Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

# ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

# ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ КУРСУ**

студента 4 курса 431 группы

специальности 10.05.01 «Компьютерная безопасность» факультета компьютерных наук и информационных технологий

Серебрякова Алексея Владимировича

Научный руководитель

Ассистент Н.А. Артемова

подпись, дата

Саратов 2022

**Задание 1. Генерация псевдослучайных чисел.**

**Описание задания**: создать программу для генерации псевдослучайных величин следующими алгоритмами:

a. Линейный конгруэнтный метод;

b. Аддитивный метод;

c. Пятипараметрический метод;

d. Регистр сдвига с обратной связью (РСЛОС);

e. Нелинейная комбинация РСЛОС;

f. Вихрь Мерсенна;

g. RC4;

h. ГПСЧ на основе RSA;

i. Алгоритм Блюма-Блюма-Шуба;

Для управления приложением предлагается следующий формат параметров командной строки:

/g:<код\_метода> - параметр указывает на метод генерации ПСЧ, при этом код\_метода может быть одним из следующих:

· lc – линейный конгруэнтный метод;

· add – аддитивный метод;

· 5p – пятипараметрический метод;

· lfsr – регистр сдвига с обратной связью (РСЛОС);

· nfsr – нелинейная комбинация РСЛОС;

· mt – вихрь Мерсенна;

· rc4 – RC4;

· rsa – ГПСЧ на основе RSA;

· bbs – алгоритм Блюма-Блюма-Шуба;

/i:<число> - инициализационный вектор генератора.

/n:<длина> - количество генерируемых чисел. Если параметр не указан, - генерируется 10000 чисел.

/f:<полное\_имя\_файла> - полное имя файла, в который будут выводиться данные. Если параметр не указан, данные должны записываться в файл с именем rnd.dat.

/h – информация о допустимых параметрах командной строки программы.

**Алгоритм 1. Линейный конгруэнтный метод.**

**Описание алгоритма.**

Одним из простых и популярных методов сейчас является линейный конгруэнтный метод (ЛКМ), предложенный Д.Г. Лехмером в 1949 году. В его основе лежит выбор четырех ключевых чисел:

* m > 0, модуль;
* 0 ≤ a ≤ m, множитель;
* 0 ≤ c ≤ m, приращение (инкремент);
* 0 ≤ ≤ m, начальное значение.

Последовательность ПСЧ, получаемая по формуле:

называется линейной конгруэнтной последовательностью (ЛКП). Ключом для неё служит

**Параметры запуска программы**:

/g:lc /i:1223,7,11,3 /n:10000 /f:rnd.dat

**Код программы:**

За работу данного генератора отвечает функция:

**def** **lc**(IV, seq\_len, path):

seq\_num = []

md, mult, inc, init\_value = IV

seq\_num.append(init\_value)

acc = init\_value

perc\_25, perc\_50, perc\_75 = seq\_len // 4, seq\_len // 2, seq\_len // 4 \* 3

print('[STATUS] Генерация чисел: 0%')

**for** i **in** range(1, seq\_len):

acc = (mult \* acc + inc) % md

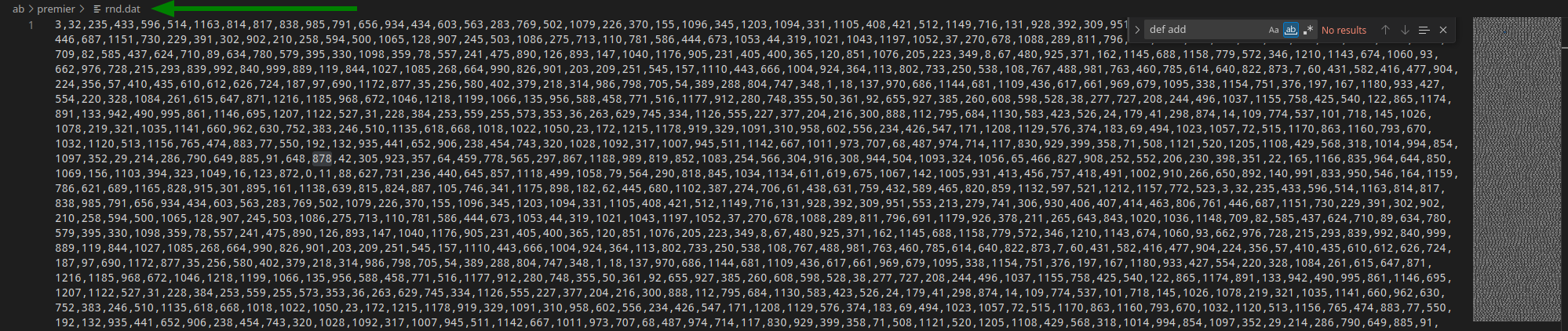
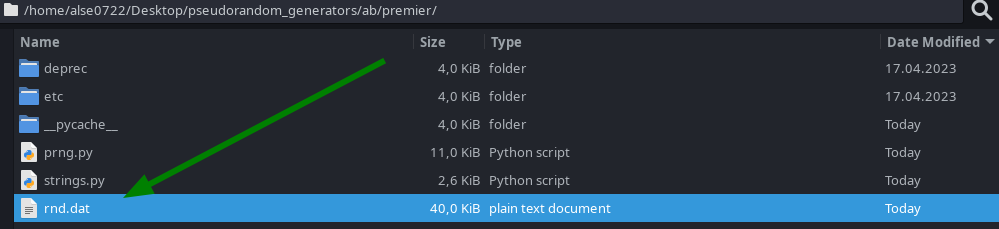
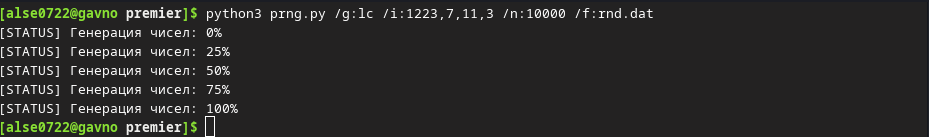
seq\_num.append(acc)

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 2. Аддитивный метод.**

**Описание алгоритма.**

Формула:

Параметры, поступающие на вход: модуль m, коэффициент k, коэффициент j, числа .

**Параметры запуска программы**:

/g:add/i:100,24,55,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55 /n:10000 /f:rnd\_add.dat

**Код программы:**

За работу данного генератора отвечает функция:

**def** **add**(IV, seq\_len, path):

md, k\_delay, j\_delay, seq\_num = IV[0], IV[1], IV[2], IV[3:]

perc\_25, perc\_50, perc\_75 = seq\_len // 4, seq\_len // 2, seq\_len // 4 \* 3

print('[STATUS] Генерация чисел: 0%')

