Отчёт по лабораторной работе 7

МОЗИиИБ

Папикян Гагик Тигранович

Содержание

[1 Цель работы 1](#_Toc91360906)

[2 Задание 1](#_Toc91360907)

[3 Теоретическое введение 1](#_Toc91360908)

[3.1 Алгоритм Полларда 1](#_Toc91360909)

[4 Выполнение лабораторной работы 2](#_Toc91360910)

[5 Выводы 4](#_Toc91360911)

# 1 Цель работы

Познакомиться с алгоритмом Полларда для дискретного логарифмирования в конечном поле

# 2 Задание

1. Реализовать алгоритм Полларда

# 3 Теоретическое введение

## 3.1 Алгоритм Полларда

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. Сложность алгоритма оценивается как O(N^{1/4}).

ρ-алгоритм Полларда строит числовую последовательность, элементы которой образуют цикл, начиная с некоторого номера n, что может быть проиллюстрировано, расположением чисел в виде греческой буквы ρ, что послужило названием семейству алгоритмов

# 4 Выполнение лабораторной работы

Был написан следующий скрипт на python

import sys  
  
def ext\_euclid(a, b):  
 """  
 Extended Euclidean Algorithm  
 :param a:  
 :param b:  
 :return:  
 """  
 if b == 0:  
 return a, 1, 0  
 else:  
 d, xx, yy = ext\_euclid(b, a % b)  
 x = yy  
 y = xx - (a / b) \* yy  
 return d, x, y  
  
  
def xab(x, a, b, G, H, P, Q):  
 """  
 Pollard Step  
 :param x:  
 :param a:  
 :param b:  
 :return:  
 """  
 sub = x % 3 # Subsets  
  
 if sub == 0:  
 x = x\*G % P  
 a = (a+1) % Q  
  
 if sub == 1:  
 x = x \* H % P  
 b = (b + 1) % Q  
  
 if sub == 2:  
 x = x\*x % P  
 a = a\*2 % Q  
 b = b\*2 % Q  
  
 return x, a, b  
  
  
def pollard(G, H, P):  
  
 # P: prime  
 # H:  
 # G: generator  
 Q = (P - 1) / 2 # sub group  
  
  
 x = G\*H  
 a = 1  
 b = 1  
  
 X = x  
 A = int(a)  
 B = int(b)  
  
 # Do not use range() here. It makes the algorithm amazingly slow.  
 for i in range(1, P):  
 # Who needs pass-by reference when you have Python!!! ;)  
  
 # Hedgehog  
 x, a, b = xab(x, a, b, G, H, P, Q)  
  
 # Rabbit  
 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 = int(a-A)  
 denom = int(B-b)  
  
 print (nom, denom)  
  
 # It is necessary to compute the inverse to properly compute the fraction mod q  
 res = ( ext\_euclid(denom, int(Q) \* nom)[1] % int(Q))  
  
 # I know this is not good, but it does the job...  
 if verify(G, H, P, res):  
 return res  
  
 return int(res + Q)  
  
  
def verify(g, h, p, x):  
 """  
 Verifies a given set of g, h, p and x  
 :param g: Generator  
 :param h:  
 :param p: Prime  
 :param x: Computed X  
 :return:  
 """  
 return pow(int(g), int(x), p) == h  
  
g=int(5)  
h=int(22)  
p=int(53)  
  
  
print ("g=",g)  
print ("h=",h)  
print ("p=",p)  
  
print (h,"=",g,"^x (mod",p,")")  
print ("\n==============")  
  
x = int(pollard(g,h,p))  
print ("Solution x=",x)  
  
print ("Solution:",verify(g, h, p, x))  
print ("Checking h=",pow(int(g), int(x), p))

Результат исполнения скрипта приведен на рисунке 1 (рис. 1)

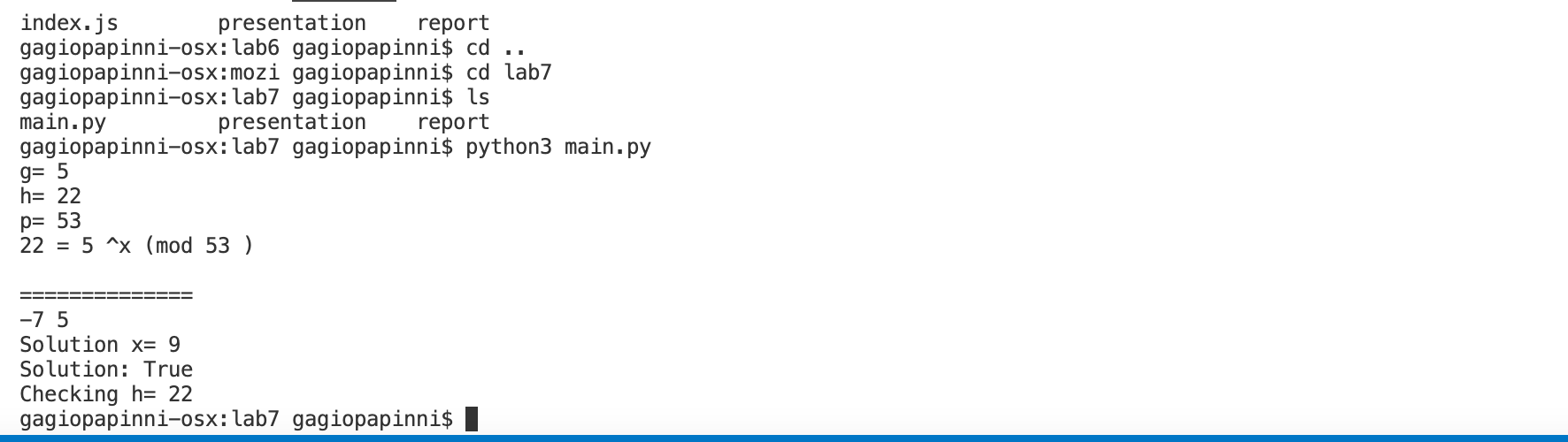


Figure 1: Выполнение лабораторной работы

# 5 Выводы

Были реализован алгоритм, и продемонстирован результат его выполнения