Шаблон отчёта по лабораторной работе №7

Дискретное логарифмирование в конечном поле

Коне Сирики

Содержание

# 1 Цель работы

Целью данной лабораторной работы является краткое ознакомление с задачей дискретного логарифмирования и -методом Полларда для её решения, а также его последующая программная реализация.

# 2 Задание

Рассмотреть и реализовать на языке программирования Python -метод Полларда для задачи дискретного логарифмирования.

# 3 Теоретическое введение

## 3.1 Основные понятия из теории групп и теории чисел

Для начала введём некоторые базовые понятия.

Опр. 1.

*Группа* – это непустое множество с бинарной операцией , обладающей свойством ассоциативности () и относительно которой существует нейтральный () и обратный элемент (). Если операция коммутативна, группа называется *абелевой* [1].

Опр. 2.

Если , то *подгруппа, порождённая* , , – это пересечение всех групп, содержащих . Если существует такой, что , то группа – циклическая. Все циклические группы абелевы.

Опр. 3.

Пусть . Целые и называются *сравнимыми по модулю* , если , т.е. является делителем . Отношение сравнимости записывается следующим образом: [2].

Опр. 4.

Для любого множество чисел называется *классом вычетов по модулю* . Существует ровно классов вычетов по модулю , причём .

Опр. 5.

*Кольцо* – это множество с двумя операциями , для которых выполняются свойства: – абелева группа, , и . Кольцо коммутативно, если . Кольцо – с единицей, если [1].

Опр. 6.

*Полем* называется коммутативное кольцо, содержащее не менее двух элементов, в котором все ненулевые элементы образуют группу по умножению [3]. Конечное поле с элементами, где – простое число, обозначается [4].

Опр. 7.

Через (или ) обозначим *множество классов вычетов по модулю* : . На нём можно определить операции сложения и умножения: , . является коммутативным кольцом с единицей , в котором нулевой элемент: , а обратный по сложению элемент: . Если простое, то – поле.

Опр. 8.

Пусть . Тогда – обратный к , если , а является обратимым, если имеет обратный класс. Множество всех обратимых классов в обозначается , является группой относительно умножения классов и называется *мультипликативной группой кольца вычетов* [5].

## 3.2 Дискретное логарифмирование

Задача дискретного логарифмирования – наравне с задачей факторизации – является одной из фундаментальных в криптоанализе. На её сложности зиждется стойкость ряда криптосистем, включая такие известные, как:

* схема распределения ключей Диффи-Хеллмана (1976);
* схема Эль-Гамаля (1985), лежащая в основе алгоритма DSA;
* криптосистема Мэсси-Омуры (1978) для передачи сообщений [4].

Для конечного поля (в частности, в простейшем и важнейшем случае , где – большое простое число) *задача дискретного логарифмирования* определяется следующим образом [4]: при заданных ненулевых найти такое целое , что:

Пусть число также имеет порядок , то есть .

## 3.3 -метод Полларда для задачи дискретного логарифмирования

Рассмотрим -метод Полларда, который можно применить и для задач дискретного логарифмирования [6]. Здесь, как и в аналогичном методе факторизации, рассмотренном в предыдущей лабораторной, строится последовательность итеративных значений функции , в которой требуется найти цикл. Для этого, как и ранее, используем алгоритм “черепахи и зайца” Флойда: к одному значению, , на каждом шаге будем применять функции единожды, к другому, , – дважды, пока их значения не совпадут и мы не сможем их приравнять.

Так, пусть , где случайные целые числа, – их начальные значения. Поскольку по условию задачи , мы также можем записать . Тогда . Таким образом, логарифмы и по основанию могут быть представлены линейно.

Теперь зададим отображение . Оно должно обладать не только сжимающими свойствами, но и вычислимостью логарифма, чтобы по мере изменения значений и мы могли также отслеживать изменения в линейном представлении их логарифмов. Будем использовать ветвящееся отображение следующего вида:

Таким образом, будет умножаться или на , или на . В первом случае получим , и тогда . Во втором случае же получаем , и отсюда .

