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

**ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ**

**ОТЧЕТ ПО ПРАКТИЧЕСКОМУ ЗАДАНИЮ 1**

студента 4 курса 431 группы

специальности 10.05.01 «Компьютерная безопасность»

факультета компьютерных наук и информационных технологий

Гендляра Сергея Максимовича

Научный руководитель

Доцент И.И. Слеповичев

подпись, дата

Саратов 2024

**Задание 1. Генерация псевдослучайных чисел**

Описание задания: создать программу, которая генерирует псевдослучайные числа из заданного диапазона. Входные параметры алгоритмы передаются программе через строку параметров, которая считывается из файла txt. Выходные значения записываются в файл dat, название которого указывается в строке параметров запуска программы.

**Алгоритм 1. Линейный конгруэнтный метод**

Функция: LC

**Описание алгоритма:** Линейный конгруэнтный генератор (LCG) — это простой и широко используемый метод для генерации псевдослучайных чисел. Он основан на рекуррентной формуле:

Xn+1= (a \* Xn+c) mod (m)

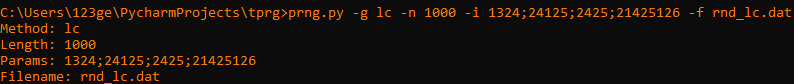
где:

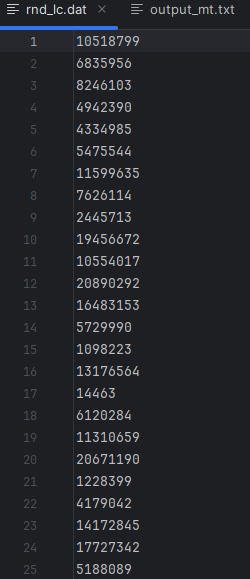
* X — последовательность псевдослучайных чисел,
* a — множитель,
* c — приращение,
* m — модуль,
* X0​ — начальное значение (seed).

**Параметры:**

* seed: Начальное значение, используемое для инициализации генератора.
* a: Множитель.
* c: Приращение.
* m: Модуль.

Запуск алгоритма через консоль выглядит следующим образом:



Можно наблюдать следующий результат:  


Код программы:

def linear\_congruential(length, params):  
 seed, a, c, m = map(int, params.split(';'))  
 result = []  
 for \_ in range(length):  
 seed = (a \* seed + c) % m  
 result.append(seed)  
 return result

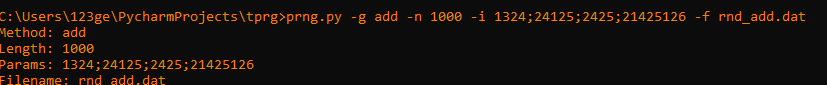
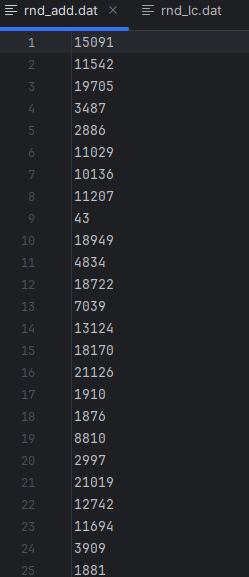
**Алгоритм 2. Аддитивный метод**

Функция: additive

**Описание алгоритма:** Аддитивный метод генерации случайных чисел основан на сложении предыдущего значения с некоторым случайным числом и взятии результата по модулю m.

**Параметры:**

* seed: Начальное значение.
* m: Модуль.
* others: Дополнительные параметры.

