Лабораторная работа №5

Вычисление наибольшего общего делителя

- Кюнкриков Даниил Саналович
- студент уч. группы НПИмд-01-24
- Российский университет дружбы народов
- 1132249574@pfur.ru
- https://github.com/DanzanK/2025-2026_math-sec/tree/main

Объект и предмет исследования

- Объект исследования - Алгоритмы проверки чисел на простоту
- Предмет исследования - алгоритмы:
  - Тест Ферма
  - вычисление символа Якоби
  - Тест Соловэя-Штрассена
  - Тест Миллера-Рабина

Цели и задачи

- Цель работы: Программная реализация алгоритмов проверки чисел на простоту на языке программирования python
- Задачи: реализовать программно алгоритмы:
  - Тест Ферма
  - вычисление символа Якоби
  - Тест Соловэя-Штрассена
  - Тест Миллера-Рабина
- продемонстрировать работу алгоритмов

Теоретическая часть

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.

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 что число n составное 4. Если n проходит тест по основанию a, алгоритм выдает что число n вероятно простое

Процесс выполнения работы

Реализация теста Ферма

```
def fermat_test_single(n: int) -> str:
    a = random.randint(2, n -2)
    r = pow(a, n -1, n)
    return "possible prime" if r == 1 else "composite"

def run_feramn_test(n: int, t: int) -> str:
    if n< 5:
        return "input должен быть >= 5"
    if n % 2 == 0:
        return "composite even"
    for _ in range(t):
        if fermat_test_single(n) == "composite":
            return "composite"
    return "possible prime"
```

Реализация алгоритма вычисления символа Якоби

```
def jacobi(a:int, n: int) -> int:
    if n <= 0 or n % 2 == 0:
        raise ValueError("n должно бытьнечетным и положительным")
    a %= n
    result = 1
    while a != 0:
        while a % 2 == 0:
            a //= 2
            r = n % 8
            if r == 3 or r == 5:
                result = - result
        a, n = n, a

        if a % 4 == 3 and n % 4 == 3:
            result = -result
        a %= n

    return result if n == 1 else 0
```

Реализация теста Соловэя-Штрассена

```
def solov_strass_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 = jacobi(a,n)
    return "possible prime" if r == (s+n) % n else "composite"

def run_solov_strass_test(n: int, t: int) -> str:
    if n < 5:
        return "input должен быть >=5"
    if n%2 ==0:
        return "composite even"
    for _ in range(t):
        if solov_strass_test_single(n) == "composite":
            return "composite"
    return " possible prime"
```

Реализация теста Миллера-Рабина

```
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 "possible prime"

def run_miller_rabin_test(n: int, t: int) -> str:
    if n < 5:
        return "input должен быть >=5"
    if n%2 ==0:
        return "composite even"
    for _ in range(t):
        if miller_rabin_test_single(n) == "composite":
            return "composite"
    return " possible prime"
```

Код запуска тестов алгоритмов

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

        if n < 5 or n % 2 ==0:
            print("Err n < 5")
            if n == 2 or n ==3:
                print("n - простое число")
            elif n % 2 == 0:
                print("n - составное число")
        elif t < 1:
            print("t < 1")
        else:
            result_ferma = run_feramn_test(n,t)
            print(f"тест Ферма: \t\t{result_ferma}")
            result_solov_strass = run_solov_strass_test(n,t)
            print(f"тест Соловэя-Штрассена: \t{result_solov_strass}")
            result_mill_rab = run_miller_rabin_test(n,t)
            print(f"тест Миллера-Рабина: \t{result_mill_rab}")

            if result_mill_rab=="possible prime":
                print("n - вероятно простое")
            else:
                print("составное")
    except ValueError:
        print("Error")
```

Результат работы программы

```
введи n:131
введи t:20
тест Ферма:             possible prime
тест Соловэя-Штрассена:          possible prime
тест Миллера-Рабина:     possible prime
```

Вывод

В ходе выполнения лабораторной работы были успешно реализованы все четыре алгоритма по проверки чисел на простоту. Разаработанные функции корректно классифицируют числа как “составные” и “вероятно простые”