Выяснение простоты натурального числа N, используя разложение чисел $N^2 \pm 1$

Сидоренко Максим Сергеевич СКБ181 и Орлов Кирилл Владимирович СКБ182

1 июня 2022 г.

Аннотация

Пусть N достаточно большое натуральное число. Предполагается, что известно частичное разложение числа $N^2 + 1$ на множители: $N^2 + 1 = F_4 * R_4$, где число F_4 разложено на простые множители, а R_4, возможно, не разложено, причем F_4 и R_4 - взаимно простые. Тогда приведенный ниже алгоритм определяет простоту числа N

1 Описание алгоритма

1. Проверяем, что исходное чиссло является натуральным и больше 1 (по определнию простых чисел)
2. Прямолинейно проверяем все делители N до 10^6
3. Если ничего не найдено, а исходное число меньше 10^{12}, то число простое
4. Иначе идём дальше и проверяем по малой теореме Ферма: ищем простые делители меньше 10 млн у числа $N-1$, затем для каждого простого делителя p случайно выбираем число m от 2 до $N-2$ такое, чтобы $m^{N-1}-1$ делилось на N, а $m^{\frac{N-1}{p}}-1$ не делилось (6 параграф статьи пункт (I))
5. Затем берём неразложенный остаток R1 числа N-1 и проверяем для $m^{\frac{N-1}{R_1}}-1$ (6 параграф статьи пункт (I))
6. Если предыдущие условия выполнены, то говорим, что число F1-псевдопростое, и если оно меньше 10^{21}, то оно простое
7. Если больше 10^{21}, то идём дальше и проверяем по последовательности Люка (6 параграф статьи пункт (III)). Т.е. задаём последовательность $x_n = a * x_{n-1} + b * x_{n-2}$, предварительно проверив, чтобы дискриминант многочлена $x^2 - a*x - b$ не был квадратичным вычетом (чтобы символ Якоби равнялся -1). Задав начальные условия $x_0 = 0, x_1 = 1$, считаем x_{N+1}и для каждого простого делителя N+1, меньшего 1 млн, считаем $x_{(N+1)/q}$ Если $\mathrm{x}_{N+1}N, \mathrm{x}_{(N+1)/q}$ не делится, то всё хорошо
8. Аналогично проверяем для $x_{(N+1)/R_2}$ (6 параграф статьи пункт (IV))

Таким образом мы находим простое разложение F_1, F_2 и определяем их псевдослучайность
Далее алгоритм проверки N^2+1, F_4, R_4 последовательно описан в следующих сносках непосредственно, а так же выделенных во вложении (в коде)

1.1 Сноска 1

(Из параграфа 4 статьи)
Выбираются такие натуральные числа C и D, чтобы $(D|N) = (C^2 - 16D|N) = -1$, где запись с

запись $(X|N)$ обозначает символ Якоби X по отношению к N. Далее для некоторых натуральных H и K вычисляются числа P_1, P_2, Q :

$$
\begin{aligned}
&P_1 = 4\left(2H^2 + HKC + 2K^2D\right)\\
&4P_2 = P_1^2 - 16D\\
&16Q = P_1^2 - 16\left(H^2C + K^2CD + 8HKD\right) + 16D
\end{aligned}
$$

Если обозначить $\triangle = 16$D, то имеет место разложение на множители:

$$
\begin{gathered}
P_1^2 + \Delta - 16Q + 2P_1\sqrt{\Delta} = 16(H + k\sqrt{D})^2(C + 4\sqrt{D})\\
E = \frac{1}{16}\left(P_1^2 + \Delta - 16Q + 2P_1\sqrt{\Delta}\right)\left(P_1^2 + \Delta - 16Q - 2P_1\sqrt{\Delta}\right) =\\
= 16\left(H^2 - K^2D\right)^2\left(C^2 - 16D\right)
\end{gathered}
$$

