

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 2023

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;
- $0 \leq a \leq m$, множитель;
- $0 \leq c \leq m$, приращение (инкремент);
- $0 \leq x_0 \leq m$, начальное значение.

Последовательность ПСЧ, получаемая по формуле:

$$x_{i+1} = ax_i + c \pmod{m}$$

называется линейной конгруэнтной последовательностью (ЛКП).

Ключом для неё служит x_0 .

Параметры запуска программы:

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

Код программы:

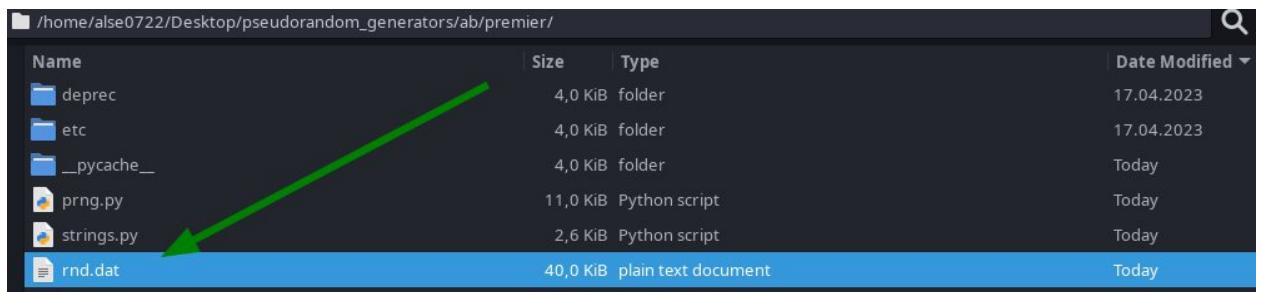
За работу данного генератора отвечает функция:

```
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:

```
[else0722@gavno premier]$ python3 prng.py /g:lc /i:1223,7,11,3 /n:10000 /f:rnd.dat
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[else0722@gavno premier]$
```



Алгоритм 2. Аддитивный метод.

Описание алгоритма.

Формула:

$$X_{n+1} = (X_{n-k} + X_{n-j}) \bmod m, n > j, j > k \geq 1, m > 0.$$

Параметры, поступающие на вход: модуль m, коэффициент k, коэффициент j, числа X_0, \dots, X_n .

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:

```
[alse0722@gavno premier]$ python3 ping.py /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 /f:'rnd_add.dat'
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$ █
```

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
__pycache__	4,0 KiB	folder	Today
prng.py	11,0 KiB	Python script	Today
strings.py	2,6 KiB	Python script	Today
rnd.dat	40,0 KiB	plain text document	Today
rnd_add.dat	28,3 KiB	plain text document	Today

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:

$$X_{n+p} = X_{n+q_1} + X_{n+q_2} + X_{n+q_3} + X_n, \quad n = 1, 2, 3, \dots$$

При этом $p > w > q_1 > q_2 > q_3 > 0, w > 0$.

Параметры (p, q_1, q_2, q_3, w) и X_1, \dots, X_p , 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 Ifsr_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:]
            binaryConverted = sum([0 if current_binary[i] == 0 else 2 ** i for i in range(w)])
            seq_num.append(binaryConverted)

    progress_output(i, perc_25, perc_50, perc_75)

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

```
def five_p(IV, seq_len, path):
```

```
seq_num = lfsr_machinerie(seq_len, p, w, nonzero_coeffs)
save_as_file(seq_num, seq_len, path)
```

Пример работы программы:

```
[alse0722@gavno premier]$ python3 prng.py /g:5p /i:89,20,40,69,10 /n:10000 /f:rnd_five.dat  
[STATUS] Генерация чисел: 0%  
[STATUS] Генерация чисел: 25%  
[STATUS] Генерация чисел: 50%  
[STATUS] Генерация чисел: 75%  
[STATUS] Генерация чисел: 100%  
[alse0722@gavno premier]$
```

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
__pycache__	4,0 KiB	folder	Today
strings.py	2,6 KiB	Python script	Today
prng.py	10,7 KiB	Python script	Today
rnd_add.dat	28,3 KiB	plain text document	Today
rnd.dat	40,0 KiB	plain text document	Today
rnd_five.dat	38,2 KiB	plain text document	Today

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:

$$x_{i+1+p} = a_1 x_{i+1} \oplus \dots \oplus a_p x_{i+p}, \quad \text{исходно заданы значения: } x_1, \dots, x_p, a_1, \dots, a_p \in \{0, 1\}.$$

Функция:

$$Y_{b:s} = 2^{s-1} x_b + 2^{s-2} x_{b+1} + \dots + 2 x_{b+s-2} + x_{b+s-1}$$

На вход подаются параметры: $p, s, m, j_1 \dots j_m, Y_{1:p}$.

При этом $Y_{1:p}, 1 \leq j_1 < \dots < j_m \leq p, m \leq p$.

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:]
        binaryConverted = sum([0 if current_binary[i] == 0 else 2 ** i for i in range(w)])
        seq_num.append(binaryConverted)

    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:

```
[alse0722@gavno premier]$ python3 prng.py /g:lfsr /i:1000010001,1010101110100011100010101010,10 /n:10000 /f:'rnd_lfsr.dat'
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$
```

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
__pycache__	4,0 KiB	folder	Today
strings.py	2,6 KiB	Python script	Today
prng.py	10,7 KiB	Python script	Today
rnd_add.dat	28,3 KiB	plain text document	Today
rnd.dat	40,0 KiB	plain text document	Today
rnd_five.dat	38,2 KiB	plain text document	Today
rnd_ifsr.dat	38,2 KiB	plain text document	Today

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: $x_{1,1} \dots x_{i,1} \dots$

РСЛОС 2 выдаёт последовательность бит: $x_{1,2} \dots x_{i,2} \dots$

.....

РСЛОС k выдаёт последовательность бит: $x_{1,k} \dots x_{i,k} \dots$

Формируется последовательность бит: $x_i = \oplus_{i_1, \dots, i_k \in \{0, 1\}} a_{i_1, \dots, i_k} x_{i,1}^{i_1} \dots x_{i,k}^{i_k}$

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

$$Y_t = 2^{w-1}x_{wt+1} + 2^{w-2}x_{wt+2} + \dots + 2x_{wt+w-1} + x_{wt+w}$$

На вход подаются параметры: k, w, p_1 , $X_{1,0}$, m_1 , $j_{1,1} \dots j_{1,m_1}$, ..., p_k , $X_{k,0}$, m_k , $j_{k,1} \dots j_{k,m_k}$, q , $j_1 \dots j_q$.

При этом $0 < j_{i,z} \leq p$, где $1 \leq i \leq k$, $1 \leq z \leq m_k$

$$|j_i| = k, \text{ где } 0 < i < q.$$

Параметры запуска программы:

```
/g:nfsr/i:10101,1000011001,1010100000001,10101001,10101000111001,101001  
0101010000101011100001,10 /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:

```

[alse0722@gavno premier]$ python3 prng.py /g:nfsr /i:10101,1000011001,10101000000001,10101001,10101000111001,1010010
10101000010101100001,10 /n:10000 /f:'rnd_nfsr.dat'
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$ █

```

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
__pycache__	4,0 KiB	folder	Today
strings.py	2,6 KiB	Python script	Today
prng.py	10,7 KiB	Python script	Today
rnd_add.dat	28,3 KiB	plain text document	Today
rnd.dat	40,0 KiB	plain text document	Today
rnd_five.dat	38,2 KiB	plain text document	Today
rnd_lfsr.dat	38,2 KiB	plain text document	Today
rnd_nfsr.dat	38,2 KiB	plain text document	Today

Алгоритм 6. Вихрь Мерсенна.

