МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра теоретических основ компьютерной безопасности и криптографии

**Генерация псевдослучайных чисел**

ОТЧЕТ ПО ДИСЦИПЛИНЕ

«ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ»

студента 4 курса 431 группы

специальности 10.05.01 Компьютерная безопасность

факультета компьютерных наук и информационных технологий

Ухова Александра Андреевича

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Научный руководитель  Ст. преподаватель | \_\_\_\_\_\_\_\_\_\_ | И.И. Слеповичев |
|  | подпись, дата |  |

Саратов 2024

**СОДЕРЖАНИЕ**

[1 Описание технической задачи 3](#_gjdgxs)

[2 Линейный конгруэнтный метод 4](#_30j0zll)

[3 Аддитивный метод 6](#_1fob9te)

[4 Регистр сдвига с обратной линейной связью (РСЛОС) 9](#_tyjcwt)

[5 Пятипараметрический метод 12](#_4d34og8)

[6 Нелинейная комбинация РСЛОС 15](#_2s8eyo1)

[7 Вихрь Мерсенна 18](#_17dp8vu)

[8 Алгоритм RC4 22](#_3rdcrjn)

[9 ГПСЧ на основе RSA 25](#_26in1rg)

[10 Алгоритм Блюма-Блюма-Шуба 28](#_lnxbz9)

[ПРИЛОЖЕНИЕ 31](#_35nkun2)

1. **Описание технической задачи**

Нужно написать программу, генерирующую псевдослучайные числа из заданного диапазона. Входные параметры алгоритма передаются через строку параметров (через файл с параметрами). Выходные значения записываются в файл, указанный в параметре запуска программы.

Для запуска программы требуется использовать следующие ключи:

**-g –** определение типа генератора псевдослучайных чисел

**-i** – передаваемые параметры для выбранного генератора

**-n** – количество генерируемых чисел (если не указан параметр, то по умолчанию генерируется 10000)

**-f** – полный путь к файлу куда требуется сохранить последовательность сгенерированных чисел (если не указан параметр, то по умолчанию данные записываются в файл rnd.dat)

**Пример запуска программы:**

Линейный конгруэнтный метод.

python prng.py /g lc /i “191216 1273 1921 5732” /n 10000 /f outfile.txt

1. **Линейный конгруэнтный метод**

Линейный конгруэнтный метод является одним из простых и популярных методов, в основе которого лежит выбор четырех ключевых чисел:

* , модуль;
* , множитель;
* , приращение (инкремент);
* , начальное значение.

*Линейная конгруэнтная последовательность* (ЛКП) – это последовательность ПСЧ, получаемая по формуле:

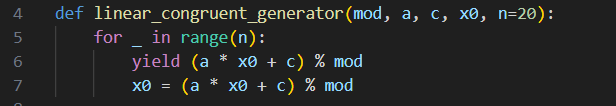
,

ключом для которой служит .

Существует обобщение формулы ЛКМ:

.

**Исходный текст программы**

****

1. **Аддитивный метод**

Идею рекурсивного вычисления значения можно обобщить до формулы, использующей два предыдущих значения и заданной начальной последовательности. Тогда, максимальная длина последовательности в лучшем случае будет равна *m*2, так как последовательность не будет повторяться, пока не будет получено равенство:

Простейшая последовательность, в которой зависит более чем от одного предыдущего значений, это последовательность Фибоначчи. Однако числа, получаемые с помощью такого рекуррентного соотношения, недостаточно случайны. Поэтому, обобщив, можно использовать следующую формулу:

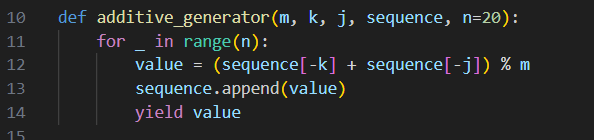
.

Дж. Ж. Митчелом и Д.Ф. Муром в 1958 году были предложены значения, называемые запаздываниями, а именно  и . Последовательность с такими параметрами была названа последовательностью Фибоначчи с запаздыванием и определялась следующим образом:

Приведем другие варианты запаздываний для аддитивного ГПСЧ: (9,49), (19,58), (18,65), (25,73), (38,89), (2,93), (21,94), (11,95), (37, 100), (33,118), (10,111), (37,124), (29,132), (52,145), (57,134), (83, 258) (107,378), (273, 607), (1029, 2281), (576, 3217), (4187, 9689), (7083, 19937), (9739, 23209).

