**Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования** **«Новгородский государственный университет имени Ярослава Мудрого»**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра ИТиС

Лабораторная работа №1

По дисциплине: Защита информации

Отчет

Выполнил: студент гр. 9091

\_\_\_\_\_\_\_\_\_\_\_Чалый.С.М

Проверил: Преподаватель

\_\_\_\_\_\_\_\_\_\_\_Жгун.Т.В

Великий Новгород

2022

1. **Формулировка цели и задач**

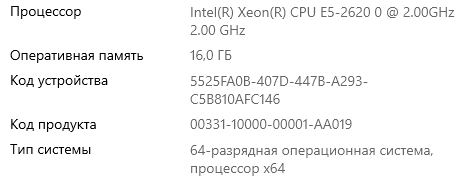
Целью данной работы является реализация работыгенераторов псевдослучайных величин.

1. Линейного конгруэнтного генератора (*LCG*)
2. Генератора псевдослучайных чисел *BBS*

3 Линейным рекуррентным генератором (*LFSR*) при *n*=5.

1. **Текст программы**

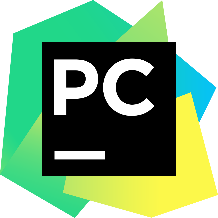
**2.1 Тестирование и запуск.**



Необходимые средства применяемые мною при выполнении данной работы:

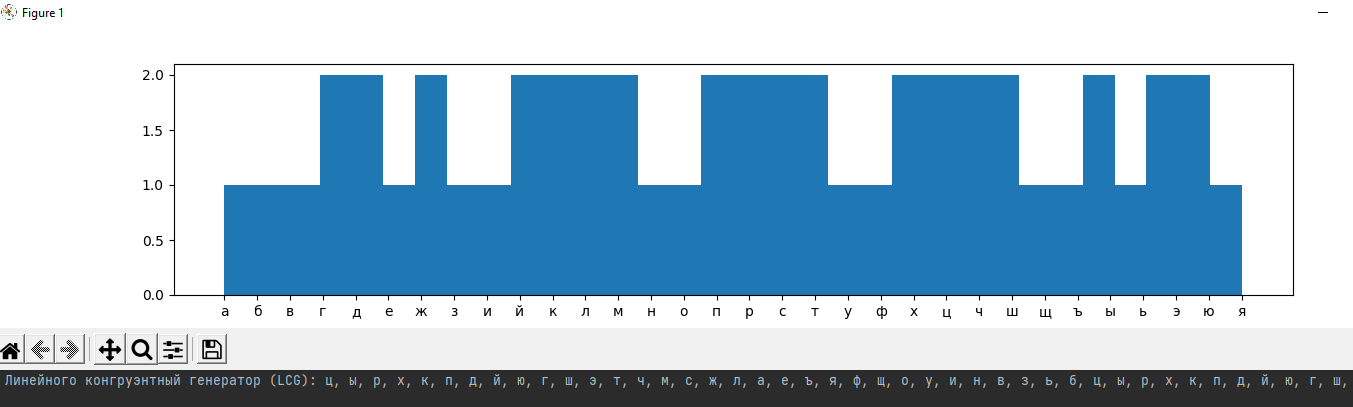
IntelliJ IDEA Educational Edition 2021.2.2

