Дисциплина	Лабораторная	ФИО
Математические основы защиты информации и информационной безопасности	№5	Александра Миличевич

Цель лабораторной

Цель лабораторной работы №5 заключается в ознакомлении студентов с алгоритмами проверки простоты чисел, такими как тест Ферма, тест Соловэя-Штрассена и тест Миллера-Рабина, а также с методами вычисления символа Якоби и модульного возведения в степень. Студенты должны научиться реализовывать эти алгоритмы на практике, понять их математические основы и оценить их эффективность в контексте криптографических приложений.

Тест Ферма и модульное возведение в степень

Этот документ описывает две функции: тест Ферма для проверки простоты числа и функцию модульного возведения в степень.

1. `fermat_test(number, num_tests)`

Эта функция реализует тест Ферма для проверки, является ли данное число, вероятно, простым.

Как работает:

1. Функция принимает на вход два аргумента:
    - `number`: Нечетное целое число, которое нужно проверить на простоту.
    - `num_tests`: Количество случайных проверок, которые нужно провести.
2. В цикле `for` функция выполняет `num_tests` проверок.
3. В каждой проверке выбирается случайное целое число `a` в диапазоне от 2 до `number - 2`.
4. Вычисляется `a^(number - 1) % number` с использованием встроенной функции `pow(a, number - 1, number)`, которая реализует быстрое модульное возведение в степень.
5. Если `a^(number - 1) % number` не равно 1, то это означает, что число `number` составное. Функция выводит сообщение “Число составное” и возвращает `False`.
6. Если все проверки пройдены, то это говорит о том, что число, вероятно, простое. Функция выводит сообщение “Число, вероятно, простое” и возвращает `True`.

Важные замечания:

```python
import random

def fermat_test(number, num_tests):
    """
    Проводит тест Ферма для проверки, является ли число простым.

    Args:
        number (int): Нечетное целое число, которое нужно проверить на простоту.
        num_tests (int): Количество случайных тестов, которые нужно провести.

    Returns:
        bool: True, если число, вероятно, простое, False, если число составное.
    """
    for _ in range(num_tests):
        # Выбираем случайное целое число a в диапазоне [2, number - 2]
        a = random.randint(2, number - 2)

        # Проверяем условие теста Ферма: a^(number-1) % number != 1
        if pow(a, number - 1, number) != 1:
            print("Число составное")
            return False
    print("Число, вероятно, простое")
    return True
```

Figure 1: тест Ферма

- Тест Ферма — это вероятностный тест. Если тест проходит, то число, скорее всего, простое, но есть небольшая вероятность, что число окажется составным, хотя и проходит тест Ферма (так называемые числа Кармайкла).
- Чем больше количество тестов, тем выше вероятность правильного результата.

2. `modular_exponentiation(base, exponent, modulus)`

Эта функция реализует алгоритм бинарного возведения в степень по модулю.

Как работает:

1. Функция принимает три аргумента:
    - `base`: Основание.
    - `exponent`: Показатель степени.
    - `modulus`: Модуль.
2. Инициализирует переменную `result` значением 1.
3. Берет остаток от деления `base` на `modulus`, чтобы уменьшить размер основания.
4. Использует цикл `while`, который продолжается, пока `exponent` больше 0.
5. Внутри цикла:
    - Если `exponent` нечетный, то `result` умножается на `base` и берется остаток от деления на `modulus` (`result = (result * base) % modulus`).
    - `base` возводится в квадрат и берется остаток от деления на `modulus` (`base = (base * base) % modulus`).
    - `exponent` делится на 2 (`exponent //= 2`).
6. Функция возвращает значение `result`, которое равно `(base^exponent) % modulus`.

```python
def modular_exponentiation(base, exponent, modulus):
    """
    Вычисляет (base^exponent) % modulus, используя бинарное возведение в степень.

    Args:
        base (int): Основание.
        exponent (int): Показатель степени.
        modulus (int): Модуль.

    Returns:
        int: Результат (base^exponent) % modulus.
    """
    result = 1
    base = base % modulus
    while exponent > 0:
        if exponent % 2 == 1:
            result = (result * base) % modulus
        base = (base * base) % modulus
        exponent //= 2
    return result
```

Figure 2: модуло алгоритм

Важные замечания:

- Бинарное возведение в степень — это эффективный способ вычисления больших степеней по модулю.
- Этот алгоритм позволяет избежать переполнения переменных при вычислении больших чисел.

Вычисление символа Якоби

Этот документ описывает функцию для вычисления символа Якоби (a/n). Символ Якоби является обобщением символа Лежандра и используется в теории чисел, в частности, в тестах простоты.