При больших запаздываниях данный метод становится неэффективным из-за высоких требований к памяти.

**Исходный текст программы**



1. **Регистр сдвига с обратной линейной связью (РСЛОС)**

*Регистр сдвига* – упорядоченный набор битов, допускающий операцию изменения позиций битов на одну и ту же величину влево или вправо.

*Регистр сдвига с обратной линейной связью* (РСЛОС) – регистр сдвига битовых слов, у которого входной (вдвигаемый) бит является линейной функцией остальных битов. Вдвигаемый вычисленный бит заносится в ячейку с номером 0. Количество ячеек *p* называют длиной регистра.

Для натурального числа *p* и , принимающих значения 0 или 1, определяют следующую рекуррентную формулу:

Из которой видно, что для РСЛОС функция обратной связи является линейной булевой функцией от состояний всех или некоторых битов регистра.

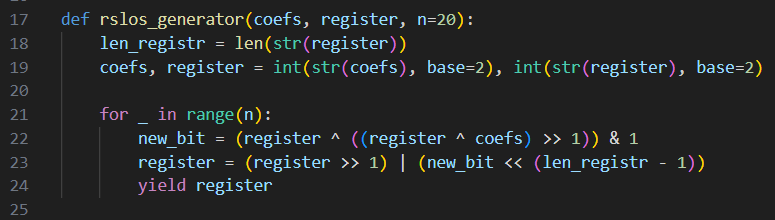
Одна итерация алгоритма, генерирующего последовательность, состоит из следующих шагов:

1. Содержимое ячейки формирует очередной бит ПСП битов.
2. Содержимое ячейки 0 определяется значением функции обратной связи, являющейся линейной булевой функцией с коэффициентами . Его вычисляют по формуле, описанной выше.
3. Содержимое каждого *i*-го бита перемещается в -й, .
4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

Максимальный период составляет Для того, чтобы получить максимальный период используют примитивные многочлены, коэффициенты которых представлены в виде двоичной последовательности.

*Период последовательности* – это наименьшее положительное целое , такое, что для всех значений .

**Исходный текст программы**



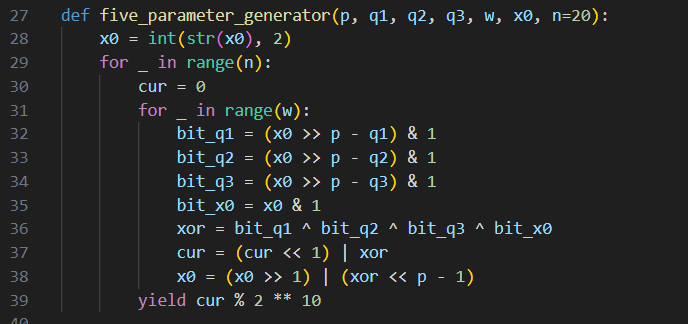
1. **Пятипараметрический метод**

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:

Параметры и , первоначально задают как начальный вектор. Примеры параметров с наибольшим периодом приведены в следующей таблице.:

| ***p*** | ***q1*** | ***q2*** | ***q3*** |
| --- | --- | --- | --- |
| 89 | 20 | 40 | 69 |
| 107 | 31 | 57 | 82 |
| 127 | 22 | 63 | 83 |
| 521 | 86 | 197 | 447 |
| 607 | 167 | 307 | 461 |
| 1279 | 339 | 630 | 988 |
| 2203 | 585 | 1197 | 1656 |
| 2281 | 577 | 1109 | 1709 |
| 3217 | 809 | 1621 | 2381 |
| 4253 | 1093 | 2254 | 3297 |

**Исходный текст программы**

****

1. **Нелинейная комбинация РСЛОС**

*Алгебраическая нормальная форма функции* – запись функции суммой по модулю 2 произведений порядков *m* независимых переменных .

*Нелинейным порядком* функции *f* называется максимальный порядок членов в записи её алгебраической нормальной формы.

Предположим теперь, что у нас регистров сдвига с линейной обратной связью , их длины попарно различны и больше двух. Подобно комбинированному методу Таусворта, мы можем объединить эти РСЛОС, но уже при помощи нелинейной функции *f:*

.

