6-1992

Primality Testing

Per Brinch Hansen
Syracuse University, School of Computer and Information Science, pbh@top.cis.syr.edu

SU-CIS-92-13

Primality Testing

Per Brinch Hansen

June 1992

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, NY 13244-4100

Primality Testing[1]

PER BRINCH HANSEN

Syracuse University, Syracuse, New York 13244

June 1992

This tutorial describes the Miller-Rabin method for testing the primality of large integers. The method is illustrated by a Pascal algorithm. The performance of the algorithm was measured on a Computing Surface.

Categories and Subject Descriptors: G.3 [**Probability and Statistics**: [Probabilistic algorithms (Monte Carlo)

General Terms: Algorithms

Additional Key Words and Phrases: Primality testing

CONTENTS

INTRODUCTION

This tutorial describes a probabilistic method for testing the *primality* of large integers. The method was developed by Miller [1976] and Rabin [1980].

In the *RSA cryptosystem* large primes play an essential role in the encoding and decoding of messages [Rivest et al., 1978]. A user chooses two large random primes. These primes are used to compute a public encoding key and a secret decoding key. Both keys include the product of the primes. The user can receive encoded messages from anyone who knows the public key. But only the user (who knows the secret key) can decode the messages.

The crucial assumption is that it is feasible to generate large primes using a computer, but there is no known algorithm for finding the prime factors of large composite numbers in reasonable amounts of computer time. If that ever becomes possible, we will be able to break the code by factorizing the public product of the secret primes.

The RSA cryptosystem is believed to be secure for keys of 150 decimal digits. The simplest way to find a 150-digit prime is to generate random 150-digit numbers until we discover a prime. The probability that a 150-digit number is a prime is about 1 in $150 \ln 10$ [Courant and Robbins, 1941]. We must therefore expect to test about 350 numbers for primality before we find a prime. (Half of these tests can be skipped if we only examine odd numbers.)

So the generation of primes is reduced to the problem of testing the primality of random numbers. Since it is not feasible to compute the prime factors of large numbers, we will use a probabilistic method that almost never fails to distinguish correctly between primes and composites.

We will describe the Miller-Rabin method of primality testing and illustrate it by a Pascal algorithm. The performance of the algorithm was tested on a Computing Surface.

1. FERMAT'S THEOREM

The primality test uses a famous theorem discovered by Pierre de Fermat [1640].

Consider a prime p and any positive integer x that is not divisible by p. We now define the following sequence of numbers:

$$0x \bmod p,\ 1x \bmod p,\ 2x \bmod p,\ \ldots,\ (p-1)x \bmod p$$

These numbers are obviously integers in the range from 0 to $p-1$. And they are distinct integers. For, if we assume that two of them are equal, say

$$jx \bmod p = ix \bmod p$$

where $0 \leq i < j \leq p-1$, then

$$(j-i)x \bmod p = 0$$

Since p is a prime, it cannot be expressed as a product of factors. Consequently, either $j - i$ or x (or both) must be divisible by p. But that is impossible, since $j - i$ is less than p, and x is not divisible by p.

Well, then we know that the sequence is simply a permutation of the integers $0, 1, \ldots, p-1$. Since the first number in the sequence is zero, the rest of it must be a permutation of the integers $1, 2, \ldots, p-1$. Consequently,

$$(1x \bmod p)(2x \bmod p)\ldots((p-1)x \bmod p) = 1^*2^*\ldots(p-1)$$

which is equivalent to

$$(p-1)!(x^{p-1} - 1) \bmod p = 0$$

Since none of the factors of $(p-1)!$ are divisible by p, the rest of the product must be divisible by p:

$$(x^{p-1} - 1) \bmod p = 0$$

In short, if p is a prime and x is a positive integer that is not divisible by p, then

$$x^{p-1} \bmod p = 1 \tag{1}$$

This is *Fermat's theorem.*

2. THE FERMAT TEST

Fermat's theorem suggests a simple way to test the primality of a positive integer p:

1. Generate a random integer x in the range

$$1 \leq x \leq p - 1$$

Since x is less than p, x is obviously not divisible by p.

2. Raise x to the power of $p - 1$ modulo p.

3a. If the result is not 1, then p does not satisfy Eq. (1). This proves that p is not a prime. In that case, the integer x is called a *witness* to the compositeness of p.

3b. If the result is 1, p may be a prime. But the Fermat test is not foolproof. For each base value x, there are an infinite number of composites p that satisfy Eq. (1). These composites are known as *pseudoprimes* [Burton 1980].