Описание алгоритма.

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

$$x_{n+p} = x_{n+q} \oplus (X_n^r | X_{n+1}^l) A \quad (n = 0, 1, 2, \dots),$$

где p, q, r – целые константы;

p – степень рекуррентности, $1 \leq q \leq p$;

X_n – w -битное двоичное целое число;

$(X_n^r | X_{n+1}^l)$ – двоичное целое число, получено по конкатенации чисел X_n^r и X_{n+1}^l

когда первые $(w - r)$ битов взяты из X_n , а последние r битов из X_{n+1} в том же порядке;

A – матрица размера $w \times w$ состоящая из нулей и единиц, определенная посредством a .

XA – произведение, при вычислении которого сначала выполняют операцию $X >> 1$ (сдвига битов на одну позицию вправо), если последний бит X равен 0, а затем, когда последний бит $X = 1$, то вычисляют $XA = (X >> 1) \oplus a$.

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) & 0xffffffff)
y ^= ((y << 7) & 0x9d2c5680)
y ^= ((y << 15) & 0xefc60000)
y ^= (y >> 18)

index += 1

return index, y & 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 & 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:

```

[alse0722@gavno premier]$ python3 prng.py /g:mt /i:2048,113 /n:1000 /f:'rnd_mt.dat'
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$ 

```

File list for /home/alse0722/Desktop/pseudorandom_generators/ab/premier/

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
pycache	4,0 KiB	folder	Today
strings.py	2,6 KiB	Python script	Today
prng.py	10,7 KiB	Python script	Today
rnd_add.dat	28,3 KiB	plain text document	Today
rnd.dat	40,0 KiB	plain text document	Today
rnd_five.dat	38,2 KiB	plain text document	Today
rnd_lfsr.dat	38,2 KiB	plain text document	Today
rnd_nfsr.dat	38,2 KiB	plain text document	Today
rnd_mt.dat	4,3 KiB	plain text document	Today

```
ab > premier > E rnd_mt.dat ←
1 165,732,162,1610,1900,770,662,791,1277,240,1017,1888,559,2018,1900,1242,479,7,315,443,244,2031,773,2021,230,1521,1924,793,725,69,11,1326,2019,519,2007,252,869,1424,11,287,1473,1790,833,791,1945,1419,400,1537,274,12
```

Алгоритм 7. RC4.

Описание алгоритма.

1. Инициализация $S_i, i = 0, 1, \dots, 255$.

a) $for i = 0 \text{ to } 255 : S_i = i;$

b) $j = 0;$

c) $for i = 0 \text{ to } 255: j = (j + S_i + K_i) \bmod 256; Swap(S_i, S_j)$

2. $i = 0, j = 0.$

3. Итерация алгоритма:

a) $i = (i + 1) \bmod 256;$

b) $j = (j + S_i) \bmod 256;$

c) $Swap(S_i, S_j);$

d) $t = (S_i + S_j) \bmod 256;$

e) $K = S_t;$

Где К – 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:

```

[alse0722@gavno premier]$ stream=$(python3 -c "import random; input=''.join([str(random.randint(1, 1024)) + ('' if i == 255 else ',') for i in range(256)])"; print(input)")
python3 prng.py /g:rc4 /i:$stream /n:1000 /f:'rnd_rc4.dat'
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$

```

Screenshot of a file explorer showing the contents of the 'ab/premier/' directory. A green arrow points from the terminal output above to the 'rnd_rc4.dat' file in the file list.

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
__pycache__	4,0 KiB	folder	Today
strings.py	2,6 KiB	Python script	Today
prng.py	10,7 KiB	Python script	Today
rnd_add.dat	28,3 KiB	plain text document	Today
rnd_five.dat	38,2 KiB	plain text document	Today
rnd_lfsr.dat	38,2 KiB	plain text document	Today
rnd_nfsr.dat	38,2 KiB	plain text document	Today
rnd_mt.dat	4,3 KiB	plain text document	Today
rnd_rc4.dat	3,5 KiB	plain text document	Today

Screenshot of a terminal window showing the content of the 'rnd_rc4.dat' file. The file contains a sequence of integers separated by commas.

```

ab > premier > E rnd_rc4.dat
1 228,172,16,34,255,132,186,18,134,15,227,211,138,27,106,153,233,210,143,229,89,41,162,218,45,40,76,65,74,228,124,35,136,213,208,8,119,20,130,2,12,78,12,241,115,167,145,211,85,32,59,18,109,8,227,225,217,5,65,212,250,
2

```

Алгоритм 8. ГПСЧ на основе RSA.

Описание алгоритма.

1. Считать два секретных простых числа p и q , а также $n = pq$ и $f = (p - 1)(q - 1)$. Выбрать случайное целое число e , $1 < e < f$, такое, что $\text{НОД}(e,f) = 1$.
2. Выбрать случайное целое x_0 – начальный вектор из интервала $[1, n - 1]$.
3. For $i = 1$ to l do
 - a. $x_i \leftarrow x_{i-1}^e \bmod n$
 - b. $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть z_1, z_2, \dots, z_l .

Параметры запуска программы:

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

Код программы:

За работу данного генератора отвечают функции:

```
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)
            init_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:

```

[alse0722@gavno premier]$ python3 prng.py /i:514081,99991,0,127,10 /n:10000 /f:'rnd_rsa.dat'
[WARNING] Некорректное значение e! Этот параметр будет сгенерирован автоматически.
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$

```

Name	Size	Type	Date Modified
deprec	4,0 KiB	folder	17.04.2023
etc	4,0 KiB	folder	17.04.2023
__pycache__	4,0 KiB	folder	Today
strings.py	2,6 KiB	Python script	Today
prng.py	10,7 KiB	Python script	Today
rnd_add.dat	28,3 KiB	plain text document	Today
rnd_five.dat	38,2 KiB	plain text document	Today
rnd_lfsr.dat	38,2 KiB	plain text document	Today
rnd_nfsr.dat	38,2 KiB	plain text document	Today
rnd_mt.dat	4,3 KiB	plain text document	Today
rnd_rc4.dat	3,5 KiB	plain text document	Today
rnd_rsa.dat	38,3 KiB	plain text document	Today

Алгоритм 9. Алгоритм Блюма-Блюма-Шуба.

Описание алгоритма.

На входе: Длина l .

На выходе: Последовательность псевдослучайных бит z_1, z_2, \dots, z_l .

- Считать два простых числа p и q , сравнимых с 3 по модулю 4.

Произведение этих чисел, n , является целым числом Блюма. Выберем другое случайное целое число x , взаимно простое с n .

- Вычислим $x_0 = x^2 \bmod n$, которое будет начальным вектором.

3. For $i=1$ to l do

- $x_i \leftarrow x_{i-1}^e \bmod n$

- $z_i \leftarrow$ последний значащий бит x_i

- Вернуть z_1, z_2, \dots, z_l .

Параметры запуска программы:

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

Код программы:

За работу данного генератора отвечают функции:

```
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)
                init_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)

