Отчёт по лабораторной работе №5

Вероятностные алгоритмы проверки чисел на простоту

Логинов Сергей НФИмд 01-22

Содержание

Список иллюстраций

Цель работы

Изучение алгоритмов Ферма, Соловэя-Штрассена, Миллера-Рабина.

Теоретические сведения

Для построения многих систем защиты информации требуются простые числа большой разрядности. В связи с этим актуальной является задача тестирования на простоту натуральных чисел.

Существует два типа критериев простоты: детерминированные и вероятностные. Детерминированные тесты позволяют доказать, что тестируемое число - простое. Практически применимые детерминированные тесты способны дать положительный ответ не для каждого простого числа, поскольку используют лишь достаточные условия простоты. Детерминированные тесты более полезны, когда необходимо построить большое простое число, а не проверить простоту, скажем, некоторого единственного числа. В отличие от детерминированных, вероятностные тесты можно эффективно использовать для тестирования отдельных чисел, однако их результаты, с некоторой вероятностью, могут быть неверными. К счастью, ценой количества повторений теста с модифицированными исходными данными вероятность ошибки можно сделать как угодно малой. На сегодня известно достаточно много алгоритмов проверки чисел на простоту. Несмотря на то, что большинство из таких алгоритмов имеет субэкспоненциальную оценку сложности, на практике они показывают вполне приемлемую скорость работы. На практике рассмотренные алгоритмы чаще всего по отдельности не применяются. Для проверки числа на простоту используют либо их комбинации, либо детерминированные тесты на простоту. Детерминированный алгоритм всегда действует по одной и той же схеме и гарантированно решает поставленную задачу. Вероятностный алгоритм использует генератор случайных чисел и дает не гарантированно точный ответ. Вероятностные алгоритмы в общем случае не менее эффективны, чем детерминированные (если используемый генератор случайных чисел всегда дает набор одних и тех же чисел, возможно, зависящих от входных данных, то вероятностный алгоритм становится детерминированным).

Тест Ферма

- Вход. Нечетное целое число $n \geq 5$.
- Выход. «Число n, вероятно, простое» или «Число n составное».

1. Выбрать случайное целое число $a, 2 \leq a \leq n - 2$.
2. Вычислить $r = a^{n-1}(modn)$
3. При $r = 1$ результат: «Число n, вероятно, простое». В противном случае результат: «Число n составное».

Тест Соловэя-Штрассена

- Вход. Нечетное целое число $n \geq 5$.
- Выход. «Число n, вероятно, простое» или «Число n составное».

1. Выбрать случайное целое число $a, 2 \leq a \leq n - 2$.
2. Вычислить $r = a^{(\frac{n-1}{2})}(modn)$
3. При $r \neq 1$ и $r \neq n - 1$ результат: «Число n составное».
4. Вычислить символ Якоби $s = (\frac{a}{n})$
5. При $r = s(modn)$ результат: «Число n, вероятно, простое». В противном случае результат: «Число n составное».

Тест Миллера-Рабина.

- Вход. Нечетное целое число $n \geq 5$.

- Выход. «Число n, вероятно, простое» или «Число n составное».

1. Представить $n-1$ в виде $n-1=2^s r$, где r - нечетное число
2. Выбрать случайное целое число $a, 2 \leq a \leq n-2$.
3. Вычислить $y=a^r (mod n)$
4. При $y \neq 1$ и $y \neq n-1$ выполнить действия
    - Положить $j=1$
    - Если $j \leq s-1$ и $y \neq n-1$ то
        - Положить $y=y^2 (mod n)$
        - При $y=1$ результат: «Число n составное».
        - Положить $j=j+1$
    - При $y \neq n-1$ результат: «Число n составное».
5. Результат: «Число n, вероятно, простое».

