Отчёт по лабораторной работе №7

Дискретное логарифмирование в конечном поле

Милёхин Александр НПМмд-02-21

Содержание

# Цель работы

Изучение задачи дискретного логарифмирования.

# Теоретические сведения

Пусть в некоторой конечной мультипликативной абелевой группе задано уравнение

Решение задачи дискретного логарифмирования состоит в нахождении некоторого целого неотрицательного числа , удовлетворяющего уравнению. Если оно разрешимо, у него должно быть хотя бы одно натуральное решение, не превышающее порядок группы. Это сразу даёт грубую оценку сложности алгоритма поиска решений сверху — алгоритм полного перебора нашёл бы решение за число шагов не выше порядка данной группы.

Чаще всего рассматривается случай, когда группа является циклической, порождённой элементом . В этом случае уравнение всегда имеет решение. В случае же произвольной группы вопрос о разрешимости задачи дискретного логарифмирования, то есть вопрос о существовании решений уравнения , требует отдельного рассмотрения.

## p-алгоритм Полларда

* 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.

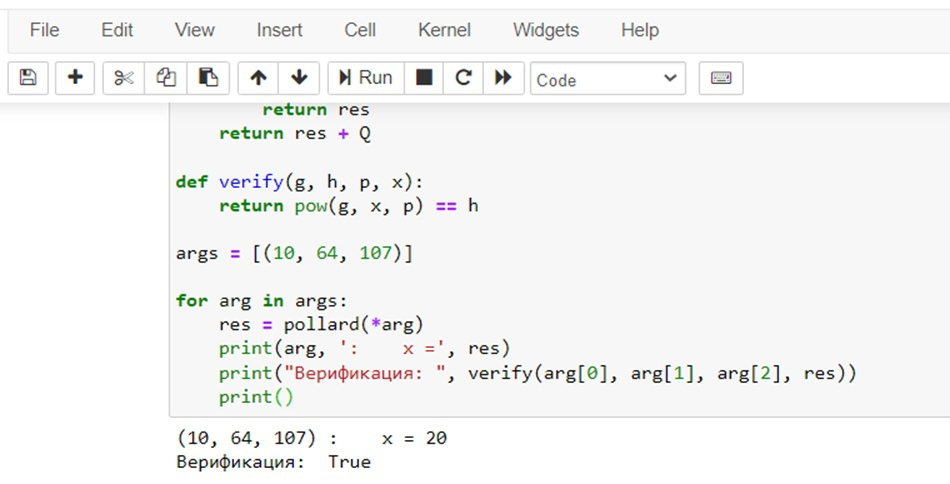
1. Выбрать произвольные целые числа и положить
2. Выполнять c=f(c)(mod p), d=f(f(d))(mod p), вычисляя при этом логарифмы для и как линейные функции от по модулю , до получения равенства
3. Приняв логарифмы для и , вычислить логарифм решением сравнения по модулю . Результат или “Решения нет”.

# Выполнение работы

## Реализация алгоритмов на языке Python

def euclid\_extended(a, b):  
 if b == 0:  
 return a, 1, 0  
 else:  
 d, xx, yy = euclid\_extended(b, a%b)  
 x = yy   
 y = xx - (a // b) \* yy  
 return d, x, y  
  
def inverse(a, n):  
 return (euclid\_extended(a, n)[1])  
  
def xab(x, a, b, x\_swap):  
 (G, H, P, Q) = x\_swap  
 sub = x % 3  
 if sub == 0:  
 x = x\*x\_swap[0] % x\_swap[2]  
 a = (a+1) % Q  
 if sub == 1:  
 x = x \* x\_swap[1] % x\_swap[2]  
 b = (b + 1) % x\_swap[2]  
 if sub == 2:  
 x = x\*x % x\_swap[2]  
 a = a\*2 % x\_swap[3]  
 b = b\*2 % x\_swap[3]  
 return x, a, b  
  
def pollard(G, H, P):  
 Q = int((P - 1) // 2)  
 x = G\*H  
 a = 1  
 b = 1  
 X = x  
 A = a  
 B = b  
 for i in range(1, P):  
 x, a, b = xab(x, a, b, (G, H, P, Q))  
 X, A, B = xab(X, A, B, (G, H, P, Q))  
 X, A, B = xab(X, A, B, (G, H, P, Q))  
 if x == X:  
 break  
 nom = a-A  
 denom = B-b  
 res = (inverse(denom, Q) \* nom) % Q  
 if verify(G, H, P, res):  
 return res  
 return res + Q  
  
def verify(g, h, p, x):  
 return pow(g, x, p) == h  
  
args = [(10, 64, 107)]  
  
for arg in args:  
 res = pollard(\*arg)  
 print(arg, ': x =', res)  
 print("Верификация: ", verify(arg[0], arg[1], arg[2], res))  
 print()

## Контрольный пример



Пример работы алгоритма

Получаем x = 20 для значений в данном примере.

# Выводы

Я изучил задачу дискретного логарифмирования, повторил p-алгоритм Полларда, а также реализовал алгоритм программно на языке Python.

# Список литературы

1. [Дискретное логарифмирование](https://e-maxx.ru/algo/discrete_log#:~:text=Дискретное%20логарифмирование.%20Задача%20дискретного%20логарифмирования,модифицировать%2C%20чтобы%20он%20по-прежнему%20работал))
2. [Доступно о криптографии на эллиптических кривых](https://habr.com/ru/post/335906/)