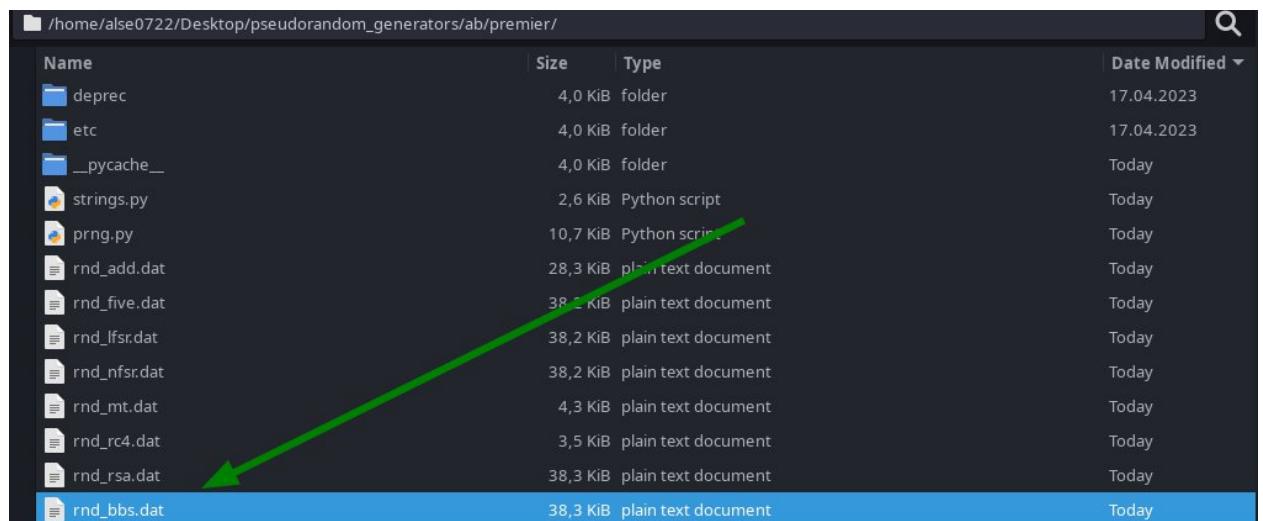

```

Пример работы программы:

За работу данного генератора отвечают функции:

```

[alse0722@gavno premier]$ python3 prng.py /g:bbs /i:113,10 /n:10000 /f:'rnd_bbs.dat'
[STATUS] Генерация чисел: 0%
[STATUS] Генерация чисел: 25%
[STATUS] Генерация чисел: 50%
[STATUS] Генерация чисел: 75%
[STATUS] Генерация чисел: 100%
[alse0722@gavno premier]$
```



Name	Type	Date Modified
deprec	folder	17.04.2023
etc	folder	17.04.2023
__pycache__	folder	Today
strings.py	Python script	Today
prng.py	Python script	Today
rnd_add.dat	plain text document	Today
rnd_five.dat	plain text document	Today
rnd_lfsr.dat	plain text document	Today
rnd_nfsr.dat	plain text document	Today
rnd_mt.dat	plain text document	Today
rnd_rc4.dat	plain text document	Today
rnd_rsa.dat	plain text document	Today
rnd_bbs.dat	plain text document	Today

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

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:

$$Y = b * U + a.$$

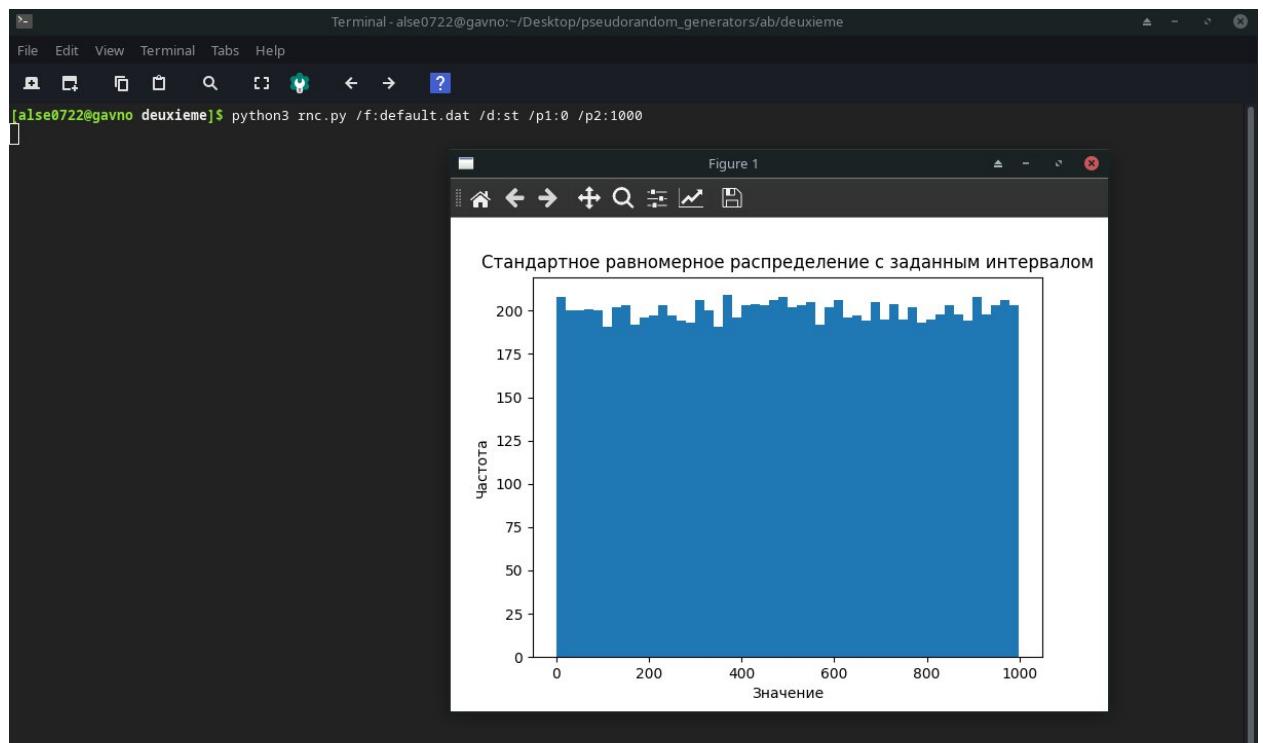
Параметры запуска программы:

/f:default.dat /d:st /p1:0 /p2:1000

Код программы:

```
def st(num_seq, a, b):  
  
    divisor = max(num_seq) + 1  
    num_seq = list(map(lambda num : int(num / divisor * b + a), num_seq))  
  