Algorithm 1 defines the *Fermat test.* The boolean value of the function defines whether or not x is a witness to the compositeness of p.

```
function witness(x, p: integer): boolean;
var e, m, y: integer;
begin {1 <= x <= p - 1}
  m := 1; y := x; e := p - 1;
  while e > 0 do
    if e mod 2 = 1 then
    begin
      m := (m*y) mod p; e := e - 1
    end else
    begin
      y := sqr(y) mod p; e := e div 2
    end;
  witness := (m <> 1)
end
```

Algorithm 1

The function defines *modular exponentiation* by repeated squaring. The loop maintains the invariant

$$my^e \bmod p = x^{p-1} \bmod p$$

where $1 \leq m \leq p-1$ and $e \geq 0$.

When the loop terminates with $e = 0$, we have

$$m = x^{p-1} \bmod p$$

If m is not 1, then x is a witness; otherwise, it is not.

The algorithm assumes that numbers are represented by standard integers. This is obviously not possible for 150-digit decimal integers. In practice, the algorithm must be reprogrammed using multiple-length arithmetic. We will discuss this requirement later.

We will use three methods to reduce the probability that the primality test gives the wrong answer:

1. Test large numbers.

It can be shown that the probability that a random number is a pseudoprime approaches zero as the number of digits goes toward infinity [Pomerance, 1981].

2. Repeat the Fermat test for different *base values* x.

Although this helps, it is not watertight either. There are composite integers p, which satisfy Eq. (1) for any base value x. Fortunately, these *Carmichael numbers*

are extremely rare [Carmichael, 1912].

3. Supplement the Fermat test with another test.

The supplementary test is based on a theorem about quadratic remainders.

3. QUADRATIC REMAINDERS

Consider the quadratic equation

$$y^2 \bmod p = 1 \tag{2}$$

where y and p are positive integers. This equation is equivalent to

$$(y-1)(y+1) \bmod p = 0$$

If p is a prime, either $y-1$ or $y+1$ (or both) must be divisible by p:

$$(y-1) \bmod p = 0$$

or

$$(y+1) \bmod p = 0$$

In that case, the only solutions to Eq. (2) are the *trivial* square roots of 1 modulo p:

$$y \bmod p = 1 \qquad y \bmod p = p-1$$

A *nontrivial* square root of 1 is an integer y modulo p in the range

$$1 < y \bmod p < p-1$$

which satisfies Eq. (2). If we can find such an integer y, then p is not a prime.

4. THE MILLER-RABIN TEST

Algorithm 2 is an extension of the Fermat test proposed by Miller [1976] and refined by Rabin [1980]. Every time the function squares the current value of y, it checks whether y modulo p is a nontrivial square root of 1. In that case, the function stops the Fermat test and returns the value true, indicating that p surely is composite.

```
function witness(x, p: integer): boolean;
var e, m, p1, r, y: integer;
  sure: boolean;
begin {1 <= x <= p-1}
  m := 1; y := x; e := p - 1;
  p1 := e; sure := false;
  while not sure and (e > 0) do
    if e mod 2 = 1 then
    begin
      m := (m*y) mod p; e := e - 1
    end else
    begin
      r := y;
      y := sqr(y) mod p; e := e div 2;
      if y = 1 then
        sure := (1 < r) and (r < p1)
    end;
  witness := sure or (m <> 1)
end
```

Algorithm 2

The loop invariant is unchanged, but the termination condition is slightly different:

$$\text{sure } \mathbf{or} \ (m = x^{p-1} \ \mathbf{mod} \ p)$$

The Miller-Rabin test is also probabilistic.

5. A PROBABILISTIC ALGORITHM

The Miller-Rabin test gives the wrong answer if it fails to discover a witness to a composite number. Rabin [1980] proved that the probability of failure is less than $\frac{1}{4}$. To improve the chance of finding the correct result, he suggested repeating the test m times with different random base values (Algorithm 3).

```
function prime(p, m: integer): boolean;
var sure: boolean; i: integer;
begin
  sure := false;
  for i := 1 to m do
    if witness(random(1, p - 1), p) then
      sure := true;
  prime := not sure
end
```

Algorithm 3

If the algorithm discovers a single witness, then p is definitely composite. At this point we could skip further tests. Instead, we let the algorithm complete the sequence of trials to make it obvious that the m trials can be performed simultaneously on a parallel computer.

If the algorithm finds no witnesses in, say, 40 trials, then p is a prime with overwhelming probability: The probability that the algorithm fails to detect a composite is less than $(\frac{1}{4})^{40} \approx 10^{-24}$.

This is far less than the probability of a computer error. A computer that performs one million operations per second with the same probability of failure per operation will fail only once in thirty billion years. That is roughly the age of the universe since the Big Bang [Sagan, 1980]!