Выполнение работы

Реализация алгоритмов на языке Python

```
import random


# тест ферма где n число которое проверяется
# a test_count это количество прогонов
def ferma(n, test_count):
    for i in range(test_count):

        a = random.randint(2, n - 1)

        if (a ** (n - 1) % n != 1):
            print("Составное")
            return False
    print("Простое")
    return True




# функция для бинарного эксп
def modulo(base, exponent, mod):
    x = 1
```

```
    y = base
    while (exponent > 0):
        if (exponent % 2 == 1):
            x = (x * y) % mod

        y = (y * y) % mod
        exponent = exponent // 2

    return x % mod



# нахождение символа якоби
# алгоритм в принципе расписан в лабе

def calculateJacobian(a, n):
    if (a == 0):
        return 0  # (0/n) = 0

    ans = 1
    if (a < 0):

        # (a/n) = (-a/n)*(-1/n)
        a = -a
        if (n % 4 == 3):
            # (-1/n) = -1 if n = 3 (mod 4)
            ans = -ans

    if (a == 1):
        return ans  # (1/n) = 1
```

```
while (a):
    if (a < 0):

        # (a/n) = (-a/n)*(-1/n)
        a = -a
        if (n % 4 == 3):
            # (-1/n) = -1 if n = 3 (mod 4)
            ans = -ans

    while (a % 2 == 0):
        a = a // 2
        if (n % 8 == 3 or n % 8 == 5):
            ans = -ans

    # меняем местами
    a, n = n, a

    if (a % 4 == 3 and n % 4 == 3):
        ans = -ans
    a = a % n

    if (a > n // 2):
        a = a - n

if (n == 1):
    return ans

return 0
```

```
# тест соловея штрассена, для получения более-менее правильного результата,
# нужно прогнать его много раз,

def solovoyStrassen(p, iterations):
    if (p < 2):
        return False
    if (p != 2 and p % 2 == 0):
        return False

    for i in range(iterations):

        # генерация рандомного числа  a
        a = random.randrange(p - 1) + 1
        jacobian = (p + calculateJacobian(a, p)) % p
        mod = modulo(a, (p - 1) / 2, p)

        if (jacobian == 0 or mod != jacobian):
            return False

    return True

# Алгоритм Миллера Рабина
# true = значит простое
# false = значит составное
def miller_rabin(n):
    if n != int(n):
        print("не простое!")
```

```
    return False
n = int(n)
if n == 0 or n == 1 or n == 4 or n == 6 or n == 8 or n == 9:
    print("не простое!")
    return False

if n == 2 or n == 3 or n == 5 or n == 7:
    print("простое!")
    return True
s = 0
d = n - 1
while d % 2 == 0:
    d >>= 1
    s += 1
assert (2 ** s * d == n - 1)

def trial_composite(a):
    if pow(a, d, n) == 1:
        print("не простое!")
        return False
    for i in range(s):
        if pow(a, 2 ** i * d, n) == n - 1:
            print("не простое!")
            return False
    print("простое!")
    return True

for i in range(8):  # number of trials
    a = random.randrange(2, n)
```

```
        if trial_composite(a):
            print("не простое!")
            return False
    print("простое!")
    return True


def main():
    n = int(input("Введите число для ферма"))
    print("Тест ферма для числа ", n )
    ferma(n, 500)
    print("Тест миллера рабина")
    n = int(input("Введите число для миллера рабина"))
    miller_rabin(n)
    n = int(input("Введите число для соловея штрассена"))
    if (solovoyStrassen(n, 500)):
        print(n, "простое число ");
    else:
        print(n, "составное число");лида
```

Контрольный пример

```
In [2]:  1 main()

Введите число для ферма15898
Тест ферма для числа  15898
Составное
Тест миллера рабина
Введите число для миллера рабина15898
простое!
не простое!
Введите число для соловея штрассена15898
15898 составное число
```

Рис. 1: Работа алгоритмов

Выводы

Изучили алгоритмы Ферма, Соловэя-Штрассена, Миллера-Рабина.

Список литературы

1. Алгоритмы тестирования на простоту и факторизации
2. Алгоритм проверки на простоту