Отчёт по лабораторной работе №5

Вероятностные алгоритмы проверки чисел на простоту

Каймакджыоглу Мерич Дорук

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной лабораторной работы является изучение и практическая реализация фундаментальных вероятностных алгоритмов для проверки чисел на простоту. Особое внимание уделяется тесту Ферма , тесту Соловэя-Штрассена и тесту Миллера-Рабина. Понимание и реализация этих алгоритмов имеет критическое значение, поскольку они составляют основу для генерации больших простых чисел, что является ключевой операцией в асимметричных криптосистемах, таких как RSA.

2 Задание

Реализовать программно все рассмотренные алгоритмы: - Алгоритм, реализующий тест Ферма - Алгоритм вычисления символа Якоби (как вспомогательный для следующего теста) - Алгоритм, реализующий тест Соловэя-Штрассена - Алгоритм, реализующий тест Миллера-Рабина - Создать главный скрипт, который импортирует эти алгоритмы и демонстрирует их работу на примерах.

3 Теоретическое введение

Целое число p называется **простым**, если оно не имеет других делителей, кроме тривиальных (± 1, $\pm p$). В противном случае число называется **составным**. Проверка чисел на простоту является составной частью генерации простых чисел, применяемых в криптографии с открытым ключом.

Алгоритмы проверки на простоту можно разделить на **детерминированные** и **вероятностные**. Детерминированный алгоритм всегда гарантированно решает задачу. Вероятностный алгоритм использует генератор случайных чисел и дает не гарантированно точный ответ.

Общая схема вероятностного теста: 1. Для числа n выбирается случайное «свидетельствующее» число a ($1 < a < n$). 2. Проверяется некоторое математическое условие. 3. Если n **не проходит** тест по основанию a, алгоритм выдает «Число n составное», и n действительно является составным. 4. Если n **проходит** тест, алгоритм выдает «Число n, вероятно, простое». Это не дает 100% гарантии.

Последовательно проведя t независимых проверок, можно утверждать, что число n является простым с высокой вероятностью. Вероятность ошибки (того, что составное число t раз будет объявлено простым) не превосходит $\frac{1}{2^t}$ (для тестов Соловэя-Штрассена и Миллера-Рабина).

1. **Тест Ферма**. Основан на Малой теореме Ферма: если p — простое число, то для любого a выполняется $a^{p-1} \equiv 1 (mod\ p)$. Алгоритм проверяет это равенство для случайного a.

2. **Тест Соловэя-Штрассена**. Основан на критерии Эйлера: нечетное n является простым тогда и только тогда, когда $a^{\frac{n-1}{2}} \equiv (\frac{a}{n})(mod\ n)$. Этот тест требует вычисления **символа Якоби** $(\frac{a}{n})$.

3. **Тест Миллера-Рабина**. Наиболее часто используемый тест. Он основан на том, что $n-1$ представляется в виде $n-1=2^s r$, где r нечетное, и использует более строгую проверку, связанную со свойствами квадратных корней из 1 по модулю n.

4 Выполнение лабораторной работы

Для выполнения задания был разработан единый скрипт на языке Python.

1. Алгоритм, реализующий тест Ферма (`fermat_test.py`)

```
import random

def fermat_test_single(n: int) -> str:
    a = random.randint(2,n-2)
    r = pow(a, n - 1, n)

    if r == 1:
        return "probably prime"
    else:
        return "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n < 5:
        return "input must be bigger than or = 5"
    if n % 2 == 0:
        return "composite even"

    for _ in range(t):
```

```
        if fermat_test_single(n) == "composite":
            return "composite"
    return "probably prime"
```

2. Алгоритм вычисления символа Якоби (jacobi_symbol.py)

```
def jacoby_symbol(a: int, n: int) -> int:
    if n < 3 or n % 2 == 0:
        raise ValueError("must be odd integer")
    a = a % n
    g = 1

    while a != 0:
        if a == 1:
            return g
        k = 0
        a1 = a
        while a1 % 2 == 0 and a1 > 0:
            a1 //= 2
            k += 1

        s = 1
        if k % 2 != 0:
            n_mod_8 = n % 8
            if n_mod_8 == 3 or n_mod_8 == 5:
                s = -1

        if a1 == 1:
            return g * s
```

```
        if (n % 4) == 3 and (a1 % 4) == 3:
            s = -s

        a = n % a1
        g = g * s

    return 0
```

3. Алгоритм, реализующий тест Соловэя-Штрассена (solovay_strassen_test.py)

```
import random
from jacobi_test import jacoby_symbol

def solovay_strassen_test_single(n: int) -> str:
    a = random.randint(2, n - 2)
    r = pow(a, (n - 1) // 2, n)

    if r != 1 and r != n - 1:
        return "composite"
    s = jacoby_symbol(a, n)

    if r == (s + n) % n:
        return "probably prime"
    else:
        return "composite"

def run_solovay_strassen_test(n: int, t: int) -> str:
    if n < 5:
```

```
        return "input must be bigger than or = 5"
    if n % 2 == 0:
        return "composite even"

    for _ in range(t):
        if solovay_strassen_test_single(n) == "composite":
            return "composite"
    return "probably prime"
```