Следующие обозначения введены в параграфе 2 статьи на странице 158. Пусть ρ_1, ρ_2 - корни уравнения $x^2 - P_1x + P_2 = 0$, (1) и α_1, β_1 и α_2, β_2 - корни уравнений

$$x^2 - \rho_1 x + Q = 0 \text{ и } x^2 - \rho_2 x + Q = 0 \ (2)$$

Обозначим $\delta = \rho_1 = \rho_2, \triangle = \delta^2 = P_1^2 = -4 * P_2$ - дискриминант уравнения (1) , $E = (P_2 + 4Q)^2 - 4QP_1^2$ - дискриминант уравнения (2) $U_n = \frac{1}{\delta}\begin{vmatrix} 1 & \alpha_1^n + & \beta_1^n \\ 1 & \alpha_2^n + \beta_2^n & \end{vmatrix}$ $(n = 1, 2, \ldots)$ Из свойств чисел последовательностей U_n (параграф 3) отметим, что для любых натуральных m, n $U_n | U_{mn}$

1.2 Сноска 2

Для i = 1, 2, ... при выбранных C и D и натуральных H_i, K_i таких что НОД $\left(N, H_i^2 - K_i^2D\right) = 1$, вычисляются $P_1^{(i)} = 4\left(2H_i^2 + H_iK_iC + 2K_i^2D\right)$

$$
\begin{aligned}
&P_2^{(i)} = 4\left(2H_i^2 + H_iK_iC + 2K_i^2D\right)^2 - 4D\\
&Q^{(i)} = \left(2H_i^2 + H_iK_iC + 2K_iD\right)^2 - \left(H_i^2C + 8H_iK_iD + K_i^2CD\right) + D.
\end{aligned}
$$

Обозначается $\bar{F}_4 = F_4/2$ рассмотрим высказывания (α) : Для любого простого делителя $q|\bar{F}_4$ существуют такие K_i, H_i, что $N \mid U^{(i)}_{N^2+1}$ и НОД$\left(U^{(i)}_{\frac{N^2+1}{q}} N\right) = 1$

1.3 Сноска 3

Повторяем предыдущие шаги, однако рассматриваем высказывание (β): Для некотрых K_i, H_i $N \mid U^{(i)}_{N^2+1}$ и НОД$\left(U^{(i)}_{\frac{N^2+1}{R_4}}, N\right) = 1$

1.4 Сноска 4

Если (α) и (β) - верны, то N - простое (теорема 4, параграф 4 статьи)