**for** i **in** range(seq\_len):

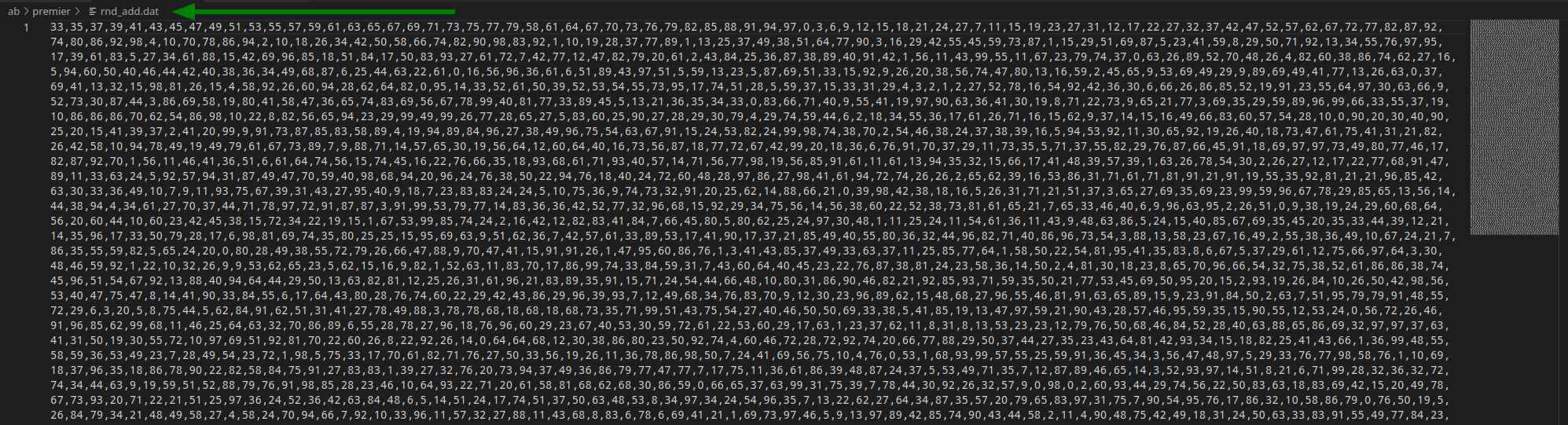
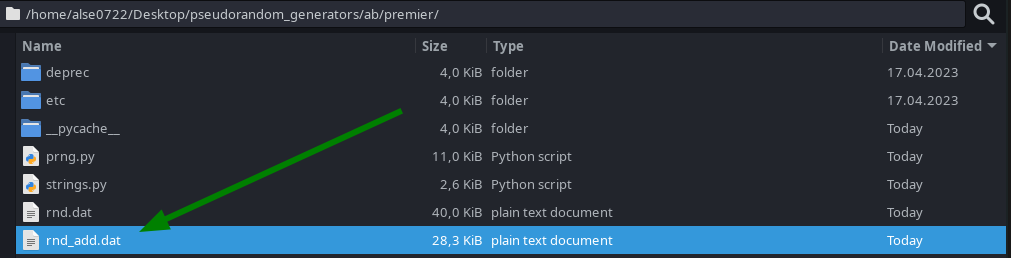
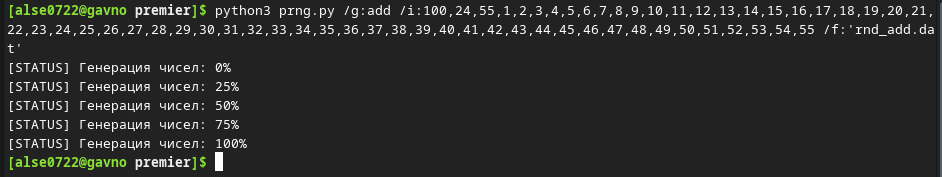
seq\_num.append((seq\_num[j\_delay - k\_delay + i] + seq\_num[i]) % md)

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

save\_as\_file(seq\_num[-seq\_len:], seq\_len, path)

**Пример работы программы:**



**Алгоритм 3. Пятипараметрический метод.**

**Описание алгоритма.**

Данный метод является частным случаем РСЛОС, использует характеристический многочлен из 5 членов и позволяет генерировать последовательности w-битовых двоичных целых чисел в соответствии со следующей рекуррентной формулой:

При этом

Параметры и , первоначально задают как начальный вектор.

**Параметры запуска программы**:

/g:5p /i:89,20,40,69,10 /n:10000 /f:rnd\_five.dat

**Код программы:**

За работу данного генератора отвечают функции:

**def** **lfsr\_machinerie**(seq\_len, p, w, nonzero\_coeffs, init\_register=[]):

seq\_num, bit\_seq = [], []

perc\_25, perc\_50, perc\_75 = seq\_len // 4 \* w, seq\_len // 2 \* w, seq\_len // 4 \* 3 \* w

print('[STATUS] Генерация чисел: 0%')

**if** len(init\_register) == 0:

bit\_seq = [random.randint(0, 1) **for** k **in** range(p)]

bit\_seq.reverse()

**else**:

bit\_seq = init\_register

**for** i **in** range(seq\_len \* w):

bit\_seq.append(sum(bit\_seq[i + nonzero\_coeffs[k]] **for** k **in** range(len(nonzero\_coeffs))) % 2)

**if** i % (w - 1) == 0 **and** i != 0:

current\_binary = bit\_seq[-w:]

binary\_converted = sum([0 **if** current\_binary[i] == 0 **else** 2 \*\* i **for** i **in** range(w)])

seq\_num.append(binary\_converted)

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

**return** seq\_num

**def** **five\_p**(IV, seq\_len, path):

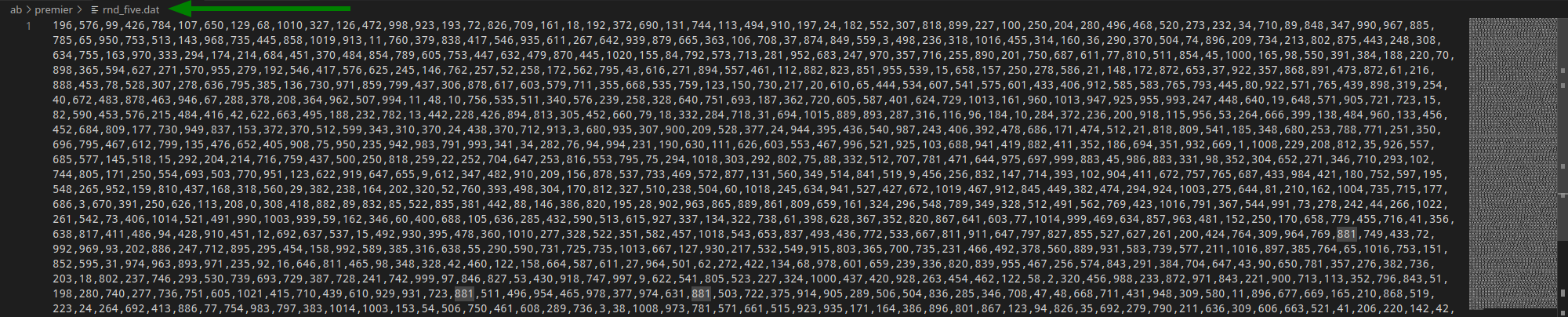
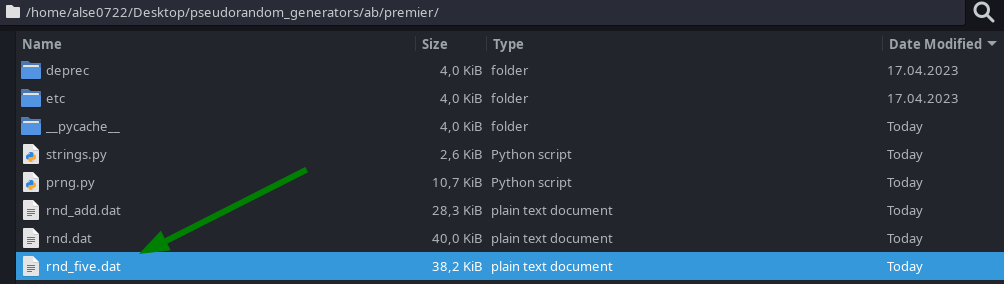
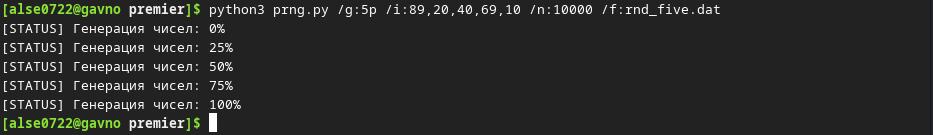
p, q\_1, q\_2, q\_3, w = IV

nonzero\_coeffs = [0, q\_1, q\_2, q\_3]

seq\_num = lfsr\_machinerie(seq\_len, p, w, nonzero\_coeffs)

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 4. Регистр сдвига с обратной связью (РСЛОС).**

**Описание алгоритма.**

Регистр сдвига с обратной связью по переносу (РСОСП) в отличие от РСЛОС имеет дополнительный регистр – регистр переноса. Данный регистр является не битом, а числом, которое есть результат преобразования битов, полученных из основного регистра. Это преобразование называется функцией обратной связи.

Рассмотрим двоичную последовательность, получаемую следующим образом:

*,* исходно заданы значения:

Функция:

На вход подаются параметры: p, s, m, , .

