Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение   
высшего образования**  
**«Новгородский государственный университет имени Ярослава Мудрого» (НовГУ)**

Великий Новгород

**Методы защиты информации.**

Лабораторная работа №2.

Отчёт.

Студент гр. 3091

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ильин Д. А.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Жгун Т. В.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г.

г. Великий Новгород

-2024-

Оглавление

[Задание №1 3](#_Toc183094958)

[Теоретическое положение 3](#_Toc183094959)

[Исходный код 3](#_Toc183094960)

[Результаты вычислений 5](#_Toc183094961)

[Задание №2 6](#_Toc183094962)

[Теоретическое положение 6](#_Toc183094963)

[Исходный код 6](#_Toc183094964)

[Результаты вычислений 8](#_Toc183094965)

[Задание №3 9](#_Toc183094966)

[Теоретическое положение 9](#_Toc183094967)

[Исходный код 9](#_Toc183094968)

[Результаты вычислений 10](#_Toc183094969)

# Задание №1

## Теоретическое положение

Линейный конгруэнтный генератор использует следующую формулу для генерации чисел:

*X n + 1 = (a X n + c) mod m*

Где a, c и m — параметры генератора, *X0*​ — начальное значение. Для корректности генератора важно, чтобы:

* c и m были взаимно простыми.
* b=a−1было кратно каждому простому делителю m.

## Исходный код

import matplotlib.pyplot as plt

import math

# 1. Линейный конгруэнтный генератор (LCG)

# Проверка условий для LCG

def check\_lcg\_conditions(a, c, m):

# Проверка: c и m должны быть взаимно просты

if math.gcd(c, m) != 1:

raise ValueError(f"c ({c}) и m ({m}) не являются взаимно простыми (НОД ≠ 1).")

# Проверка: b = a - 1 должно быть кратно каждому простому делителю m

b = a - 1

prime\_factors = prime\_factors\_of(m)

for p in prime\_factors:

if b % p != 0:

raise ValueError(f"b ({b}) не кратно простому делителю {p} числа m ({m}).")

# Функция для нахождения простых делителей числа

def prime\_factors\_of(n):

factors = set()

while n % 2 == 0:

factors.add(2)

n //= 2

for i in range(3, int(math.sqrt(n)) + 1, 2):

while n % i == 0:

factors.add(i)

n //= i

if n > 2:

factors.add(n)

return factors

# Линейный конгруэнтный генератор (LCG) с проверкой условий

def lcg(a, c, m, seed, size):

check\_lcg\_conditions(a, c, m)

# Генерация чисел

numbers = []

x = seed

for \_ in range(size):

x = (a \* x + c) % m

numbers.append(x)

return numbers

# Построение гистограммы

def plot\_histogram(data, title, bins=256, range=(0, 255)):

plt.hist(data, bins=bins, range=range, color='blue', edgecolor='black')

plt.title(title)

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

a, c, m, seed = 1664525, 1013904223, 2\*\*8, 40

lcg\_numbers = lcg(a, c, m, seed, 50)

plot\_histogram(lcg\_numbers, "LCG Histogram 50")

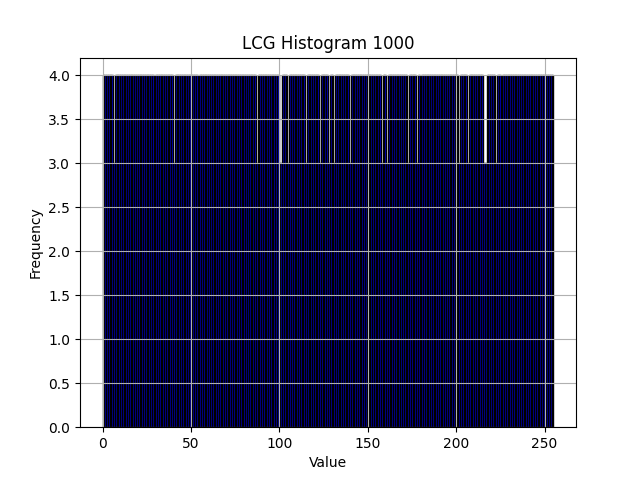
lcg\_numbers = lcg(a, c, m, seed, 100)

