p\}$.

Any natural greater than 1 which is not a prime is called a *composite*.

A Naive Primality Test

Algorithm 1 Naive Primality Test

```
procedure NaivePrimalityTest(n)
    d ← 2
    while d ≤ n − 1 do
        r ← n mod d
        if r = 0 then
            return false                ▷ n is composite
        d ← d + 1
    return true                         ▷ n is prime
```

Time Complexity - $O(n)$

An Optimization

If $n, a, b \in \mathbb{Z}$ such that $n = ab$, then $min(a, b) \leq \sqrt(n)$.

Algorithm 2 Optimized Naive Primality Test

```
procedure OptimizedNaivePrimalityTest(n)
    d ← 2
    while d ≤ min(n − 1, √n)) do
        r ← n  mod d
        if r = 0 then
            return false                ▷ n is composite
        d ← d + 1
    return true                         ▷ n is prime
```

Time Complexity - $O(\sqrt{n})$

Compositeness Tests

A successful *primality test* proves that a given number is prime, whereas a successful *compositeness test* proves that a given number is composite.

e.g. If $n > 2$ and $2 \mid n$, then n is composite.

If a compositeness test is not successful, then we can't comment on the primality of the given number.

Composite numbers which the compositeness test labels as primes are called the *pseudoprimes* for the test.

Fermat's (Little) Theorem

Theorem (Fermat's Theorem)

Given prime p, and $a \in \mathbb{Z}$, $(a, p) = 1$ we have,

$$a^{p-1} \equiv 1 \mod p$$

Corollary (1)

Given prime p, and $a \in \mathbb{Z}$ we have,

$$a^p \equiv a \mod p$$

Fermat's Theorem as a Compositeness Test

The following Corollary 2 is a simple compositeness test using *Fermat's Theorem*.

Corollary (2)

If $n \in \mathbb{N}$, $n \geq 2$ *and* $\exists a \in \mathbb{Z}$ *such that,*

$$a^n \not\equiv a \mod n$$

then n *is not a prime.*

For instance, for $n = 9$, $2^9 \equiv 8 \not\equiv 2 \mod 9$, indicating the compositeness of 9.

Fermat Pseudoprimes

There do exist combinations of a and composite n which satisfy the *Fermat's Theorem*.

For instance $n = 341 = 11.31$ gives $2^{341} \equiv 2 \mod 341$. This makes 341 a pseudoprime to the Fermat's Compositeness Test, or a *Fermat Pseudoprime*.

Although, in this case a change of base a from 2 to 3 yields $3^{341} \equiv 168 \not\equiv 3 \mod 341$ which indicates that 341 is not a prime.

Carmichael Numbers

Given $n \in \mathbb{Z}$ is a *Carmichael Number*, if $a^{n-1} \equiv 1 \mod n$, $\forall a \in \mathbb{Z}, (a, n) = 1$.

The smallest example of *Carmichael Numbers* is 561, and there exist infinitely many of them.

Carmichael Numbers are *Fermat Pseudoprimes* for each base a comprime to n.

Eucledian Algorithm for G.C.D.

For $a, b \in \mathbb{Z}$, we have $(a, 0) = a$, $(a, b) = (a, b - a)$ and therefore $(a, b) = (a, b \mod a)$.

Algorithm 3 Euclidean Algorithm

```
procedure EuclideanAlgorithm(a, b)
    a ← Abs(a)
    b ← Abs(b)                         ▷ Eliminating negative signs
    if a > b then
        Swap(a, b)
    while a ≠ 0 do
        c ← b mod a
        b ← a
        a ← c
    return b
```

Time Complexity - $O(\log \min(|a|, |b|))$

Logarithmic Exponentiation

$$a^n = \begin{cases} 1 & n = 0 \\ (a^{\frac{n}{2}})^2 & n \equiv 0 \mod 2 \\ a(a^{\frac{n-1}{2}})^2 & n \equiv 1 \mod 2 \end{cases}$$

