Лабораторная работа №5

Вероятностные алгоритмы проверки чисел на простоту

Хохлачева Яна Дмитриевна, НФИмд-02-22

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 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).

2.1 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 $r = a^{n-1}(mod n)$
3. При $r = 1$ результат: «Число n, вероятно, простое». В противном случае результат: «Число n составное».

Подробнее об алгоритме: [1]

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

Подробнее об алгоритме: [2]

2.3 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».

Подробнее об алгоритме: [3]

3 Выполнение работы

3.1 Реализация алгоритмов

```
import random


def ferma(n, test_count):
    for i in range(test_count):
        a = random.randint(2, n - 1)
        if (a ** (n - 1) % n != 1):
            print("Число n составное")
            return False
    print("Число n, вероятно, простое")
    return True



def find_jacobian(a, n):
    if (a == 0):
        return 0
    ans = 1
    if (a < 0):
        a = -a
        if (n % 4 == 3):
            ans = -ans
```

```
    if (a == 1):
        return ans

    while(a):
        if (a < 0):
            a = -a
            if (n % 4 == 3):
                ans = -ans
        while (a % 2 == 0):
            a = a // 2
            if (n % 8 == 3 or n % 8 == 5):
                ans = -ans

        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


def modul(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


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

    for i in range(iterations):
        a = random.randrange(p - 1) + 1
        jacobian = (p + find_jacobian(a, p)) % p
        mod = modul(a, (p - 1) / 2, p)
        if (jacobian == 0 or mod != jacobian):
            return False
    return True


def miller_rabin(n):
    if (n != int(n)):
        print("Число n составное")
        return False
    n = int(n)
```

```
if (n == 0 or n == 1 or n == 4 or n == 6 or n == 8 or n == 9):
    print("Число n составное")
    return False
if (n == 2 or n == 3 or n == 5 or n == 7):
    print("Число n, вероятно, простое")
    return True
s = 0
d = n - 1
while (d % 2 == 0):
    d >> 1
    s += 1
assert(2 ** s * d == n - a)

def probn_sost(a):
    if pow(a, d, n) == 1:
        print("Число n составное")
        return False
    for i in range(s):
        if pow(a, 2 ** i * d, n) == n - a:
            print("Число n составное")
            return False
    print("Число n, вероятно, простое")
    return True

for i in range(8):
    a = random.randrange(2, n)
    if probn_sost(a):
        print("Число n составное")
        return False
```

```
    print("Число n, вероятно, простое")
    return True


print("Тест Ферма")
n = int(input("Введите число для теста Ферма: "))
ferma(n, 500)

print("\nТест Миллера-Рабина")
n = int(input("Введите число для теста Миллера-Рабина: "))
miller_rabin(n)

print("\nСоловэя-Штрассена")
n = int(input("Введите число для теста Соловэя-Штрассена: "))
if (solovay_strassen(n, 500)):
    print("Число n, вероятно, простое")
else:
    print("Число n составное")
```

3.2 Пример работы алгоритма

```
Тест Ферма
Введите число для теста Ферма: 5
Число n, вероятно, простое

Тест Миллера-Рабина
Введите число для теста Миллера-Рабина: 5
Число n, вероятно, простое

Тест Соловэя-Штрассена
Введите число для теста Соловэя-Штрассена: 5
Число n, вероятно, простое
```

Рис. 3.1: Пример работы алгоритмов для n - простого

3.3 Пример работы алгоритма

```
Тест Ферма
Введите число для теста Ферма: 8
Число n составное

Тест Миллера-Рабина
Введите число для теста Миллера-Рабина: 8
Число n составное

Тест Соловэя-Штрассена
Введите число для теста Соловэя-Штрассена: 8
Число n составное
```

Рис. 3.2: Пример работы алгоритмов для n - составного

4 Выводы

В ходе выполнения работы мне удалось изучить алгоритмы Ферма, Соловэя-Штрассена, Миллера-Рабина, а также реализовать данные алгоритмы программно на языке Python.

Список литературы

1. Алгоритм Ферма [Электронный ресурс]. Википедия, 2021. URL: https://ru.wikipedia.org/wiki/Метод_факторизации_Ферма.
2. Алгоритм Соловэя-Штрассена [Электронный ресурс]. Википедия, 2020. URL: https://ru.wikipedia.org/wiki/Тест_Соловея_—_Штрассена.
3. Расширенный Миллера-Рабина [Электронный ресурс]. Википедия, 2021. URL: https://ru.wikipedia.org/wiki/Тест_Миллера_—_Рабина.