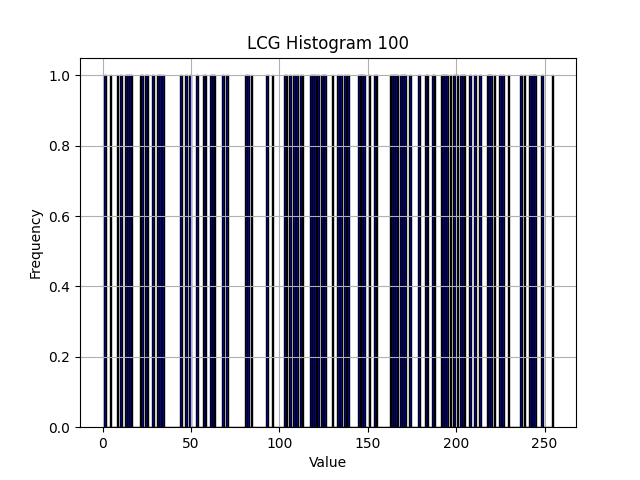
plot\_histogram(lcg\_numbers, "LCG Histogram 100")

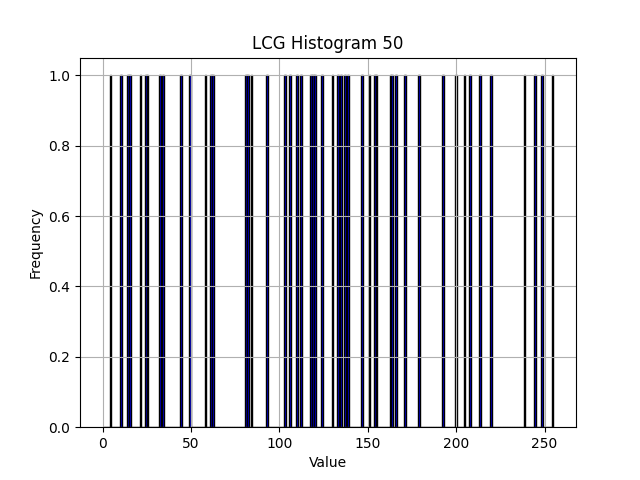
lcg\_numbers = lcg(a, c, m, seed, 1000)

plot\_histogram(lcg\_numbers, "LCG Histogram 1000")

## Результаты вычислений







# Задание №2

## Теоретическое положение

Генератор BBS основан на использовании двух простых чисел ppp и qqq, таких что p≡q≡3 mod  4, и начального значения *x0*​ ​, взаимно простого с M=p⋅q. Числа генерируются по формуле:



Для корректности генератора важно, чтобы p и q были простыми числами, а начальное значение *x0*​ — взаимно простым с M.

## Исходный код

import matplotlib.pyplot as plt

import numpy as np

import math

from collections import Counter

# 2. Генератор BBS (Blum Blum Shub)

# Проверка условий для генератора BBS

def check\_bbs\_conditions(p, q, x):

# Проверка: p и q - простые числа

if not is\_prime(p) or not is\_prime(q):

raise ValueError(f"p ({p}) и/или q ({q}) не являются простыми числами.")

# Проверка: x взаимно прост с M

M = p \* q

if math.gcd(x, M) != 1:

raise ValueError(f"seed ({x}) не является взаимно простым с M = p \* q ({M}).")

# Проверка, является ли число простым

def is\_prime(n):

if n <= 1:

return False

if n <= 3:

return True

if n % 2 == 0 or n % 3 == 0:

return False

i = 5

while i \* i <= n:

if n % i == 0 or n % (i + 2) == 0:

return False

i += 6

return True

# Генератор BBS с проверкой условий

def bbs(p, q, seed, size):

check\_bbs\_conditions(p, q, seed)

# Генерация чисел

M = p \* q

x = (seed \* seed) % M

numbers = []

for \_ in range(size):

x = (x \* x) % M

numbers.append(x)

return numbers

# Построение гистограммы

def plot\_histogram(data, title, bins=256, range=(0, 255)):

plt.hist(data, bins=bins, range=range, color='blue', edgecolor='black')

plt.title(title)

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

p, q, seed = 11, 19, 2\*\*8

bbs\_numbers = bbs(p, q, seed, 50)

plot\_histogram(bbs\_numbers, "BBS Histogram 50")

