

Отчёт по лабораторной работе №6

Математические основы защиты информации и информационной безопасности

Разложение чисел на множители

Выполнил: Мануэл Марсия Педру,
НФИмд-02-25, 1032255503

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Реализация вычисления символа Якоби	6
3	Список литературы. Библиография	10

Список иллюстраций

2.1	Реализация р-Метода Полларда	6
2.2	Реализация р-Метода Полларда	7
2.3	Реализация р-Метода Полларда	7
2.4	Реализация р-Метода Полларда	8
2.5	Реализация р-Метода Полларда	8
2.6	Реализация р-Метода Полларда	8
2.7	Проверка	9

Список таблиц

1 Цель работы

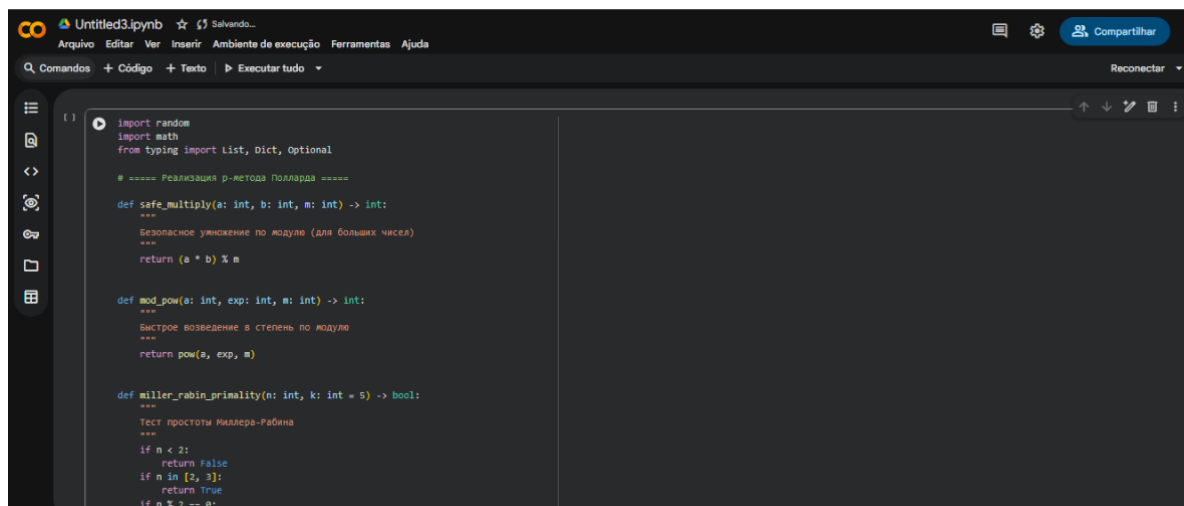
Изучить алгоритм разложения чисел на множители и научиться его реализовывать.

2 Выполнение лабораторной работы

2.1 Реализация вычисления символа Якоби

Ро-алгоритм — предложенный Джоном Поллардом в 1975 году алгоритм, служащий для факторизации (разложения на множители) целых чисел. Данный алгоритм основывается на алгоритме Флойда поиска длины цикла в последовательности и некоторых следствиях из парадокса дней рождения. Алгоритм наиболее эффективен при факторизации составных чисел с достаточно малыми множителями в разложении.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.1 - рис. 2.6):



```
import random
import math
from typing import List, Dict, Optional

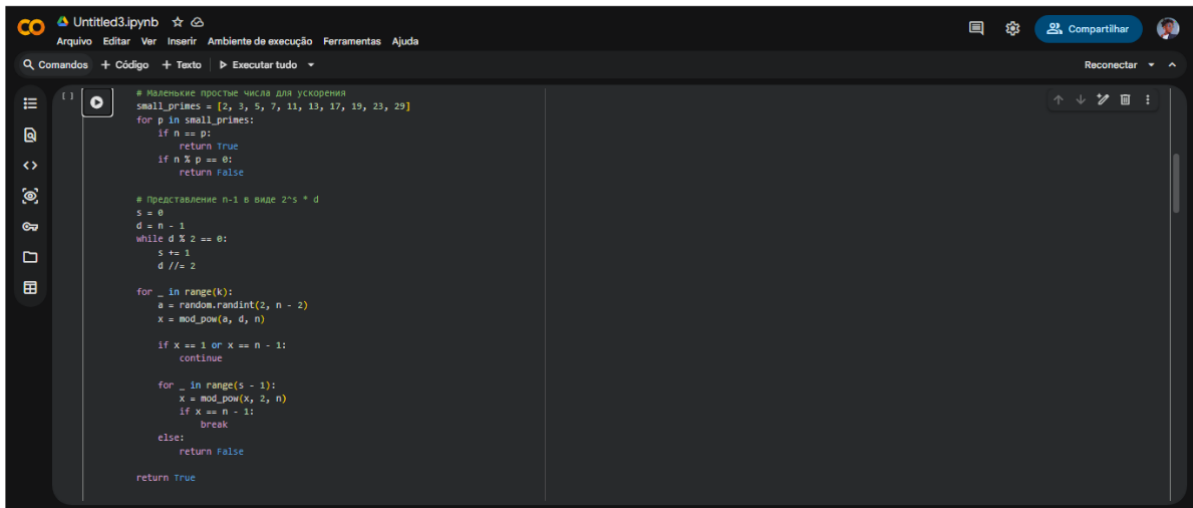
# ===== Реализация р-метода Полларда =====

def safe_multiply(a: int, b: int, m: int) -> int:
    """
    Безопасное умножение по модулю (для больших чисел)
    """
    return (a * b) % m

def mod_pow(a: int, exp: int, m: int) -> int:
    """
    Быстрое возведение в степень по модулю
    """
    return pow(a, exp, m)

def miller_rabin_primality(n: int, k: int = 5) -> bool:
    """
    Тест простоты Миллера-Рабина
    """
    if n < 2:
        return False
    if n in [2, 3]:
        return True
    if n % 2 == 0:
```