При этом

**Параметры запуска программы**:

/g:lfsr /i:1000010001,1010101110100011100010101010,10 /n:10000 /f:'rnd\_lfsr.dat'

**Код программы:**

За работу данного генератора отвечают функции:

**def** **lfsr\_machinerie**(seq\_len, p, w, nonzero\_coeffs, init\_register=[]):

seq\_num, bit\_seq = [], []

perc\_25, perc\_50, perc\_75 = seq\_len // 4 \* w, seq\_len // 2 \* w, seq\_len // 4 \* 3 \* w

print('[STATUS] Генерация чисел: 0%')

**if** len(init\_register) == 0:

bit\_seq = [random.randint(0, 1) **for** k **in** range(p)]

bit\_seq.reverse()

**else**:

bit\_seq = init\_register

**for** i **in** range(seq\_len \* w):

bit\_seq.append(sum(bit\_seq[i + nonzero\_coeffs[k]] **for** k **in** range(len(nonzero\_coeffs))) % 2)

**if** i % (w - 1) == 0 **and** i != 0:

current\_binary = bit\_seq[-w:]

binary\_converted = sum([0 **if** current\_binary[i] == 0 **else** 2 \*\* i **for** i **in** range(w)])

seq\_num.append(binary\_converted)

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

**return** seq\_num

**def** **lfsr**(IV, seq\_len, path):

coef\_vec, init\_register, w = IV

coef\_vec, init\_register = list(map(int, str(coef\_vec))), list(map(int, str(init\_register)))

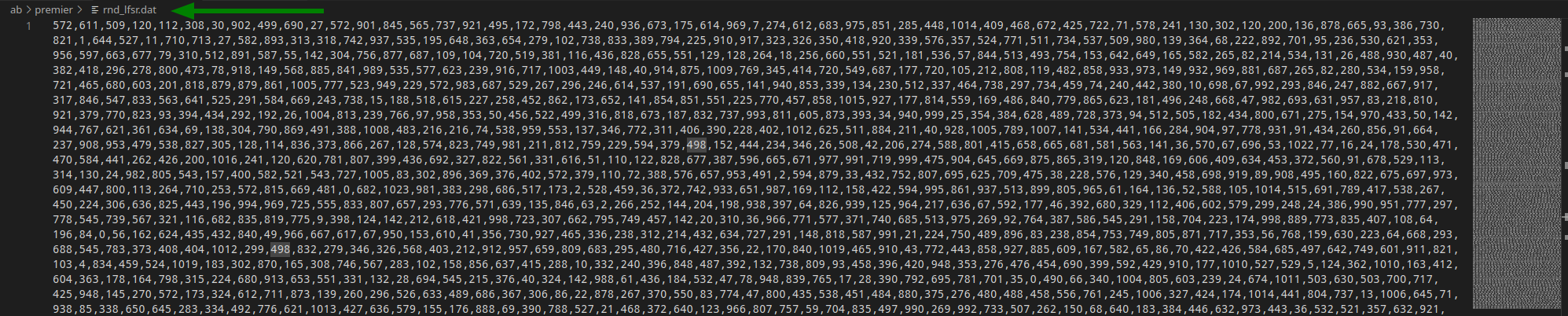
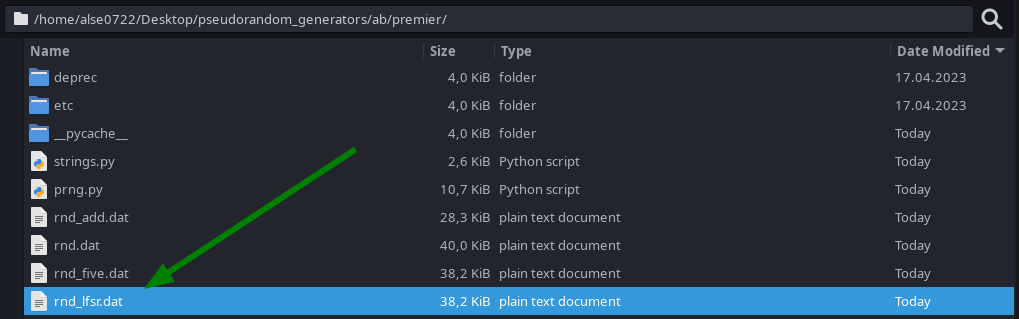
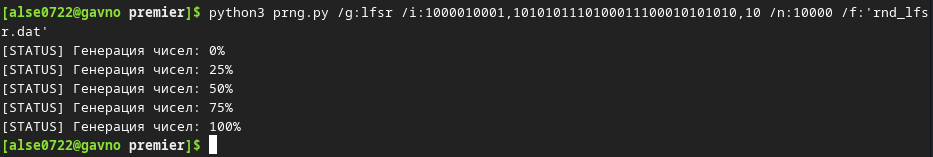
coef\_vec.reverse(), init\_register.reverse()

nonzero\_coeffs = [i **for** i **in** range(len(coef\_vec)) **if** coef\_vec[i] == 1]

seq\_num = lfsr\_machinerie(seq\_len, 0, w, nonzero\_coeffs, init\_register)

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 5. Нелинейная комбинация РСЛОС.**

**Описание алгоритма.**

Имеется комбинация РСЛОС. Каждый из РСЛОС генерирует очередной бит. Вычисляется функция от значений битов, полученных из РСЛОС и полученный бит подаётся на вывод.

РСЛОС 1 выдаёт последовательность бит:

РСЛОС 2 выдаёт последовательность бит:

.................................................

РСЛОС k выдаёт последовательность бит:

Формируется последовательность бит:

Она разбивается на блоки по w бит и формируется последовательность выходных чисел:

На вход подаются параметры: ,…,.

При этом

**Параметры запуска программы**:

/g:nfsr/i:10101,1000011001,10101000000001,10101001,10101000111001,1010010101010000101011100001,10 /n:10000 /f:'rnd\_nfsr.dat'

**Код программы:**

За работу данного генератора отвечают функции:

**def** **bit\_array\_conversion**(input\_register):

input\_register = list(map(int, str(input\_register)))

input\_register.reverse()

**return** input\_register

**def** **bitwise\_or**(fb\_rev, sb\_rev):

fb\_rev\_s, sb\_rev\_s = len(fb\_rev), len(sb\_rev)

min\_size = min(fb\_rev\_s, sb\_rev\_s)

**return** [1 **if** fb\_rev[i] == 1 **or** sb\_rev[i] == 1 **else** 0 **for** i **in** range(min\_size)] + (fb\_rev[min\_size:] **if** fb\_rev\_s > sb\_rev\_s **else** sb\_rev[min\_size:])

**def** **xor**(fb\_rev, sb\_rev):

fb\_rev\_s, sb\_rev\_s = len(fb\_rev), len(sb\_rev)

min\_size = min(fb\_rev\_s, sb\_rev\_s)

**return** [(fb\_rev[i] + sb\_rev[i]) % 2 **for** i **in** range(min\_size)] + (fb\_rev[min\_size:] **if** fb\_rev\_s > sb\_rev\_s **else** sb\_rev[min\_size:])

**def** **nfsr**(IV, seq\_len, path):

R1, R2, R3 = IV[:3]

reg\_R1, reg\_R2, reg\_R3, w = IV[3:]

R1, R2, R3 = list(map(bit\_array\_conversion, [R1, R2, R3]))

reg\_R1, reg\_R2, reg\_R3 = list(map(bit\_array\_conversion, [reg\_R1, reg\_R2, reg\_R3]))

