Лабораторная работа: Вероятностные алгоритмы проверки чисел на простоту

Цель работы

Реализовать и протестировать вероятностные алгоритмы проверки чисел на простоту:

1. Алгоритм, реализующий тест Ферма.
2. Алгоритм вычисления символа Якоби.
3. Алгоритм, реализующий тест Соловея-Штрассена.
4. Алгоритм, реализующий тест Миллера-Рабина.

Эти алгоритмы позволяют определить, является ли заданное число **вероятно простым** или **составным** с высокой степенью надёжности.

---

1. Алгоритм, реализующий тест Ферма

Описание: Тест Ферма основан на **малой теореме Ферма**. Если (p) — простое число, и (a) — любое целое число, такое что (1 < a < p-1), то выполняется следующее сравнение: [a^{p-1} \equiv 1 \ (\text{mod} \ p)] Если это условие не выполняется для случайного значения (a), то (n) является составным.

Реализация на Python:

```
import random

def fermat_primality_test(n, k=5):
    """Проверяет, является ли число n вероятно простым с помощью теста Ферма."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    for _ in range(k):
        a = random.randint(2, n - 2)
        if pow(a, n - 1, n) != 1:
            return False  # n составное
    return True  # n вероятно простое
```

2. Алгоритм вычисления символа Якоби

```
def jacobi_symbol(a, n):
    """Вычисление символа Якоби (a/n)."""
    result = 1
    while a != 0:
```

```
        while a % 2 == 0:
            a //= 2
            if n % 8 in [3, 5]:
                result = -result
        a, n = n, a  # Меняем a и n местами
        if a % 4 == 3 and n % 4 == 3:
            result = -result
        a %= n
    return result if n == 1 else 0
```

3. Алгоритм, реализующий тест Соловея-Штрассена

Описание: Тест Соловея-Штрассена проверяет простоту числа, используя символ Якоби и критерий Эйлера.

```
def solovay_strassen_test(n, k=5):
    """Проверяет, является ли число n вероятно простым с помощью теста Соловея-Штрассена."""
    if n < 2:
        return False
    if n != 2 and n % 2 == 0:
        return False
    for _ in range(k):
        a = random.randint(2, n - 2)
        x = jacobi_symbol(a, n)
        if x == 0 or pow(a, (n - 1) // 2, n) != (x % n):
            return False  # n составное
    return True  # n вероятно простое
```

4. Алгоритм, реализующий тест Миллера-Рабина

```
def miller_rabin_primality_test(n, k=5):
    """Проверяет, является ли число n вероятно простым с помощью теста Миллера-Рабина."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False

    # Разложение n - 1 в вид 2^s * d
    s = 0
    d = n - 1
```

```
    while d % 2 == 0:
        d //= 2
        s += 1

    for _ in range(k):
        a = random.randint(2, n - 2)
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(s - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False  # n составное
    return True  # n вероятно простое
```

Пример использования

Данный пример позволяет пользователю вводить число n , количество итераций k для повышения надёжности, и значение a для вычисления символа Якоби.

```
# Ввод от пользователя
n = int(input("Введите число для проверки на простоту: "))
k = int(input("Введите количество итераций для повышения надёжности: "))

# Вывод результатов
print("\nРезультаты проверки числа", n, "на простоту:")
print("Тест Ферма:", "Вероятно простое" if fermat_primality_test(n, k) else
"Составное")

# Символ Якоби вычисляется для примера с a = 2, можно изменить на другое значение
a_for_jacobi = int(input("\nВведите значение a для вычисления символа Якоби (a/n):
"))
print(f"Символ Якоби для a={a_for_jacobi} и n={n}:", jacobi_symbol(a_for_jacobi,
n))

print("Тест Соловея-Штрассена:", "Вероятно простое" if solovay_strassen_test(n, k)
else "Составное")
print("Тест Миллера-Рабина:", "Вероятно простое" if miller_rabin_primality_test(n,
k) else "Составное")
```