Рис. 2.1: Реализация р-Метода Полларда



```

# маленькие простые числа для ускорения
small_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
for p in small_primes:
    if n == p:
        return True
    if n % p == 0:
        return False

# представление n-1 в виде 2^s * d
s = 0
d = n - 1
while d % 2 == 0:
    s += 1
    d //= 2

for _ in range(k):
    a = random.randint(2, n - 2)
    x = mod_pow(a, d, n)

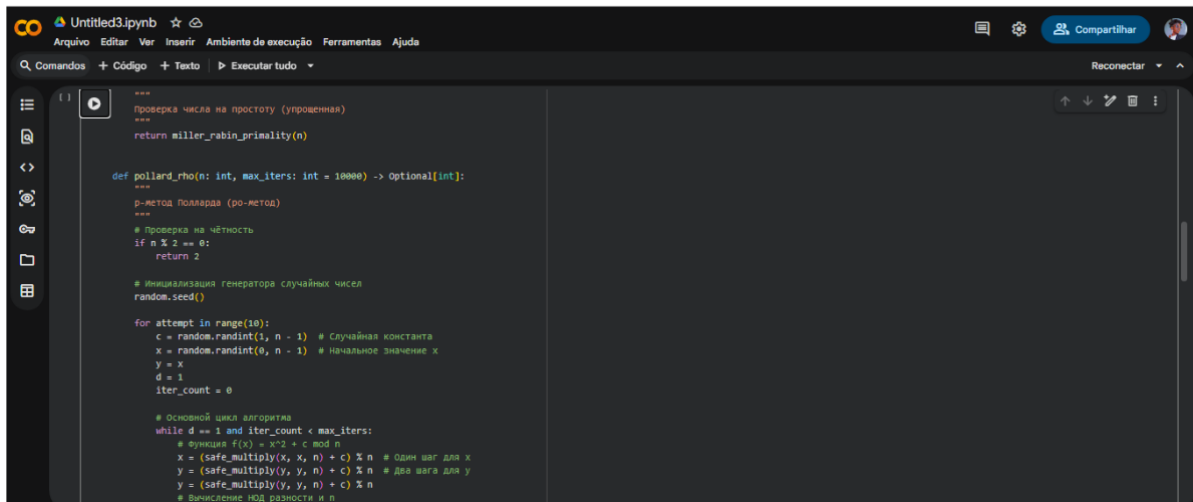
    if x == 1 or x == n - 1:
        continue

    for _ in range(s - 1):
        x = mod_pow(x, 2, n)
        if x == n - 1:
            break
    else:
        return False

return True