    save_to_file(num_seq, 'st.dat')  
    visualize(num_seq, 'Стандартное равномерное распределение с заданным интервалом')
```

Пример работы программы:



Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today

Алгоритм 2. Треугольное распределение.

Описание алгоритма.

Если стандартные случайные числа U_1 и U_2 независимо получены методом генерации стандартного равномерного числа, то случайное число Y , подчиняющееся треугольному распределению, определяют по формуле:

$$Y = a + b * (U_1 + U_2 - 1).$$

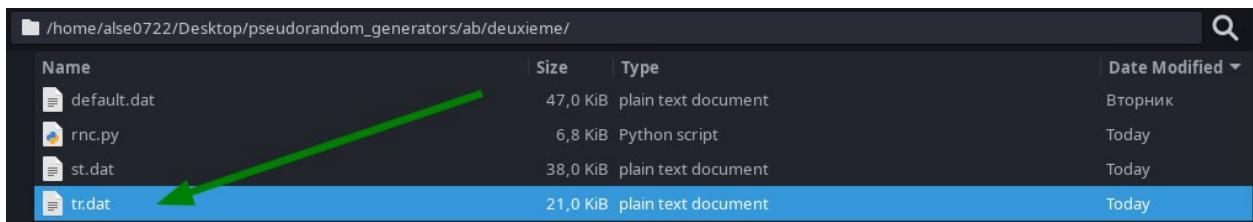
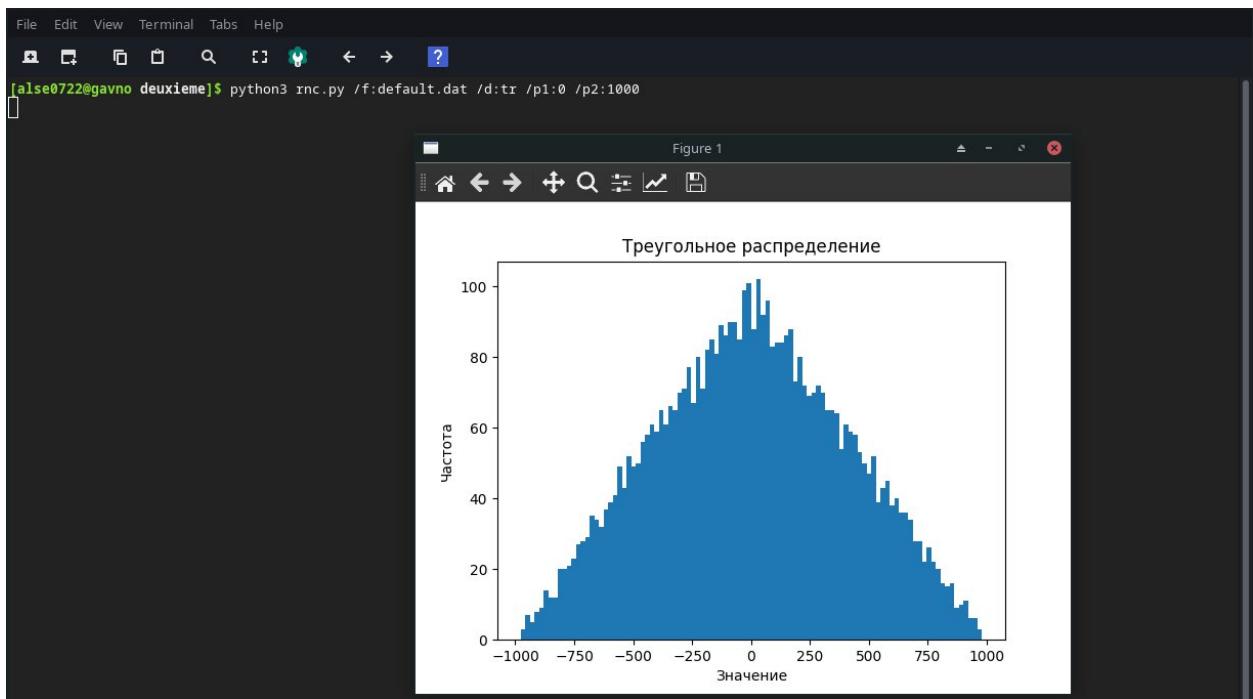
Параметры запуска программы

/f:default.dat /d:tr /p1:0 /p2:1000

Код программы:

```
def tr(num_seq, a, b):  
  
    divisor = max(num_seq) + 1  
    num_seq, u_f, u_s = list(map(lambda num : num / divisor, num_seq)), list(), list()  
  
    for i in range(len(num_seq)):  
        if i % 2 == 0:  
            u_f.append(num_seq[i])  
        else: u_s.append(num_seq[i])  
  
    num_seq = list(map(lambda f, s : int(a + b * (f + s - 1)), u_f, u_s))  
  
    save_to_file(num_seq, 'tr.dat')  
    visualize(num_seq, 'Треугольное распределение', 100)
```

Пример работы программы:



Алгоритм 3. Общее экспоненциальное распределение.

Описание алгоритма.

Случайное число, соответствующее экспоненциальному распределению, получают по формуле:

$$Y = -b * \ln(U) + a.$$

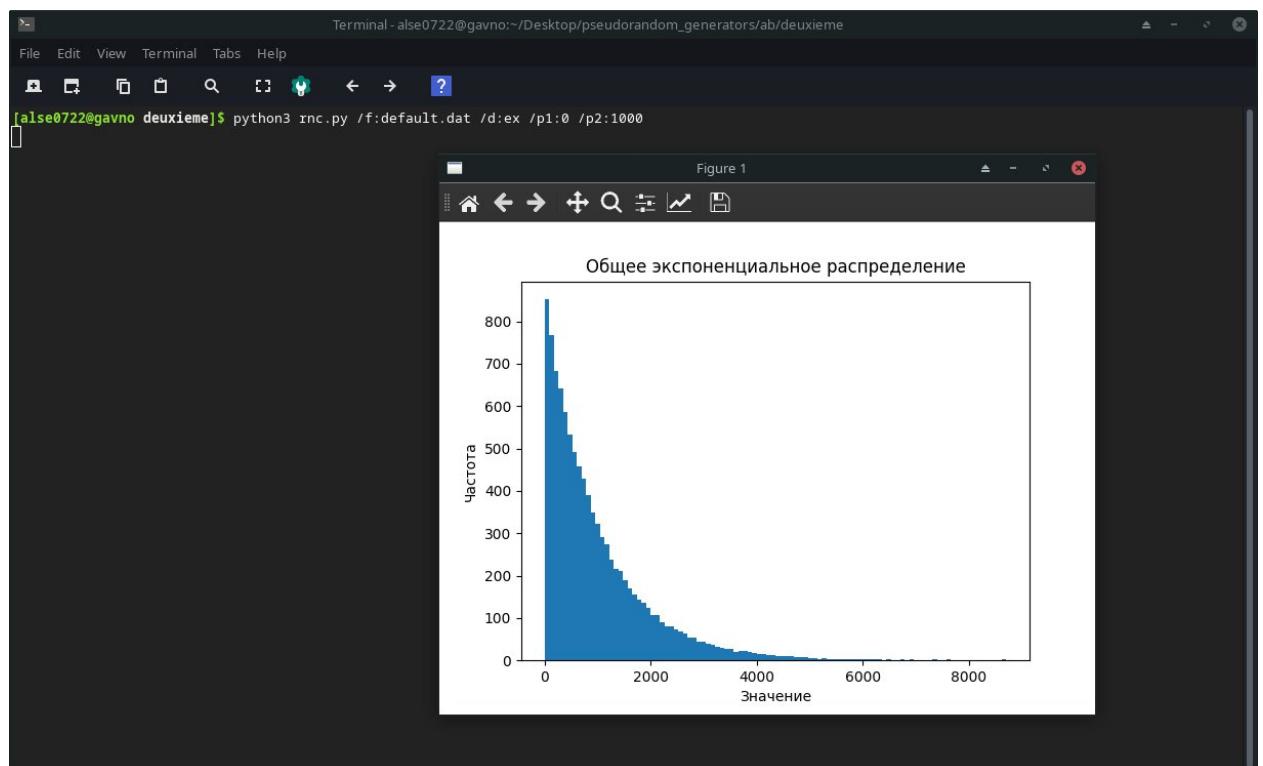
Параметры запуска программы

/f:default.dat /d:ex /p1:0 /p2:1000

Код программы:

