Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Политехнический институт

Кафедра прикладной математики и информатики

**Методы защиты информации**

Лабораторная работа 2

**Генератор псевдослучайных чисел**

по дисциплине «Методы защиты информации»

по направлению подготовки 09.03.01  
 «Информатика и вычислительная техника»

направленность (профиль) «Программное обеспечение   
вычислительной техники и автоматизированных систем»

Проверил

Доцент КПМИ, доц., к. ф.-м.н.

Жгун Т.В.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Выполнил

студент группы 3091

Ильин Данила Александрович

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2024 г.

**Оглавление**

[План работы 3](#_Toc185336322)

[Теоретическое положение. 4](#_Toc185336323)

[Алгоритм работы программы 6](#_Toc185336324)

[Приложения А 7](#_Toc185336325)

[Приложение Б 10](#_Toc185336326)

# План работы

Реализовать алгоритмы:

1. Линейного конгруэнтного генератора (*LCG*)
2. Генератора псевдослучайных чисел *BBS* (L. Blum, M. Blum, M. Shub)
3. Линейным рекуррентным генератором (*LFSR*) при *n*=8.

Реализовать приложения для работы с 3 генераторами псевдослучайных чисел

# Теоретическое положение

**Линейный конгруэнтный генератор (LCG) -** Основан на использовании рекуррентного соотношения:

где a, c, m, *X0*​ являются целыми числами. Особенность LCG заключается в том, что при правильном выборе параметров последовательность достигает максимального периода m. Для этого необходимо выполнение следующих условий:

1. Параметры c и m должны быть взаимно простыми, что означает, что их наибольший общий делитель равен 1.
2. Разность b = a - 1 должна быть кратной каждому простому числу, являющемуся делителем mm.
3. Если m делится на 4, то b также должно быть кратным 4.

Часто для m выбираются степени двойки 2n или простые числа Мерсенна. Это делает генератор эффективным в вычислительном плане и удобным для реализации. Однако диапазон m в данной задаче ограничен значениями от 200 до 255, что может влиять на период и свойства получаемой последовательности.

**Генератор BBS (Blum, Blum, Shub)** - основывается на свойствах квадратных вычетов и является примером криптографически стойкого метода. Его работа строится следующим образом:

1. Выбираются два больших простых числа p и q, которые затем перемножаются для получения числа M= p ⋅ q, называемого числом Блюма.
2. Выбирается случайное целое число x, взаимно простое с M, что обеспечивает единственность последовательности.
3. Вычисляется начальное значение x, после чего последовательность определяется рекуррентным соотношением:

Результатом каждого шага служит младший бит числа xn+1 или его бит четности. Этот метод отличается высокой степенью безопасности и используется в криптографических приложениях. Примечательно, что параметрыp и q для задачи находятся в диапазоне [0, 255], что ограничивает величину M.

**Линейный рекуррентный генератор (LFSR)** - представляет собой модель генерации последовательностей в поле GF(2) и задается следующим уравнением:

где A — матрица перехода, связанная с неприводимым многочленом в поле GF(2). Для максимальной длины последовательности выбирается многочлен , что гарантирует достижение полной периодичности. Последовательности, генерируемые LFSR, имеют вид битовых строк, которые удобно представлять в виде восьмибитных отрезков. Это позволяет проводить частотный анализ, например, с использованием гистограмм.

**Применение генераторов и анализ результатов:**

Для оценки качества работы генераторов производится построение гистограмм на основе последовательностей длиной 50, 100 и 1000 байт. Каждая гистограмма отражает частотное распределение значений в диапазоне [0, 255]. Этот анализ позволяет визуально оценить равномерность распределения и случайность последовательности, что важно для проверки корректности генерации.

# Алгоритм работы программы

Данная программа представляет собой графическое приложение, разработанное с использованием библиотеки **tkinter**, которое позволяет генерировать и анализировать последовательности псевдослучайных чисел (ПСЧ). Программа создана с целью демонстрации работы различных алгоритмов генерации случайных чисел и предоставляет пользователю удобный интерфейс для выбора метода, задания параметров генерации и визуального анализа полученных данных.

