Лабораторная работа №5

Математические основы защиты информации и информационной безопасности

Леонтьева К. А., НПМмд-02-23

19 октября 2023

Российский университет дружбы народов

Москва, Россия

Цель лабораторной работы

1) Реализовать на языке программирования вероятностные алгоритмы проверки чисел на простоту

Пусть a - целое число. Числа ± 1, $\pm a$ называются **тривиальными делителями** числа a.

Целое число $p \in Z/\{0\}$ называется **простым**, если оно не является делителем единицы и не имеет других делителей, кроме тривиальных. В противном случае число $p \in Z/\{-1, 0, 1\}$ называется **составным**.

Детерминированный алгоритм всегда действует по одной и той же схеме и гарантированно решает поставленную задачу (или не дает никакого ответа).

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, зависящих от входных данных, то вероятностный алгоритм становится детерминированным).

Для проверки на простоту числа n вероятностным алгоритмом выбирают случайное число a $(1 < a < n)$ и проверяют условия алгоритма. Если число n не проходит тест по основанию a, то алгоритм выдает результат “Число n составное”, и число n действительно является составным.

Если же n проходит тест по основанию a, ничего нельзя сказать о том, действительно ли число n является простым. Последовательно проведя ряд проверок таким тестом для разных a и получив для каждого из них ответ “Число n, вероятно, простое”, можно утверждать, что число n является простым с вероятностью, близкой к 1.

- Реализуем тест Ферма

```
import numpy as np
import math
```

```
def Fermat(n):
    a = np.random.randint(2, (n - 2) + 1)
    r = (a ** (n - 1)) % n
    if r == 1:
        return "Число " + str(n) + ", вероятно, простое"
    else:
        return "Число " + str(n) + " составное"
```

Figure 1: Рис.1: Тест Ферма

Ход выполнения лабораторной работы

- Реализуем вычисление символа Якоби

```
def Jacobi(a,n):
    g = 1
    s = 0
    while (a != 0) and (a != 1):
        a1 = a
        k = 0
        if a1 % 2 != 0:
            k = 0
        if a1 % 2 == 0:
            while a1 % 2 == 0:
                a1 = int(a1 / 2)
            while a != (2 ** k) * a1:
                k = k + 1
        if k % 2 == 0:
            s = 1
        else:
            if (n % 8 == 1 % 8) or (n % 8 == -1 % 8):
                s = 1
            elif (n % 8 == 3 % 8) or (n % 8 == -3 % 8):
                s = -1
        if a1 == 1:
            return g * s

        if (n % 4 == 3 % 4) and (a1 % 4 == 3 % 4):
            s = -s
        a = n % a1
        n = a1
        g = g * s

    if a == 0:
        return 0
    else:
        return g
```

Figure 2: Рис.2: Вычисление символа Якоби

- Реализуем тест Соловэя-Штрассена

```
def S_SH(n):
    a = np.random.randint(2, (n - 2) + 1)
    r = (a ** ((n - 1) / 2)) % n
    if (r != 1) and (r != n - 1):
        return "Число " + str(n) + " составное"
    s = Jacobi(a,n)
    if r != s % n:
        return "Число " + str(n) + " составное"
    return "Число " + str(n) + ", вероятно, простое"
```

Figure 3: Рис.3: Тест Соловэя-Штрассена

- Реализуем тест Миллера-Рабина

```
def M_R(n):
    r = n - 1
    s = 0
    if r % 2 != 0:
        s = 0
    if r % 2 == 0:
        while r % 2 == 0:
            r = int(r / 2)
        while n - 1 != (2 ** s) * r:
            s = s + 1
    a = np.random.randint(2, (n - 2) + 1)
    y = (a ** r) % n
    if (y != 1) and (y != n - 1):
        j = 1
        if (j <= s - 1) and (y != n - 1):
            y = (y ** 2) % n
            if y == 1:
                return "Число " + str(n) + " составное"
            j = j + 1
        if y != n - 1:
            return "Число " + str(n) + " составное"
    return "Число " + str(n) + ", вероятно, простое"
```

Figure 4: Рис.4: Тест Миллера-Рабина

- Получили следующие результаты

```
for n in range (5, 50 , 2):
    print(Fermat(n))
    print(S_SH(n))
    print(M_R(n))
    print('-------')
```

```
Число 5, вероятно, простое
Число 5, вероятно, простое
Число 5, вероятно, простое
-------
Число 7, вероятно, простое
Число 7, вероятно, простое
Число 7, вероятно, простое
-------
Число 9 составное
Число 9 составное
Число 9 составное
-------
Число 11, вероятно, простое
Число 11, вероятно, простое
Число 11, вероятно, простое
-------
Число 13, вероятно, простое
Число 13, вероятно, простое
Число 13, вероятно, простое
```

Figure 5: Рис.5: Результаты

Ход выполнения лабораторной работы

- Получили следующие результаты

```
Число 15 составное
Число 15 составное
Число 15 составное
-------
Число 17, вероятно, простое
Число 17, вероятно, простое
Число 17 составное
-------
Число 19, вероятно, простое
Число 19, вероятно, простое
Число 19, вероятно, простое
-------
Число 21 составное
Число 21 составное
Число 21 составное
-------
Число 23, вероятно, простое
Число 23, вероятно, простое
Число 23, вероятно, простое
-------
Число 25 составное
Число 25, вероятно, простое
Число 25 составное
```

Figure 6: Рис.6: Результаты

Вывод

- В ходе выполнения данной лабораторной работы были реализованы вероятностные алгоритмы проверки чисел на простоту