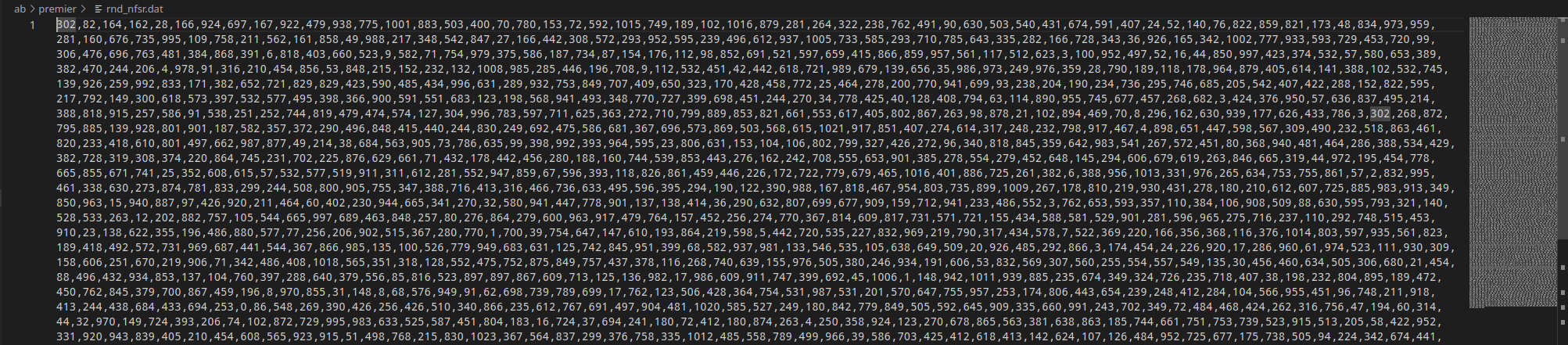
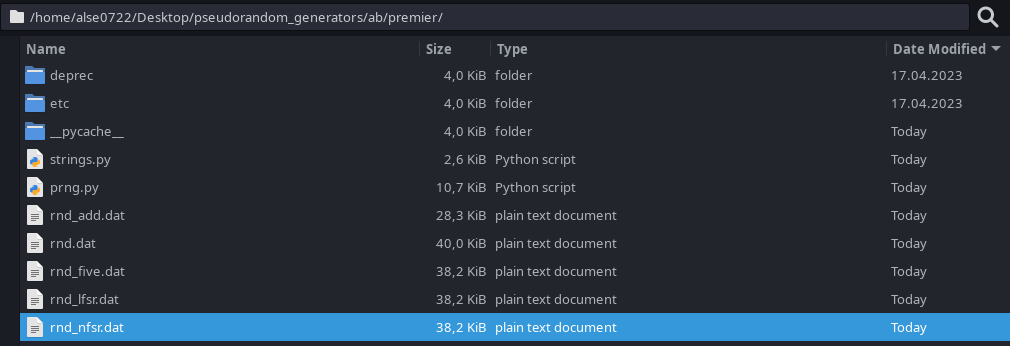
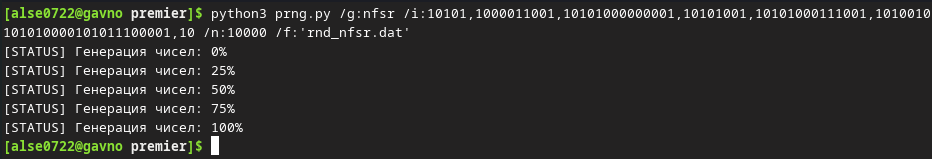
R = bitwise\_or( bitwise\_or( xor(R1, R2), xor(R2, R3) ), R3 )

reg\_R = bitwise\_or( bitwise\_or( xor(reg\_R1, reg\_R2), xor(reg\_R2, reg\_R3) ), reg\_R3 )

seq\_num = lfsr\_machinerie(seq\_len, 0, w, [i **for** i **in** range(len(R)) **if** R[i] == 1], reg\_R)

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 6. Вихрь Мерсенна.**

**Описание алгоритма.**

Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых w-битных чисел в соответствии с рекуррентной формулой:

где – целые константы;

– степень рекуррентности, ;

– *w*-битное двоичное целое число;

– двоичное целое число, полученное конкатенацией чисел и когда первые битов взяты из , а последние битов из в том же порядке;

– матрица размера состоящая из нулей и единиц, определенная

посредством .

– произведение, при вычислении которого сначала выполняют операцию (сдвига битов на одну позицию вправо), если последний бит равен 0, а затем, когда последний бит , то вычисляют .

**Параметры запуска программы**:

/g:mt /i:2048,113 /n:1000 /f:'rnd\_mt.dat'

**Код программы:**

За работу данного генератора отвечают функции:

**def** **extract\_number**(index, mt):

**if** index >= 624:

twist(mt)

index = 0

y = mt[index]

y ^= ((y >> 11) &amp; 0xffffffff)

y ^= ((y << 7) &amp; 0x9d2c5680)

y ^= ((y << 15) &amp; 0xefc60000)

y ^= (y >> 18)

index += 1

**return** index, y &amp; 0xffffffff

**def** **MT**(IV, seq\_len, path):

md, seed = IV

register\_len, seq\_num = 624, []

index, gen\_num = register\_len, 1812433253

mt[0] = seed

perc\_25, perc\_50, perc\_75 = seq\_len // 4, seq\_len // 2, seq\_len // 4 \* 3

**for** i **in** range(1, register\_len):

temp = gen\_num \* (mt[i - 1] ^ (mt[i - 1] >> 30)) + i

mt[i] = temp &amp; 0xffffffff

print('[STATUS] Генерация чисел: 0%')

**for** i **in** range(seq\_len):

index, y = extract\_number(index, mt)

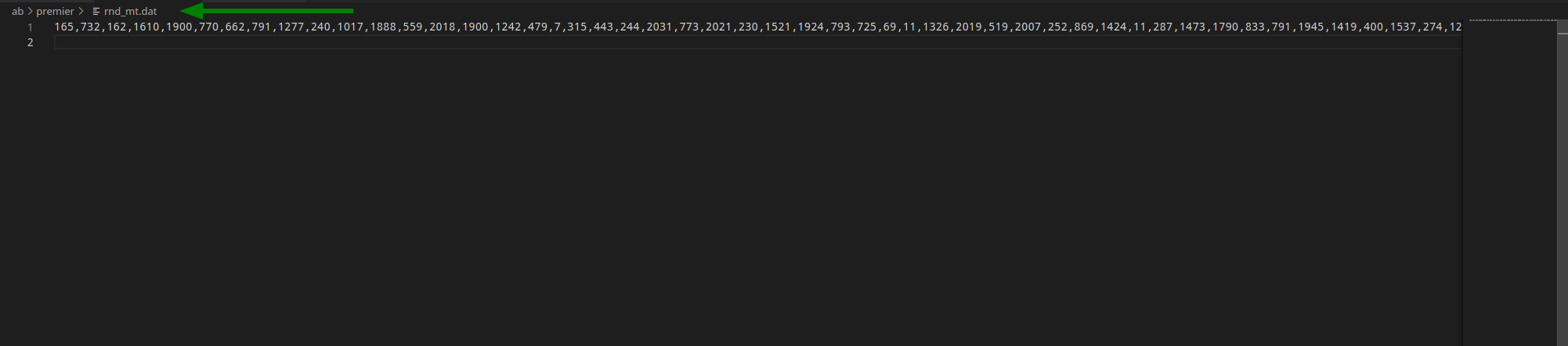
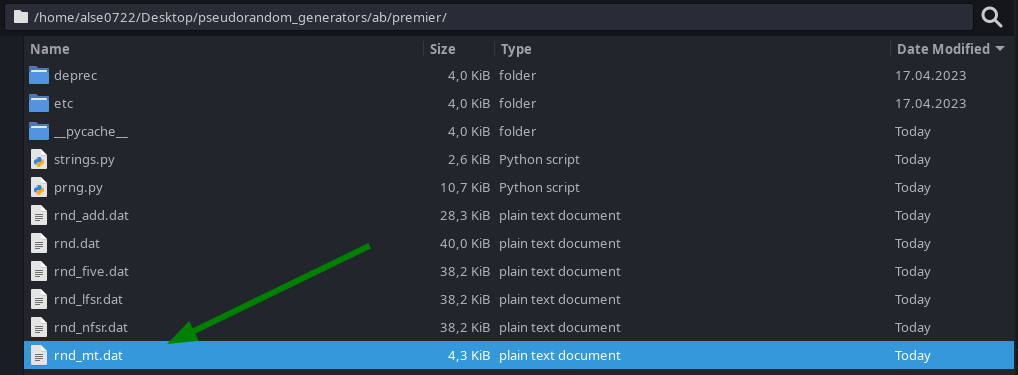
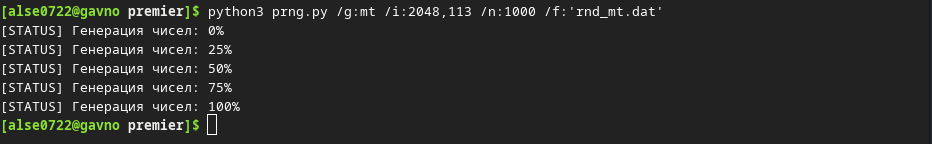
seq\_num.append(y % md)

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 7. RC4.**

**Описание алгоритма.**

1. Инициализация

a) 𝑓𝑜𝑟 ;

b) ;

c) 𝑓𝑜𝑟 𝑡𝑜 255:

2. .