Тогда *линейная сложность* потока ключей равна . Если – попарно взаимно простые числа, то длина периода последовательности равна:

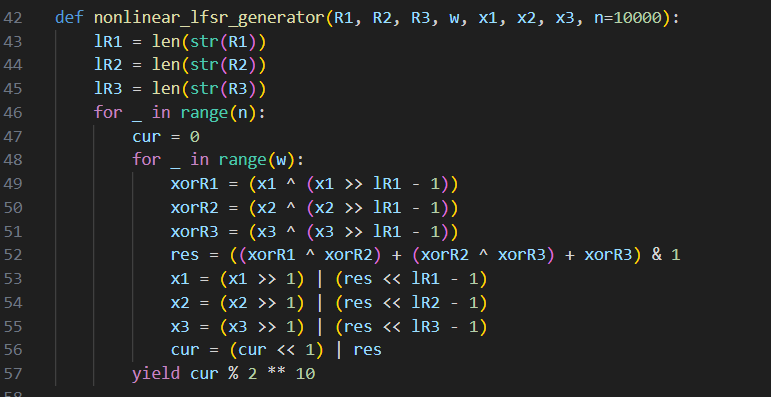
.

Генератор Геффа является примером нелинейной комбинации РСЛОС. В этом генераторе используются три РСЛОС, объединённые нелинейным образом. Длины этих регистров – попарно простые числа. Нелинейная функция генератора выглядит так:

Длина периода ПСЧ: .

Линейная сложность: .

**Исходный текст программы**

**

1. **Вихрь Мерсенна**

Метод основан на свойствах простых чисел Мерсенна и обладает рядом достоинств относительно многих других ГПСЧ. «Вихрь» – это преобразование, которое обеспечивает равномерное распределение ПСЧ.

*Числом Мерсенна* называется натуральное число , определяемое формулой

Одним из важных свойств чисел Мерсенна является то, что если является простым, то и – тоже простое. Обратное в общем случае не верно. Однако это свойство дает эффективный способ проверки числа на простоту.

Данный ГПСЧ является РСЛОС, состоящим из 624 ячеек по 32 бита. Метод Вихрь Мерсенна позволяет генерировать последовательность двоичных псевдослучайных целых *w*-битовых чисел в соответствии со следующей рекуррентной формулой:

где *p,q,r* – целые константы, *p* – степень рекуррентности, ;

– *w*-битовое двоичное целое число;

– двоичное целое число, полученное конкатенацией чисел и , когда первые (*w-r*) битов взяты из , а последние *r* битов из в том же порядке;

*А* – матрица размера *w*×*w,* состоящая из нулей и единиц, определенная посредством *а*;

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

**Шаги алгоритма:**

**Шаг 1а**. Инициализируются значения по формуле:

– всего бит, – всего *r* бит,

*–* последняя строка матрицы *А.*

**Шаг 1б**. заполняются начальными значениями.

**Шаг 2**. Вычисляется

**Шаг 3**. Вычисляется новое значение :

**Шаг 4**. Вычисляется

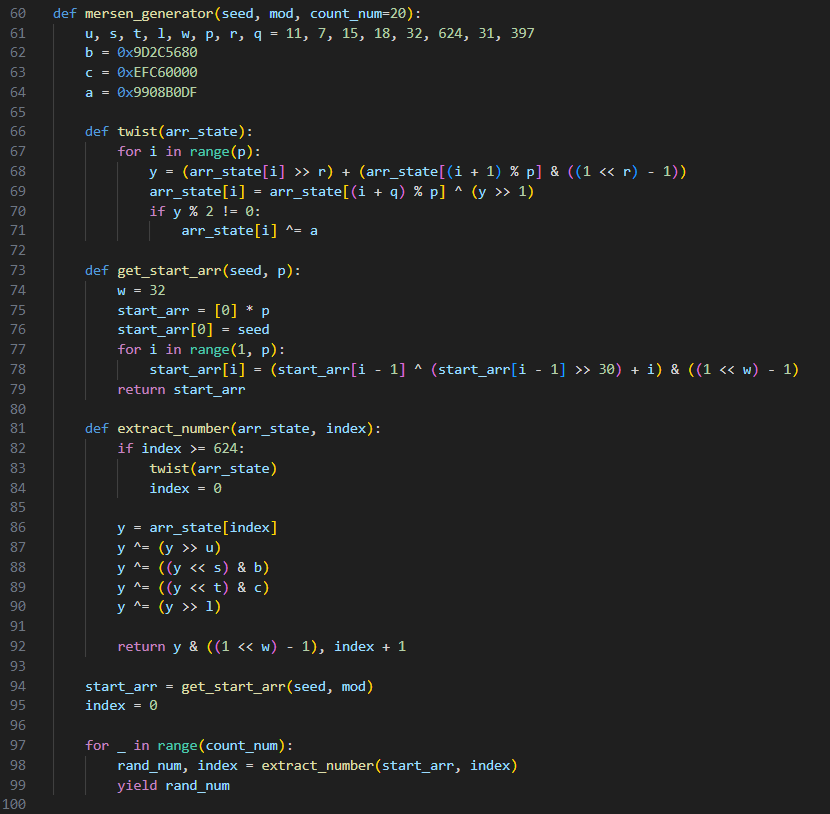
*Z* подается на выход, как результат.

