

Московский Авиационный Институт

Институт №8 «Информационные технологии и прикладная
математика»

Кафедра 806 «Вычислительная математика и
программирование»

Лабораторная работа №3 по курсу «Криптография»

Студент: В. П. Будникова

Преподаватель: А. В. Борисов

Группа: М80-307Б-19

Дата:

Оценка:

Подпись:

Москва, 2022

Задание:

Подобрать такую эллиптическую кривую, порядок точки которой полным перебором находится за 10 минут на ПК. Упомянуть в отчёте результаты замеров работы программы, характеристики вычислителя. Также указать какие алгоритмы и/или теоремы существуют для облегчения и ускорения решения задачи полного перебора.

Оборудование:

Ноутбук, процессор: 1,6 GHz 2-ядерный процессор Intel Core i5, память 8ГБ.

Описание:

Множество точек эллиптической кривой:

$$\{ (x, y) \in (F_p)^2 \mid y^2 \equiv x^3 + ax + b \pmod{p}, \\ 4a^3 + 27b^2 \not\equiv 0 \pmod{p} \} \cup \{0\}$$

Операция сложения:

$$P = (x_p, y_p), \quad Q = (x_q, y_q), \quad R = (x_r, y_r) \\ R = P + Q$$

1. Если $Q = (0, 0)$, то $R = P + 0 = P$

2. Если $Q = -P$, то $R = P + (-P) = 0$

3.

$$x_r = (m^2 - x_p - x_q) \pmod{p} \\ y_r = (y_p + m(x_r - x_p)) \pmod{p}$$

3.1 Если $P = Q$, то

$$m = (3x_p^2 + a)(2y_p)^{-1} \pmod{p}$$

3.2 Если $P \neq Q$, то

$$m = (y_p - y_q)(x_p - x_q)^{-1} \pmod{p}$$

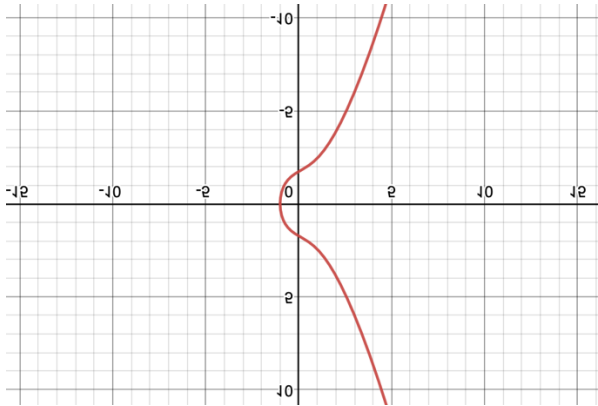
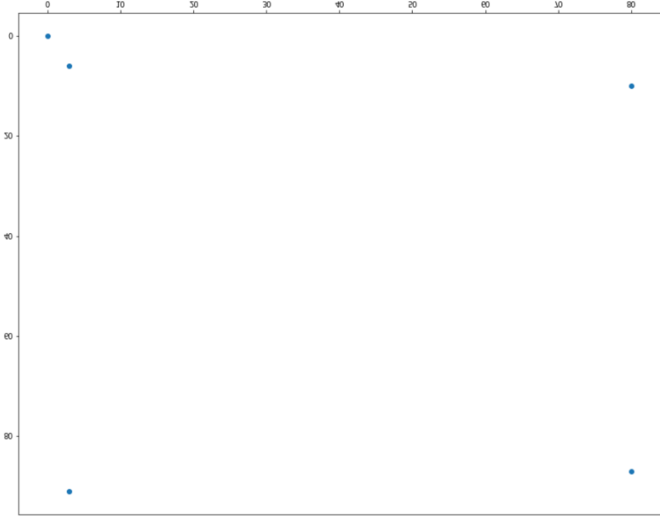
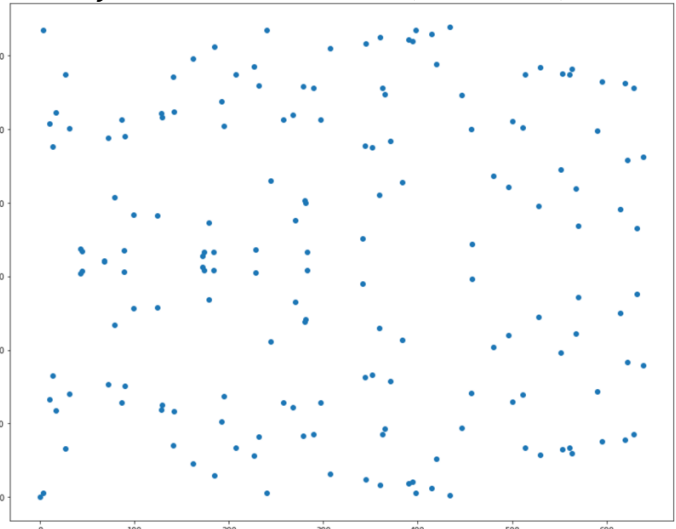
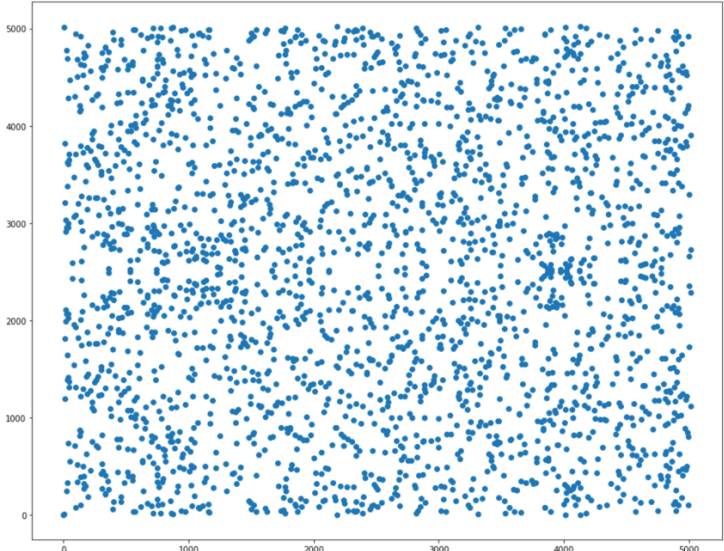
Операция умножения:

$$R = P * n$$

1. Если $n = 0$, то $R = P * 0 = 0$

2. Иначе $R = P * n = P + P + P + \dots + P$ – сложить P n раз.

Результаты работы:

<p>График эллиптической кривой</p> $y^2 = x^3 + ax + b$	<p>График эллиптической кривой</p> $y^2 \equiv x^3 + ax + b \pmod{p}$ <p>(точки найдены с помощью реализованной программы)</p>
<p>$y^2 = x^3 + 2x + 3$</p> 	<p>$y^2 \equiv x^3 + 2x + 3 \pmod{97}$</p>  <p>$y^2 \equiv x^3 + 2x + 3 \pmod{641}$</p> 
	<p>$y^2 \equiv x^3 + 2x + 3 \pmod{5023}$</p> 

Выполнение задания:

Алгоритм поиска порядка начальной точки заключается в суммировании начальной точки, пока результат суммы не станет нулем:

```
p0 = Point(x0, y0)
p_next = p0 + p0
X = [p0.x, p_next.x]
Y = [p0.y, p_next.y]
n = 2
start_time = time.time()
while not(p_next.x == p_next.y == 0):
    n += 1
    p_next = p_next + p0
    if (not check(p_next.x, p_next.y)) and p_next != Point(0, 0) :
        print(n)
        print("Error!")
        break;
end_time = time.time()

print(n)
print("time: " + str((end_time - start_time) / 60) + " min")
```

Мой алгоритм каждый раз проверяет, удовлетворяет ли точка уравнению с помощью функции *check*. Также для удобства реализован класс *Point*, в котором перегружены операторы суммы и сравнения.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, q):
        if self.x == q.x and self.y == q.y: return True
        return False

    def __ne__(self, q):
        return not(self == q)

    def __add__(self, q):
        if q == Point(0, 0): return self
        if self == Point(0, 0): return q
        if q.x == self.x and q.y != self.y: return Point(0, 0)

        if (self == q):
            sign = -1 if self.y < 0 else 1
            m = ((3 * self.x * self.x + A) * sign * inverse_of(abs(2 * self.y), P)) % P
        else:
            sign = -1 if self.x - q.x < 0 else 1
            m = ((self.y - q.y) * sign * inverse_of(abs(self.x - q.x), P)) % P

        x = ((m * m) % P - self.x - q.x) % P
        y = (q.y + m * (x - q.x)) % P
        return Point(x, (-y) % P)
```

Коэффициенты *A*, *B*, *P* являются глобальными. Для поиска mod *p* от числа, возведенного в -1-ю степень используется алгоритм Евклида.

Для решения поставленной задачи я подобрала коэффициенты:

$$y^2 \equiv x^3 - 3x + 7 \pmod{p}$$

$$a = -3$$

$$b = 7$$

Начальная точка:

$$x_0 = 2$$

$$y_0 = 3$$

$$p = 64526213$$

Результат:

```
64527109  
time: 12.125018580754597 min
```

$$p = 63521179$$

Результат:

```
63520025  
time: 12.411431229114532 min
```

Для ускорения решения данной задачи существуют некоторые алгоритмы:

1. Алгоритм больших и маленьких шагов

Суть алгоритма заключается в том, чтобы перебором случайных значений x , пока правая часть $(x^3 + ax + b)$ не будет являться квадратом. Далее выбирается $P = (x_p, y_p)$ такой, что $y_p = (x^3 + ax + b)^{1/2}$

Используется Теорема Хассе и Лагранжа для нахождения интервала мощности M множества и поиск самого M , через условие: $MP = 0$.

Для данного алгоритма есть исключение: существуют два таких M , что выполняется условие. Для решения этой проблемы предлагается выбрать новую точку P .

Можно получить ускорение на F_p до $4\sqrt[4]{p}$

2. Алгоритм Шуфа

Алгоритм использует идею теоремы Хассе, о том, что существует конечное значение числа точек (используется факт ограничения сверху), а также китайскую теорему об остатках и многочленами деления.

Вычисляется $M \bmod N$, $N > 4\sqrt{p}$. Идея состоит в том, что вычисление сводится к вычислению M по модулю простых чисел, произведение которых не превосходит N (для данных вычислений используется многочлены деления).

Далее применяется китайская теорема об остатках

Сложность алгоритма: $O(n^3)$

3. Алгоритм Шуфа-Элкиса-Аткина

Данный алгоритм является улучшением алгоритма Шуфа.

Простые числа делятся на простые числа Эткинса и простые числа Аткина

Сложность алгоритма: $O(n^3 \log(n))$

Программа (файл *Budnikova_lab3.ipynb*)

```
import matplotlib.pyplot as plt
import numpy as np
import time

def plot_points(x, y):
    plt.figure(figsize=(15, 12))
    plt.plot(x, y, 'o')
    plt.show()

def check(x, y):
    return (y*y - (x**3 + (A % P)* x + B % P)) % P == 0

A = -3
B = 7
P = 64526213
x0 = 2
y0 = 3

# P = 64526213
# P = 63521179

# A = 2
# B = 3
# P = 5023
# x0 = 3
# y0 = 6

# 50231

def extended_euclidean_algorithm(a, b):
    s, old_s = 0, 1
    t, old_t = 1, 0
    r, old_r = b, a

    while r != 0:
        quotient = old_r // r
        old_r, r = r, old_r - quotient * r
        old_s, s = s, old_s - quotient * s
        old_t, t = t, old_t - quotient * t

    return old_r, old_s, old_t

def inverse_of(n, p):
    gcd, x, y = extended_euclidean_algorithm(n, p)
    assert (n * x + p * y) % p == gcd

    if gcd != 1:
        raise ValueError(
            '{} has no multiplicative inverse '
            'modulo {}'.format(n, p))
    else:
        return x % p
```



```

class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, q):
        if self.x == q.x and self.y == q.y: return True
        return False

    def __ne__(self, q):
        return not(self == q)

    def __add__(self, q):
        if q == Point(0, 0): return self
        if self == Point(0, 0): return q
        if q.x == self.x and q.y != self.y: return Point(0, 0)

        if (self == q):
            sign = -1 if self.y < 0 else 1
            m = ((3 * self.x * self.x + A) * sign * inverse_of(abs(2 * self.y), P)) % P
        else:
            sign = -1 if self.x - q.x < 0 else 1
            m = ((self.y - q.y) * sign * inverse_of(abs(self.x - q.x), P)) % P

        x = ((m * m) % P - self.x - q.x) % P
        y = (q.y + m * (x - q.x)) % P
        return Point(x, (-y) % P)

p0 = Point(x0, y0)
p_next = p0 + p0
X = [p0.x, p_next.x]
Y = [p0.y, p_next.y]
n = 2
start_time = time.time()
while not(p_next.x == p_next.y == 0):
    n += 1
    p_next = p_next + p0
    if (not check(p_next.x, p_next.y)) and p_next != Point(0, 0) :
        print(n)
        print("Error!")
        break;
end_time = time.time()

print(n)
print("time: " + str((end_time - start_time) / 60) + " min")

```

Вывод:

При выполнении данной лабораторной работы я познакомилась с эллиптическими кривыми, реализовала операцию сложения, а также метод, вычисляющий порядок точки полным перебором и ознакомилась с алгоритмами ускорения решения задачи. После подбора коэффициентов время выполнения поиска порядка точки заняло около 12 минут.