Запуск алгоритма через консоль выглядит следующим образом:  
В результате чего, в файле rnd\_add.dat можно видеть:  


Код программы:

def additive(length, params):  
 seed, m, \*others = map(int, params.split(';'))  
  
 # Check if m is less than or equal to 1  
 if m <= 1:  
 m = 2 # Set m to a default value of 2  
  
 result = []  
 for \_ in range(length):  
 seed = (seed + random.randint(1, m - 1)) % m  
 result.append(seed)  
 return result

**Алгоритм 3. Пятипараметрический метод**

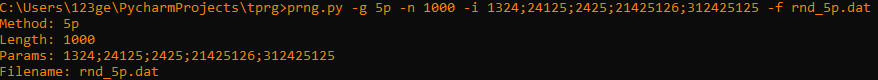
Функция: five\_parameter

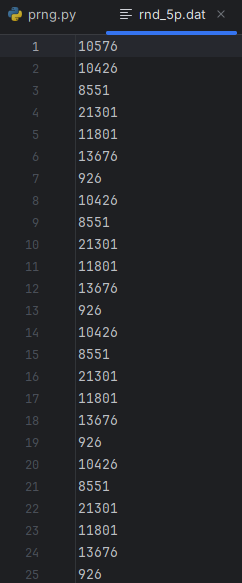
**Описание алгоритма:** Этот метод расширяет линейный конгруэнтный генератор, добавляя пятый параметр e, который может быть использован для дополнительных преобразований.

**Параметры:**

* seed: Начальное значение.
* m: Модуль.
* a: Множитель.
* c: Приращение.
* e: Дополнительный параметр.

Запуск алгоритма через командную строку имеет следующий вид:



Тогда в файле можно увидеть:  


Алгоритм написан при помощи следующего кода:

def five\_parameter(length, params):  
 seed, m, a, c, e = map(int, params.split(';'))  
 result = []  
 for \_ in range(length):  
 seed = (a \* seed + c + e) % m  
 result.append(seed)  
 return result

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

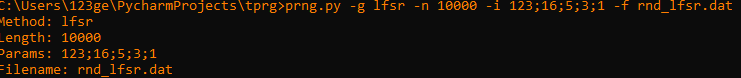
Функция: lfsr

**Описание алгоритма:** LFSR использует сдвиг регистра и линейную обратную связь для генерации псевдослучайных чисел. Обратная связь определяется с использованием битовых позиций (taps).

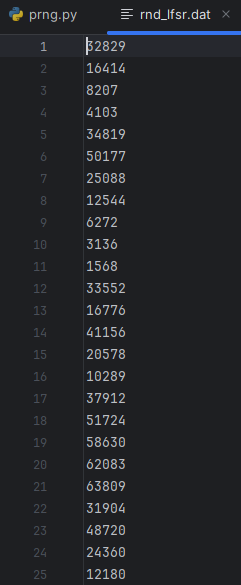
**Параметры:**

* seed: Начальное значение.
* size: Размер регистра.
* taps: Позиции для обратной связи.

Алгоритм запускается при помощи следующих команд в строке:



В файле можно увидеть:



Функция имеет следующий код:

def lfsr(length, params):  
 seed, size, \*taps = map(int, params.split(';'))  
 state = seed  
 result = []  
 for \_ in range(length):  
 lsb = state & 1  
 state >>= 1  
 feedback = 0  
 for tap in taps:  
 feedback ^= (state >> (tap - 1)) & 1  
 state |= (feedback << (size - 1))  
 result.append(state)  
 return result

**Алгоритм 5. Нелинейная комбинация РСЛОС**

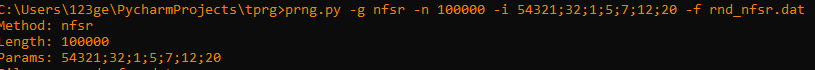
Функция: nfsr

**Описание алгоритма:** NFSR работает аналогично LFSR, но использует нелинейную обратную связь для генерации чисел, что делает последовательности менее предсказуемыми.

