

Лабораторная работа №6

Математические основы защиты информации и информационной безопасности

Мануэл Марсия Педру

2026

Российский университет дружбы народов имени Патриса Лумумбы, Москва, Россия

..... {.columns align=center} ::: {.column width="70%"}

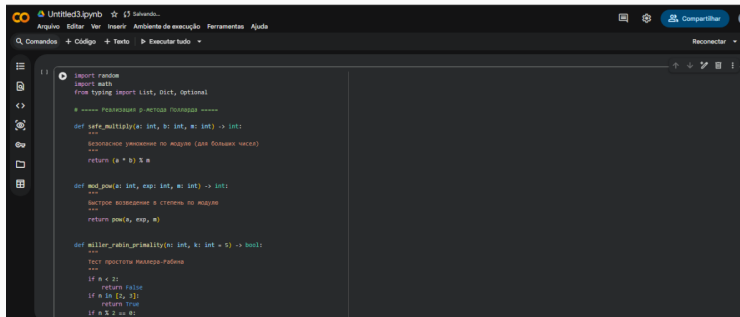
- Мануэл Марсия Педру
- Студент группы НФИмд-02-25
- Студ. билет 1032255503
- Российский университет дружбы народов имени Патриса Лумумбы

::: ::: {.column width="30%"}

- Изучить алгоритм разложения чисел на множители и научиться его реализовывать.

Выполнение лабораторной работы

Реализация р-Метода Полларда



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

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

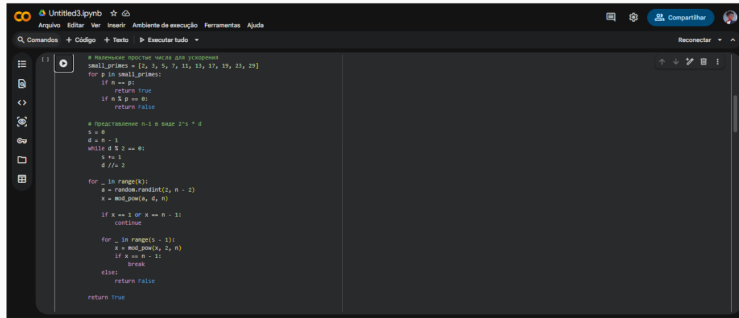
def safe_multiply(a: int, b: int, m: int) -> int:
    """
    Безопасное умножение по модулю (для больших чисел)
    """
    return (a * b) % m

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. 1: Реализация р-Метода Полларда

Реализация р-Метода Полларда



```

# Маленькие простые числа для ускорения
small_primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

def is_prime(n):
    for p in small_primes:
        if n == p:
            return True
        if n % p == 0:
            return False

# Представление n-1 в виде 2^s * d
def rho(n):
    s = 0
    d = n - 1
    while d % 2 == 0:
        s += 1
        d //= 2

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

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

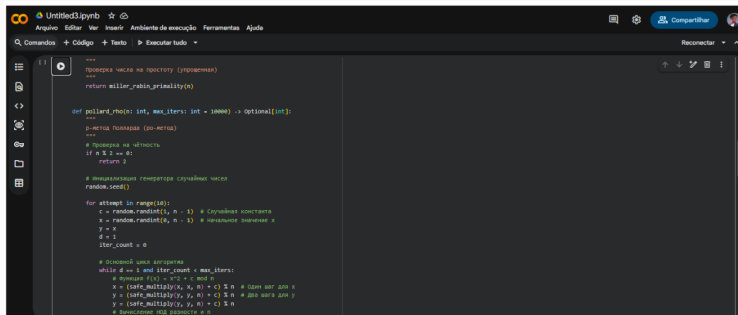
        for _ in range(s - 1):
            x = mod_pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False

    return True