Algorithm 4 Recursive Logarithmic Exponentiation

procedure $\textsc{LogarithmicExponentiation}(a, n, m)$
  $result \leftarrow 1 \mod m$ ▷ Calculates $a^n \mod m$
  if $n > 0 \;\&\; n \equiv 0 \mod 2$ **then**
    $result \leftarrow \textsc{LogarithmicExponentiation}(a, \frac{n}{2}, m)$
    $result \leftarrow result * result \mod m$
  else if $n > 0 \;\&\; n \equiv 1 \mod 2$ **then**
    $result \leftarrow \textsc{LogarithmicExponentiation}(a, \frac{n-1}{2}, m)$
    $result \leftarrow result * result \mod m$
    $result \leftarrow result * a \mod m$
  return $result$

Logarithmic Exponentiation

If $n = b_{d-1}b_{d-2}\dots b_0 = \sum_{i=0}^{d-1} b_i 2^i$, then

$$a^n = a^{\sum_{i=0}^{d-1} b_i 2^i} = \prod_{i=0}^{d-1} a^{b_i 2^i}$$

Algorithm 5 Iterative Logarithmic Exponentiation

procedure LogarithmicExponentiation(a, n, m)
  $result \leftarrow 1 \mod m$ ▷ Calculates $a^n \mod m$
  $b \leftarrow a$
  while $n > 0$ **do**
    if $n \mod 2 = 1$ **then** ▷ If rightmost bit is 1
      $result \leftarrow result * b \mod m$ ▷ Multiply by b
    $b \leftarrow b * b \mod m$ ▷ b stores a^{2^i} on i^{th} step
    $n \leftarrow \frac{n}{2}$ ▷ Remove rightmost bit
  return result

Time Complexity - $O(\log n)$

Fermat's Compositeness Test

Using current discussion, *Fermat's Compositeness Test* can be implented as

Algorithm 6 Fermat's Compositeness Test

procedure $\textsc{FermatCompositenessTest}(a, n)$
 $gcd \leftarrow \textsc{EucledianAlgorithm}(a, n)$.
 if $gcd > 1$ & $gcd < n$ **then**
  return *false*
 $left \leftarrow \textsc{LogarithmicExponentiation}(a, n, n)$
 $right \leftarrow a \mod m$
 return $left \neq right$

Fermat's Probabilistic Primality Test

Every failed run of a compositness test reduces the probability of compositeness, and increases the probability of primality. So we have a *Probabilistic Primality Test*,

Algorithm 7 Fermat's Probabilistic Primality Test

```
procedure FermatProbabilisticPrimalityTest(n, iter)
    while iter > 0 do                              ▷ iter is number of iterations
        a ← Random(0, n − 1)                       ▷ Random number in [0, n − 1]
        check ← FermatCompositenessTest(a, n)
        if check then
            return false                           ▷ Composite found
        iter ← iter − 1
    return true                                    ▷ Probable prime found
```

Time Complexity - $O(\log n)$

Fermat's Primality Test

If we have a table of *pseudoprimes* then a simple check removes the flaw from *Fermat's Compositeness Test*.

D.H. Lehmer prepared a table of all Fermat pseudoprimes below 2.10^8 for the base 2 with no factor < 317. Thus a primality test to check primality for $n < 2.10^8$ can be formulated.

Fermat's Primality Test

Algorithm 8 Fermat's Primality Test

procedure FermatPrimalityTest(n)
    if $n \geq 2.10^8$ **then**
        return *false* $\triangleright$ Fail if out of range
    for $i = 2,\ i \leq \min(313, n-1),\ i \leftarrow i+1$ **do**
        if $i \mid n$ **then**
            return *false* $\triangleright$ Factor ≤ 313
    if IterativeLogarithmicExponentiation$(2, p-1, p) \not\equiv 1 \mod 2$ **then**
        return *false* $\triangleright$ Composite by Fermat's Theorem
    return !IsLehmerPseudoprime(n) $\triangleright$ Check Lehmer's Table

Future Work

Planned readings include *Lucas Sequences* based *Primality Tests*, *Lenstra's Theorem* and *A.K.S. Algorithm*.

We plan to implement key algorithms from the above (and elsewhere) as well, in a modern programming language.