Программа имеет интуитивно понятный интерфейс, который включает выпадающий список для выбора метода генерации, текстовые поля для ввода параметров (начального значения — Seed, количества чисел), кнопки управления и область для отображения результатов в текстовом формате, а также гистограммы, отображающей распределение сгенерированных чисел.

После запуска программы пользователь видит окно с основными элементами управления. Первым шагом необходимо выбрать один из трёх доступных методов генерации псевдослучайных чисел:

1. **LCG (Линейный конгруэнтный генератор)** — простой и классический метод, основанный на использовании линейного сравнения. Подходит для быстрого получения последовательностей с достаточно равномерным распределением.
2. **BBS (Blum Blum Shub)** — криптографически устойчивый генератор, обеспечивающий высокий уровень случайности. Работает медленнее, но лучше подходит для задач, требующих высокой безопасности.
3. **LFSR (Сдвиговый регистр с линейной обратной связью)** — метод, использующий регистры сдвига и обратную связь для генерации последовательностей. Отличается высокой скоростью работы.

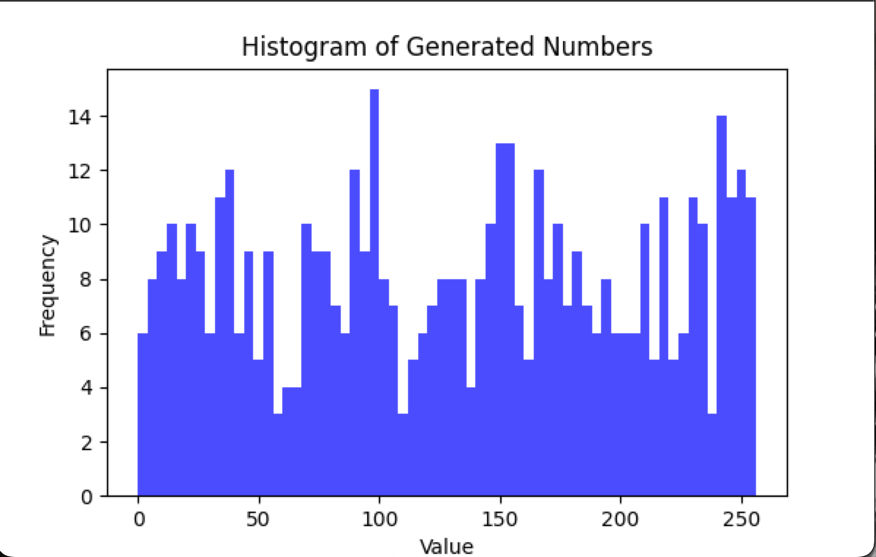
После выбора метода пользователь вводит параметры генерации:

* **Seed (начальное значение)**: число, с которого начинается генерация. Этот параметр определяет начальное состояние генератора.
* **Количество чисел**: задаёт, сколько чисел требуется сгенерировать в последовательности.

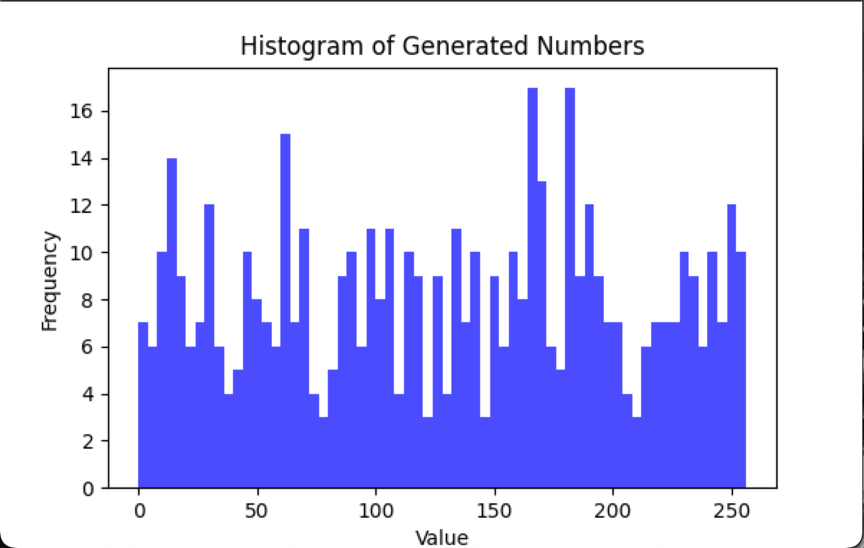
Для запуска процесса генерации пользователь нажимает кнопку **Generate**, после чего выбранный алгоритм производит расчёты и выведет гистограмму.