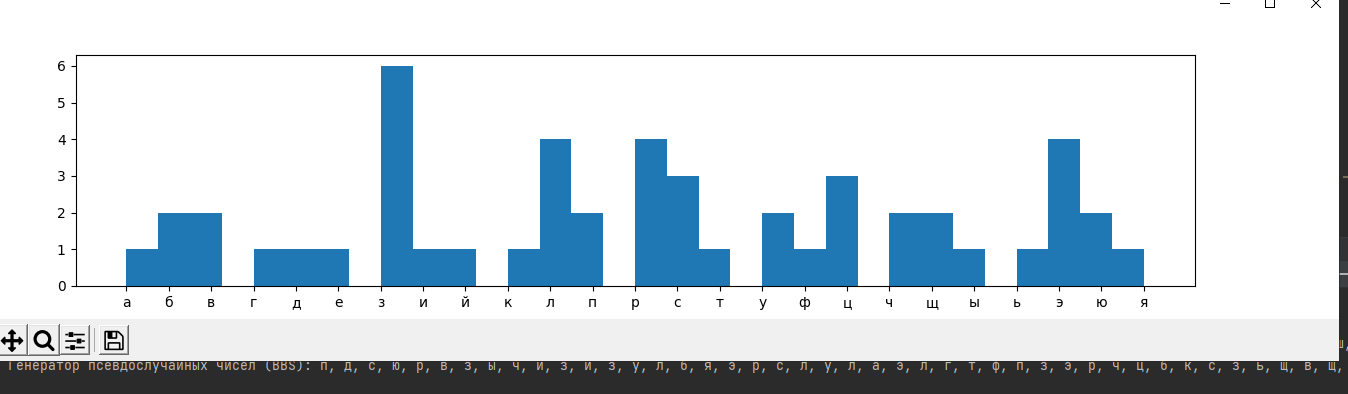
* Version: 2021.3.2
* Build: 213.6777.50
* 31 January 2022
* Community
* Python 3.10

