

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

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

## Цель лабораторной работы

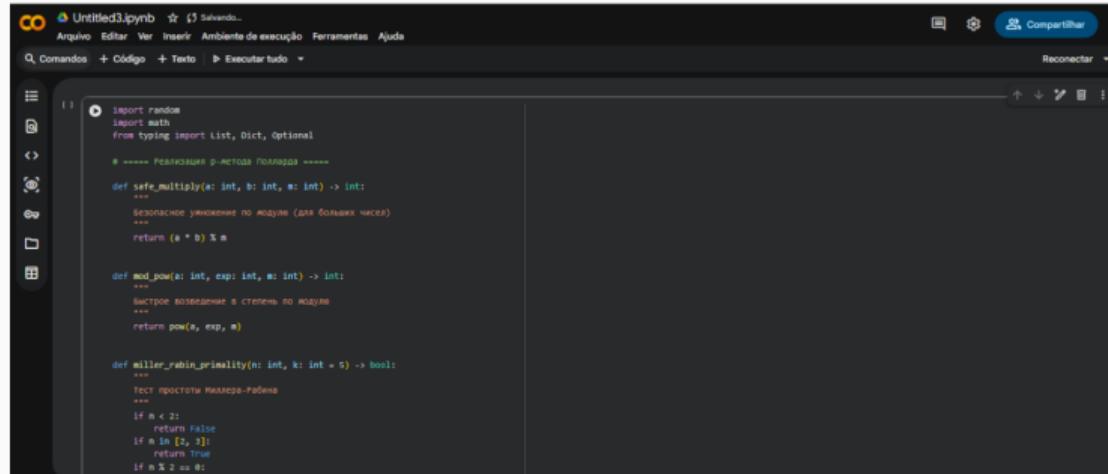
---

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

## Выполнение лабораторной работы

---

# Реализация p-Метода Полларда



```
Untitled3.ipynb ★ ⓘ Salvando...
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Q_ Comandos + Código + Texto ▶ Executar tudo Reconectar
import random
import math
from typing import List, Dict, Optional

# ----- Реализация p-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:
        return False

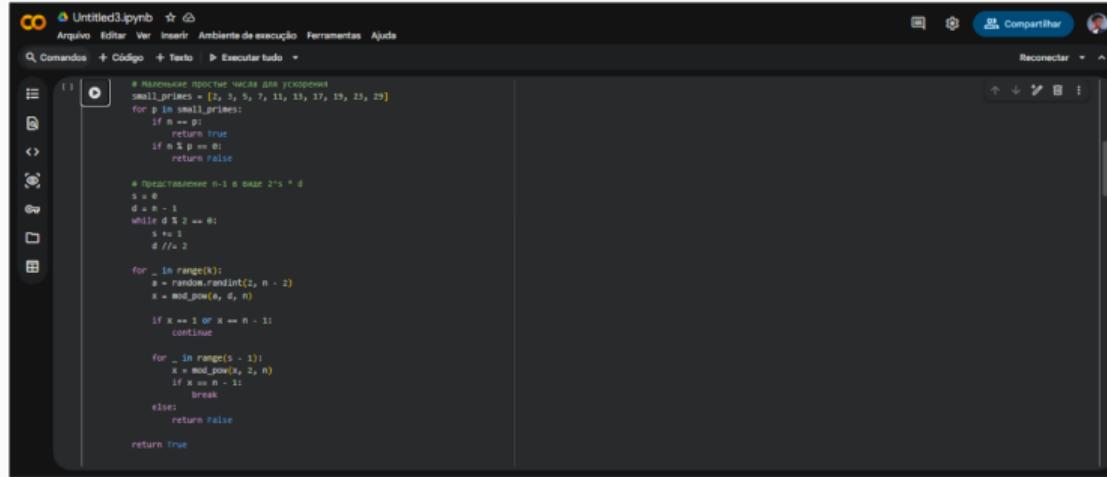
    # Выбираем случайные базы
    bases = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
    random.shuffle(bases)

    for base in bases:
        if not MillerRabinTest(n, base):
            return False

    return True
```