Для демонстрации в приложения A данного отчета предоставлены примеры работы программы. В приложение Б расположен исходный код приложения.

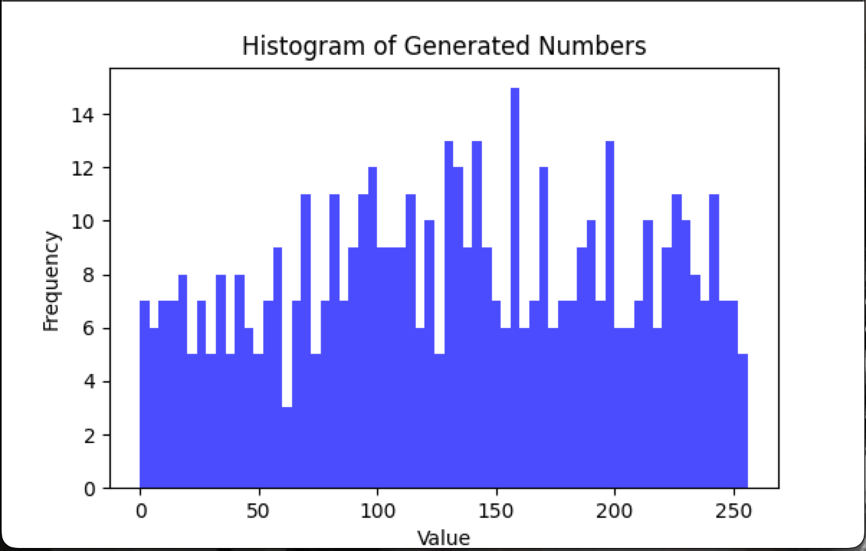
# Приложения А



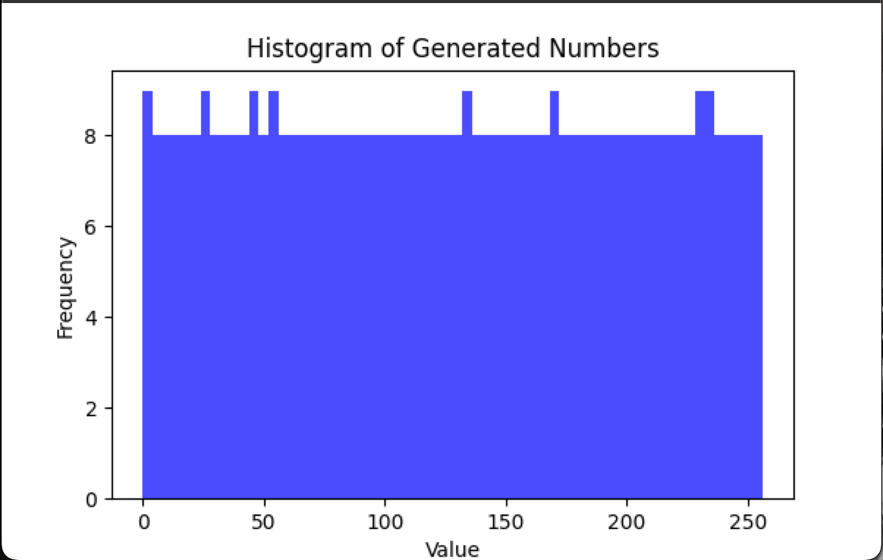
Изображение 1 seed 250 BBS-50



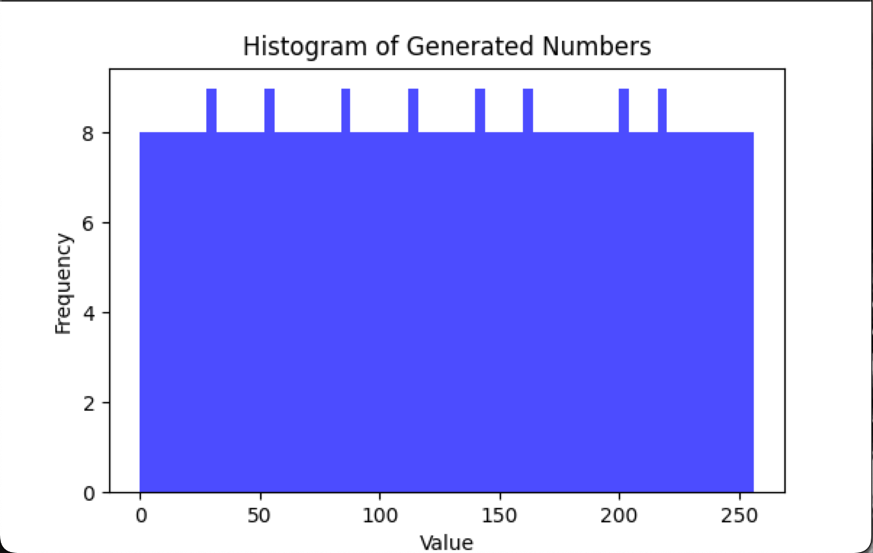