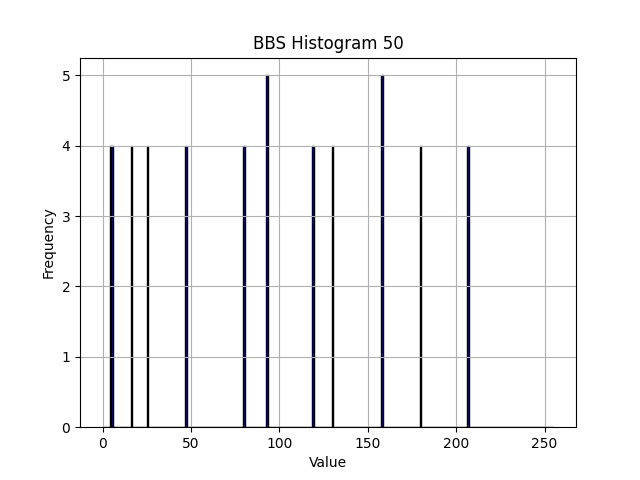
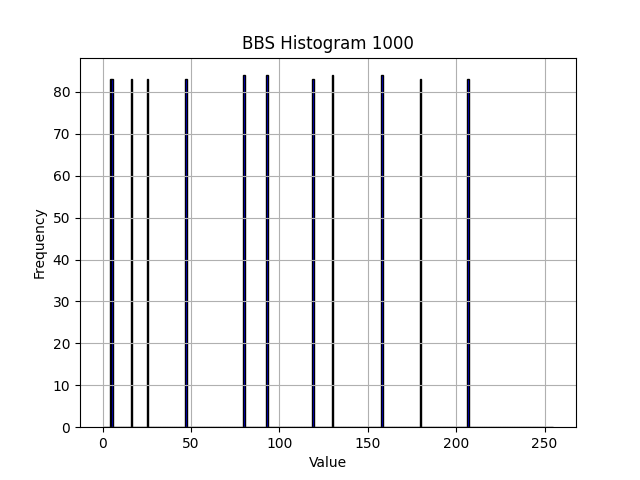
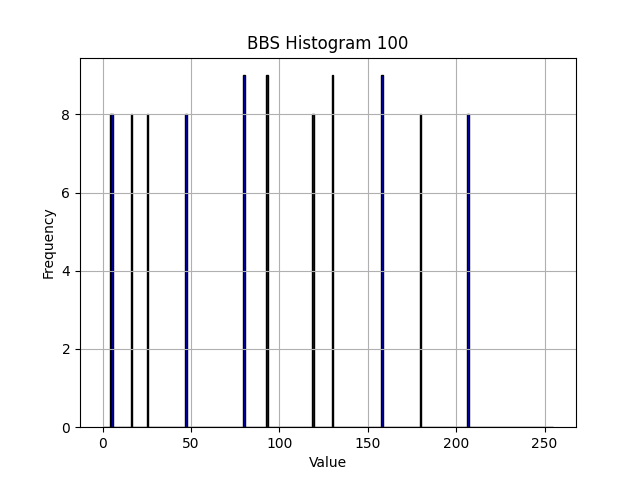
bbs\_numbers = bbs(p, q, seed, 100)

plot\_histogram(bbs\_numbers, "BBS Histogram 100")

bbs\_numbers = bbs(p, q, seed, 1000)

plot\_histogram(bbs\_numbers, "BBS Histogram 1000")

## Результаты вычислений



# Задание №3

## Теоретическое положение

LFSR — это генератор, который использует сдвиговый регистр с линейной обратной связью. Он генерирует последовательности битов, которые могут быть собраны в числа. Генератор работает с матрицей коэффициентов для создания последовательностей.

## Исходный код

import matplotlib.pyplot as plt

import numpy as np

import math

from collections import Counter

# 3. Линейный рекуррентный генератор (LFSR)

def lfsr(seed, taps, size):

state = seed

numbers = []

for \_ in range(size):

numbers.append(state[-1])

feedback = 0

for t in taps:

feedback ^= state[t]

state = [feedback] + state[:-1]

return [int(''.join(map(str, numbers[i:i+8])), 2) for i in range(0, len(numbers), 8)]

# Построение гистограммы

def plot\_histogram(data, title, bins=256, range=(0, 255)):

plt.hist(data, bins=bins, range=range, color='blue', edgecolor='black')

plt.title(title)

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.grid(True)

plt.show()

seed = [1, 0, 0, 1, 1, 0, 1, 1]

taps = [0, 1, 3, 4]

lfsr\_numbers = lfsr(seed, taps, 50)

plot\_histogram(lfsr\_numbers, "LFSR Histogram 50")

lfsr\_numbers = lfsr(seed, taps, 100)

plot\_histogram(lfsr\_numbers, "LFSR Histogram 100")

lfsr\_numbers = lfsr(seed, taps, 1000)

plot\_histogram(lfsr\_numbers, "LFSR Histogram 1000")

## Результаты вычислений