```
def ex(num_seq, a, b):  
  
    divisor = max(num_seq) + 1  
    num_seq = [num_seq[i] for i in range(len(num_seq)) if num_seq[i] != 0]  
    num_seq = list(map(lambda num : int(-b * math.log(num) + a), list(map(lambda num : num / divisor,  
    num_seq))))  
  
    save_to_file(num_seq, 'ex.dat')  
    visualize(num_seq, 'Общее экспоненциальное распределение', 100)
```

Пример работы программы:



Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today
tr.dat	21,0 KiB	plain text document	Today
ex.dat	41,6 KiB	plain text document	Today

Алгоритм 4. Нормальное распределение.

Описание алгоритма.

Два независимых нормальных случайных числа Z_1 и Z_2 определяют в соответствии со следующими формулами:

$$Z_1 = \mu + \sigma \sqrt{-2\ln(1 - U_1)} \cos(2\pi U_2),$$

$$Z_2 = \mu + \sigma \sqrt{-2\ln(1 - U_1)} \sin(2\pi U_2).$$

Параметры запуска программы

/f:default.dat /d:nr /p1:0 /p2:1000

Код программы:

```
def nr(num_seq, mu, sigma, targeted=True):

    def box_muller(f, s, func):
        temp = mu + sigma * math.sqrt(-2 * math.log(1 - f)) * func(2 * math.pi * s)
        return (int(temp) if targeted else temp)

    def aux(u_f, u_s, func):
        return list(map(lambda f, s: box_muller(f, s, func), u_f, u_s))

    divisor = max(num_seq) + 1
    num_seq, u_f, u_s = list(map(lambda num: num / divisor, num_seq)), list(), list()

    for i in range(len(num_seq)):
        if i % 2 == 0:
            u_f.append(num_seq[i])
        else:
            u_s.append(num_seq[i])

    u_f, u_s = aux(u_f, u_s, math.cos), aux(u_f, u_s, math.sin)

    num_seq = u_f + u_s

    if targeted:
        save_to_file(num_seq, 'nr.dat')
        visualize(num_seq, 'Нормальное распределение', 100)

    return num_seq
```

Пример работы программы:

Figure 1

Характеристики: x=-1.86e+03 y=283.5

Нормальное распределение

Частота

Значение

This figure is a histogram titled "Нормальное распределение" (Normal Distribution). The plot shows a bell-shaped curve representing a normal distribution. The x-axis is labeled "Значение" (Value) and ranges from -3000 to 4000 with major ticks every 1000 units. The y-axis is labeled "Частота" (Frequency) and ranges from 0 to 300 with major ticks every 50 units. The distribution is centered at approximately -1.86e+03 (x = -1860), with the highest frequency of 283.5 occurring at y = 283.5. The curve is symmetric and peaks sharply around the center.

Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today
tr.dat	21,0 KiB	plain text document	Today
ex.dat	41,6 KiB	plain text document	Today
nr.dat	46,2 KiB	plain text document	Today

Алгоритм 5. Гамма распределение.

Описание алгоритма.

Случайное число Y , подчиняющееся гамма распределению, определяют по формуле:

$$Y = a - b * \ln \{(1 - U_1) \dots (1 - U_m)\}.$$

Параметры запуска программы

/f:default.dat /d:gm /p1:0 /p2:1000000 /c:2

Код программы:

```
def gm(num_seq, a, b):  
  
    c, divisor = input('Введите значение параметра с (целое число): '), max(num_seq) + 1  
  
    while not c.isdigit():  
        c = input('Некорректное значение параметра с! Попробуйте снова: ')  
  
    c = int(c)  
  
    num_seq = list(map(lambda num : num / divisor, num_seq))  
    num_seq = [num_seq[i:i+c] for i in range(0, len(num_seq), c)]  
  
    if len(num_seq[-1]) != c:  
        del num_seq[-1]  
  
    for i in range(len(num_seq)):  
        if 0.0 in num_seq[i]:  
            num_seq[i] = [np.random.uniform() if num_seq[i][j] == 0.0 else num_seq[i][j] for j in range(c)]  
  
    num_seq = list(map(lambda _list : int(a - b * math.log(np.prod(list(map(lambda _num : 1 - _num,  
        _list)))), num_seq)))  
  
    save_to_file(num_seq, 'gm.dat')  
    visualize(num_seq, 'Гамма-распределение', 100)
```

Пример работы программы:

The figure is a histogram titled "Гамма-распределение" (Gamma Distribution). The x-axis is labeled "Значение" (Value) and ranges from 0.0 to 1.0e7. The y-axis is labeled "частота" (Frequency) and ranges from 0 to 200. The histogram bars are blue, forming a right-skewed bell-shaped curve that peaks around a value of approximately 0.1.

Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today
tr.dat	21,0 KiB	plain text document	Today
ex.dat	41,6 KiB	plain text document	Today
nr.dat	46,2 KiB	plain text document	Today
gm.dat	37,7 KiB	plain text document	Today

Алгоритм 6. Логнормальное распределение.

Описание алгоритма.

Случайное число Y , подчиняющееся логнормальному распределению, определяют по формуле:

$$Y = a + \exp(b - U).$$

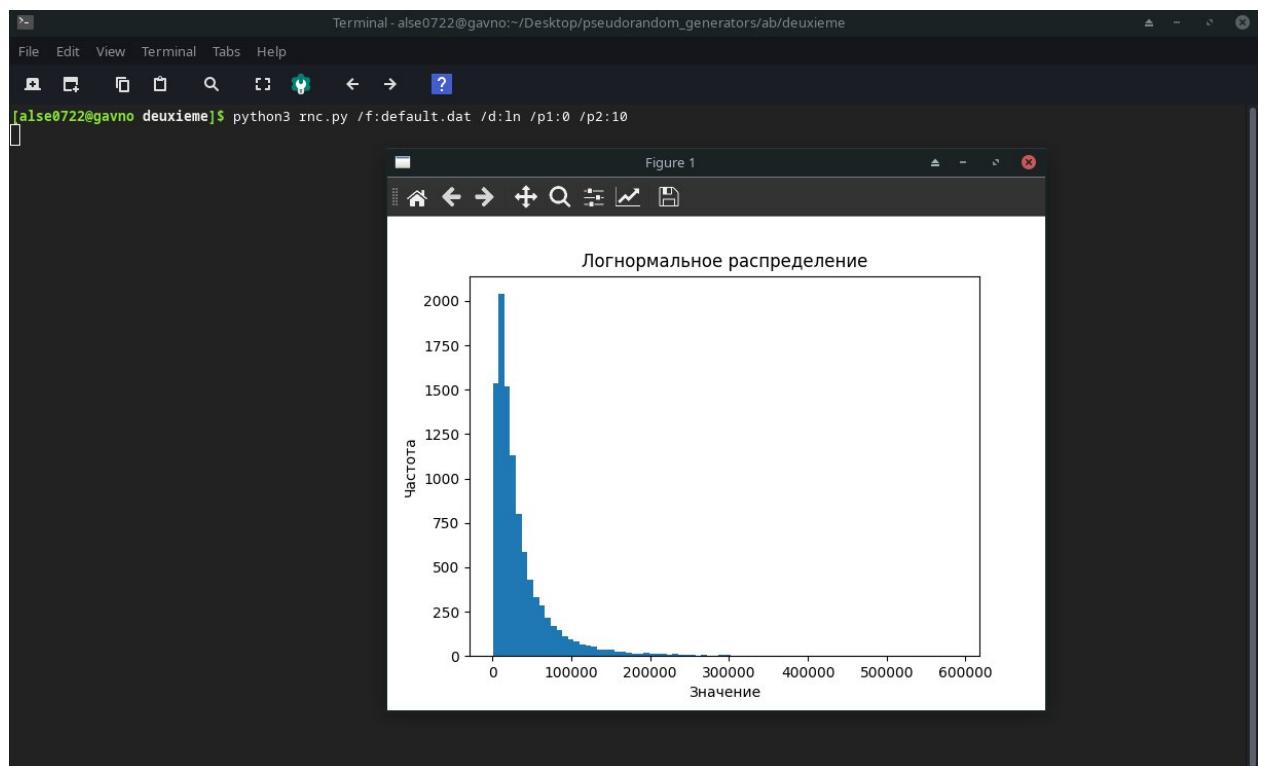
Параметры запуска программы

/f:default.dat /d:ln /p1:0 /p2:10

Код программы:

```
def ln(num_seq, a, b):  
  
    num_seq = nr(num_seq, 0, 1, False)  
    num_seq = list(map(lambda _num : int(a + math.e ** (b - _num)), num_seq))  
  
    save_to_file(num_seq, 'In.dat')  
    visualize(num_seq, 'Логнормальное распределение', 80)
```

Пример работы программы:



Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today
tr.dat	21,0 KiB	plain text document	Today
ex.dat	41,6 KiB	plain text document	Today
nr.dat	46,2 KiB	plain text document	Today
gm.dat	37,7 KiB	plain text document	Today
In.dat	57,1 KiB	plain text document	Today

Алгоритм 7. Логистическое распределение.

Описание алгоритма.

Случайное число Y , подчиняющееся логистическому распределению, определяют по формуле:

$$Y = a + b * \ln\left(\frac{U}{1 - U}\right).$$

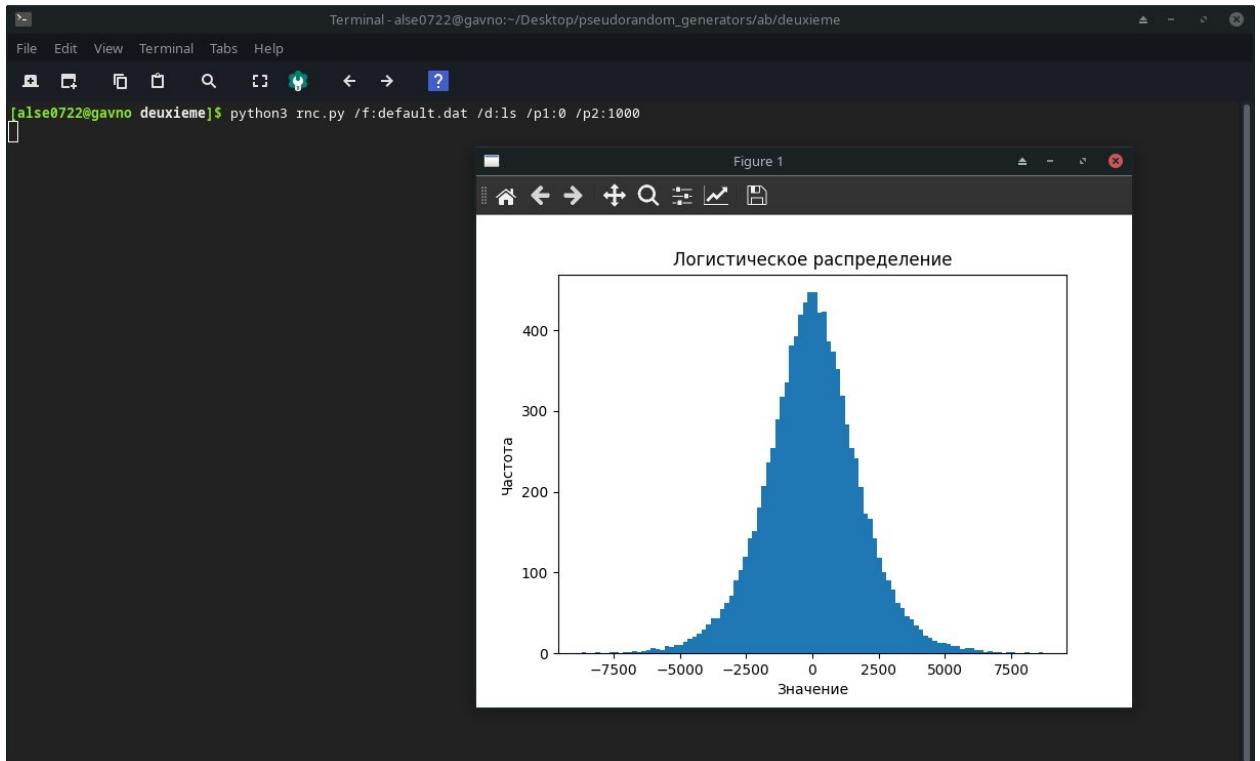
Параметры запуска программы

/f:default.dat /d:ls /p1:0 /p2:1000

Код программы:

```
def ls(num_seq, a, b):  
  
    divisor, num_seq_norm = max(num_seq) + 1, num_seq  
    num_seq = list(map(lambda _num : _num / divisor, num_seq))  
    num_seq = [np.random.uniform() if num_seq[i] == 0.0 else num_seq[i] for i in range(len(num_seq))]  
    num_seq = list(map(lambda _num : int(a + b * math.log(_num / (1 - _num))), num_seq))  
  
    save_to_file(num_seq, 'ls.dat')  
    visualize(num_seq, 'Логистическое распределение', 100)
```

Пример работы программы:



Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today
tr.dat	21,0 KiB	plain text document	Today
ex.dat	41,6 KiB	plain text document	Today
nr.dat	46,2 KiB	plain text document	Today
gm.dat	37,7 KiB	plain text document	Today
In.dat	57,1 KiB	plain text document	Today
ls.dat	48,6 KiB	plain text document	Today

Алгоритм 8. Биномиальное распределение.

Описание алгоритма.

Преобразование к биномиальному распределению осуществим методом обратной функции. Функция распределения получается следующим образом: $F(y) = \sum_{k=0}^y \binom{n}{k} p^k (1-p)^{n-k}$, $y = 0, 1, \dots, n$. Случайное число Y является наименьшим значением y , для которого $U \leq F(y)$.

Параметры запуска программы

/f:default.dat /d:bi /p1:100 /p2:0.75

Код программы:

```
def bi(num_seq, n, p):

    def inverse_function_method(n, p, val, bins=[], gen_bins=False):

        k, accum = 0, 0
        l_p, r_p = 1, (1 - p) ** n

        if gen_bins:

            bins = []
            while accum < val:

                bin_coef = math.comb(n, k)
                accum += bin_coef * l_p * r_p
                k, l_p, r_p = k + 1, l_p * p, r_p / (1 - p)
                bins.append(bin_coef)

            return bins

        else:
            while accum < val:
                accum += bins[k] * l_p * r_p
                k, l_p, r_p = k + 1, l_p * p, r_p / (1 - p)