3. Итерация алгоритма:

a);

b) ;

c) ;

d) ;

e) ;

Где K – это ключ.

**Параметры запуска программы**:

stream=$(python3 -c "import random; input=''.join([str(random.randint(1, 1024)) + ('' if i == 255 else ',') for i in range(256)]); print(input)")

/g:rc4 /i:$stream /n:1000 /f:'rnd\_rc4.dat'

**Код программы:**

За работу данного генератора отвечает функция:

**def** **rc4**(IV, seq\_len, path):

seq\_num = []

s\_block, key, j = [i **for** i **in** range(256)], IV, 0

perc\_25, perc\_50, perc\_75 = seq\_len // 4, seq\_len // 2, seq\_len // 4 \* 3

**for** i **in** range(256):

j = ( j + s\_block[i] + key[i] ) % 256

s\_block[i], s\_block[j] = s\_block[j], s\_block[i]

i, j = 0, 0

print('[STATUS] Генерация чисел: 0%')

**for** k **in** range(seq\_len):

i = ( i + 1 ) % 256

j = ( j + s\_block[i] ) % 256

s\_block[i], s\_block[j] = s\_block[j], s\_block[i]

t = ( s\_block[i] + s\_block[j] ) % 256

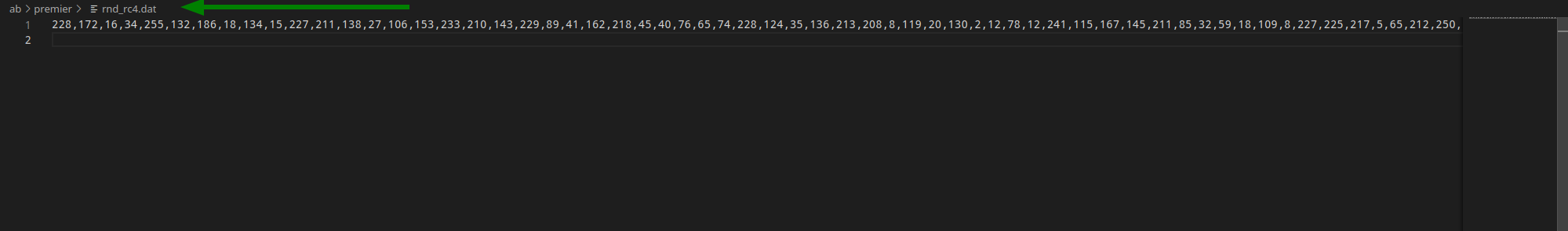
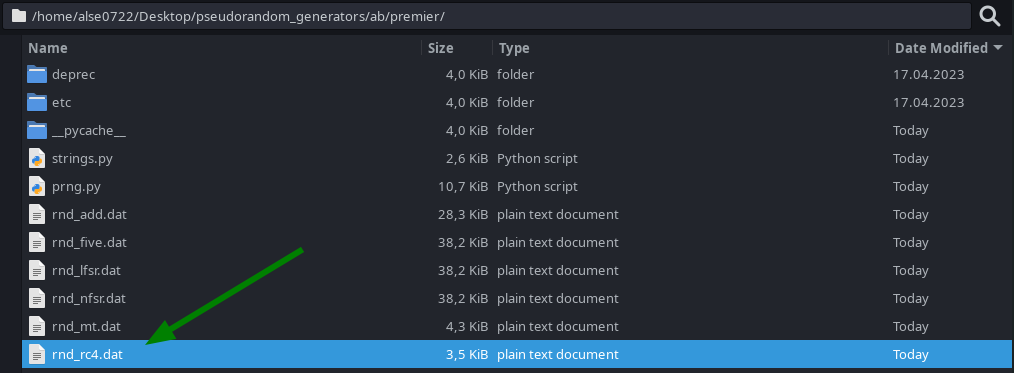
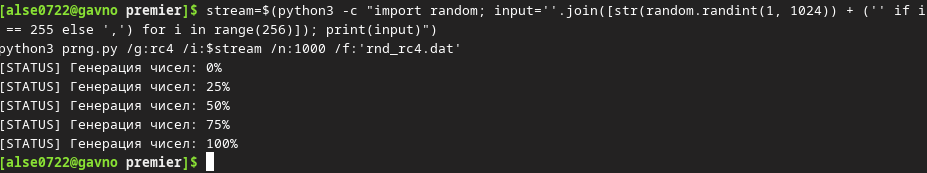
seq\_num.append(s\_block[t])

progress\_output(k, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 8.** **ГПСЧ на основе RSA.**

**Описание алгоритма.**

1. Считать два секретных простых числа и , а также и . Выбрать случайное целое число , такое, что НОД(𝑒,𝑓) = 1.

2. Выбрать случайное целое 𝑥0 –начальный вектор из интервала [1, 𝑛 − 1].

3. 𝐹𝑜𝑟 𝑡𝑜 𝑙 𝑑𝑜

a.

b. последний значащий бит

4. Вернуть 𝑧1, 𝑧2, …, 𝑧𝑙.

**Параметры запуска программы**:

/g:rsa /i:514081,99991,0,127,10 /n:10000 /f:'rnd\_rsa.dat'

**Код программы:**

**def** **rsa\_machinerie**(seq\_len, init\_x, \_pow, modulo, phi, w, is\_bbs=False):

seq\_num = list()

perc\_25, perc\_50, perc\_75 = seq\_len // 4, seq\_len // 2, seq\_len // 4 \* 3

print('[STATUS] Генерация чисел: 0%')

**for** i **in** range(seq\_len):

bit\_seq = list()

**for** j **in** range(w):

init\_x = pow(init\_x, \_pow, modulo)

bit\_seq.append(init\_x % 2)

bit\_seq.reverse()

init\_x = random.randint(2, modulo - 1)

**if** is\_bbs:

**while** math.gcd(init\_x, modulo) != 1:

init\_x = random.randint(2, modulo - 1)

inix\_x = (init\_x \* init\_x) % modulo

**if** **not** is\_bbs:

\_pow = random.randint(2, phi - 1)

**while** math.gcd(\_pow, phi) != 1:

\_pow = random.randint(2, phi - 1)

seq\_num.append(sum([0 **if** bit\_seq[k] == 0 **else** 2 \*\* k **for** k **in** range(w)]))

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

**return** seq\_num

**def** **rsa**(IV, seq\_len, path):

seq\_num = []

p, q, e, init\_x, w = IV

n, phi = p \* q, (p - 1) \* (q - 1)

**if** **not** (1 < e < phi) **or** math.gcd(e, phi) != 1:

print(strings.invalid\_e)

e = 0

**while** math.gcd(e, phi) != 1:

e = random.randint(2, phi - 1)