**Шаг 5**. . Переход на шаг 2.

**Параметры алгоритма Вихрь Мерсенна**: , s=7, t=15,

Вихрь Мерсенна имеет огромный период, равный числу Мерсенна (219937 – 1). Этот период достаточен для большинства возможных применений алгоритма.

**Исходный текст программы**



1. **RC4**

Являясь потоковым шифром, в основе которого генератор псевдослучайных чисел, RC4 широко используется в различных криптографических протоколах. Достоинством алгоритма является высокая скорость работы и переменный размер ключа.

**Шаги алгоритма:**

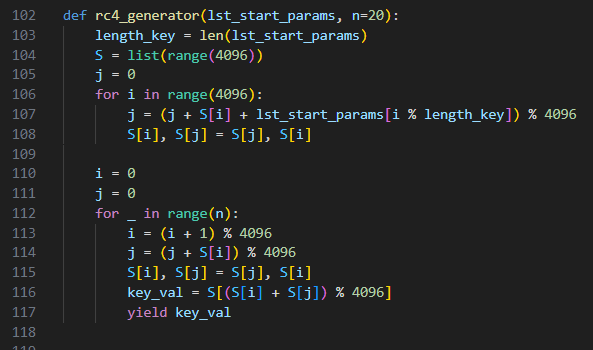
1. Инициализация ,
2. Итерация алгоритма:

Стойкость алгоритма основана на следующем наблюдении: даже если оппонент узнал ключ *К* и номер шага *i*, он может вычислить всего лишь значение , но не всё внутреннее состояние массива. Это следует из того, что оппонент не в состоянии определить значение переменной *t*, не зная *j*, .

Каждый шаг алгоритма усиливает его стойкость:

* шаг a обеспечивает однократное использование элементов массива;
* шаг b обеспечивает нелинейную зависимость выхода от массива;
* шаг c изменяет массив в процессе итераций;
* шаг d скрывает внутреннее состояние массива от анализа.

**Исходный текст программы**



1. **ГПСЧ на основе RSA**

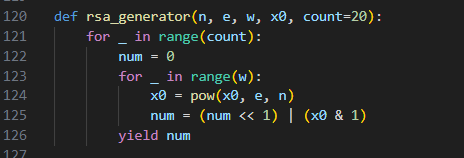
Эди Шамир предложил использовать алгоритм шифрования с открытым ключом RSA для генерации ПСЧ. В его работе показано, что предсказание выхода генератора псевдослучайных чисел равносильно взлому RSA. Очевидным недостатком такого алгоритма является низкая скорость и громоздкость реализации.

Дальнейшей модификацией этого алгоритма является ГПСЧ RSA, который также основан на сложности решения проблемы RSA.

**Шаги алгоритма:**

1. Сгенерировать два секретных простых числа *p* и *q*, а также и . Выбрать случайное целое число , такое что
2. Выбрать случайное целое – начальный вектор из интервала
3. .

**Исходный текст программы**

****

1. **Алгоритм Блюма-Блюма-Шуба (BBS)**

В основе алгоритма – использование квадратичных остатков по модулю *n*. На текущее время это один из самых простых и быстрых алгоритмов ГПСЧ, использующих вычислительно сложные задачи.

**Шаги алгоритма:**

**На входе:** Длина *l*.

**На выходе:** Последовательность псевдослучайных бит .

1. Сгенерировать два простых числа *p* и *q*, сравнимых с 3 по модулю 4. Это гарантирует, что каждый квадратичный вычет имеет один квадратный корень, который также является квадратичным вычетом. Произведение этих чисел – *n=pq* является целым числом Блюма. Выберем другое случайное целое число *x*, взаимно простое с *n*.
2. Вычислим , которое будет начальным вектором.
3. .