Изображение 2 seed 250 BBS-100



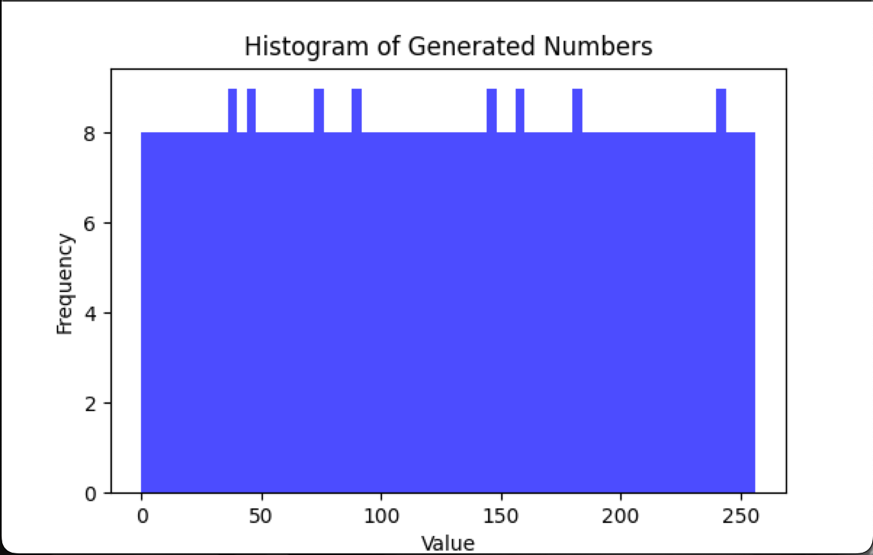
Изображение 3 seed 250 BBS-1000



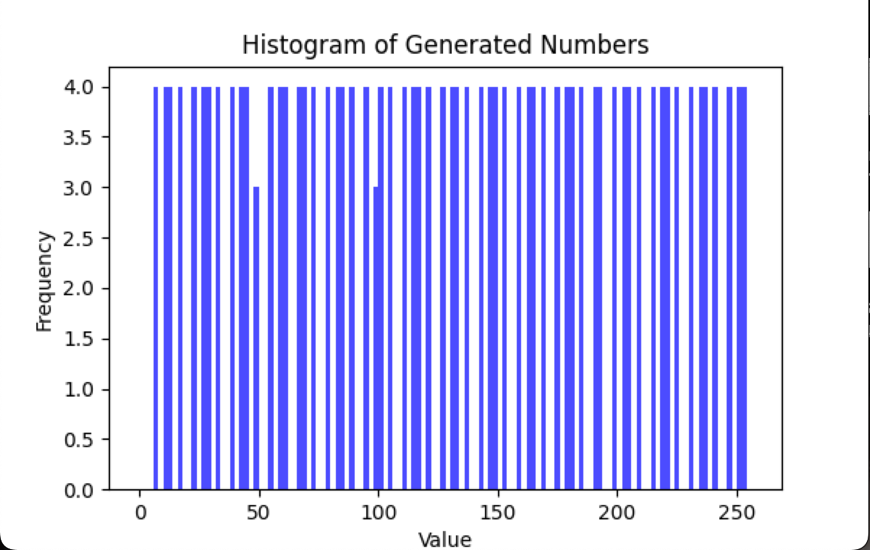
Изображение 4 seed 250 LCG-50



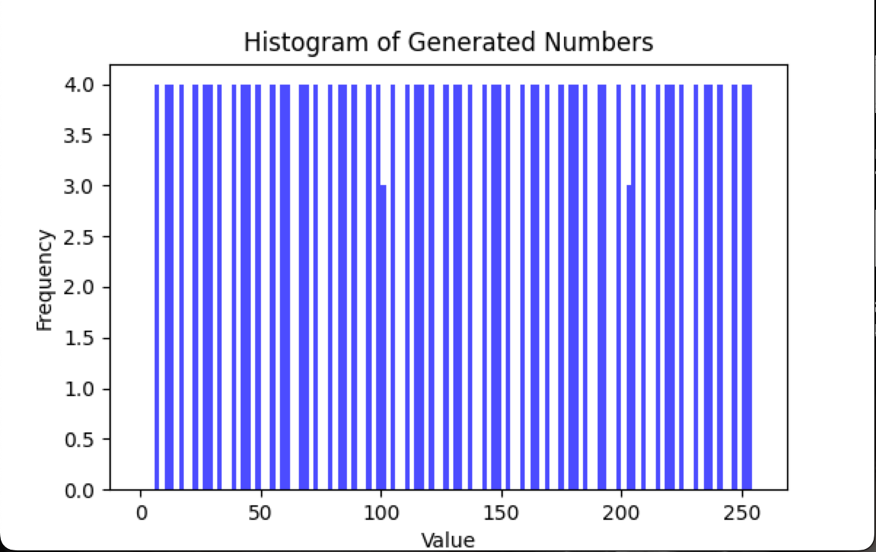
Изображение 5 seed 250 LCG-100



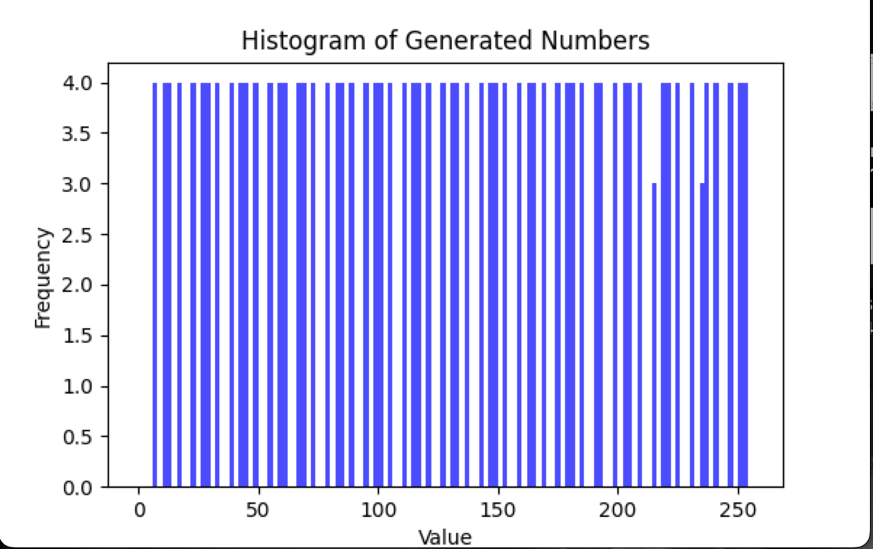
Изображение 6 seed 250 LCG-1000



Изображение 7 seed 250 LFSR-50



Изображение 8 seed 250 LFSR-100



Изображение 9 seed 250 LFSR-1000

# Приложение Б

import tkinter as tk

from tkinter import ttk

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

# Реализация линейного конгруэнтного генератора (LCG)

def lcg\_numpy(seed, length):