Рис. 1: Реализация p-Метода Полларда

# Реализация p-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-2 в виде 2^s * d
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, 2, n)

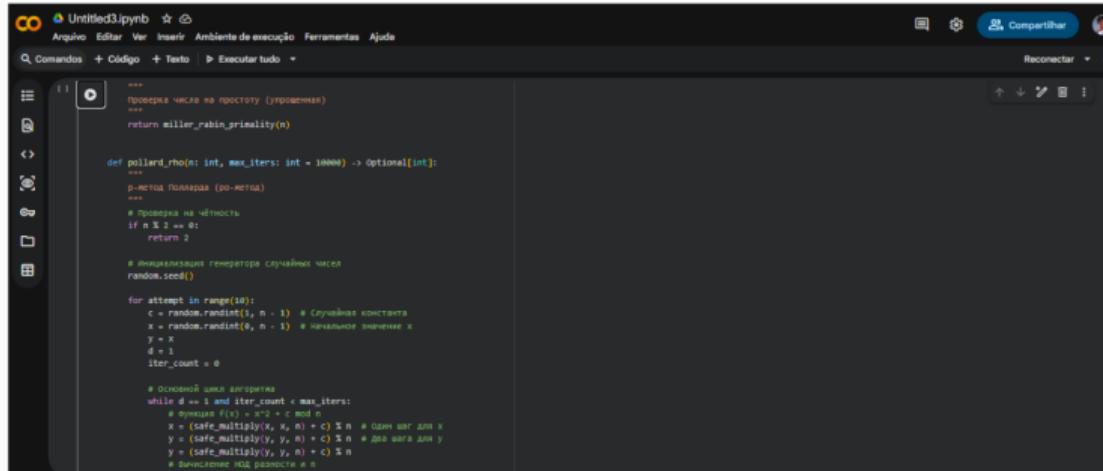
    if x == 1 or x == n - 1:
        continue

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

return True
```

Рис. 2: Реализация p-Метода Полларда

# Реализация p-Метода Полларда



```
*** проверка числа на простоту (урокенная)
...
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

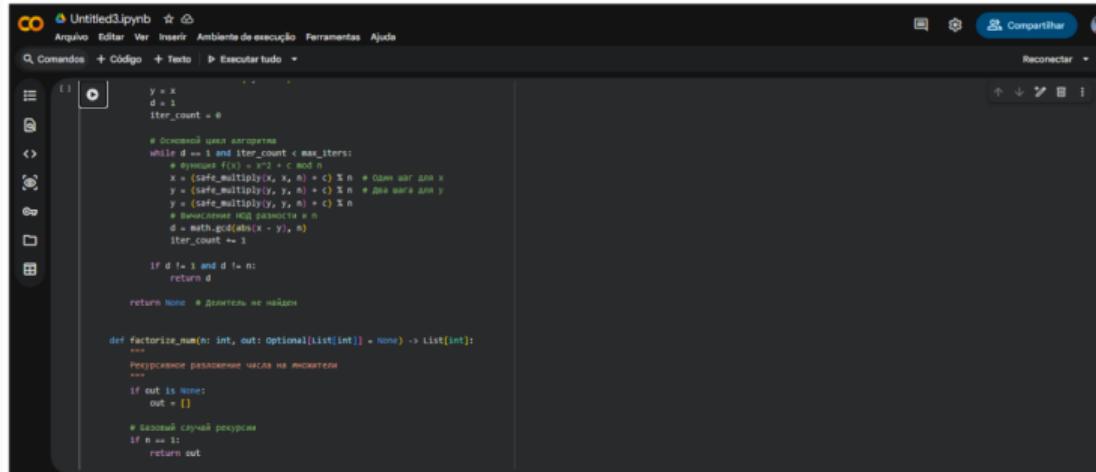
    # инициализация генератора случайных чисел
    random.seed()