**if** **not** (1 < init\_x < n):

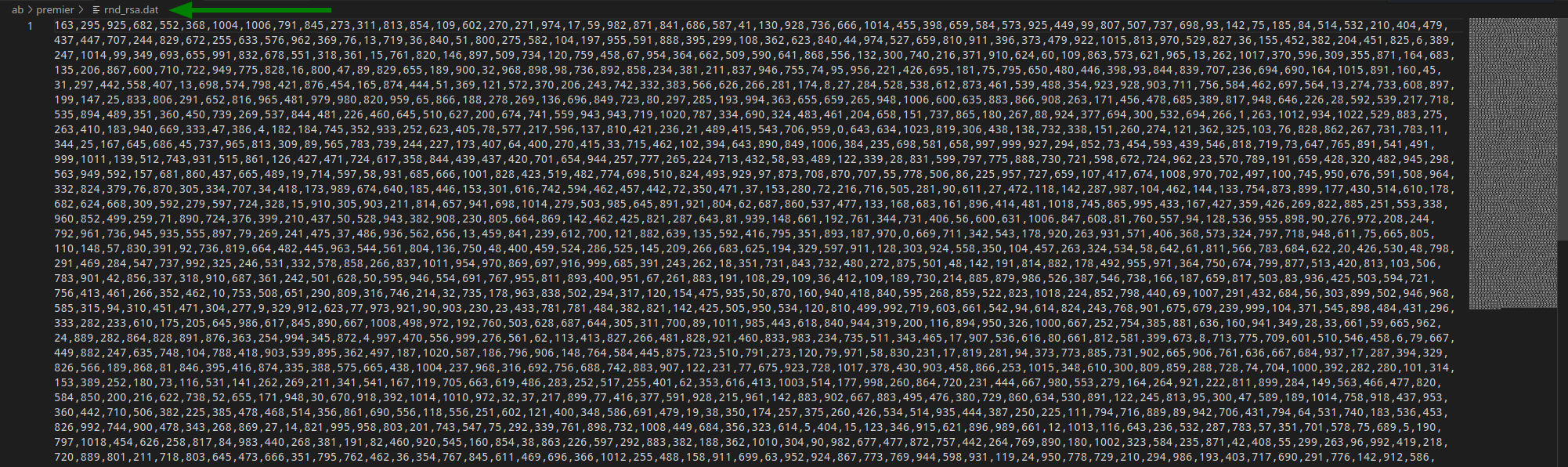
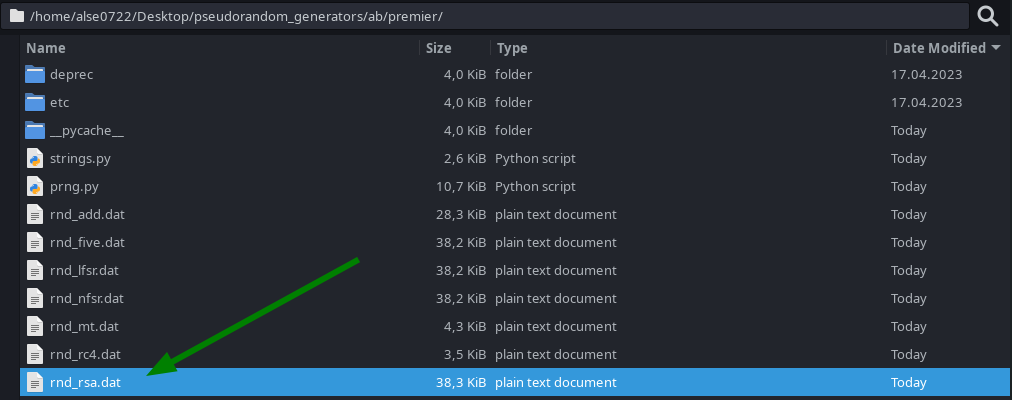
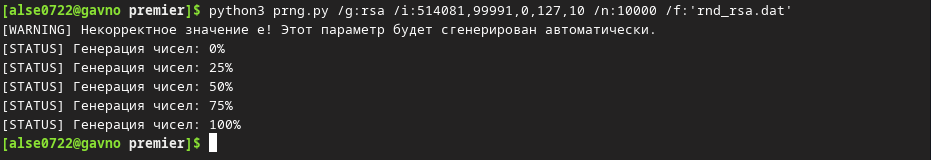
print(strings.invalid\_init\_x)

init\_x = random.randint(1, n - 1)

seq\_num = rsa\_machinerie(seq\_len, init\_x, e, n, phi, w)

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Алгоритм 9.** **Алгоритм Блюма-Блюма-Шуба.**

**Описание алгоритма.**

**На входе:** Длина *l*.

**На выходе:** Последовательность псевдослучайных бит .

1. Считать два простых числа и , сравнимых с 3 по модулю 4. Произведение этих чисел, *n*, является целым числом Блюма. Выберем другое случайное целое число *x*, взаимно простое с *n*.

2. Вычислим , которое будет начальным вектором.

3. 𝐹𝑜𝑟 𝑖=1 𝑡𝑜 𝑙 𝑑𝑜

1.

2. последний значащий бит 𝑥*i*

4. Вернуть .

**Параметры запуска программы**:

/g:bbs /i:113,10 /n:10000 /f:'bbs\_rsa.dat'

**Код программы:**

**def** **rsa\_machinerie**(seq\_len, init\_x, \_pow, modulo, phi, w, is\_bbs=False):

seq\_num = list()

perc\_25, perc\_50, perc\_75 = seq\_len // 4, seq\_len // 2, seq\_len // 4 \* 3

print('[STATUS] Генерация чисел: 0%')

**for** i **in** range(seq\_len):

bit\_seq = list()

**for** j **in** range(w):

init\_x = pow(init\_x, \_pow, modulo)

bit\_seq.append(init\_x % 2)

bit\_seq.reverse()

init\_x = random.randint(2, modulo - 1)

**if** is\_bbs:

**while** math.gcd(init\_x, modulo) != 1:

init\_x = random.randint(2, modulo - 1)

inix\_x = (init\_x \* init\_x) % modulo

**if** **not** is\_bbs:

\_pow = random.randint(2, phi - 1)

**while** math.gcd(\_pow, phi) != 1:

\_pow = random.randint(2, phi - 1)

seq\_num.append(sum([0 **if** bit\_seq[k] == 0 **else** 2 \*\* k **for** k **in** range(w)]))

progress\_output(i, perc\_25, perc\_50, perc\_75)

print('[STATUS] Генерация чисел: 100%')

**return** seq\_num

**def** **bbs**(IV, seq\_len, path):

p, q, n = 127, 131, 16637

phi = (p - 1) \* (q - 1)

x, w = IV

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

print(strings.invalid\_init\_x)