Когда значения и совпадут, мы сможем приравнять их логарифмы и получим сравнение по : .

**Алгоритм 1. Алгоритм, реализующий -метод Полларда для задач дискретного логарифмирования**

*Вход.* Простое число , число порядка по модулю , целое число ; отображение , обладающее сжимающими свойствами и сохраняющее вычислимость логарифма.

*Выход.* Показатель , для которого , если такой показатель существует.

1. Выбрать произвольные целые числа и положить .
2. Выполнять , вычисляя при этом логарифмы для и как линейные функции от по модулю , до получения равенства .
3. Приравняв логарифмы для и , вычислить логарифм решением сравнения по модулю . Результат: или “Решений нет”.

Пример 1.

Решим задачу . Выберем отображение: $ при , при . Порядок числа по модулю равен . Пусть , . Результаты вычислений представлены в Таблице 1.

Таблица 1: Применение -метода Полларда для решения примера №1

| Шаг | c |  | d |  |
| --- | --- | --- | --- | --- |
| 0 | 4 |  | 4 |  |
| 1 | 40 |  | 79 |  |
| 2 | 79 |  | 56 |  |
| 3 | 27 |  | 75 |  |
| 4 | 56 |  | 3 |  |
| 5 | 53 |  | 86 |  |
| 6 | 75 |  | 42 |  |
| 7 | 92 |  | 23 |  |
| 8 | 3 |  | 53 |  |
| 9 | 30 |  | 92 |  |
| 10 | 86 |  | 30 |  |
| 11 | 47 |  | 47 |  |

Приравниваем полученные логарифмы: . Отсюда . Чтобы решить данное сравнение, нужно найти обратный элемент для по модулю () и умножить на него левую и правую часть сравнения. Так как этот обратный элемент – сравним сам с собой по модулю , подобное сравнение будет справедливо [7].

В общем виде пусть решается сравнение . Если и – взаимно простые, т.е. НОД, мы можем применить расширенный алгоритм Евклида, разобранный в рамках 4-ой лабораторной работы, и получить линейное представление единицы в виде: [8]. Отсюда , что эквивалентно , что эквивалентно , т.е. . Если же НОД не равен единице, то мы предполагаем, что (поскольку в противном случае обратного элемента не существует), и тогда сравнение можно поделить на [7], и получим .

Возвращаясь к примеру, получаем . Проверка: .

# 4 Выполнение лабораторной работы

Реализуем описанный выше алгоритм на языке **Python** в среде Jupyter Notebook. Для работы нам понадобится функция вычисления порядка числа по модулю, расширенный алгоритм Евклида, реализацию которого мы возьмем из 4-ой лабораторной работы, а также основанная на нём функция решения сравнения вида :

import math  
import numpy as np  
  
def multiplicative\_order(a, n):  
 """  
 Вычисляет порядок числа a по модулю n  
 """  
 k = 1; flag = True # начнем перебор с единицы  
  
 while flag:  
 if (a \*\* k - 1) % n == 0: # если порядок найден  
 flag = False # "опускаем" флаг и выходим из цикла  
 else: # иначе  
 k += 1 # увеличиваем порядок на единицу  
  
 return k  
  
def euclidean\_algorithm\_extended(a, b):  
 """  
 Находит d = НОД(a, b), а также такие целые числа x и y, что ax + by = d,  
 с помощью расширенного алгоритма Евклида  
 """  
 (a, b) = (int(a), int(b))  
  
 reversed = True if abs(b) > abs(a) else False # флаг  
 (a, b) = (b, a) if reversed else (a, b) # меняем местами a и b, если нужно  
  
 (r, x, y) = ([a, b], [1, 0], [0, 1]) # шаг 1  
  
 while r[1] != 0:  
 (r[0], r[1], q) = (r[1], r[0] % r[1], r[0] // r[1])  
  
 if r[1] != 0: # если остаток ещё не нулевой..  
 (x[0], x[1]) = (x[1], x[0] - q \* x[1])  
 (y[0], y[1]) = (y[1], y[0] - q \* y[1])  
  