```

Рис. 2.2: Реализация р-Метода Полларда



```

# Проверка числа на простоту (упрощенная)
def miller_rabin_primality(n):
    # ... (code from previous image) ...

def pollard_rho(n: int, max_iters: int = 10000) -> Optional[int]:
    """
    p-метод Полларда (rho-метод)
    """
    # Проверка на чётность
    if n % 2 == 0:
        return 2

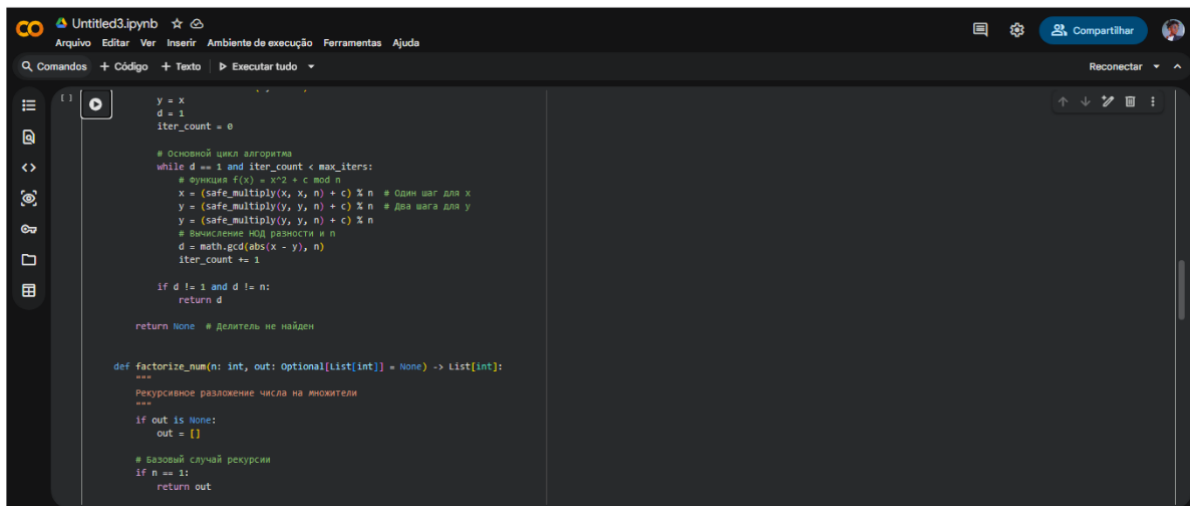
    # Инициализация генератора случайных чисел
    random.seed()

    for attempt in range(10):
        c = random.randint(1, n - 1) # Случайная константа
        x = random.randint(0, n - 1) # начальное значение x
        y = x
        d = 1
        iter_count = 0

        # Основной цикл алгоритма
        while d == 1 and iter_count < max_iters:
            # Функция f(x) = x^2 + c mod n
            x = (safe_multiply(x, x, n) + c) % n # Один шаг для x
            y = (safe_multiply(y, y, n) + c) % n # Два шага для y
            y = (safe_multiply(y, y, n) + c) % n
            # Вычисление НОД разности x и y

```

Рис. 2.3: Реализация р-Метода Полларда



```
def pollard_rho(n):
    y = x
    d = 1
    iter_count = 0

    # Основной цикл алгоритма
    while d == 1 and iter_count < max_iters:
        # Функция f(x) = x^2 + c mod n
        x = (safe_multiply(x, x, n) + c) % n # Один шаг для x
        y = (safe_multiply(y, y, n) + c) % n # Два шага для y
        y = (safe_multiply(y, y, n) + c) % n
        # Вычисление НОД разности x и y
        d = math.gcd(abs(x - y), n)
        iter_count += 1

    if d != 1 and d != n:
        return d

    return None # Делитель не найден

def factorize_num(n: int, out: Optional[List[int]] = None) -> List[int]:
    """
    Рекурсивное разложение числа на множители
    """
    if out is None:
        out = []

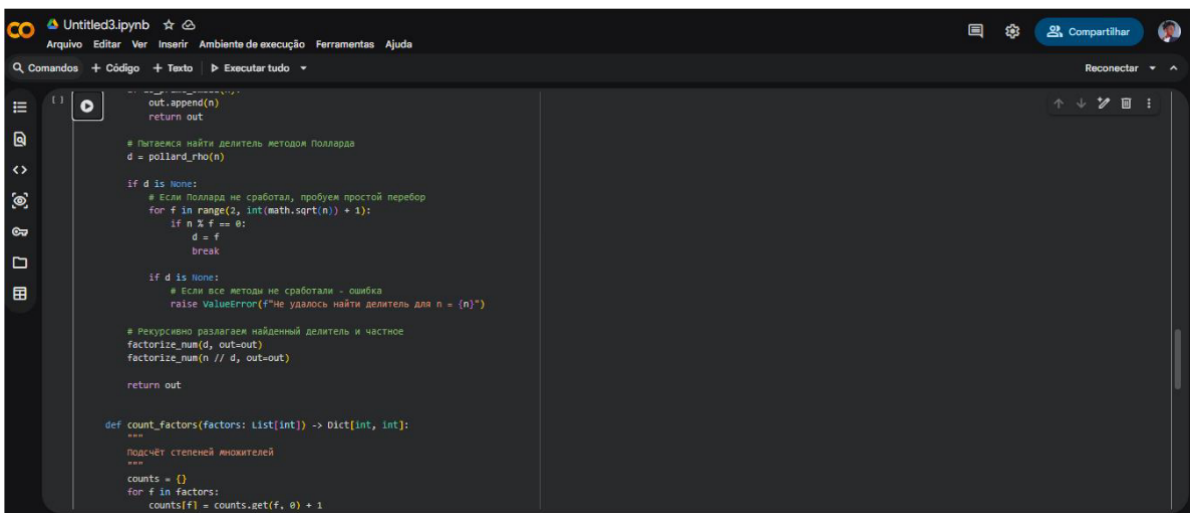
    # Базовый случай рекурсии
    if n == 1:
        return out

    d = pollard_rho(n)
    if d is None:
        out.append(n)
        return out
    else:
        factorize_num(d, out)
        factorize_num(n // d, out)
        return out
```