**while** math.gcd(x, n) != 1:

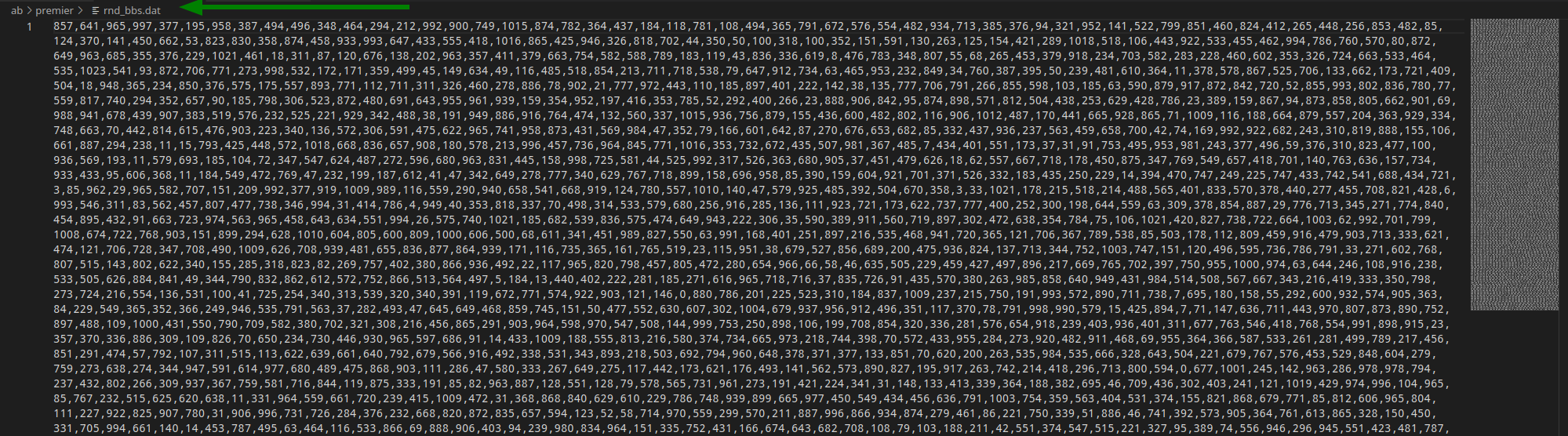
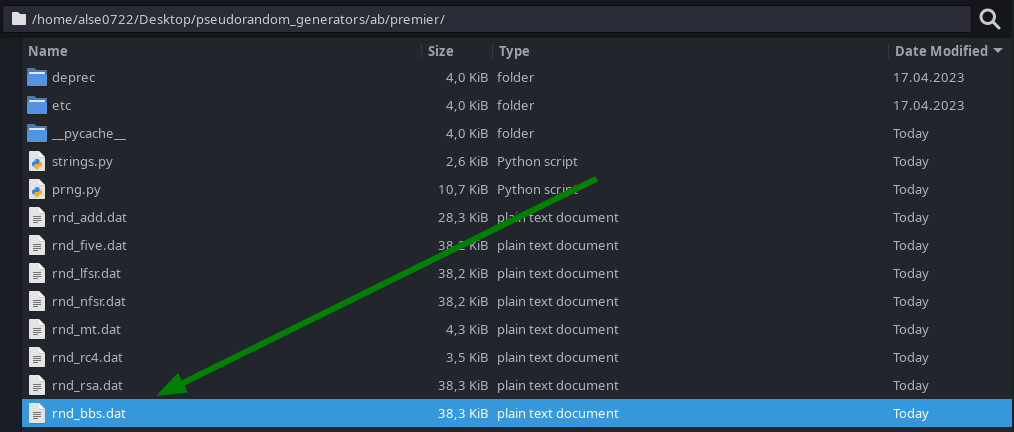
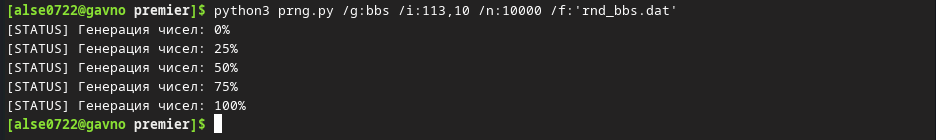
x = random.randint(2, n - 1)

x = (x \* x) % n

seq\_num = rsa\_machinerie(seq\_len, x, 2, n, phi, w, is\_bbs=True)

save\_as\_file(seq\_num, seq\_len, path)

**Пример работы программы:**



**Задание 2.** Преобразование ПСЧ к заданному распределению.

Описание задания: создать программу для преобразования последовательности ПСЧ в другую последовательность ПСЧ с заданным распределением:

1. Стандартное равномерное с заданным интервалом;
2. Треугольное распределение;
3. Общее экспоненциальное распределение;
4. Нормальное распределение;
5. Гамма распределение;
6. Логнормальное распределение;
7. Логистическое распределение;
8. Биномиальное распределение.

На входе: Текстовый файл с десятичными числами (разделитель – любой), интервал преобразуемых значений, параметры распределения.

Для управления приложением предлагается следующий формат параметров командной строки:

/f:<имя\_файла> - имя файла с входной последовательностью.

/d:<распределение> - код распределения для преобразования последовательности.

С помощью линейного конгруэнтного метода были сгенерированы 10000 чисел. Параметры метода: /i:7;106;1283;6075.

**Алгоритм 1. Стандартное равномерное распределение с заданным интервалом.**

**Описание алгоритма.**

Если стандартное равномерное случайное число 𝑈 получено методом, установленным в предыдущем параграфе, то равномерное случайное число должно быть получено в соответствии со следующей формулой:

𝑌 = 𝑏 \* 𝑈 + 𝑎.

**Параметры запуска программы**: /f:gen.txt /d:st /p:0 /q:1 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> St\_U(vector<int> R)

{

vector<double> res;

for (int i = 0; i < R.size(); i++)

{

double tmp;

tmp = R[i] / (double)m;

res.push\_back(tmp);

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

  vector<double> res;

res = St\_U(R);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

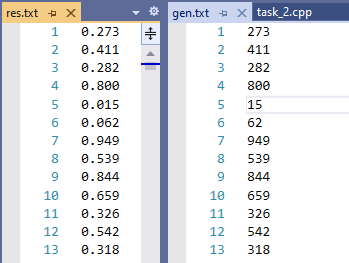
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 2. Треугольное распределение.**

**Описание алгоритма.**

Если стандартные случайные числа и независимо получены методом генерации стандартного равномерного числа, то случайное число Y, подчиняющееся треугольному распределению, определяют по формуле:

𝑌 = 𝑎 + 𝑏 \* ( + − 1).

**Параметры запуска программы** /f:gen.txt /d:tr /p:0 /q:1 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> tr(vector<int> R, double a, double b)

{

vector<double> res;

for (int i = 0; i < R.size() - 1; i++)

{

double tmp = a + b \* ((R[i] / (double)m) + (R[i + 1] / (double)m) - 1);

res.push\_back(tmp);

i++;

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

  vector<double> res;

double a = config.p;

double b = config.q;

res = tr(R, a, b);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

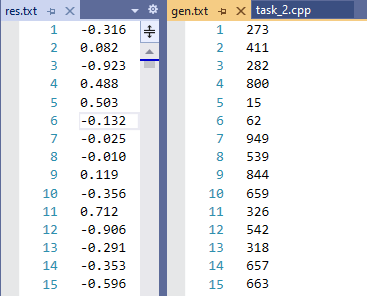
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 3. Общее экспоненциальное распределение.**

**Описание алгоритма.**

Случайное число, соответствующее экспоненциальному распределению, получают по формуле:

.

**Параметры запуска программы** /f:gen.txt /d:ex /p:0 /q:1 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> ex(vector<int> R, double a, double b)

{

vector<double> res;

for (int i = 0; i < R.size(); i++)

{

double tmp = a - b \* log(R[i] / (double)m);

res.push\_back(tmp);

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

vector<double> res;

double a = config.p;

double b = config.q;

res = ex(R, a, b);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

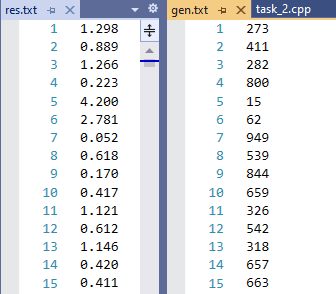
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 4. Нормальное распределение.**

**Описание алгоритма.**

Два независимых нормальных случайных числа определяют в соответствии со следующими формулами:

**Параметры запуска программы** /f:gen.txt /d:nr /p:0 /q:1 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> nr(vector<int> R, double a, double b)

{

vector<double> res;

for (int i = 0; i < R.size() - 1; i++)

{

double y1, y2;

y1 = a + b \* sqrt(-2 \* log(1 - (R[i] / (double)m))) \* cos(2 \* PI \* (R[i + 1] / (double)m));

y2 = a + b \* sqrt(-2 \* log(1 - (R[i] / (double)m))) \* sin(2 \* PI \* (R[i + 1] / (double)m));

res.push\_back(y1);

res.push\_back(y2);

i++;

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

 vector<double> res;

double a = config.p;

double b = config.q;

res = nr(R, a, b);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

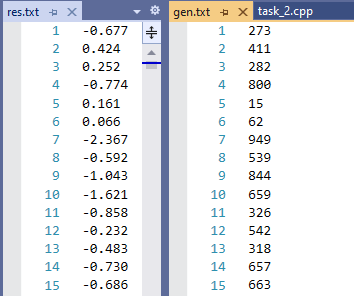
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 5. Гамма распределение.**

**Описание алгоритма.**

Случайное число , подчиняющееся гамма распределению, определяют по формуле:

.