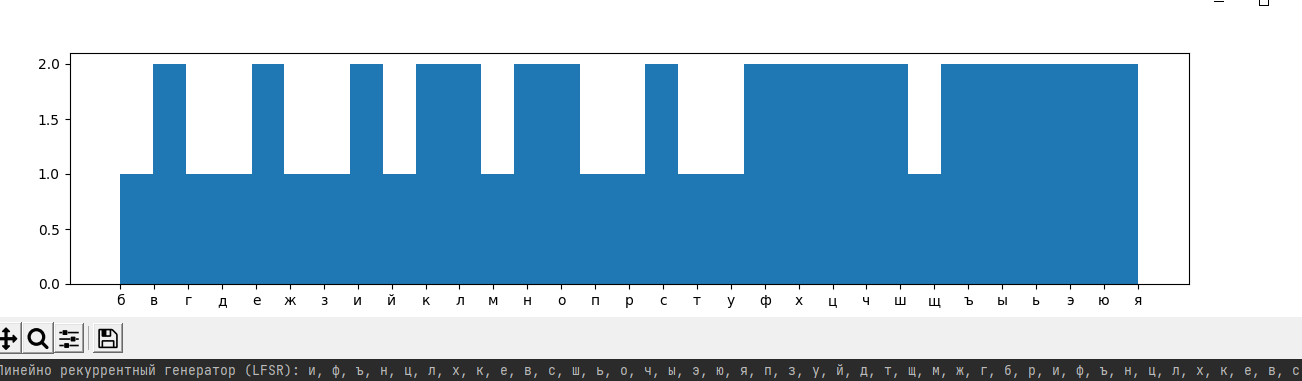


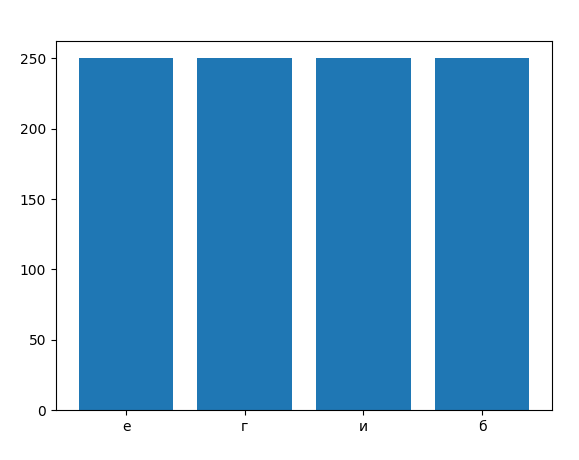
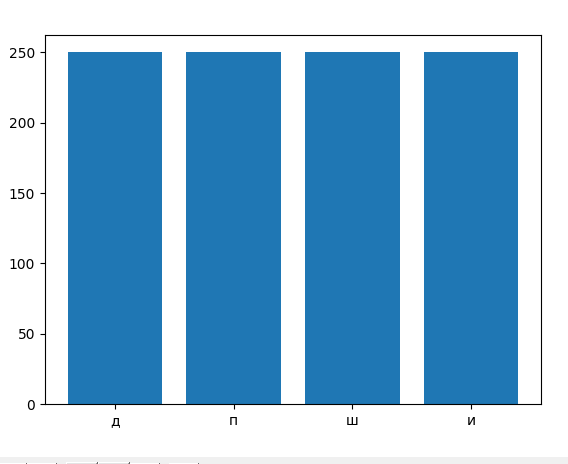
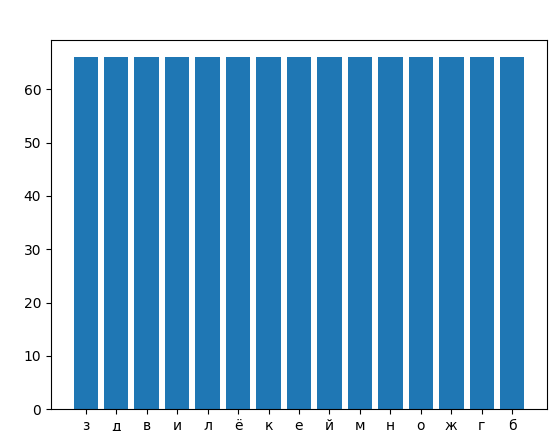
"**2.1 Тестирование и запуск**".

import numpy as np  
import matplotlib.pyplot as plt  
from itertools import islice  
import itertools  
  
  
def is\_coprime(x, y):  
 return np.gcd(x, y) == 1  
  
  
def is\_prime(n):  
 if n < 2:  
 return False  
 for number in islice(itertools.count(2), int(np.sqrt(n) - 1)):  
 if n % number == 0:  
 return False  
  
 return True  
  
  
class Generator:  
 # linear congruential generator  
 def lcg(self, limit, a, c, modulus, seed=1):  
 fvalue = 0  
 x = seed  
 while fvalue < limit:  
 fvalue += 1  
 x = (a \* x + c) % modulus  
 yield x  
  
 # Blum Blum Shub  
 def bbs(self, limit, p, q, seed=1):  
 np.random.seed(seed)  
 if not (is\_prime(p) and p % 4 == 3 and is\_prime(q) and q % 4 == 3):  
 raise ValueError("Wrong p or q")  
 M = p \* q  
 # print(f"p = {p}")  
 # print(f"q = {q}")  
 # print(f"M = {M}")  
 xs = np.array(np.nonzero(is\_coprime(np.arange(2, M, dtype="int"), M)))[0]  
 x\_ind = np.random.randint(0, len(xs))  
 np.random.seed()  
 x = xs[x\_ind]  
 # print(x)  
 # print(f"x = {x}")  
 fvalue = 0  
 while fvalue < limit:  
 fvalue += 1  
 out = 0  
 for i in range(5):  
 x = x \*\* 2 % M  
 # print(x)  
 out \*= 2  
 # print(x % 2)  
 out += x % 2  
 yield out  
  