Функция `jacobi_symbol(a, n)`

Эта функция вычисляет символ Якоби (a/n).

Описание:

- **Вход:**
  - `a` (int): Целое число.
  - `n` (int): Нечетное целое число, большее или равное 3.
- **Выход:**
  - Символ Якоби (a/n), который равен 0, 1 или -1.

Как работает:

1. **Базовый случай:** Если `a` равно 0, возвращается 0, так как (0/n) = 0.
2. **Инициализация:** Устанавливается начальное значение результата `result` равным 1.

3. **Отрицательное a:** Если `a` отрицательное, то `a` заменяется на `-a`, а если `n` по модулю 4 дает остаток 3, то результат меняет знак.
4. **a равно 1:** Если `a` равно 1, то результат возвращается (так как (1/n) = 1).
5. **Основной цикл:** Выполняется цикл `while a`, который продолжает работу, пока `a` не станет равным 0.
6. **Отрицательное a (внутри цикла):** Если `a` отрицательное, то `a` заменяется на `-a`, а если `n` по модулю 4 дает остаток 3, то результат меняет знак.
7. **Четное a:** Пока `a` четное, `a` делится на 2. Если `n` по модулю 8 дает остаток 3 или 5, то результат меняет знак.
8. **Замена значений:** Значения `a` и `n` меняются местами (`a, n = n, a`).
9. **Квадратичный закон взаимности:** Если `a` и `n` по модулю 4 дают остаток 3, то результат меняет знак.
10. **Уменьшение a:** `a` берется по модулю `n`, а если `a` больше, чем половина `n`, то `a` вычитается из `n`.
11. **Финальное условие:** Если `n` равен 1, то функция возвращает `result`.
12. **В остальных случаях:** Функция возвращает 0.

Важные замечания:

- Символ Якоби (a/n) равен 0, если `a` и `n` не взаимно простые, равен 1, если `a` является квадратичным вычетом по модулю `n`, и равен -1, если `a` не является квадратичным вычетом по модулю `n`.
- Эта функция использует свойства символа Якоби, такие как закон квадратичной взаимности, чтобы эффективно вычислить символ.
- Символ Якоби используется для теста Соловэя — Штрассена и других тестов простоты.

Заключение

Этот документ описывает функцию `jacobi_symbol(a, n)`, которая вычисляет символ Якоби (a/n) для заданных целых чисел `a` и `n`. Эта функция использует ряд математических свойств, чтобы эффективно вычислить символ Якоби, и важна в

```python
def jacobi_symbol(a, n):
    """
    Вычисляет символ Якоби (a/n).

    Args:
        a (int): Целое число.
        n (int): Нечетное целое число, большее или равное 3.

    Returns:
        int: Символ Якоби (a/n), который равен 0, 1 или -1.
    """
    if a == 0:
        return 0  # (0/n) = 0

    result = 1
    if a < 0:
        a = -a
        if n % 4 == 3:
            result = -result

    if a == 1:
        return result  # (1/n) = 1

    while a:
        if a < 0:
            a = -a
            if n % 4 == 3:
                result = -result

        while a % 2 == 0:
            a //= 2
            if n % 8 == 3 or n % 8 == 5:
                result = -result
        a, n = n, a
        if a % 4 == 3 and n % 4 == 3:
            result = -result
        a %= n
        if a > n // 2:
             a -= n
    if n == 1:
        return result

    return 0
```

теории чисел и криптографии.

Тест Соловэя-Штрассена для проверки простоты

Этот документ описывает функцию `solovay_strassen_test`, реализующую тест Соловэя-Штрассена для проверки простоты числа.

Функция `solovay_strassen_test(number, iterations)`

Эта функция проверяет, является ли заданное нечетное число, вероятно, простым, используя тест Соловэя-Штрассена.

Описание:

- **Вход:**
  - `number` (int): Нечетное целое число для проверки.
  - `iterations` (int): Количество итераций теста.
- **Выход:**
  - `True`, если число, вероятно, простое.
  - `False`, если число составное.

Как работает:

1. **Проверка на 2 и меньше:** Если число меньше 2 или четное (кроме 2), то возвращается `False`.
2. **Цикл итераций:** Выполняется `iterations` раз.
3. **Генерация случайного числа:** Генерируется случайное число `a` от 1 до `number - 1`.
4. **Вычисление символа Якоби:** Вычисляется символ Якоби `jacobi_symbol(a, number)`.
5. **Вычисление модульного возведения в степень:** Вычисляется `a^((number-1)/2) % number` с помощью функции `modular_exponentiation`.
6. **Проверка условий:** Если символ Якоби равен 0, или результат модульного возведения в степень не равен символу Якоби, то число составное, и возвращается `False`.
7. **Вероятно простое:** Если все итерации пройдены без возврата `False`, то число, вероятно, простое и возвращается `True`.