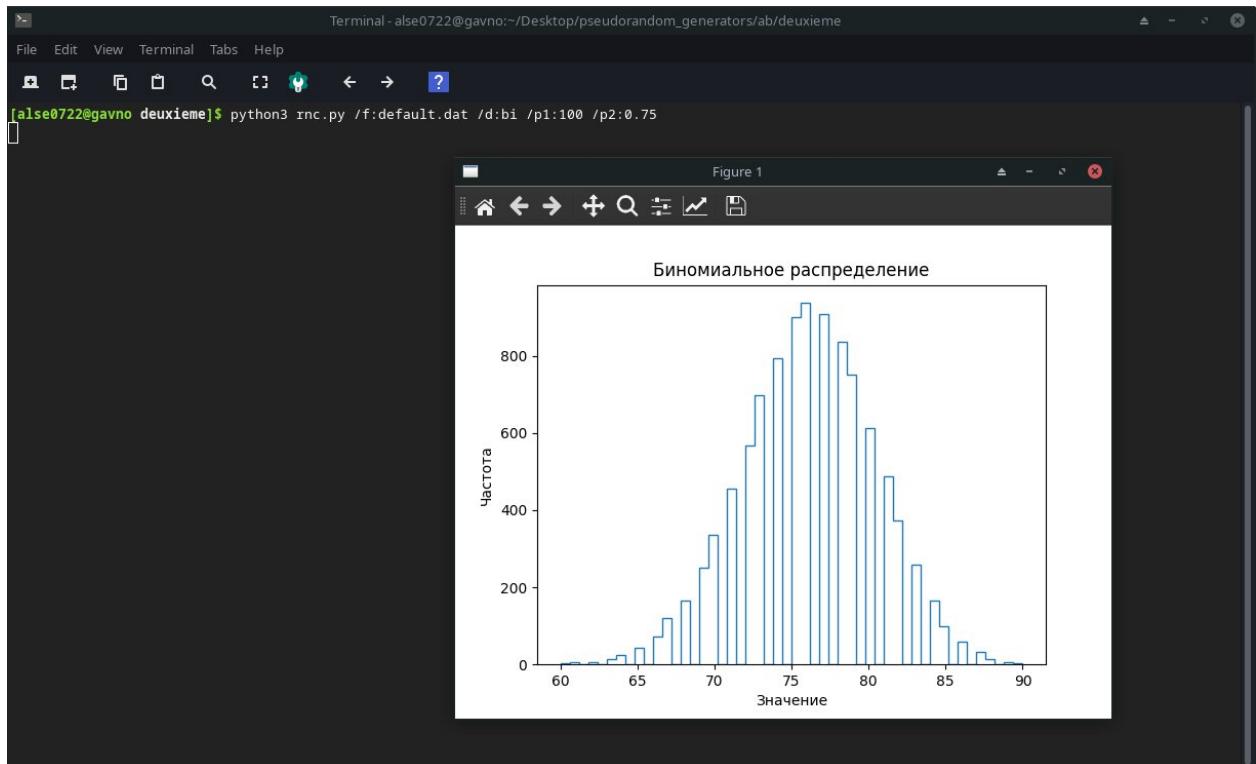
            return k

    divisor, n = max(num_seq) + 1, int(n)
    num_seq = list(map(lambda _num : _num / divisor, num_seq))
    num_seq = [np.random.uniform() if num_seq[i] == 0.0 else num_seq[i] for i in range(len(num_seq))]

    uni_max = max(num_seq)
    bins = inverse_function_method(n, p, uni_max, gen_bins=True)
    num_seq = list(map(lambda _num : inverse_function_method(n, p, _num, bins), num_seq))

    save_to_file(num_seq, 'bi.dat')
    visualize(num_seq, 'Биномиальное распределение', cont=False)
```

Пример работы программы:



Name	Size	Type	Date Modified
default.dat	47,0 KiB	plain text document	Вторник
rnc.py	6,8 KiB	Python script	Today
st.dat	38,0 KiB	plain text document	Today
tr.dat	21,0 KiB	plain text document	Today
ex.dat	41,6 KiB	plain text document	Today
nr.dat	46,2 KiB	plain text document	Today
gm.dat	37,7 KiB	plain text document	Today
In.dat	57,1 KiB	plain text document	Today
ls.dat	48,6 KiB	plain text document	Today
bi.dat	29,3 KiB	plain text document	Today

ПРИЛОЖЕНИЕ А

Листинг программы prng.py (Задание 1)

```
import random
import sys
import strings
import math

def get_input():
    opts = {}
    args = sys.argv[1:]

    for i in range(len(args)):
        opts[args[i][1]] = args[i][3:]

    if len(opts) == 0:
        opts = {'g': 'lc', 'i': [6075, 106, 1283, random.randint(1, 1000)], 'n': 10000, 'f': 'rnd.dat'}
    return opts

keys = opts.keys()

if 'h' in keys:
    opts['h'] = True

if 'g' not in keys:
    opts['g'] = 'lc'

if 'i' in keys:
    opts['i'] = list(map(int, opts['i'].split(',')))
else:
    opts['i'] = [6075, 106, 1283, random.randint(1, 1000)]

if 'n' in keys:
    opts['n'] = int(opts['n'])
else:
    opts['n'] = 10000

if 'f' not in keys:
    opts['f'] = 'rnd.dat'

return opts

def save_as_file(seq_num, seq_len, path):

    with open(path, 'w') as f:
        for i in range(seq_len):
            f.write(str(seq_num[i]) + ('\n' if i == seq_len - 1 else ','))

def progress_output(perc_cur, perc_25, perc_50, perc_75):

    if perc_cur == perc_25:
        print('[STATUS] Генерация чисел: 25%')
    elif perc_cur == perc_50:
```

```

print('[STATUS] Генерация чисел: 50%')
elif perc_cur == perc_75:
    print('[STATUS] Генерация чисел: 75%')

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)

#python3 prng.py /g:rsa /i:514081,99991,0,127,10
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)

"""

stream=$(python3 -c "import random; input=".join([str(random.randint(1, 1024)) + (" if i == 255 else ',' ) for i in range(256)])"; print(input)")
python3 prng.py /g:rc4 /i:$stream
"""

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)

mt = [0 for i in range(624)]
def twist(mt):

    lower_mask, upper_mask, reg_len = 0x7fffffff, 0x80000000, 624

    for i in range(reg_len):
        x = (mt[i] & upper_mask) + (mt[(i + 1) % reg_len] & lower_mask)
        xA = x >> 1
        if (x % 2) != 0:
            xA ^= 0x9908b0df
        mt[i] = mt[(i + 397) % reg_len] ^ xA

```

```

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; 0xfc60000)
    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)

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 = Ifsr_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)

def Ifsr_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 Ifsr(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 = Ifsr_machinerie(seq_len, 0, w, nonzero_coeffs, init_register)

```

```

save_as_file(seq_num, seq_len, path)

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)

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)

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)

def main():
    opts = get_input()
    method, IV, seq_len, path = opts['g'], opts['i'], opts['n'], opts['f']

    if 'h' in opts.keys():
        print(strings.help)
        return

```

```
if method=='lc':
    lc(IV, seq_len, path)
elif method=='add':
    add(IV, seq_len, path)
elif method=='5p':
    five_p(IV, seq_len, path)
elif method=='lfsr':
    lfsr(IV, seq_len, path)
elif method=='nfsr':
    nfsr(IV, seq_len, path)
elif method=='mt':
    MT(IV, seq_len, path)
elif method=='rc4':
    rc4(IV, seq_len, path)
elif method=='rsa':
    rsa(IV, seq_len, path)
elif method=='bbs':
    bbs(IV, seq_len, path)
else:
    print('Такого генератора нет!')
    return

if __name__ == '__main__':
    main()
```

ПРИЛОЖЕНИЕ Б

Листинг программы гнс.ру (Задание 2)