# Параметры для LCG

a, c, m = 1664525, 1013904223, 2 \*\* 32 # Классические параметры LCG

x = np.empty(length, dtype=np.uint8)

value = seed

for i in range(length):

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

x[i] = value % 256 # Последние 8 бит результата

return x.tolist()

# Реализация генератора BBS (Blum Blum Shub) с использованием двоичных операций

def bbs\_binary(seed, length):

# Параметры для BBS

p, q = 383, 503 # Простые числа Блюма

M = p \* q

x = seed % M

sequence = []

for \_ in range(length):

x = pow(x, 2, M)

sequence.append(x & 0xFF) # Последние 8 бит результата

return sequence

# Реализация LFSR с использованием битовых операций и двоичного представления

def lfsr\_bitwise(seed, length):

# Параметры для LFSR

# bit = (seed ^ (seed >> 3) ^ (seed >> 4) ^ (seed >> 5)) & 1

# Параметры для LFSR

taps = [7, 4, 3, 2, 0] # Полином x^8 + x^5 + x^4 + x^3 + 1

state = seed & 0xFF # Обрезаем если сид больше 8

sequence = []

for \_ in range(length):

sequence.append(state & 0xFF)

new\_bit = 0

for tap in taps:

new\_bit ^= state >> (tap % 8)

sequence.append(state)

state = ((state >> 1) | (new\_bit << 7)) & 0xFF # Последние 8 бит результата

return sequence

class RNGApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Pseudorandom Number Generator")

# Начальные размеры окна

self.root.geometry("600x600")

self.root.resizable(width=True, height=True)

# Переменные

self.method\_var = tk.StringVar(value="LCG")

self.length\_var = tk.StringVar()

self.seed\_var = tk.StringVar()

# Выбор метода

ttk.Label(root, text="Select Method:").pack(pady=5)

self.method\_menu = ttk.Combobox(root, textvariable=self.method\_var, values=["LCG", "BBS", "LFSR"])

self.method\_menu.pack()

# Ввод длины

ttk.Label(root, text="Number of Values to Generate:").pack(pady=5)

self.length\_entry = ttk.Entry(root, textvariable=self.length\_var)

self.length\_entry.pack()

# Ввод начального значения

ttk.Label(root, text="Seed Value:").pack(pady=5)

self.seed\_entry = ttk.Entry(root, textvariable=self.seed\_var)

self.seed\_entry.pack()

# Кнопка генерации

self.generate\_button = ttk.Button(root, text="Generate", command=self.generate\_numbers)

self.generate\_button.pack(pady=10)

# Фрейм для графика

self.canvas\_frame = tk.Frame(root)

self.canvas\_frame.pack(fill=tk.BOTH, expand=True)

self.canvas\_frame.pack\_propagate(True)

def generate\_numbers(self):

method = self.method\_var.get()

length = int(self.length\_var.get())

seed = int(self.seed\_var.get())

# Выбор метода генерации

if method == "LCG":

numbers = lcg\_numpy(seed, length)

elif method == "BBS":

numbers = bbs\_binary(seed, length)

elif method == "LFSR":

numbers = lfsr\_bitwise(seed, length)

self.plot\_histogram(numbers)

def plot\_histogram(self, numbers):

# Очистка старого графика

for widget in self.canvas\_frame.winfo\_children():

widget.destroy()

# Создание графика

fig, ax = plt.subplots(figsize=(6, 4))

bins = 64 # Увеличиваем количество интервалов для более узких столбцов

hist, bin\_edges, \_ = ax.hist(numbers, bins=bins, range=(0, 256), color='blue', alpha=0.7)

ax.set\_title("Histogram of Generated Numbers")

ax.set\_xlabel("Value")

ax.set\_ylabel("Frequency")

# Добавление графика в tkinter

canvas = FigureCanvasTkAgg(fig, master=self.canvas\_frame)

canvas.draw()

canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

# Обновление высоты окна в зависимости от графика

self.root.update\_idletasks()

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

app = RNGApp(root)

root.mainloop()