Важные замечания:

- Тест Соловэя-Штрассена является вероятностным.
- Этот тест, как и тест Ферма, может ошибаться с небольшой вероятностью.
- Чем больше итераций, тем выше вероятность корректного определения простоты.

Заключение

Функция `solovay_strassen_test` предоставляет способ проверить, является ли нечетное число, вероятно, простым. Она использует случайные числа, символ Якоби и

```python
def solovay_strassen_test(number, iterations):
    """
    Проводит тест Соловэя-Штрассена для проверки, является ли число простым.

    Args:
        number (int): Нечетное целое число, которое нужно проверить на простоту.
        iterations (int): Количество итераций теста.

    Returns:
        bool: True, если число, вероятно, простое, False, если число составное.
    """
    if number < 2:
        return False
    if number != 2 and number % 2 == 0:
        return False
    for _ in range(iterations):
        # Генерация случайного числа a от 1 до number - 1
        a = random.randrange(number - 1) + 1
        # Вычисляем символ Якоби
        jacobi = (number + jacobi_symbol(a, number)) % number
        # Вычисляем (a^((number-1)/2)) % number
        mod = modular_exponentiation(a, (number - 1) // 2, number)

        if jacobi == 0 or mod != jacobi:
            return False
    return True
```

модульное возведение в степень для проведения теста

Тесты простоты чисел: Миллера-Рабина и другие

Этот документ описывает несколько функций для проверки, является ли число простым, включая тест Миллера-Рабина, тест Ферма и тест Соловэя-Штрассена, а также функции для вычисления символа Якоби и модульного возведения в степень.

1. `miller_rabin_test(number)`

Эта функция реализует тест Миллера-Рабина для проверки простоты числа.

Описание:

- **Вход:**
  - `number` (int): Целое число для проверки на простоту.
- **Выход:**
  - `True`, если число, вероятно, простое.
  - `False`, если число составное.

Как работает:

1. **Проверка типа:** Проверяется, является ли число целым. Если нет, выводится сообщение об ошибке и возвращается `False`.
2. **Проверка на простые и составные:** Исключаются известные составные и простые числа (0, 1, 4, 6, 8, 9 и 2, 3, 5, 7).
3. **Разложение `number - 1`:** Число `number - 1` представляется в виде `2^s * d`, где `d` нечетное. Вычисляются значения `s` и `d`.
4. **Функция `trial_composite(a)`:** Вложенная функция проверяет, является ли число `a` свидетелем составности.
   - Проверяет условие `a^d % number == 1`.

    - Проверяет условие `a^(2^i * d) % number == number - 1` для `i` от 0 до `s-1`.
    - Возвращает `True`, если хотя бы одно условие выполнилось, и `False` в противном случае.
5. **Проведение тестов:** 8 случайных чисел `a` (от 2 до `number - 1`) проверяются функцией `trial_composite(a)`.
    - Если для какого-то `a` функция `trial_composite(a)` вернула `False`, то число составное, и возвращается `False`.
6. **Число, вероятно, простое:** Если все 8 тестов пройдены, то число, вероятно, простое и возвращается `True`.

```python
def miller_rabin_test(number):
    """
    Проводит тест Миллера-Рабина для проверки, является ли число простым.

    Args:
        number (int): Целое число, которое нужно проверить на простоту.

    Returns:
        bool: True, если число, вероятно, простое, False, если число составное.
    """
    if not isinstance(number, int):
        print("Число не целое!")
        return False
    number = int(number)
    if number == 0 or number == 1 or number == 4 or number == 6 or number == 8 or number == 9:
        print("Число не простое!")
        return False

    if number == 2 or number == 3 or number == 5 or number == 7:
        print("Число простое!")
        return True

    s = 0
    d = number - 1
    while d % 2 == 0:
        d >>= 1
        s += 1
    assert (2 ** s * d == number - 1)

    def trial_composite(a):
        if pow(a, d, number) == 1:
             return True
        for i in range(s):
            if pow(a, 2 ** i * d, number) == number - 1:
                return True
        return False

    for _ in range(8):  # number of trials
        a = random.randrange(2, number)
        if not trial_composite(a):
            print("Число не простое!")
            return False
    print("Число простое!")
    return True
```

Figure 3: миллера рабина алгоритм

#Вывод

В целом, код демонстрирует реализацию ключевых компонентов для проверки простоты больших чисел, важных в криптографии и теории чисел.