Интересным достоинством этого генератора является то, что для получения *i*-го бита при известных *p* и *q* достаточно воспользоваться формулой

Это дает преимущества данному ГПСЧ при работе с массивами данных с произвольной точкой доступа (random access data).

**Исходный текст программы.**

# 

# **ПРИЛОЖЕНИЕ**

**Листинг программы**

import click

def linear\_congruent\_generator(mod, a, c, x0, n=20):

    for \_ in range(n):

        yield (a \* x0 + c) % mod

        x0 = (a \* x0 + c) % mod

def additive\_generator(m, k, j, sequence, n=20):

    for \_ in range(n):

        value = (sequence[-k] + sequence[-j]) % m

        sequence.append(value)

        yield value

def rslos\_generator(coefs, register, n=20):

    len\_registr = len(str(register))

    coefs, register = int(str(coefs), base=2), int(str(register), base=2)

    for \_ in range(n):

        new\_bit = (register ^ ((register ^ coefs) >> 1)) & 1

        register = (register >> 1) | (new\_bit << (len\_registr - 1))

        yield register

def five\_parameter\_generator(p, q1, q2, q3, w, x0, n=20):

    x0 = int(str(x0), 2)

    for \_ in range(n):

        cur = 0

        for \_ in range(w):

            bit\_q1 = (x0 >> p - q1) & 1

            bit\_q2 = (x0 >> p - q2) & 1

            bit\_q3 = (x0 >> p - q3) & 1

            bit\_x0 = x0 & 1

            xor = bit\_q1 ^ bit\_q2 ^ bit\_q3 ^ bit\_x0

            cur = (cur << 1) | xor

            x0 = (x0 >> 1) | (xor << p - 1)

        yield cur % 2 \*\* 10

def nonlinear\_lfsr\_generator(R1, R2, R3, w, x1, x2, x3, n=10000):

    lR1 = len(str(R1))

    lR2 = len(str(R2))

    lR3 = len(str(R3))

    for \_ in range(n):

        cur = 0

        for \_ in range(w):

            xorR1 = (x1 ^ (x1 >> lR1 - 1))

            xorR2 = (x2 ^ (x2 >> lR1 - 1))

            xorR3 = (x3 ^ (x3 >> lR1 - 1))

            res = ((xorR1 ^ xorR2) + (xorR2 ^ xorR3) + xorR3) & 1

            x1 = (x1 >> 1) | (res << lR1 - 1)

            x2 = (x2 >> 1) | (res << lR2 - 1)

            x3 = (x3 >> 1) | (res << lR3 - 1)

            cur = (cur << 1) | res

        yield cur % 2 \*\* 10

def mersen\_generator(seed, mod, count\_num=20):

    u, s, t, l, w, p, r, q = 11, 7, 15, 18, 32, 624, 31, 397

    b = 0x9D2C5680

    c = 0xEFC60000

    a = 0x9908B0DF

    def twist(arr\_state):

        for i in range(p):

            y = (arr\_state[i] >> r) + (arr\_state[(i + 1) % p] & ((1 << r) - 1))

            arr\_state[i] = arr\_state[(i + q) % p] ^ (y >> 1)

            if y % 2 != 0:

                arr\_state[i] ^= a

    def get\_start\_arr(seed, p):

        w = 32

        start\_arr = [0] \* p

        start\_arr[0] = seed

        for i in range(1, p):

            start\_arr[i] = (start\_arr[i - 1] ^ (start\_arr[i - 1] >> 30) + i) & ((1 << w) - 1)

        return start\_arr

    def extract\_number(arr\_state, index):

        if index >= 624:

            twist(arr\_state)

            index = 0

        y = arr\_state[index]

        y ^= (y >> u)

        y ^= ((y << s) & b)

        y ^= ((y << t) & c)

        y ^= (y >> l)

        return y & ((1 << w) - 1), index + 1

    start\_arr = get\_start\_arr(seed, mod)

    index = 0

    for \_ in range(count\_num):

        rand\_num, index = extract\_number(start\_arr, index)

        yield rand\_num

def rc4\_generator(lst\_start\_params, n=20):

    length\_key = len(lst\_start\_params)

