Artículo de investigación

The development of an effective algorithm searching for strong pseudoprime numbers

El desarrollo de un algoritmo eficaz en busca de fuertes números pseudoprimos
O desenvolvimento de um algoritmo eficiente em busca de números pseudo-primos fortes

Recibido: 20 de abril de 2018. Aceptado: 10 de mayo de 2018

Written by:
Nikolai A. Antonov[I]
[I]*KAZAN FEDERAL UNIVERSITY*
e-mail: ***nikoljaany@mail.ru***
Tel. +79047667430

Abstract

The problem of searching for strictly pseudoprime numbers is relevant in the field of number theory, and it also has a number of applications in cryptography: in particular, with the help of numbers in this class one can strengthen the efficiency of the Miller-Rabin simplicity test by transforming it from probabilistic into deterministic. At the present time, several algorithms for constructing sequences of such numbers are known, but they have a rather high complexity, which makes it impossible to obtain strictly pseudoprime numbers of large magnitude in an acceptable time. The theme of this paper is the construction of strictly pseudoprime numbers of the special form n = pq = (u + 1) (2u + 1), where p, q are prime numbers, u is a natural number. Numbers of this kind are present in the sequence Ψk, used to estimate the number of iterations in the Miller-Rabin simplicity test. We denote by Fk the smallest odd composite number of the above-mentioned type, which successfully passes the Miller-Rabin test with k first prime numbers. The paper proposes a new algorithm for constructing Fk numbers, gives data on its speed and efficiency on the memory used, and specifies the features of the software implementation.

Keywords: strictly pseudoprime numbers, strongly pseudo-simple numbers, algorithm for constructing strictly pseudoprime numbers, Miller-Rabin test, special form of a number.

Resumen

El problema de la búsqueda de números estrictamente pseudoprimos es relevante en el campo de la teoría de números, y también tiene varias aplicaciones en criptografía: en particular, con la ayuda de números en esta clase se puede fortalecer la eficiencia de la simplicidad de Miller-Rabin Prueba transformándolo de probabilístico en determinístico. En la actualidad, se conocen varios algoritmos para construir secuencias de tales números, pero tienen una complejidad bastante alta, lo que hace imposible obtener números estrictamente pseudoprimos de gran magnitud en un tiempo aceptable. El tema de este artículo es la construcción de números estrictamente pseudoprime de la forma especial n = pq = (u + 1) (2u + 1), donde p, q son números primos, u es un número natural. Los números de este tipo están presentes en la secuencia Ψk, utilizada para estimar el número de iteraciones en la prueba de simplicidad de Miller-Rabin. Denotamos por Fk el número compuesto impar más pequeño del tipo mencionado anteriormente, que pasa con éxito la prueba de Miller-Rabin con k primeros números primos. El documento propone un nuevo algoritmo para construir números Fk, proporciona datos sobre su velocidad y eficiencia en la memoria utilizada y especifica las características de la implementación del software.

Palabras claves: Números estrictamente pseudoprimos, números fuertemente pseudo-simples, algoritmo para construir números estrictamente pseudoprimos, prueba de Miller-Rabin, forma especial de un número.

Resumo

O problema de encontrar números pseudoprimos é estritamente relevante no campo da teoria dos números, e também tem muitas aplicações em criptografia: em particular, com a ajuda de números nesta classe pode fortalecer a eficiência da simplicidade de Miller Teste de Rabin transformando-o de probabilístico para determinístico. Atualmente, vários algoritmos são conhecidos por construir seqüências de tais números, mas eles têm uma complexidade bastante alta, o que torna impossível obter números estritamente pseudoprimo de grande magnitude em um tempo aceitável. O assunto deste artigo é a construção de números estritamente pseudoprimo forma especial com n = pq = (L + 1) (2u + 1) em que p, q são números primos, u é um número natural. Números desse tipo estão presentes na seqüência Ψk, usada para estimar o número de iterações no teste de simplicidade de Miller-Rabin. Denotamos por Fk o menor número composto ímpar do tipo mencionado acima, que passa com sucesso no teste de Miller-Rabin com k primeiros números primos. O documento propõe um novo algoritmo para a construção de números Fk, fornece dados sobre sua velocidade e eficiência na memória utilizada e especifica as características da implementação do software.