 # linear feedback shift register  
 def lfsr(self, limit, polynom):  
 n = 5  
 A = np.eye(5, 5, -1)  
 # https://books.ifmo.ru/file/pdf/958.pdf  
 A[0] = polynom  
 # print(A)  
 x = (1, 0, 0, 0, 0)  
 fvalue = 0  
 while fvalue < limit:  
 fvalue += 1  
  
 x = A.dot(x) % 2  
 # print(x)  
 yield np.sum(x \* (16, 8, 4, 2, 1))  
  
  
count = 1000  
m = 32  
a = 17  
c = 5  
  
generator = Generator()  
values = []  
for value in generator.lcg(count, a, c, m):  
 values.append(value)  
  
plt.hist(values, bins=m)  
plt.title("Линейного конгруэнтный генератор (LCG)")  
plt.xlim(0, 31)  
plt.show()  
  
values = []  
for value in generator.bbs(10000, 2851, 3559, seed=1):  
 values.append(value)  
# print(values)  
plt.hist(values, bins=m)  
plt.xlim(0, 31)  
plt.title("Генератор псевдослучайных чисел (BBS)")  
plt.show()  
  
values = []  
for value in generator.lfsr(1000, (0, 1, 1, 1, 1)):  
 values.append(value)  
c = np.bincount(values)  
plt.hist(values, bins=31)  
plt.xlim(0, 31)  
plt.title("Линейно рекуррентный генератор (LFSR)")  
plt.show()  
  
generators = [  
 {  
 "name": "Линейного конгруэнтный генератор (LCG)",  
 "generator": generator.lcg,  
 "params": {"a": 17, "c": 5, "m": 32},  
 "bins": 32  
 },  
 {  
 "name": "Генератор псевдослучайных чисел (BBS)",  
 "generator": generator.bbs,  
 "params": {"p": 2851, "q": 3559},  
 "bins": 32  
 },  
 {  
 "name": "Линейно рекуррентный генератор (LFSR)",  
 "generator": generator.lfsr,  
 "params": {"polynom": (0, 1, 1, 1, 1)},  
 "bins": 31  
 }  
]  
  
  
def to\_char(i):  
 return chr(ord('а') + int(i))  
  
  
for i in [50, 100, 1000]:  
 print(f"Длинна: {i}")  
 for gen in generators:  
 xs = [x for x in gen["generator"](i, \*(gen["params"]).values())]  
 xs\_str = list(map(to\_char, xs))  
 print(f"{gen['name']}: {', '.join(xs\_str)}")  
 plt.rcParams["figure.figsize"] = (20, 3)  
  
 plt.hist(sorted(xs\_str), bins=gen["bins"])  
 plt.show()  
 # print(i, \*(gen["params"]).values())  
 # for x in :  
 # print(x)

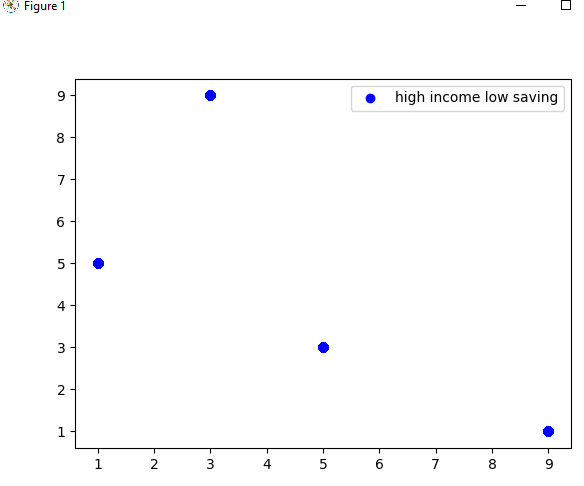
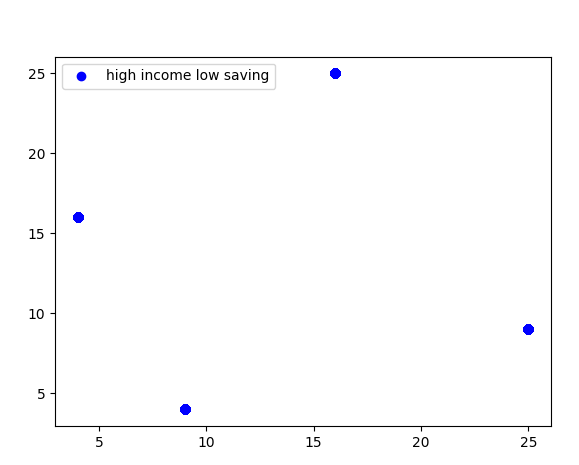
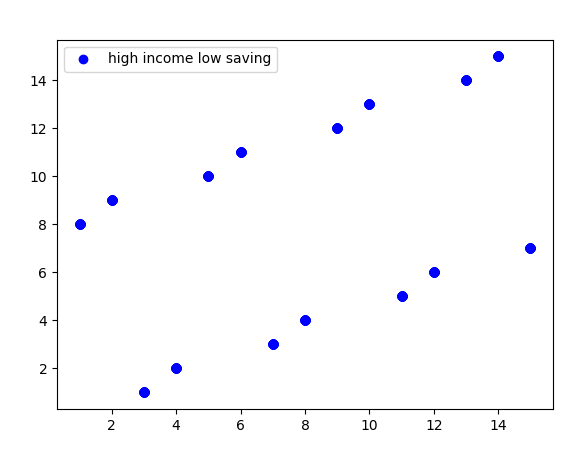