```
import sys
import math
import matplotlib.pyplot as plt
import numpy as np

def get_input():
    params = {}
    args = sys.argv[1:]

    for i in range(len(args)):
        flag = i == 0 or i == 1
        params[args[i][1:(2 if flag else 3)]] = args[i][(3 if flag else 4):]

    keys = params.keys()

    if 'f' not in keys:
        params['f'] = 'default.dat'

    if 'd' not in keys or 'p1' not in keys or 'p2' not in keys:
        params['d'], params['p1'], params['p2'] = 'st', 0, 100
    else:
        params['p1'], params['p2'] = float(params['p1']), float(params['p2'])

    return params

def save_to_file(num_seq, file_name='default.dat'):
    penult = len(num_seq) - 1

    with open(file_name, 'w') as file:
        for i in range(penult):
            file.write(str(num_seq[i]) + ',')
        file.write(str(num_seq[-1]) + '\n')

def visualize(num_seq, distrib_str, bin_num=50, cont=True):
    num_seq = np.array(num_seq)

    if cont:
        plt.hist(num_seq, bins=bin_num)
    else:
        plt.hist(num_seq, bins=bin_num, histtype="step")

    plt.xlabel('Значение')
    plt.ylabel('Частота')
    plt.title(distrib_str)

    plt.show()

def bi(num_seq, n, p):
```

```

def inverse_function_method(n, p, val, bins=[], gen_bins=False):

    k, accum = 0, 0
    l_p, r_p = 1, (1 - p) ** n

    if gen_bins:

        bins = list()
        while accum < val:

            bin_coef = math.comb(n, k)
            accum += bin_coef * l_p * r_p
            k, l_p, r_p = k + 1, l_p * p, r_p / (1 - p)
            bins.append(bin_coef)

    return bins

else:
    while accum < val:
        accum += bins[k] * l_p * r_p
        k, l_p, r_p = k + 1, l_p * p, r_p / (1 - p)

    return k

divisor, n = max(num_seq) + 1, int(n)
num_seq = list(map(lambda _num : _num / divisor, num_seq))
num_seq = [np.random.uniform() if num_seq[i] == 0.0 else num_seq[i] for i in range(len(num_seq))]

uni_max = max(num_seq)
bins = inverse_function_method(n, p, uni_max, gen_bins=True)
num_seq = list(map(lambda _num : inverse_function_method(n, p, _num, bins), num_seq))

save_to_file(num_seq, 'bi.dat')
visualize(num_seq, 'Биномиальное распределение', cont=False)

def ls(num_seq, a, b):

    divisor, num_seq_norm = max(num_seq) + 1, num_seq
    num_seq = list(map(lambda _num : _num / divisor, num_seq))
    num_seq = [np.random.uniform() if num_seq[i] == 0.0 else num_seq[i] for i in range(len(num_seq))]
    num_seq = list(map(lambda _num : int(a + b * math.log(_num / (1 - _num))), num_seq))
    num_seq_norm = nr(num_seq_norm, 0, 1, False)

    save_to_file(num_seq, 'ls.dat')
    visualize(num_seq, 'Логистическое распределение', 100)
    #visualize(num_seq_norm, 'Нормальное распределение', 100)

def ln(num_seq, a, b):

    num_seq = nr(num_seq, 0, 1, False)
    num_seq = list(map(lambda _num : int(a + math.e ** (b - _num)), num_seq))

    save_to_file(num_seq, 'ln.dat')
    visualize(num_seq, 'Логнормальное распределение', 80)
    num_seq_logged = list(map(lambda _num : math.log(_num), num_seq))
    #visualize(num_seq_logged, 'Логарифм логнормального распределения', 80)

```

```

def gm(num_seq, a, b):
    c, divisor = input('Введите значение параметра с (целое число): '), max(num_seq) + 1

    while not c.isdigit():
        c = input('Некорректное значение параметра с! Попробуйте снова: ')

    c = int(c)

    num_seq = list(map(lambda num : num / divisor, num_seq))
    num_seq = [num_seq[i:i+c] for i in range(0, len(num_seq), c)]

    if len(num_seq[-1]) != c:
        del num_seq[-1]

    for i in range(len(num_seq)):
        if 0.0 in num_seq[i]:
            num_seq[i] = [np.random.uniform() if num_seq[i][j] == 0.0 else num_seq[i][j] for j in range(c)]

    num_seq = list(map(lambda _list : int(a - b * math.log(np.prod(list(map(lambda _num : 1 - _num,
    _list)))), num_seq))

    save_to_file(num_seq, 'gm.dat')
    visualize(num_seq, 'Гамма-распределение', 100)

def nr(num_seq, mu, sigma, targeted=True):

    def box_muller(f, s, func):
        temp = mu + sigma * math.sqrt(-2 * math.log(1 - f)) * func(2 * math.pi * s)
        return (int(temp) if targeted else temp)

    def aux(u_f, u_s, func):
        return list(map(lambda f, s : box_muller(f, s, func), u_f, u_s))

    divisor = max(num_seq) + 1
    num_seq, u_f, u_s = list(map(lambda num : num / divisor, num_seq)), list(), list()

    for i in range(len(num_seq)):
        if i % 2 == 0:
            u_f.append(num_seq[i])
        else: u_s.append(num_seq[i])

    u_f, u_s = aux(u_f, u_s, math.cos), aux(u_f, u_s, math.sin)

    num_seq = u_f + u_s

    if targeted:
        save_to_file(num_seq, 'nr.dat')
        visualize(num_seq, 'Нормальное распределение', 100)

    return num_seq

def ex(num_seq, a, b):

    divisor = max(num_seq) + 1
    num_seq = [num_seq[i] for i in range(len(num_seq)) if num_seq[i] != 0]

```

```

    num_seq = list(map(lambda num : int(-b * math.log(num) + a), list(map(lambda num : num / divisor,
num_seq))))
    save_to_file(num_seq, 'ex.dat')
    visualize(num_seq, 'Общее экспоненциальное распределение', 100)

def tr(num_seq, a, b):
    divisor = max(num_seq) + 1
    num_seq, u_f, u_s = list(map(lambda num : num / divisor, num_seq)), list(), list()

    for i in range(len(num_seq)):
        if i % 2 == 0:
            u_f.append(num_seq[i])
        else:
            u_s.append(num_seq[i])

    num_seq = list(map(lambda f, s : int(a + b * (f + s - 1)), u_f, u_s))

    save_to_file(num_seq, 'tr.dat')
    visualize(num_seq, 'Треугольное распределение', 100)

def st(num_seq, a, b):
    divisor = max(num_seq) + 1
    num_seq = list(map(lambda num : int(num / divisor * b + a), num_seq))

    save_to_file(num_seq, 'st.dat')
    visualize(num_seq, 'Стандартное равномерное распределение с заданным интервалом')

def main():
    params = get_input()
    file, distrib, p1, p2 = params.values()

    f = open(file, "r")
    num_seq = list(map(int, (f.read()).split(',')))

    if distrib=='st':
        st(num_seq, p1, p2)
    elif distrib=='tr':
        tr(num_seq, p1, p2)
    elif distrib=='ex':
        ex(num_seq, p1, p2)
    elif distrib=='nr':
        nr(num_seq, p1, p2)
    elif distrib=='gm':
        gm(num_seq, p1, p2)
    elif distrib=='ln':
        ln(num_seq, p1, p2)
    elif distrib=='ls':
        ls(num_seq, p1, p2)
    elif distrib=='bi':
        bi(num_seq, p1, p2)
    else:
        print("Такого распределения нет!")
    return

```

```
if __name__ == '__main__':
    main()
```