    S = list(range(4096))

    j = 0

    for i in range(4096):

        j = (j + S[i] + lst\_start\_params[i % length\_key]) % 4096

        S[i], S[j] = S[j], S[i]

    i = 0

    j = 0

    for \_ in range(n):

        i = (i + 1) % 4096

        j = (j + S[i]) % 4096

        S[i], S[j] = S[j], S[i]

        key\_val = S[(S[i] + S[j]) % 4096]

        yield key\_val

def rsa\_generator(n, e, w, x0, count=20):

    for \_ in range(count):

        num = 0

        for \_ in range(w):

            x0 = pow(x0, e, n)

            num = (num << 1) | (x0 & 1)

        yield num

def bbs\_generator(x0, w, p=127, q=131, count\_num=20):

    n = p \* q

    for \_ in range(count\_num):

        num = 0

        for \_ in range(w):

            x0 = pow(x0, 2, n)

            num = (num << 1) | (x0 & 1)

        yield num

generators = {

    'lc': lambda lst: linear\_congruent\_generator(lst[0], lst[1], lst[2], lst[3], lst[4]),

    'add': lambda lst: additive\_generator(lst[0], lst[1], lst[2], lst[3:-1:], lst[-1]),

    '5p': lambda lst: five\_parameter\_generator(lst[0], lst[1], lst[2], lst[3], lst[4], lst[5]),

    'lfsr': lambda lst: rslos\_generator(lst[0], lst[1], lst[2]),

    'nfsr': lambda lst: nonlinear\_lfsr\_generator(lst[0], lst[1], lst[2], lst[3], lst[4], lst[5], lst[6], lst[7]),

    'mt': lambda lst: mersen\_generator(lst[0], lst[1], lst[2]),

    'rc4': lambda lst: rc4\_generator(lst[0:-1:], lst[-1]),

    'rsa': lambda lst: rsa\_generator(lst[0], lst[1], lst[2], lst[3], lst[4]),

    'bbs': lambda lst: bbs\_generator(lst[0], lst[1], lst[2], lst[3], lst[4])

}

def write\_to\_file(data\_list, filepath):

    f = open(filepath, "w", encoding="UTF-8")

    data = " ".join(map(str, data\_list))

    f.write(data)

    f.close()

@click.command()

@click.option(

    '/g', type=click.Choice(["lc", "add", "5p", "lfsr", "nfsr", "mt", "rc4", "rsa", "bbs"]), required=True,

    help=(

            "Параметр указывает на метод генерации ПСЧ. "

            "код\_метода может быть одним из следующих:\n"

            "  • lc – линейный конгруэнтный метод (i: a c m x[0] – целые неотрицательные числа; m > 0)\n"

            "  • add – аддитивный метод (i: c m a[1] a[2] ... a[s] x[1] ... x[s] – целые неотрицательные числа, меньше m; m > 0)\n"

            "  • 5p – пятипараметрический метод (i: p q1 q2 q3 w x0 – целые неотрицательные числа; w – разрядность выходного числа; p > q1, q2, q3 > 0; q1, q2, q3 – попарно различны; x0 – p-битное число)\n"

            "  • lfsr – регистр сдвига с обратной связью (РСЛОС) (i: p a x[0] l; a, x0 – p-битные неотрицательные числа, a и x0 интерпретируются как двоичный вектор)\n"

            "  • nfsr – нелинейная комбинация РСЛОС (i: n l p[1] a[1] x0[1] p[2] a[2] x0[2] ... p[n] a[n] x0[n] c[1] ... c[m]; n – количество генераторов, n > 0; l – количество бит в сгенерированном значении; p[i], a[i], x0[i] – параметры соответствующего РСЛОС; c[i] – n-битные числа)\n"

            "  • mt – вихрь Мерсенна (i: p w r q a u s t l b c; w – размерность слова; r – позиция разделения, r <= w; p, q – два положительных числа, 0 < q <= p; a, b, c – w-разрядные неотрицательные числа, 0 <= a, b, c < 2^w; u, s, t, l – коэффициенты, 0 <= u, s, t, l <= w)\n"

            "  • rc4 – RC4 (i: w K0 … K255, где w – длина в битах, на которую разбиваются блоки; K0 … K255 – некоторая перестановка чисел от 0 до 255)\n"

            "  • rsa – ГПСЧ на основе RSA (i: n e x0 w l - положительные целые числа; w > 0, x0 < n; w – количество генерируемых бит за шаг, l – разрядность выходного числа)\n"

            "  • bbs – алгоритм Блюма-Блюма-Шуба (i: n x0 l – положительные целые числа; x0 < n; l – разрядность выходного числа)"

    )

)

@click.option('/n', type=int, default=10000,

              help="- количество генерируемых чисел. Если параметр не указан, то генерируется 10000 чисел.")