Palavras-chave: Números estritamente pseudoprimarios, números fortemente pseudo-simples, algoritmo para construção de números estritamente pseudoprimarios, teste de Miller-Rabin, forma especial de um número.

Introduction

Of all the types of numbers used in cryptography, the most widely used are prime numbers. A natural number is said to be simple if it shares only one unit and itself. Otherwise it is called compound. Such a definition always makes it possible to distinguish a prime number from a composite one, but it quickly becomes inapplicable in practice - with increasing numbers, their verification for simplicity (search of all possible divisors) takes more and more time (Crandall & Pomerance, 2011; Golovchun, 2017).

The first step in solving this problem was made by the French mathematician Pierre Fermat. Having formulated the statement, later called "Fermat's small theorem", he proposed the necessary condition for the given number to be simple. Later, this condition was turned into sufficient to make the number composite. Nevertheless, the use of Fermat's small theorem as a criterion for checking the number for simplicity turned out to be impossible: there were composite numbers that could not be distinguished from simple ones, using only the condition of the theorem (Crandall & Pomerance, 2005; Villalobos Antúnez, 2013). Such numbers began to be called pseudoprimes.

The next significant contribution to the solution of the problem of checking the simplicity of the number was the Miller-Rabin test (Ishmukhametov, 2011). In general, the test result for this test is correct only with a certain probability, but the probability of error decreases fourfold after the next stage of verification. If the given number n is simple, then the next round of the test, performed with any number a (1 <a <n), will show in favor of the fact that n is simple. The number a in this case is called the witness of the simplicity of the number n. According to the theorem proved by Rabin, all numbers less than a given prime number will be the witnesses of simplicity. At the same time, the number of witnesses to the simplicity of any composite number does not exceed a quarter of the value of the Euler function from it (Ishmukhametov & Mubarakov, 2014; Ishmukhametov & Mubarakov, 2013).

Thus, this number can be compound, but some of the smaller numbers will indicate its simplicity. Let a natural number n be given and a (1 <a <n) is a witness of its simplicity. In this case, the number n is called strictly (strongly) pseudoprime with respect to the base "a" (Pomerance et al, 1980; Jaeschke, 1993).

Methods

Consider a set of numbers of the form n = pq = (u + 1) (2u + 1), where p, q are prime numbers, u is a natural number. The smallest number of a given set that successfully passes the Miller-Rabin test with k first prime numbers is noted by Fk. By definition, the number Fk has the following form: Fk = pq = (u + 1) (2u + 1),

where p, q are prime numbers, u is a natural number. It is known that the remainder of dividing any prime number by 6 can be equal to either 1 or 5. We use this property to refine the form of the numbers Fk. We write and solve four systems of linear comparisons (Jiang & Deng, 2012):

(1) $\begin{cases} u+1 \equiv 1 \pmod 6 \\ 2u+1 \equiv 1 \pmod 6 \end{cases}$

(2) $\begin{cases} u+1 \equiv 1 \pmod 6 \\ 2u+1 \equiv 5 \pmod 6 \end{cases}$

(3) $\begin{cases} u+1 \equiv 5 \pmod 6 \\ 2u+1 \equiv 1 \pmod 6 \end{cases}$

(4) $\begin{cases} u+1 \equiv 5 \pmod 6 \\ 2u+1 \equiv 5 \pmod 6 \end{cases}$

Solving system (1), we obtain that u = 6t, where t is a natural number. Systems (2), (3), (4) have no solutions. Thus, the form of the number Fk can be written as follows:

Fk = pq = (6t + 1) (12t + 1), where p, q are prime numbers, t is a natural number.