2 Код программы

2.1 Основной код программы с комментариями

```
import math
import random
import numpy as np

def powmod(a, p, N):
    '''Возведение в степень и сравнение по модулю, т.е. a^(p-1) эквивалентно 1 по модулю p'''
    pow2 = 1
    result = 1
```

```
    while (pow2 <= p):
        pow2 *= 2
    pow2 //= 2
    while (pow2 > 0):
        if ((p // pow2) % 2 == 1):
            result *= a
        if (pow2 > 1):
            result **= 2
        pow2 //= 2
        result %= N
    return result

def matrix_powmod(a, p, N):
    '''Возведение в степень матрицы по модулю'''
    pow2 = 1
    result = np.eye(a.shape[0], dtype=int)
    while (pow2 <= p):
        pow2 *= 2
    pow2 //= 2
    while (pow2 > 0):
        if ((p // pow2) % 2 == 1):
            result = np.dot(result, a)
        if (pow2 > 1):
            result = np.dot(result, result)
        pow2 //= 2
        result %= N
    return result


def jacobi(P, Q):
    '''Проверка на число Якоби'''
    P %= Q
    R = 0
    result = 1
    if (Q % 2 == 0):
        return 0
    while (P > 1):
        while (P % 2 == 0):
            if ((Q ** 2 - 1) % 16 != 0):
                result *= -1
            P //= 2
        R = Q % P
        if ((Q - 1) % 2 == 1 and (P - 1) % 2 == 1):
            result *= -1
        Q = P
        P = R
    return result

def is_prime(N):
    '''Определяем - простое число или нет'''

    prime = True
    # Так как простые числа натуральные и больше 1
    if (N < 2):
        return False
```

```
# Ищем все делители до 10^6, если число меньше, то выполняем обычный алгоритм для поиска простых
for i in range(2, min([N, 1000000])):
    prime = prime and (N % i != 0)
    if not prime:
        return prime
# Если ничего не найдено, а исходное число меньше 10^12, то оно простое (ищем выше до половины)
if (N < 10 ** 12):
    return prime
divisors_f1 = []
f1 = 1
r1 = N - 1
# Ищем простые делители меньше 1 млн у числа N-1,
# затем для каждого простого делителя p случайно выбираем число m от 2 до N-2 такое,
# чтобы m^(N-1) - 1 делилось на N, а m^((N-1)/p) - 1 не делилось
for i in range(2, 1000000):
    while (r1 % i == 0):
        r1 //= i
        f1 *= i
        if (i != 2):
            divisors_f1.append(i)
random.seed()
##
a_example = random.randrange(2, N - 1)
prime = prime and (powmod(a_example, N - 1, N) == 1)
if not prime:
    return prime
suits_a = False
exists_a = False
is_primitive_root = True
for p in divisors_f1:
    for i in range(7):
        a1_example = random.randrange(2, N - 1)
        suits_a = (powmod(a1_example, N-1, N) == 1) and (powmod(a1_example, (N - 1)//p, N) != 1)
        exists_a = exists_a or suits_a
    is_primitive_root = is_primitive_root and exists_a
    exists_a = False
prime = prime and is_primitive_root
for i in range(12):
    # берём неразложенный остаток R1 числа N-1 и проверяем для m^((N-1)/R1) - 1
    a1_example = random.randrange(2, N - 1)
    suits_a = (powmod(a1_example, N-1, N) == 1) and (powmod(a1_example, (N - 1)//r1, N) != 1)
    exists_a = exists_a or suits_a
prime = prime and exists_a
if not prime:
    return prime
# Если у числа не делителей меньше, то очевидно, что нет смысла проверять числа меньше 10^21
if (N < 10 ** 21):
 return prime
# Если больше 10^21, то идём дальше и проверяем f2 по последовательности Люка
divisors_f2 = []
f2 = 1
r2 = N + 1
for i in range(2, 10000000):
    while (r2 % i == 0):
        r2 //= i
        f2 *= i
```

```
        if (i != 2):
            divisors_f2.append(i)
suits_a = False
exists_a = False
is_primitive_root = True
iters = 0
for q in divisors_f2:
    while (iters < 288 and not exists_a):
        # задаём последовательность x_n = ax_{n-1} + bx_{n-2}, предварительно проверив, чтобы дис
        # многочлена x^2 - ax - b не был квадратичным вычетом (чтобы символ Якоби равнялся -1)
        a2_example = random.randrange(2, N - 1)
        b2_example = random.randrange(2, N - 1)
        if (jacobi(a2_example ** 2 + 4 * b2_example, N) == -1):
            # Задав начальные условия x_0 = 0, x_1 = 1, считаем x_{N+1}