```

Рис. 2: Реализация р-Метода Полларда

Реализация р-Метода Полларда



```
"""
проверка числа на простоту (эвристика)
"""
return miller_rabin_primality(n)

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

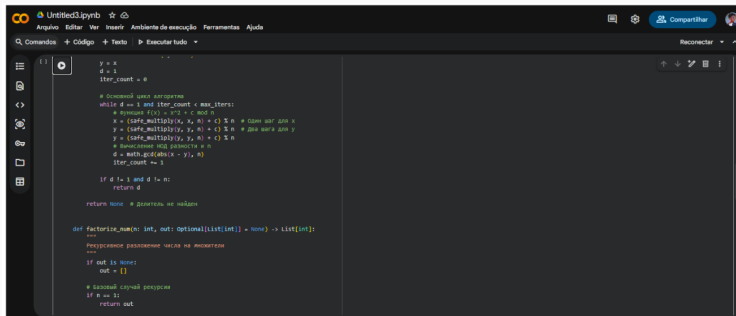
    # инициализация генератора случайных чисел
    random.seed()

    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

        # Оценили 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. 3: Реализация р-Метода Полларда

Реализация р-Метода Полларда



```

y = x
d = 1
iter_count = 0

# Проверка если x равен 0
while d == 1 and iter_count < max_iters:
    # Вычисляем 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 # два шага для y
    # Вычисляем 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 # делителей не найдено

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

    # базовый случай поиска
    if n == 1:
        return out

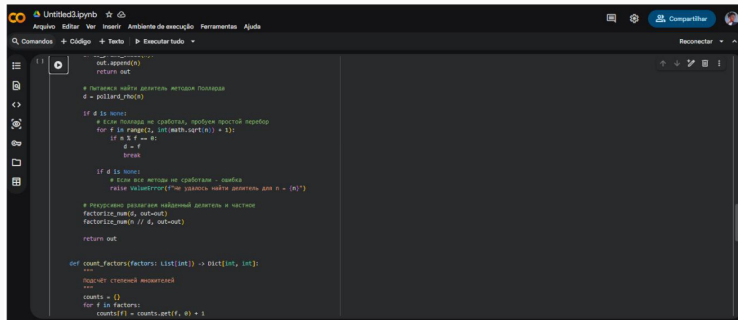
```

Рис. 4: Реализация р-Метода Полларда

Реализация р-Метода Полларда

Рис. 5: Реализация р-Метода Полларда

Реализация р-Метода Полларда



```
def factorize_num(n, out):
    """
    Рекурсивно разлагаем найденный делитель и частное
    """
    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

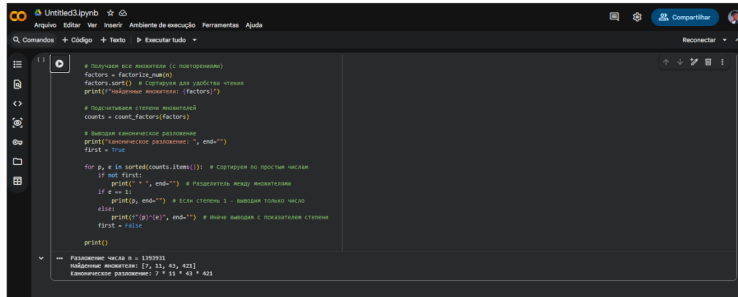
if d is None:
    # Если все методы не сработали - ошибка
    raise ValueError("Не удалось найти делитель для n = {n}")

# 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]:
    """
    Подсчёт степеней множителей
    """
    counts = {}
    for f in factors:
        counts[f] = counts.get(f, 0) + 1
```

Рис. 6: Реализация р-Метода Полларда



The screenshot shows a Jupyter Notebook window 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 running code, saving, and other functions. The main area displays a Python script for prime factorization. The script defines a function to factorize a number, sorts the factors, and prints the canonical decomposition. The output shows the factorization of the number 1393931 into the product of prime factors 7, 11, 43, and 421.

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

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

Рис. 7: Проверка

Вывод

- В ходе выполнения лабораторной работы был изучен алгоритм разложения чисел на множители, а также написан его алгоритм на языке python.

Список литературы. Библиография