6. COMPLEXITY

In practice, Algorithms 2–3 must be reprogrammed to perform *multiple-length arithmetic* on large natural numbers. These serial operations imitate the familiar paper-and-pencil methods [Knuth, 1969]. Multiple-length division turns out to be a problem of surprising difficulty [Brinch Hansen, 1992a].

Consider primality testing of an integer p with N *decimal digits*. During the computation, an integer with $O(N)$ decimal digits is represented by an array of $O(n)$ digits in a *radix* b, which is a power of ten:

$$b = 10^{\log b} \qquad n \approx N/\log b$$

The witness test performs $O(N)$ iterations (unless it is interrupted by the discovery of nontrivial square root of 1). Each iteration involves multiplication and division of $O(n^2)$ complexity. Each step also requires operations of $O(n)$ complexity, including the time-consuming computation of trial digits during division [Brinch Hansen, 1992a].

Consequently, primality testing has the complexity

$$T = O((n^2 + n)N) = O((n + 1)nN)$$

In other words,

$$T \approx (cN/\log b + d)N^2/\log b \tag{3}$$

where c and d are system-dependent constants of decimal arithmetic.

7. EXPERIMENTS

We programmed Rabin's algorithm in occam and tested it on a Computing Surface with T800 transputers using the random number generator of Park and Miller [1988].

After systematic testing of the multiple-length arithmetic, we performed several experiments. In each experiment a random integer p was tested 40 times for primality using different random base values x. The trials were performed in parallel by 40 transputers [Brinch Hansen, 1992b].

The program correctly identified the 121-digit *Mersenne number*

$$2^{400} - 1$$

as a composite, and confirmed that

$$2^{400} - 593$$

almost certainly is a prime [Rabin, 1980].

Table I shows measured (and predicted) run times T in seconds for a random integer p with $N = 120$ decimal digits. The integer is represented by an array of random digits in radix b. In theory, radix 10,000 reduces the run time by a factor 16 (or less) compared to radix 10. In practice, it makes the program run 13 times faster.

Table I

b	T	s
10	219	(226)
100	59	(61)
1,000	29	(29)
10,000	17	(18)

Table II shows measured (and predicted) run times T for primality testing of random numbers with N decimal digits represented in radix $b = 10,000$.

Table II

N	T	s
120	17	(18)
160	39	(39)
200	73	(73)
240	124	(122)
280	190	(190)

The empirical formula

$$T \approx (0.12N/\log b + 1.3)N^2/\log b \; ms$$

defines the estimated run times shown in parentheses in Tables I and II.

8. SUMMARY

We have described a probabilistic algorithm for testing the primality of a large integer without factorizing it. The Miller-Rabin algorithm is always right when it identifies a number as composite. A number that is not recognized as composite is prime with extremely high probability. On a Computing Surface the algorithm tests the primality of a 150-digit decimal integer 40 times in about 30 seconds.

ACKNOWLEDGEMENTS

I am grateful to Jonathan Greenfield for valuable advice.

REFERENCES

BRINCH HANSEN, P. 1992a. Multiple-length division revisited: A tour of the minefield. School of Computer and Information Science, Syracuse University, Syracuse, NY.

BRINCH HANSEN, P. 1992b. Parallel Monte Carlo trials. School of Computer and Information Science, Syracuse University, Syracuse, NY.

BURTON, D. M. 1980. *Elementary Number Theory.* Allyn and Bacon, Boston, MA.

CARMICHAEL, R. D. 1912. On composite numbers p which satisfy the Fermat congruence $a^{p-1} = 1 \bmod p$. *American Mathematical Monthly* 19:22–27.

COURANT, R., and ROBBINS, H. 1941. *What is Mathematics?* Oxford University Press, NY.

FERMAT, P. de, 1640. Letter to Bernard Frenicle de Bessy. (Oct. 18).

KNUTH, D. 1969. *The Art of Computer Programming.* Volume 2: *Seminumerical Algorithms.* Addison-Wesley, Reading, MA.

MILLER, G. L., 1976. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences* **13**:300–317.

PARK, S. K., and MILLER, K. W. 1988. Random number generators: good ones are hard to find. *Communications of the ACM* **31**:1192–1201.

POMERANCE, C. 1981. On the distribution of pseudoprimes. *Mathematics of Computation* **37**:587–593.

RABIN, M. O. 1980. Probabilistic algorithms for testing primality. *Journal of Number Theory* **12**:128–138.

RIVEST, R. L., SHAMIR, A., and ADLEMAN, L. M. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* **21**:120–126.

SAGAN, C. 1980. *Cosmos.* Random House, NY.