            # и для каждого простого делителя N+1, меньшего 1 млн, считаем x_{(N+1)/q}.
            # чтобы x_{N+1} делился на N, а x_{(N+1)/q} не делился,
            U_example = np.array([[0, 1], [b2_example, a2_example]])
            vec = np.array([0, 1])
            vecres = np.dot(matrix_powmod(U_example, N+1, N), vec)
            vecresq = np.dot(matrix_powmod(U_example, (N+1)//q, N), vec)
            suits_a = (vecres[0] == 0) and (vecresq[0] % N != 0)
            exists_a = exists_a or suits_a
        iters += 1
    is_primitive_root = is_primitive_root and exists_a
    exists_a = False
    iters = 0
prime = prime and is_primitive_root
exists_a = False
while (iters < 288 and not exists_a):
    a2_example = random.randrange(2, N - 1)
    b2_example = random.randrange(2, N - 1)
    if (jacobi(a2_example ** 2 + 4 * b2_example, N) == -1):
        # Аналогично предыдущему надо проверить для x_{(N+1)/R2}
        U_example = np.array([[0, 1], [b2_example, a2_example]])
        vec = np.array([0, 1])
        vecres = np.dot(matrix_powmod(U_example, N+1, N), vec)
        vecresq = np.dot(matrix_powmod(U_example, (N+1)//r2, N), vec)
        suits_a = (vecres[0] % N == 0 and vecresq[0] % N != 0)
        exists_a = exists_a or suits_a
    iters += 1
iters = 0
prime = prime and exists_a
if not prime:
    return prime
#дальше начинается проверка N^2 + 1, F4 и R4
divisors_f4 = []
f4 = 1
r4 = N ** 2 + 1
for i in range(2, 10000000):
    while (r4 % i == 0):
        r4 //= i
        f4 *= i
        if (i != 2):
            divisors_f4.append(i)
suits_a = False
exists_a = False
```

```
is_primitive_root = True
# сноска 1
for q in divisors_f4:
    while (iters < 288 and not exists_a):
        c_c = random.randrange(2, N - 1)
        d_d = random.randrange(2, N - 1)
        h_h = random.randrange(2, N - 1)
        k_k = random.randrange(2, N - 1)
        if (jacobi(d_d, N) == -1 and jacobi(c_c ** 2 - 16 * d_d, N) == -1):
            p1_p1 = 4 * (h_h ** 2 * 2 + 2 * h_h * k_k * c_c + 2 * k_k * k_k * d_d)
            p2_p2 = (p1_p1 ** 2 - 16 * d_d) // 4
            q_q = (p1_p1 ** 2) // 16 + d_d - (h_h * h_h * c_c + k_k * k_k * c_c * d_d +
                                              8 * h_h * k_k * d_d)
            # сноска 2
            U_example = np.array([[0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1],
                                  [-q_q ** 2, q_q * p1_p1, -p2_p2 - 2 * q_q, p1_p1]])
            vec_example = np.array([0, 1, p1_p1, p1_p1 ** 2 - p2_p2 - 3 * q_q])
            vecres = np.dot(matrix_powmod(U_example, N ** 2 + 1, N), vec_example)
            vecresq = np.dot(matrix_powmod(U_example, (N ** 2 + 1)//q, N), vec_example)
            suits_a = (vecres[0] % N == 0 and vecresq[0] % N != 0)
            exists_a = exists_a or suits_a
        iters += 1
    is_primitive_root = is_primitive_root and exists_a
    exists_a = False
    iters = 0

exists_a = False
prime = prime and is_primitive_root
# сноска 3
while (iters < 288 and not exists_a):
    c_c = random.randrange(2, N - 1)
    d_d = random.randrange(2, N - 1)
    h_h = random.randrange(2, N - 1)
    k_k = random.randrange(2, N - 1)
    if (jacobi(d_d, N) == -1 and jacobi(c_c ** 2 - 16 * d_d, N) == -1):
        p1_p1 = 4 * (h_h ** 2 * 2 + 2 * h_h * k_k * c_c + 2 * k_k * k_k * d_d)
        p2_p2 = (p1_p1 ** 2 - 16 * d_d) // 4
        q_q = (p1_p1 ** 2 + 16 * d_d - 16 * (h_h * h_h * c_c + k_k * k_k * c_c * d_d +
                                            8 * h_h * k_k * d_d)) // 16
        U_example = np.array([[0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1],
                              [-q_q ** 2, q_q * p1_p1, -p2_p2 - 2 * q_q, p1_p1]])
        vec_example = np.array([0, 1, p1_p1, p1_p1 ** 2 - p2_p2 - 3 * q_q])
        vecres = np.dot(matrix_powmod(U_example, N ** 2 + 1, N), vec_example)
        vecresq = np.dot(matrix_powmod(U_example, (N ** 2 + 1)//r4, N), vec_example)
        suits_a = (vecres[0] % N == 0 and vecresq[0] % N != 0)
        exists_a = exists_a or suits_a
    iters += 1
#сноска 4
prime = prime and exists_a
if not prime:
    return prime
return prime
```