@click.option('/f', default="rnd.dat",

              help="- полное имя файла, в который будут выводиться данные. Если параметр не указан, данные должны записываться в файл с именем rnd.dat.")

@click.option('/i', type=str, help="Перечисление параметров для выбранного генератора")

def main(g, i, n, f):

    i = list(map(int, i.split()))

    try:

        generator\_name = g

        arguments = i + [n]

        generator = generators[generator\_name]

        file = generator\_name + "\_" + f

        numbers = [i for i in generator(arguments)]

        write\_to\_file(numbers, file)

    except Exception as err:

        print(f"Произошла ошибка: {err}")

"""

python prng.py /g lc /i "1023 127 131 1024" /n 60

python prng.py /g add /i "1024 5 9 6667 9059 3718 5389 853 9038 224 249 3437 6699 7717 2761 1714 1377 3620 4124 6814 3110 7969 1477 1298" /n 600

python prng.py /g 5p /i "38 3 10 20 10 10101011010101001010110101000111010101" /n 600

python prng.py /g lfsr /i "1000010001 1011010110" /n 600

python prng.py /g nfsr /i "1101011001 1100010001 1011010110 10 857 785 726" /n 600

python prng.py /g mt /i "6423 1234" /n 60

python prng.py /g rc4 /i "2309 1203 3836 4107 1944 1438 7472 4532 9239 1077 5584 4680 5754 5801 199 5117 2565 1804 7971 6885 8378 1106 8790 5904 4452 9068 6075 2725 9864 8279 1988 4034 1876 5932 1248 2420 8835 6797 2803 5213 4804 3972 6051 7912 6017 4107 3940 3122 9365 5713 6289 5731 364 8800 7892 3089 3138 5523 3726 5040 1898 7809 1386 2869 7844 8246 5254 2503 1253 1130 3368 4382 5316 5249 6276 4071 7893 1765 4508 9952 2734 3559 2313 3554 4483 889 7323 1518 3766 672 2132 4096 6167 8673 1219 4086 3241 7996 4274 7696 5617 8175 3383 4388 519 9199 9937 8095 489 5095 7591 2498 476 1727 4817 3592 9017 9533 7800 247 8154 2620 9273 8977 8271 5786 301 8425 3318 9171 8835 3691 2379 6213 5385 8338 7764 5322 3286 2193 2187 6189 3051 9366 1614 8210 9968 4723 2478 435 3064 9248 8619 2021 3841 5971 2077 382 7749 9034 8472 8598 5136 8417 5803 9549 4291 4550 4321 1675 2637 6206 4375 7893 9847 7974 5142 6210 7902 6811 5930 5351 7533 4732 944 2317 4393 3491 9393 8178 2293 6346 172 508 9567 6172 673 6877 2249 9612 9801 6299 7482 2324 6037 2548 910 5037 5532 5077 2594 9638 4022 719 5356 1184 4081 2277 7486 9883 8842 7460 266 246 8079 6422 8838 2102 1858 9034 9417 9342 351 4897 1161 2116 3740 4348 4506 4074 8019 7703 2228 279 2787 1763 2594 4193 9236 3038 3772 9463 9862 1853 7645 721" /n 60

python prng.py /g rsa /i "37320042546258151385585748655126402275533898458677529665166237785905883512912615700605768608589299896136946304126741313488484989387521930513771133544049676415442489082044251499963695092831431208960561181495913575017558079374616598365042558446225838569076192130606654969509117761161385964776929093056043562942995644496964324119684633518554350805247688958172238783753657689734635149 21313771133544049676415442489082044251416763454483249741353934747840086819619366458506965160046575598181142166390510565647991519183809143795101153952660774855424443628817863597390818864925727668200496135675357577444496964324119684633518554350805247626192861013350488647068390875204840518919456340410569301907212781232077722298951260234859218593729374772847487435338361409891082097 10 10778408789827883569250963876980263028140342166390510565647991519183809143795122415167903694505657190781037011063318254988841723817121548695115458698533321645606935495430645111037117339877785302353945125794644703173404840518919456340410569301907212781238950110437222437807668278467459925384821541691940460003" /n 60

python prng.py /g bbs /i "620 5 127 131" /n 60

"""

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

    main()