We carry out further transformations: let the prime numbers p, q be given. The number n = pq is strongly pseudoprime with respect to the base a if and only if the following conditions hold:

Ordp (a) | GCD (p − 1; q − 1),
Ordq (a) | GCD (p − 1; q − 1),

Bin(ordp (a)) = Bin(ordq (a)).

If n = pq = (u + 1)(2u + 1) we will get
Ord(u+1) (a) | u
Ord(2u+1) (a) | u
Bin (Ord(u+1) (a)) = Bin(Ord(2u+1) (a)).

It follows that any witness of the simplicity of a number
n = pq = (u + 1) (2u + 1)
is not a generator of the Galois field GF (q). Therefore, we can say that the number a is a quadratic residue of modulo q and $\left(\frac{a}{q}\right)=1$

As the index k increases, the form of the number Fk can be sequentially refined (modified), increasing the coefficients before t.
CASE k = 1:
In this case, the simplicity of the number Fk is guaranteed to be the first prime number – 2.

$$\left(\frac{2}{q}\right)=\left(\frac{2}{12t+1}\right)=(-1)^{\frac{12t}{2}\cdot\frac{1}{2}}\left(\frac{12t+1}{2}\right)=(-1)^{3t}\cdot 1=(-1)^{3t}$$

$$\left(\frac{2}{q}\right)=1 \qquad \Rightarrow \qquad (-1)^{3t}=1$$

Thus, t can take only even values (STRONG PSEUDOPRIMES TO TWELVE PRIME BASES JONATHAN SORENSON AND JONATHAN WEBSTER). Consequently, any number Fk can be represented in the following form:
Fk = pq = (12t + 1)(24t + 1),
where p, q are prime numbers, t is a natural number.
CASE k = 2:

In this case, the first two prime numbers - 2 and 3 - are guaranteed to be the witnesses of the simplicity of the number Fk. The form of the number Fk, strictly (strongly) pseudoprime with respect to the base 2, has already been established. Now you can explore it for the possibility of refinement along with a simple number 3:

$$\left(\frac{3}{q}\right)=\left(\frac{3}{24t+1}\right)=(-1)^{\frac{24t}{2}\cdot\frac{2}{2}}\left(\frac{24t+1}{3}\right)=(-1)^{12t}\cdot 1=1$$

Form for search Fk remains the same
CASE k > 2.
Consider the example of the case k = 3. Taking as the basis the form of the number Fk, strictly pseudoprime for bases 2 and 3, we try to refine it using the third prime number - 5. We have:

$$\left(\frac{5}{q}\right)=\left(\frac{5}{24t+1}\right)=(-1)^{\frac{24t}{2}\cdot\frac{4}{2}}\left(\frac{24t+1}{5}\right)=1\cdot\left(\frac{4t+1}{5}\right)=1$$

The quadratic residues of a prime number 5 are numbers 1 and 4. The form 4t + 1 must be comparable to one of them modulo 5. We have:

$$\begin{cases} 4t+1 \equiv 1 \pmod 5 \\ 4t+1 \equiv 4 \pmod 5 \end{cases}$$

$$\begin{cases} t \equiv 0 \pmod 5 \\ t \equiv 2 \pmod 5 \end{cases}$$

$$\begin{cases} t = 5v \\ t = 5v+2 \end{cases}$$

Thus, the form for finding the numbers Fk for k = 3 can be clarified, but it will be necessary to consider not one but two more precise forms.

We note that as the index k increases, starting from the value of 3, the number of forms that will need to be considered after the next stage of refinement also increase.
From all of the above, the following scheme for constructing special forms follows: let the value of index k be given, and it is required to find the value of the corresponding number Fk.
If k = 1 or k = 2, then we searching through the natural values of the variable t according to the form:
Fk = pq = (12t + 1) (24t + 1),where p, q are prime numbers, t is a natural number.
If k> 2, this "initial" form can be refined sequentially, but no more than k - 2 times. At the same time, as noted earlier, at the next stage of refinement j (1 <= j <k-2), a set of new special forms will be obtained, and the enumeration of the values of each individual form will be performed with a much larger step. Let the initial form:pq = (12t + 1) (24t + 1) passed j - 1 stages of refinement. Accordingly, we construct the set Fj-1 of all possible special forms suitable for finding the numbers Ft, where t > = j - 1. Construct all the most accurate forms for finding the number Ft, where t > = j. For this:

1. Consider the j-th order prime pj and the set Rj of quadratic residues modulo pj.
2. For each pair (f, r) f $\in$ Fj-1, r $\in$ Rj we solve the linear comparison f ≡ r (mod pj). As a result, the variable t in the form f will be represented in the form t = pj * n + c, n is a new variable, c is a natural number smaller than pj.
3. Having made the change of variables in the f form, we get a new more precise form suitable for searching for the numbers Fj, Fj + 1, ..., Fk.

Results and Discussion

The input of the algorithm is given by values of the following parameters:

1. The index k of the number Fk.to be found
2. The boundary of the computation is the number B, which restricts the value of Fk..
3. Depth of constructing a tree of special forms d <= k
4. The number of available processors C (when using a software implementation that allows parallelization)

Comment 1:
All entered values must be natural numbers.

Comment 2:
In accordance with the value of the parameter d, special forms suitable for searching for the numbers Fj, where j > = d, will be constructed. Obviously, the number Fk.is guaranteed to be strictly pseudoprime with respect to the first k prime numbers, therefore we need to take d <= k.

General scheme of the algorithm

1. Transmission of input parameters
2. If k = 1 or k = 2, search Fk among the values of the special form n = pq = (12t + 1) (24t + 1) that do not exceed the given boundary B (t is a natural number). Early exit the algorithm.
3. Otherwise, build a list of special forms in accordance with the parameter d.

Search Fk among the values of each of the constructed special forms. The search for each form is carried out by increasing the natural arguments and until its value exceeds the specified boundary B.

Development environment

The algorithm is implemented in the C# programming language in the Microsoft Visual Studio 2012 environment. The complete list of libraries connected to the project after its development and optimization is as follows:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;
using System.Text;
using System.Threading;
```

The implementation uses the System.Threading subspace of names and provides the ability to perform parallel computations to find the desired number n. The resource costs associated with the launch of parallel threads are minimized, as a result of which the acceleration of the performance of the algorithm using parallel computing is almost exactly proportional to the number of processors involved.

Effective implementation of the Miller-Rabin test

The running time of the constructed algorithm directly depends on the speed of checking for the simplicity of the factors of a given semisimple

number. In this regard, the procedure for verifying the simplicity of the number based on the Miller-Rabin test was significantly improved in comparison with the direct implementation of its conceptual model (Damgard et al, 1993). So, in the simplicity check procedure, a method of full enumeration of prime divisors was introduced, the application of which for small numbers (n <109) yields a gain in the scan time and saves memory resources:

```
bool MR1(BigInteger value)
      {
         if (value < 2)
            return false;
         int sqrt = (int)Sqrt(value);
         for (int i = 0; Primes[i] <= sqrt; i++)
            if (value % Primes[i] == 0)
               return false;
         return true;
   }
```

A more significant improvement is the introduction into the Miller-Rabin test of the numbers of the sequence {Ψk}. Their application makes the test deterministic to the value of n, equal to the last member of this sequence, which in practice means a significant acceleration of the verification procedure:

```
// Number group {Ψk}, arranged in ascending order
 string[] Psi = {  "2047", "1373653", "2532600l",
"3215031751",                "2152302898747",
"3474749660383",          "34155007172832l",
"341550071728321",  "38251230565464l3051",
"3825l230565464l3051",
"3825l230565464l3051","318665857834031l5
1167461",  "331704406467988738596l981"};
```

For verifiable numbers n whose values exceed the maximum of the numbers {Ψk}, the verification is carried out according to the deterministic test of Miller under the condition of validity of the Extended Riemann.

Hypothesis:

```
bool MR3(BigInteger value)
      {  // Let (n - 1) be t*(2^s), t - uneven
         BigInteger t = value - 1;
         int s = 0;
         while (t % 2 == 0)
         {s++;  t /= 2;}
      // Define the bound of calculations
      // according to G. Miller, provided the
validity of the CGW boundary b < 2ln(n)*ln(n)
         double logOfvalue =
BigInteger.Log(value);
         BigInteger border = (BigInteger)(2 *
logOfvalue * logOfvalue) + 1;
      // Test conditions check
      for (int i = 2; i <= border; i++)
         if (!IsWitnessByBase(i, value, t, s))
            return false;
      return true;
   }
```

Using the task of the manufacturer and the consumer

In the process of searching for the number of Fk, several threads run in parallel. One of them constructs special forms of searching for the number Fk in accordance with the parameter d. Others perform an enumeration of the values of the obtained forms up to a given boundary. The transfer of special forms occurs through a buffer of a limited amount. Thus, the scheme for implementing the algorithm involves the model of the problem of the producer and the consumer.

Summary

During the testing of the algorithm, a sequence of numbers Fk was constructed for values of index k not exceeding thirteen.

Table 1. Elements of the sequence of numbers Fk

k, index number	Fk
1	49141
2	1373653
3	1157839381
4	307768373641
5	21569059132741
6	75451785985621
7	300208309455510001
8	84983557412237221
9	412343161357056904l
10	1955097530374556503981
11	739501024079412070938l
12	3186658578340311511674 61
13	331704406467988738596l981

Below are the data on the time spent searching for Fk values. The algorithm was launched on a personal computer with 4 virtual (2 real) cores and a processor clock speed of 2.0 GHz. The calculations were made in parallel.

Table 2. Time of parallel execution of the algorithm

Index number	The length of Fk in decimal digits	Time
1	0,49 * 105	0, 000001s
2	0,13 * 107	0, 000001s
3	0,115* 1010	0,015s
4	0,307 * 1012	0,109s
5	0,21 * 1014	0,375s
6	0,75 * 1014	0,328s
7	0,3 * 1017	3s
8	0,84 * 1017	2,5s
9	0,41 * 1020	27s
10	0,195 * 1022	2min 14s
11	0,739 * 1022	5min 33s
12	0,318 * 1024	17min 6s
13	0,331 * 1025	51min 25s

Thus, most of the above values of Fk were calculated in a few seconds or less, and the largest of the Fk numbers are less than one hour.

The algorithm can be executed simultaneously by several processors, even with an extremely small amount of available RAM. The minimum requirements do not exceed several hundred bytes and do not depend on the required number Fk. The algorithm costs for the amount of RAM used are estimated as O (1).

The search for Fk numbers can be performed in a standard way - by sorting by the even values of the argument "u" of the form Fk = pq = (u + 1) (2u + 1). In this case neither specifying special forms nor parallel calculations are used. The table below compares the search time Fk with the standard sequential search method and the new method proposed in this work.

Table 3. Search Time Comparison

Index number	Search by proposed new method	Standard sequential search
1	0, 000001s	0, 000001s
2	0, 000001s	0, 000001s
3	0,015s	0,265s
4	0,109s	6s
5	0,375s	54s
6	0,328s	1min 43 s
7	3s	35min 22s
8	2,5s	59min 20s
9	27s	> 24 hours

Obviously, the efficiency of the algorithm using the procedure for constructing more precise special forms and using parallel computations is much higher.

Estimation of algorithm complexity

Let the ordinal number k of the number Fk be given, the value of the parameter d (1 ≤ d ≤ k) and the boundary of the computation B.