    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

        # 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 под разности x-y
```

Рис. 3: Реализация p-Метода Полларда

# Реализация p-Метода Полларда



```
# Untitled3.ipynb
Arquivo Editar Ver Inserir Ambiente de execução Ferramentas Ajuda
Q Comandos + Código + Texto | Executar tudo
Reconectar
x = 2
y = 1
iter_count = 0

# Основной цикл алгоритма
while d == 1 and iter_count < max_iters:
    # функция f(x) = x2 + 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 квадратичной разности в n
    d = math.gcd(d, abs(x - y), n)
    iter_count += 1

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

return None  # делитель не найден

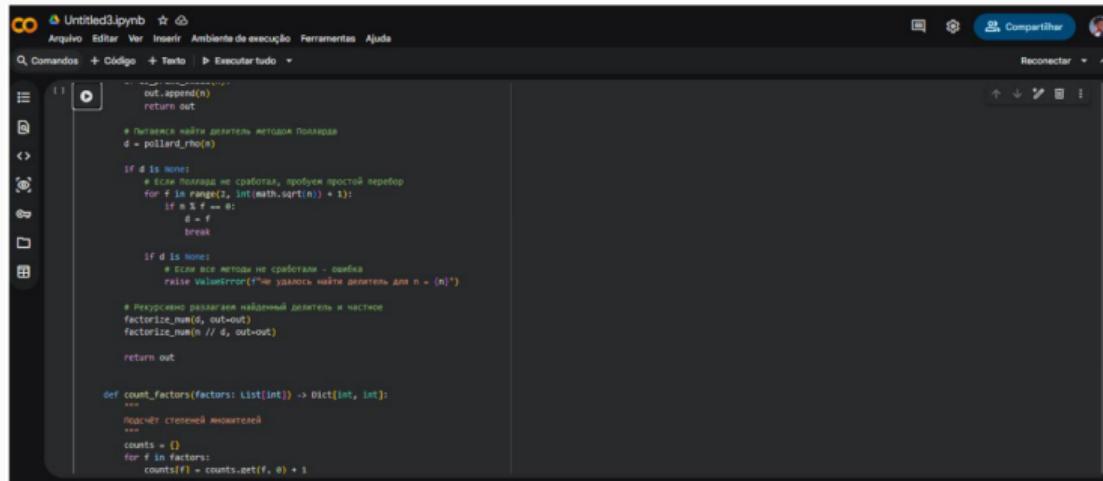
def factorize_num(n: int, out: Optional[List[int]] = None) -> List[int]:
    """Рекурсивное разложение числа на множители"""
    if out is None:
        out = []
    # базовый случай рекурсии
    if n == 1:
        return out
```

Рис. 4: Реализация p-Метода Полларда

Реализация p-Метода Полларда

Рис. 5: Реализация p-Метода Полларда

# Реализация p-Метода Полларда



```
out.append(n)
return out

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

#if d is None:
#    # Если поллард не сработал, 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 = " + str(n))

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

return out

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

Рис. 6: Реализация p-Метода Полларда

# Проверка

The screenshot shows a Jupyter Notebook interface with a single code cell. The code performs prime factorization of the number 1393931. It uses the `factorize` function from the `sympy` library to get all factors, sorts them, and then prints them. It then calculates the exponent of each prime factor in the factorization. Finally, it prints the prime factors with their exponents and multiplies them together to verify the result.

```
# 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("^(p|^e)", end="") # Иначе выводим с показателем степени
    first = False

print()
*** Разложение числа n = 1393931
Найденные множители: [7, 11, 43, 421]
Каноническое разложение: 7 ^ 1 * 11 ^ 1 * 43 * 421
```

Рис. 7: Проверка

## Вывод

---

## Вывод

---

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

## Список литературы. Библиография

---