**Параметры запуска программы** /f:gen.txt /d:gm /p:0 /q:1 /c:3 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> nr(vector<int> R, double a, double b)

{

vector<double> res;

for (int i = 0; i < R.size() - 1; i++)

{

double y1, y2;

y1 = a + b \* sqrt(-2 \* log(1 - (R[i] / (double)m))) \* cos(2 \* PI \* (R[i + 1] / (double)m));

y2 = a + b \* sqrt(-2 \* log(1 - (R[i] / (double)m))) \* sin(2 \* PI \* (R[i + 1] / (double)m));

res.push\_back(y1);

res.push\_back(y2);

i++;

}

return res;

}

double loggm(vector<int> R, double c)

{

double res;

double tmp = 1;

int int\_c = (int)c;

for (int i = 0 + gm\_cnt; i < int\_c + gm\_cnt - 1; i++)

{

double u = R[i] / (double)m;

u = 1 - u;

tmp = tmp \* u;

}

gm\_cnt = gm\_cnt + int\_c;

res = log(tmp);

return res;

}

vector<double> gm(vector<int> R, double a, double b, double c)

{

vector<double> res;

if (floor(c) == c)

{

while (gm\_cnt < R.size())

{

double y;

y = a - b \* loggm(R, c);

res.push\_back(y);

}

}

else

{

int k = floor(c);

double ln\_y1;

double ln\_y2;

if (k == 1)

{

ln\_y1 = loggm({ R[3], R[3] }, 2);

gm\_cnt = 0;

}

else

{

vector<int> R\_gm;

for (int j1 = 3; j1 < k + 2; j1++)

{

R\_gm.push\_back(R[j1]);

}

ln\_y1 = loggm(R\_gm, R\_gm.size());

gm\_cnt = 0;

}

ln\_y2 = loggm({ R[k + 3], R[k + 2] }, 2);

gm\_cnt = 0;

while (gm\_cnt + 3 < R.size())

{

double y1;

double y2;

double z1;

double z2;

vector<double> z;

z = nr({ R[gm\_cnt], R[gm\_cnt + 1] }, 0, 1);

z1 = z[0];

z2 = z[1];

y1 = a + b \* ((pow(z1, 2) / 2) - ln\_y1);

y2 = a + b \* ((pow(z2, 2) / 2) - ln\_y2);

res.push\_back(y1);

res.push\_back(y2);

gm\_cnt = gm\_cnt + 2;

}

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

 vector<double> res;

double a = config.p;

double b = config.q;

double c = config.c;

res = gm(R, a, b, c);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

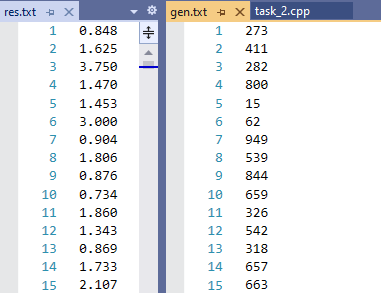
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 6. Логнормальное распределение.**

**Описание алгоритма.**

Случайное число , подчиняющееся логнормальному распределению, определяют по формуле:

.

**Параметры запуска программы** /f:gen.txt /d:ln /p:0 /q:1 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> ln(vector<int> R, double a, double b)

{

vector<double> res;

for (int i = 0; i < R.size() - 1; i++)

{

double z1, z2, y1, y2;

z1 = sqrt(-2 \* log(1 - (R[i] / (double)m))) \* cos(2 \* PI \* (R[i + 1] / (double)m));

z2 = sqrt(-2 \* log(1 - (R[i] / (double)m))) \* sin(2 \* PI \* (R[i + 1] / (double)m));

y1 = a + exp(b - z1);

y2 = a + exp(b - z2);

res.push\_back(y1);

res.push\_back(y2);

i++;

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

 vector<double> res;

double a = config.p;

double b = config.q;

res = ln(R, a, b);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

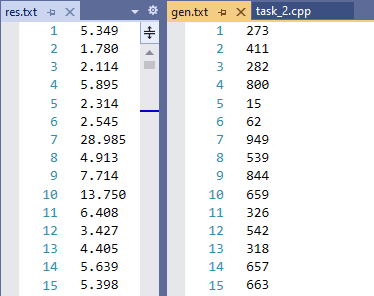
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 7. Логистическое распределение.**

**Описание алгоритма.**

Случайное число , подчиняющееся логистическому распределению, определяют по формуле:

**Параметры запуска программы** /f:gen.txt /d:ls /p:0 /q:1 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

vector<double> ls(vector<int> R, double a, double b)

{

vector<double> res;

for (int i = 0; i < R.size(); i++)

{

double u, y;

u = R[i] / (double)m;

y = a + b \* log(u / (1 - u));

res.push\_back(y);

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

  vector<double> res;

double a = config.p;

double b = config.q;

res = ls(R, a, b);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

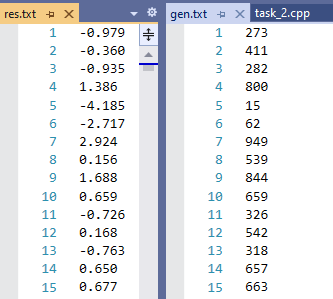
}

out\_file.close();

return 0;

}

**Пример работы программы:**



**Алгоритм 8. Биномиальное распределение.**

**Описание алгоритма.**

Преобразование к биномиальному распределению осуществим методом обратной функции. Функция распределения получается следующим образом: . Случайное число является наименьшим значением , для которого .

**Параметры запуска программы** /f:gen.txt /d:bi /p:0.5 /q:5 /r:res.txt

**Код программы:**

#include <iostream>

#include <string>

#include <vector>

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

vector<string> b\_str;

int m = 1000;

int gm\_cnt = 0;

const double PI = 3.141592653589793;

typedef struct config\_t {

string f;

string d;

double p;

double q;

double c;

string r;

} config\_t;

void parse\_params(config\_t\* config, int argc, char\* argv[])

{

int opt;

for (int j = 1; j < argc; j++)

{

string a = argv[j];

switch (a[1])

{

case 'f':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->f = b;

b.clear();

break;

}

case 'd':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->d = b;

b.clear();

break;

}

case 'p':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->p = u;

b.clear();

break;

}

case 'q':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->q = u;

b.clear();

break;

}

case 'c':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

double u = stod(b);

config->c = u;

b.clear();

break;

}

case 'r':

{

string b;

for (int k = 3; k < a.size(); k++)

{

b.push\_back(a[k]);

}

config->r = b;

b.clear();

break;

}

}

}

}

int factorial(int a)

{

if (a == 0) return 1;

else return (a \* factorial(a - 1));

}

double C(int k, int b)

{

double res;

int int\_b = (int)b;

int a1 = factorial(int\_b);

int a2 = factorial(k);

int a3 = factorial(int\_b - k);

a2 = a2 \* a3;

res = a1 / a2;

return res;

}

vector<int> bi(vector<int> R, double a, double b)

{

vector<int> res;

for (int i = 0; i < R.size(); i++)

{

double u;

int y;

double s = 0;

int k = 0;

bool flag = false;

int int\_b = (int)b;

u = R[i] / (double)m;

while (!flag)

{

s = s + C(k, b) \* pow(a, k) \* pow((1 - a), b - k);

if (s > u)

{

y = k;

break;

}

if (k < b - 1)

{

k = k + 1;

continue;

}

y = b;

break;

}

res.push\_back(y);

}

return res;

}

int main(int argc, char\* argv[])

{

setlocale(LC\_ALL, "Russian");

vector<int> R;

config\_t config{ "gen.txt", "st" , 0, 1, 0, "res.txt"};

parse\_params(&config, argc, argv);

ifstream in\_file(config.f);

if (in\_file.is\_open())

{

while (!in\_file.eof())

{

int tmp;

in\_file >> tmp;

R.push\_back(tmp);

}

}

in\_file.close();

ofstream out\_file(config.r);

vector<int> res;

double a = config.p;

double b = config.q;

res = bi(R, a, b);

if (out\_file.is\_open())

{

for (int i = 0; i < res.size(); i++)

{

out\_file << fixed;

out\_file.precision(3);

out\_file << res[i] << endl;

}

}

out\_file.close();

return 0;

}

**Пример работы программы:**