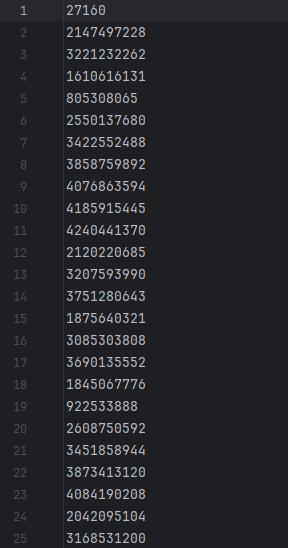
**Параметры:**

* seed: Начальное значение.
* size: Размер регистра.
* taps: Позиции для обратной связи.

Запуск алгоритма через консоль выглядит следующим образом:



В файле можно наблюдать:



**Алгоритм 6. Вихрь Мерсенна**

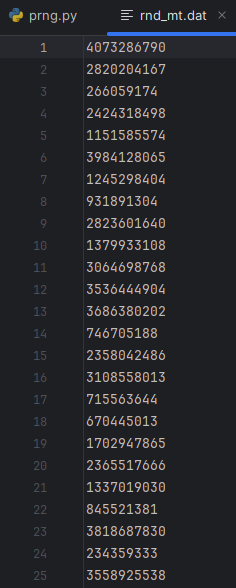
Функция: mersenne\_twister

**Описание алгоритма:** Генератор Мерсенна (MT) — это широко используемый метод генерации случайных чисел, известный своей высокой периодичностью и равномерным распределением.

Запуск алгоритма через консоль имеет следующий вид:



В файле можно наблюдать:



Алгоритм имеет следующий код:

def mersenne\_twister(length, params):  
 mt = random.sample(range(0, 2 \*\* 32), 624)  
 result = []  
 for \_ in range(length):  
 if not mt:  
 mt = random.sample(range(0, 2 \*\* 32), 624)  
 y = mt.pop(0)  
 y ^= y >> 11  
 y ^= (y << 7) & 0x9D2C5680  
 y ^= (y << 15) & 0xEFC60000  
 y ^= y >> 18  
 result.append(y)  
 return result

**Алгоритм 7. RC4**

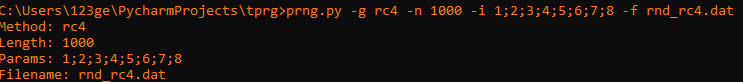
Функция: rc4

**Описание алгоритма:** RC4 — это потоковый шифр, который генерирует псевдослучайные числа с использованием ключа для инициализации массива байт (S-блок).

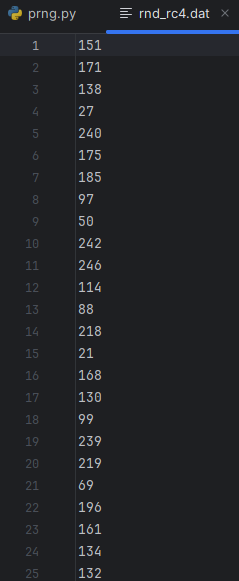
**Параметры:**

* key: Ключ, используемый для инициализации S-блока.

Запуск алгоритма через командную строку выглядит следующим образом:



Файл содержит следующие значения:



Код алгоритма выглядит следующим образом:

def rc4(length, params):  
 key = list(map(int, params.split(';')))  
 S = list(range(256))  
 j = 0  
 for i in range(256):  
 j = (j + S[i] + key[i % len(key)]) % 256  
 S[i], S[j] = S[j], S[i]  
 i = j = 0  
 result = []  
 for \_ in range(length):  
 i = (i + 1) % 256  
 j = (j + S[i]) % 256  
 S[i], S[j] = S[j], S[i]  
 result.append(S[(S[i] + S[j]) % 256])  
 return result

**Алгоритм 8. Алгоритм Блюм–Блюма–Шуба (BBS)**

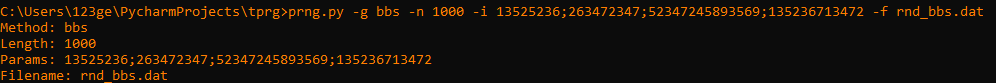
Функция: bbs

**Описание алгоритма:** BBS — это криптографически безопасный генератор псевдослучайных чисел, основанный на квадратичных вычетах.