 (d, x\_r, y\_r) = (r[0], x[1], y[1])  
  
 if reversed: # если a и b были в неправильном порядке  
 (x\_r, y\_r) = (y\_r, x\_r) # меняем найденные коэффициенты местами  
  
 return (d, x\_r, y\_r)  
  
def solve\_congruence(c, d, p):  
 """  
 Решает сравнение вида k\_1 \* x + b\_1 = k\_2 \* x + b\_2 (mod p), где  
 c = (k\_1, b\_1), d = (k\_2, b\_2)  
 """  
 (k\_1, b\_1) = c; (k\_2, b\_2) = d # получаем коэффициенты  
  
 k = k\_1 - k\_2; b = b\_2 - b\_1 # kx = b (mod p)  
 # k \* k\_inverse = gcd (mod p)  
 (gcd, k\_inverse, \_) = euclidean\_algorithm\_extended(k, p)  
  
 if gcd == 1: # если k и p - взаимно простые..  
 return (b \* k\_inverse) % p  
 else: # иначе  
 k = int(k / gcd); b = int(b / gcd) # делим сравнение на gcd  
 (\_, k\_inverse, \_) = euclidean\_algorithm\_extended(k, int(p / gcd))  
  
 return (b \* k\_inverse) % p

Примеры работы функции multiplicative\_order(a, n) представлены на Рис. 1.

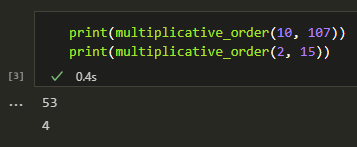


Рис. 1: Примеры нахождения порядка числа по модулю

## 4.1 Алгоритм, реализующий -метод Полларда для задачи дискретного логарифмирования

Создадим функцию pollard\_rho\_method(n, f, c) следующего вида:

def pollard\_rho\_dlog(a, b, p, def0 = True, to\_print = False):  
 """  
 Решает сравнение a^x = b (mod p) ро-методом Полларда;  
 def0 = True, если нужно использовать начальные значения u и v по умолчанию,  
 и False, если их нужно определить случайно;  
 to\_print = True, если нужно вывести на экран ход алгоритма  
 """  
 r = multiplicative\_order(a, p) # порядок числа а  
 half\_p = math.floor(p / 2) # p / 2  
  
 # отображение f  
 f = "({a} \* x % {p}) if x < {half} else ({b} \* x % {p})".format(a = a,  
 p = p, half = half\_p, b = b)  
  
 # начальные значения u и v  
 (u, v) = (2, 2) if def0 else (np.random.randint(1, half\_p),  
 np.random.randint(1, half\_p))  
  
 if not def0 and to\_print:  
 print("(u, v) = ({}, {})".format(u, v))  
  
 c = ((a \*\* u) \* (b \*\* v)) % p #   
 d = c #  
 # шаг 1  
 (k\_c, l\_c) = (u, v) #  
 (k\_d, l\_d) = (u, v) #  
  
 if to\_print:  
 print("{:^10} | {:^10} | {:^10} | {:^10}"  
 .format("c", "log\_c", "d", "log\_d"))  
 print("{:^10} | {:^10} | {:^10} | {:^10}"  
 .format("----------", "----------", "----------", "----------"))  
 print("{:^10} | {:^3} + {:^3}x | {:^10} | {:^3} + {:^3}x"  
 .format(c, l\_c, k\_c, d, l\_d, k\_d))  
  
 while True:  
 # вычисляем f(c)  
 # и log\_a f(c)  
 x = c #  
 if x < half\_p: #  
 l\_c += 1 #  
 else: #  
 k\_c += 1 #  
 c = eval(f) #  
 #  
 # вычисляем f(c) #  
 # и log\_a f(c) #  
 x = d # шаг 2  
 if x < half\_p: #  
 l\_d += 1 #  
 else: #  
 k\_d += 1 #  
 x = eval(f) #  
 if x < half\_p: #  
 l\_d += 1 #  
 else: #  
 k\_d += 1 #  
 d = eval(f) #  
  
 if to\_print:  
 print("{:^10} | {:^3} + {:^3}x | {:^10} | {:^3} + {:^3}x"  
 .format(c, l\_c, k\_c, d, l\_d, k\_d))  
  