4. Алгоритм, реализующий тест Миллера-Рабина (miller_rabin_test.py)

```
import random

def miller_rabin_test_single(n: int) -> str:
    s = 0
    r = n -1
    while r % 2 == 0:
        s += 1
        r //= 2

    a = random.randint(2, n - 2)
    y = pow(a, r, n)

    if y != 1 and y != n - 1:
        j = 1
        while j <= s - 1 and y != n - 1:
            y = pow(y, 2, n)
            if y == 1:
                return "composite"
```

```
            j += 1
        if y != n - 1:
            return "composite"
    return "probably prime"

def run_miller_rabin_test(n: int, t: int) -> str:
    if n < 5:
        return "input must be bigger than or = 5"
    if n % 2 == 0:
        return "composite even"

    for _ in range(t):
        if miller_rabin_test_single(n) == "composite":
            return "composite"
    return "probably prime"
```

5. Главный файл и демонстрация работы (lab05_primality_tests.py)

```
from fermat_test import run_feramn_test
from sol_stra_test import run_solovay_strassen_test
from mill_rab_test import run_miller_rabin_test

if __name__ == "__main__":
    try:
        n_input = input("enter n: ")
        n = int(n_input)
        t_input = input ("enter t: ")
        t = int(t_input)
```

```
        if n < 5 or n % 2 == 0:
            print(f"[illegible]: {n} [illegible] [illegible] [illegible] [illegible] >= 5.")
            if n == 2 or n == 3:
                print(f"{n} - [illegible] [illegible].")
            elif n % 2 == 0:
                print(f"{n} - [illegible] [illegible] ([illegible]).")
        elif t < 1:
            print("[illegible]: [illegible] [illegible] t [illegible] [illegible] >= 1.")
        else:
            print(f"\n--- [illegible] [illegible] {n} (c {t} [illegible]) ---")

            result_fermat = run_feramn_test(n, t)
            print(f"1. [illegible] [illegible]: \t\t{result_fermat}")

            result_ss = run_solovay_strassen_test(n, t)
            print(f"2. [illegible] [illegible]-[illegible]: \t{result_ss}")

            result_mr = run_miller_rabin_test(n, t)
            print(f"3. [illegible] [illegible]-[illegible]: \t{result_mr}")

            print("\n--- [illegible] ---")
            if result_mr == "probably prime":
                print(f"[illegible] {n}, [illegible], [illegible].")
                print(f"([illegible] [illegible] < 1 / (4^{t}))")
            else:
                print(f"[illegible] {n} - [illegible].")
    except ValueError:
        print("[illegible]: [illegible], [illegible] [illegible] [illegible] [illegible].")
```

```
    except Exception as e:
        print(f"□□□□□□□□□ □□□□□□: {e}")
```

4.1 Результаты выполнения

Пример 1: Число Кармайкла (561)

Числа Кармайкла — это составные числа, которые проходят тест Ферма для любого a.

```
enter n: 561
enter t: 20

--- □□□□□□□□□□□□ □□□□□ 561 (с 20 □□□□□□□□□□) ---
1. □□□□ □□□□□:           composite
2. □□□□ □□□□□□□-□□□□□□□□□:      composite
3. □□□□ □□□□□□□-□□□□□□:          composite

--- □□□□□ ---
□□□□□ 561 - □□□□□□□□□.
```

5 Выводы

В ходе выполнения данной лабораторной работы были успешно реализованы все три вероятностных алгоритма проверки на простоту, указанные в задании: тест Ферма , тест Соловэя-Штрассена и тест Миллера-Рабина. Также был реализован алгоритм вычисления символа Якоби, необходимый для теста Соловэя-Штрассена.

Разработанные функции, объединенные в модульную структуру, корректно классифицируют числа как «составные» или «вероятно, простые». Демонстрация на числе Кармайкла (561) показала практическую слабость теста Ферма и надежность тестов Миллера-Рабина и Соловэя-Штрассена.

Наиболее важным результатом является практическая реализация теста Миллера-Рабина , который является индустриальным стандартом и чаще всего используется на сегодняшний день для генерации простых чисел в криптографии.

Эта работа позволила углубить понимание математических основ, на которых строится информационная безопасность, в частности, механизм генерации больших простых чисел. Эта задача неразрывно связана с темой предыдущей лабораторной работы (вычисление модульного обратного элемента), так как обе являются ключевыми компонентами для работы асимметричных криптосистем.

Список литературы

::: {#Методические указания к лабораторной работе №5} :::