Рис. 2.4: Реализация р-Метода Полларда

Реализация р-Метода Полларда

Рис. 2.5: Реализация р-Метода Полларда



```
def pollard_rho(n):
    out.append(n)
    return out

# Пытаемся найти делитель методом Полларда
d = pollard_rho(n)

if d is None:
    # Если Поллард не работает, пробуем простой перебор
    for f in range(2, int(math.sqrt(n)) + 1):
        if n % f == 0:
            d = f
            break

if d is None:
    # Если все методы не работали - ошибка
    raise ValueError(f"Не удалось найти делитель для n = {n}")

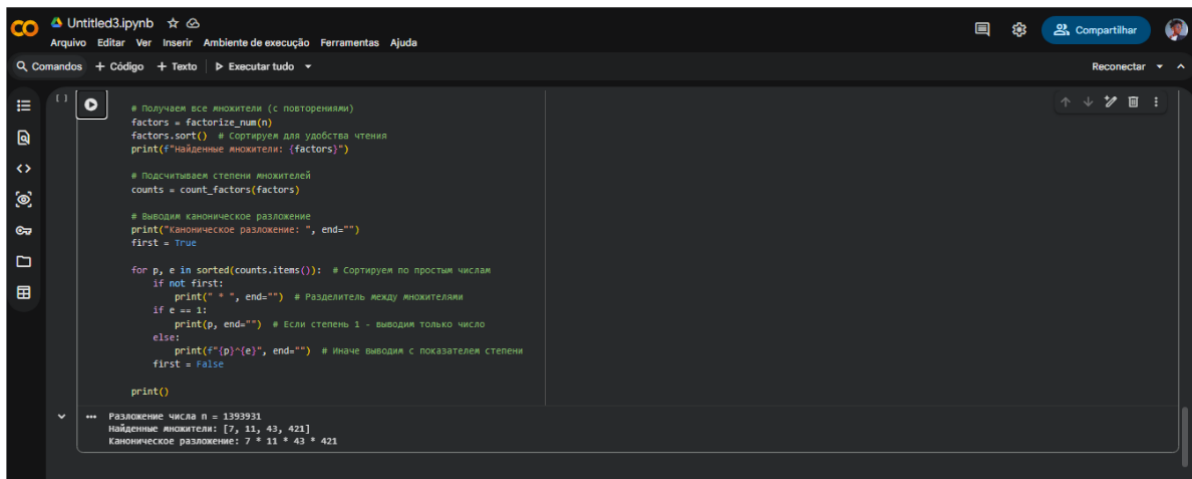
# Рекурсивно разлагаем найденный делитель и частное
factorize_num(d, out)
factorize_num(n // d, out)

return out

def count_factors(factors: List[int]) -> Dict[int, int]:
    """
    Подсчет степеней множителей
    """
    counts = {}
    for f in factors:
        counts[f] = counts.get(f, 0) + 1
```

Рис. 2.6: Реализация р-Метода Полларда

Проверим работу алгоритма (рис. 2.7):



The screenshot shows a Jupyter Notebook titled 'Untitled3.ipynb'. The interface includes a top menu bar with options like 'Arquivo', 'Editar', 'Ver', 'Inserir', 'Ambiente de execução', 'Ferramentas', and 'Ajuda'. Below the menu is a toolbar with icons for file operations and execution. The main area contains a code cell with the following Python code:

```
# Получаем все множители (с повторениями)
factors = factorize_num(n)
factors.sort() # Сортируем для удобства чтения
print(f"Найденные множители: {factors}")

# Подсчитываем степени множителей
counts = count_factors(factors)

# Выводим каноническое разложение
print("Каноническое разложение: ", end="")
first = True

for p, e in sorted(counts.items()): # Сортируем по простым числам
    if not first:
        print(" * ", end="") # Разделитель между множителями
    if e == 1:
        print(p, end="") # Если степень 1 - выводим только число
    else:
        print(f"{p}^{e}", end="") # Иначе выводим с показателем степени
    first = False

print()
```

Below the code cell, the output is displayed:

```
Разложение числа n = 1393931
Найденные множители: [7, 11, 43, 421]
Каноническое разложение: 7 * 11 * 43 * 421
```

Рис. 2.7: Проверка

3 Список литературы. Библиография