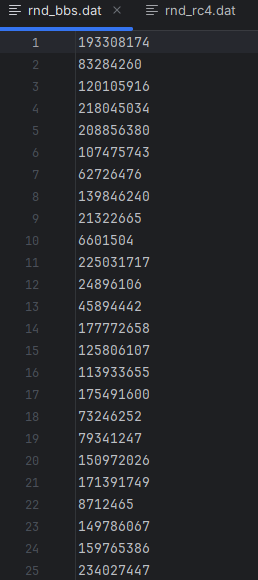
**Параметры:**

* c: Константа.
* m: Модуль.
* x: Начальное значение (seed).
* l: Дополнительный параметр).

Запуск алгоритма через командную строку выглядит следующим образом:



Файл содержит следующее:



Кодовая реализация алгоритма имеет следующий вид:

def bbs(length, params):  
 c, m, x, l = map(int, params.split(';'))  
 result = []  
 for \_ in range(length):  
 x = (x \* x + c) % m  
 result.append(x)  
 return result

**Алгоритм 9. RSA**

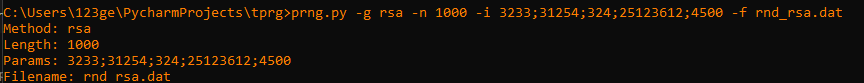
Функция: rsa\_generator

**Описание алгоритма:** Генератор RSA использует алгоритм шифрования RSA для генерации случайных чисел. Он основан на многократном возведении начального значения в степень с использованием публичного ключа и модуля.

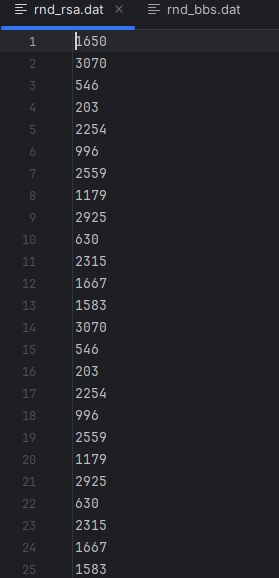
**Параметры:**

* n: Модуль.
* e: Публичный ключ.
* w: Дополнительный параметр.
* seed: Начальное значение.
* n\_count: Количество генерируемых чисел.

Запуск алгоритма через командную строку:



В файле можно наблюдать:



Данный алгоритм имеет следующую реализацию:

def rsa\_generator(n, e, w, seed, n\_count):  
 result = []  
 x = seed  
 for \_ in range(n\_count):  
 x = pow(x, e, n)  
 result.append(x)  
 return result