 # шаг 3  
 if c == d:  
 # приравниваем логарифмы и решаем сравнение  
 result = solve\_congruence((k\_c, l\_c), (k\_d, l\_d), r)  
  
 if (a \*\* result - b) % p == 0: # проверка  
 return result  
 else:  
 return 0

Теперь с помощью данной функции решим несколько задач на вычисление дискретных логарифмов: (см. Рис. 2), (см. Рис. 3) и (см. Рис. 4).

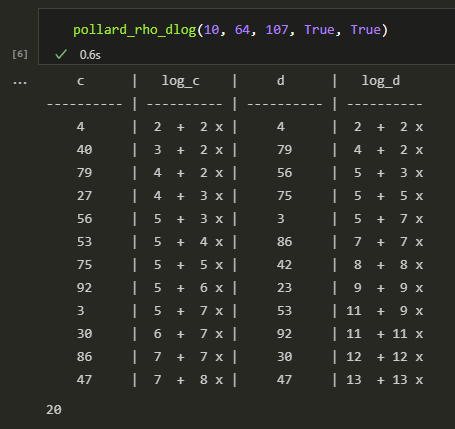


Рис. 2: Решение сравнения №1

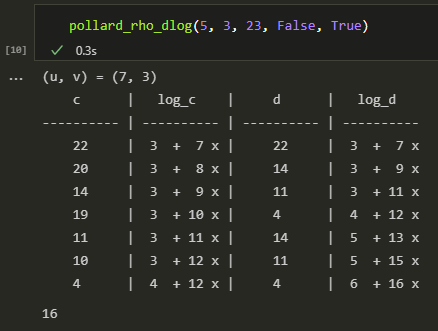


Рис. 3: Решение сравнения №2

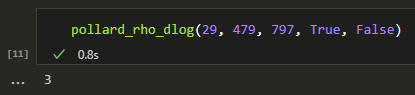


Рис. 4: Решение сравнения №3

# 5 Выводы

Таким образом, была достигнута цель, поставленная в начале лабораторной работы: было проведено краткое знакомство с задачей дискретного логарифмирования и с алгоритмом, реализующим -метод Полларда для её решения, после чего алгоритм был успешно реализован на языке программирования **Python**.

# Список литературы

1. Богданов И.И. Теория групп: конспект. ФИВТ МФТИ, 2016. С. 42.

2. Илларионов А.А. [Теория чисел: учебное пособие](\url{http://www.iam.khv.ru/articles/Illarionov/mainNumberTheory.pdf}). 2016.

3. Зельвенский И.Г. Группы, кольца, поля: Методические указания по дисциплине «Геометрия и алгебра». Спб.: ГЭТУ ЛЭТИ, 1997. С. 30.

4. Yan S. Primality Testing and Integer Factorization in Public-Key Cryptography. Boston: Springer, 2009. С. 371.

5. Веретенников Б.М., Михалева М.М. Алгебра и теория чисел : учебное пособие. Часть 1 / под ред. Чуксина Н.В. Екатеринбург: Изд-во Урал. ун-та, 2014. С. 52.

6. Бубнов С.А. Лабораторный практикум по основам криптографии: учебно-методическое пособие. Саратов; <http://elibrary.sgu.ru/uch_lit/656.pdf>: Саратовский государственный университет им. Н.Г. Чернышевского, 2012.

7. Википедия. [Сравнение по модулю — Википедия, свободная энциклопедия](\url{https://ru.wikipedia.org/?curid=13486&oldid=116785073}). 2021.

8. Occhipinti T. [Discrete logs with Pollard rho | Math 361](\url{https://www.youtube.com/watch?v=QjDC1YtQFj4}). 2021.