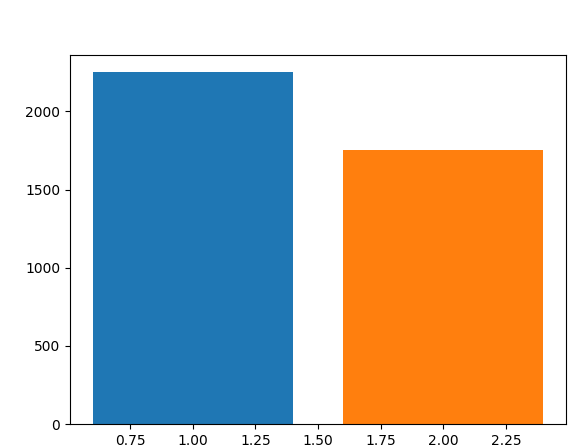
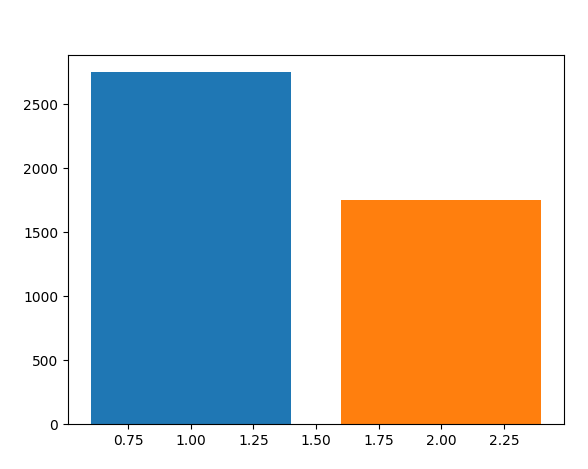
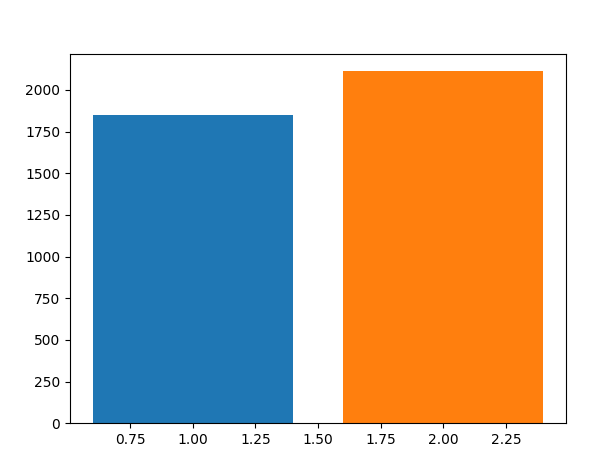


2)

3)

4)