**Приложение А – листинг программы, реализующей задание № 1**

**import argparse**

**import random**

**def linear\_congruential(length, params):**

**seed, a, c, m = map(int, params.split(';'))**

**result = []**

**for \_ in range(length):**

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

**result.append(seed)**

**return result**

**def additive(length, params):**

**seed, m, \*others = map(int, params.split(';'))**

**# Check if m is less than or equal to 1**

**if m <= 1:**

**m = 2 # Set m to a default value of 2**

**result = []**

**for \_ in range(length):**

**seed = (seed + random.randint(1, m - 1)) % m**

**result.append(seed)**

**return result**

**def five\_parameter(length, params):**

**seed, m, a, c, e = map(int, params.split(';'))**

**result = []**

**for \_ in range(length):**

**seed = (a \* seed + c + e) % m**

**result.append(seed)**

**return result**

**def lfsr(length, params):**

**seed, size, \*taps = map(int, params.split(';'))**

**state = seed**

**result = []**

**for \_ in range(length):**

**lsb = state & 1**

**state >>= 1**

**feedback = 0**

**for tap in taps:**

**feedback ^= (state >> (tap - 1)) & 1**

**state |= (feedback << (size - 1))**

**result.append(state)**

**return result**

**def nfsr(length, params):**

**seed, size, \*taps = map(int, params.split(';'))**

**state = seed**

**result = []**

**for \_ in range(length):**

**lsb = state & 1**

**state >>= 1**

**feedback = 0**

**for tap in taps:**

**feedback ^= (state >> (size - tap)) & 1**

**state |= feedback << (size - 1)**

**result.append(state)**

**return result**

**def mersenne\_twister(length, params):**

**mt = random.sample(range(0, 2 \*\* 32), 624)**

**result = []**

**for \_ in range(length):**

**if not mt:**

**mt = random.sample(range(0, 2 \*\* 32), 624)**

**y = mt.pop(0)**

**y ^= y >> 11**

**y ^= (y << 7) & 0x9D2C5680**

**y ^= (y << 15) & 0xEFC60000**

**y ^= y >> 18**

**result.append(y)**

**return result**

**def rc4(length, params):**

**key = list(map(int, params.split(';')))**

**S = list(range(256))**

**j = 0**

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

**j = (j + S[i] + key[i % len(key)]) % 256**

**S[i], S[j] = S[j], S[i]**

**i = j = 0**

**result = []**

**for \_ in range(length):**

**i = (i + 1) % 256**

**j = (j + S[i]) % 256**

**S[i], S[j] = S[j], S[i]**

**result.append(S[(S[i] + S[j]) % 256])**

**return result**

**def bbs(length, params):**

**c, m, x, l = map(int, params.split(';'))**

**result = []**

**for \_ in range(length):**

**x = (x \* x + c) % m**

**result.append(x)**

**return result**

**def rsa\_generator(n, e, w, seed, n\_count):**

**result = []**

**x = seed**

**for \_ in range(n\_count):**

**x = pow(x, e, n)**

**result.append(x)**

**return result**

**def main():**

**parser = argparse.ArgumentParser(description='Generate pseudorandom numbers using different methods.')**

**parser.add\_argument('-g', '--method', metavar='method', type=str, help='Method of PRNG to use', required=True,**

**choices=['lc', 'add', '5p', 'lfsr', 'nfsr', 'mt', 'rc4', 'bbs', 'rsa'])**

**parser.add\_argument('-n', '--length', metavar='length', type=int, help='Length of the sequence', required=True)**

**parser.add\_argument('-i', '--params', metavar='params', type=str, nargs='+',**

**help='Parameters for the method separated by semicolons', required=True)**

**parser.add\_argument('-f', '--filename', metavar='filename', type=str, help='Output file name', required=False)**

**args = parser.parse\_args()**

**params = ';'.join(args.params)**

**print('Method:', args.method)**

**print('Length:', args.length)**

**print('Params:', params)**

**print('Filename:', args.filename)**

**if args.method == 'lc':**

**result = linear\_congruential(args.length, params)**

**print('Result:', result)**

**elif args.method == 'add':**

**result = additive(args.length, params)**

**print('Result:', result)**

**elif args.method == '5p':**

**result = five\_parameter(args.length, params)**

**print('Result:', result)**

**elif args.method == 'lfsr':**

**result = lfsr(args.length, params)**

**print('Result:', result)**

**elif args.method == 'nfsr':**

**result = nfsr(args.length, params)**

**print('Result:', result)**

**elif args.method == 'mt':**

**result = mersenne\_twister(args.length, params)**

**print('Result:', result)**

**elif args.method == 'rc4':**

**result = rc4(args.length, params)**

**print('Result:', result)**

**elif args.method == 'bbs':**

**result = bbs(args.length, params)**

**print('Result:', result)**

**elif args.method == 'rsa':**

**n, e, w, seed, n\_count = map(int, params.split(';'))**

**result = rsa\_generator(n, e, w, seed, args.length)**

**print('Result:', result)**

**if args.filename:**

**with open(args.filename, 'w') as f:**

**for num in result:**

**f.write(str(num) + '\n')**

**if \_\_name\_\_ == '\_\_main\_\_':**

**main()**