2.2 Примеры в коде программы

В массиве чисел k в комментариях к каждому числу указан ответ - проверенный в wolframalpha

```
k = [-1, # false
     11099088999991232491,  # true
     1109908899999121133, # true
     1109908899999121111, #false
     0, # false
     1, # false
     2, # true
     3, # true
     10, # false
     17, # true
     15, # false
     35980201101257391549860360923563262525974949247991832187257385201689, # true
     1889, # true
     1900, # false
     1901, # true
     1109908899999123251, # true
     359802011012573915498603609235632625259749492479918321872573852016899, # true
     35980201101257391549860360923563262525974949247991832187257385201688, #false
     35980201101257391549860360923563262525974949247991832187257385202287, #true
     35980201101257391549860360923563262525974949247991832187257385202227, # false
     35980201101257391549860360923563262525974949247991832187257385203331, #true
     1234567898765432234567876543234567657 3, #true
     345123123267672138238128321372718382137127321839123, #false
     3451231232676721382381283213727183821371273218391234, #false
     34512312326767213823812832137271838213712732183912341232421412421412412421412, #false
     34512312326767213823812832137271838213712732183912341232421412421412412421327, #true
     359802011012573915498603609235632625259749492479918321872573852016890009927213123213232132312313
     359802011012573915498603609235632625259749492479918321872573852016890009927213123213232132312313
     35980201101257391549860360923563262525974949247991832187257385202287] # true
for i in k:
    n = i #35980201101257391549860360923563262525974949247991832187257385201689 - from example
    print('---------------------------')
    print(i)
    print(is_prime(n))
print('---------------------------')
```

2.3 Результаты

```
---------------------------
-1
False
---------------------------
11099088999991232491
True
---------------------------
1109908899999121133
True
---------------------------
1109908899999121111
False
---------------------------
0
False
---------------------------
1
False
```

```
---------------------------
2
True
---------------------------
3
True
---------------------------
10
False
---------------------------
17
True
---------------------------
15
False
---------------------------
359802011012573915498603609235632625259749492479918321872573852016 89
True
---------------------------
1889
True
---------------------------
1900
False
---------------------------
1901
True
---------------------------
1109908899999123251
True
---------------------------
3598020110125739154986036092356326252597494924799183218725738520168 99
True
---------------------------
359802011012573915498603609235632625259749492479918321872573852016 88
False
---------------------------
359802011012573915498603609235632625259749492479918321872573852022 87
True
---------------------------
35980201101257391549860360923563262525974949247991832187257385202 27
False
---------------------------
35980201101257391549860360923563262525974949247991832187257385203 31
True
---------------------------
1234567898765432234567876543234567 6573
True
---------------------------
3451231232676721382381283213727183821371273218391 23
False
---------------------------
3451231232676721382381283213727183821371273218391 234
False
---------------------------
345123123267672138238128321372718382137127321839123412324214124214124124 21412
```

```
False
---------------------------
34512312326767213823812832137271838213712732183912341232421412421412412421327
True
---------------------------
35980201101257391549860360923563262525974949247991832187257385201689000992721312321323213231231312313
False
---------------------------
35980201101257391549860360923563262525974949247991832187257385201689000992721312321323213231231311921
True
---------------------------
359802011012573915498603609235632625259749492479918321872573852022287
True
---------------------------
```

Где True - простое, False - не простое.