1. Let S denote the number of special forms constructed to search for Fk

$$S = S(d) = 2 \cdot \ddot{I}_{i=1}^{d} \left(\frac{p_i - 1}{2} \right),$$

where p_i - i-th prime number in order

2. Denote by N the number of non-negative values of the argument t of a separate special form, on which its values are less than the specified boundary B. Let $\ddot{I}_{i=1}^{d} p_i = P$. Then any constructed special form has the form $(2Pt + C_1)(4Pt + C_2)$, где $C_1 < 2P$, $C_2 < 4P$ and we have:

$$(2Pt + C_1)(4Pt + C_2) < B$$

$$8P^2t^2 + (2C_2 + 4C_1)Pt - (B - C_1C_2) < B$$

.

Solving this inequality with respect to t, we obtain:

$$t \le \frac{-(2C_2 + 4C_1) + \sqrt{(2C_2 - 4C_1)^2 + 32B}}{16P}$$

Thus,

$$N = N(d) = O\left(\frac{\sqrt{B}}{\ddot{I}_{i=1}^{d} p_i} \right)$$

3. Estimation of the complexity of the algorithm:

$$E = S(d) \cdot N(d)$$

$$E = O\left(\frac{\sqrt{B} \cdot \ddot{I}_{i=1}^{d} \left(\frac{p_i - 1}{p_i} \right)}{2^d} \right)$$

4. It is obvious that for a fixed boundary B the minimum E = E (d) is attained at d = k. Thus, in the sequential execution of the algorithm, we agree to take d = k. In this case we have:

$$E = E(k) = O\left(\frac{\sqrt{B} \cdot \ddot{I}_{i=1}^{k} \left(\frac{p_i - 1}{p_i} \right)}{2^k} \right)$$

Conclusions

Being adapted to use parallel computations, the proposed algorithm can certainly be useful in modern studies in the construction of strictly pseudoprime numbers. It is safe to assume that the construction of algorithms based on special forms will make it possible to approach even more closely the solution of more general problems related to the construction of numbers of this class.

Acknowledgements

The work is performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

Bibliography

Crandall R., Pomerance K. (2011). The prime numbers: Cryptographic and computational aspects. Trans. from English / Ed. and with an introduction by V.N. Chubarikov. - Moscow.: USSR: Book House "LIBROKOM", 664 p.

Crandall R., Pomerance K. (2005). The prime numbers: a computational perspective. – 2nd Ed., Springer-Verlag, Berlin, 604 p.

Sh. T. Ishmukhametov. (2011). Methods of factorization of natural numbers: a tutorial. - Kazan, KFU, 190 p.

Sh. T. Ishmukhametov, B.G. Mubarakov. (2014). On a class of strictly pseudoprime number // Heuristic Algorithms and Distributed Computing. - Book 1, No. 1, P. 64-73.

S. Ishmukhametov, B. Mubarakov. (2013). On practical aspects of the Miller-Rabin Primality Test // Lobachevskii Journal of Mathematics. – Vol. 34, No. 4, P.304–312.

C. Pomerance, C. Selfridge, S. Wagstaff. (1980). The Pseudoprimes to 25*109 // Math. Comput. P. 1003-1026.

G. Jaeschke. (1993). On Strong Pseudoprimes to Several Bases // Math. Comput. 61. P. 915-926.

J. Jiang, Y. Deng. (2012). Strong pseudoprimes to the first 9 prime bases // ArXiv:1207.0063v1 [math.NT]. P.1–12.

J. Jiang, Y. Deng. (2012). Strong pseudoprimes to the first 9 prime bases // ArXiv:1207.0063v1 [math.NT]. P.1–12.

STRONG PSEUDOPRIMES TO TWELVE PRIME BASES JONATHAN SORENSON AND JONATHAN WEBSTER Date: September 4, 2015. 2010 Mathematics Subject Classi_cation. Primary 11Y16, 11Y16; Secondary 11A41, 68W40, 68W10.

Ivan B. Damgard, Peter Landrock, and Carl Pomerance. (1993). Average case error estimates for the strong probable prime test. Math. Comp. 61(203), P. 177-194.

Golovchun A., Karimova B., Zhunissova M., Ospankulova G., Mukhamadi K. (2017). Content And Language Integrated Learning In Terms Of Multilingualism: Kazakhstani Experience, Astra Salvensis, Supplement No. 10, p. 297-306.

Villalobos Antúnez J.V. (2013). José Vicente Villalobos Antúnez. Opcón, 29 (72), Pp. 10-19.