МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

ТЕОРИЯ ПСЕВДОСЛУЧАЙНЫХ ГЕНЕРАТОРОВ

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Улитина Ивана Владимировича

Проверено:
доцент _____ И. И. Слеповичев

Саратов 2023

СОДЕРЖАНИЕ

1 Генератор псевдослучайных чисел

Создайте программу для генерации псевдослучайных величин следующими алгоритмами:

1. Линейный конгруэнтный метод;
2. Аддитивный метод;
3. Пятипараметрический метод;
4. Регистр сдвига с обратной связью (РСЛОС);
5. Нелинейная комбинация РСЛОС;
6. Вихрь Мерсенна;
7. RC4;
8. ГПСЧ на основе RSA;
9. Алгоритм Блюма-Блюма-Шуба.

1.1 Линейный конгруэнтный метод

Последовательность ПСЧ, получаемая по формуле:

$$X_{n+1} = (aX_n + c) \mod m,\ n \geq 1,$$

называется *линейной конгруэнтной последовательностью (ЛКП)*. Параметры:

— $m > 0$, модуль;
— $0 \leq a \leq m$, множитель;
— $0 \leq c \leq m$, приращение;
— $0 \leq X_0 \leq m$, начальное значение.

При запуске программы дополнительно проверяется, что выполняются следующие условия, при выполнении которых ЛКП имеет период m:

1. числа c и m взаимно простые;
2. $a - 1$ кратно p для некоторого простого p, являющегося делителем m;
3. $a - 1$ кратно 4, если m кратно 4.

1.2 Аддитивный метод

Последовательность определяется следующим образом:

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

Параметры:

— $m > 0$, модуль;
— k, младший индекс;
— j, старший индекс;
— последовательность из j начальных значений.

1.3 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:

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

Параметры:
— p, q_1, q_2, q_3, коэффициенты пятипараметрического метода;
— w, размерность чисел в битах (целое положительное число);
— начальное значение регистра (целое положительное число).

1.4 Регистр сдвига с обратной связью (РСЛОС)

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.

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

1. Содержимое ячейки $p - 1$ формирует очередной бит ПСП битов.
2. Содержимое ячейки 0 определяется значением функции обратной связи, являющейся линейной булевой функцией с коэффициентами $a_1, \ldots, a_{p-1}$.
3. Содержимое каждого i-го бита перемещается в $(i + 1)$-й, $0 \leq i < p - 1$.
4. В ячейку 0 записывается новое содержимое, вычисленное на шаге 2.

Параметры:
— двоичное представление вектора коэффициентов;
— начальное значение регистра.

1.5 Нелинейная комбинация РСЛОС

Последовательность получается из нелинейной комбинации трёх РСЛОС следующим образом:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_3$$

Параметры:

— двоичное представление вектора коэффициентов для $R1, R2, R3$;

— w, длина слова;

— x_1, x_2, x_3 — десятичное представление начальных состояний регистров $R1, R2, R3$.

1.6 Вихрь Мерсенна

Псевдокод ниже (источник: википедия) представляет собой алгоритм генерации ПСЧ с помощью вихря Мерсенна:

```
// Создание массива длины n для сохранения состояний генератора
int[0..n-1] MT
int index := n+1
const int lower_mask = (1 << r) - 1
const int upper_mask = lowest w bits of (not lower_mask)

// Initialize the generator from a seed
function seed_mt(int seed) {
    index := n
    MT[0] := seed
    for i from 1 to (n - 1) { // loop over each element
        MT[i] := lowest w bits of (f * (MT[i-1] xor (MT[i-1] >> (w-2))) + i)
    }
}

// Извлечение чисел на основе массива MT[index]
// Вызывает twist() каждые n чисел
function extract_number() {
    if index >= n {
        twist()
    }

    int y := MT[index]
    y := y xor ((y >> u) and d)
    y := y xor ((y << s) and b)
    y := y xor ((y << t) and c)
    y := y xor (y >> l)
```

```
    index := index + 1
    return lowest w bits of (y)
}

// Генерация следующих n значений
function twist() {
    for i from 0 to (n-1) {
        int x := (MT[i] and upper_mask)
                  | (MT[(i+1) mod n] and lower_mask)
        int xA := x >> 1
        if (x mod 2) != 0 { // lowest bit of x is 1
            xA := xA xor a
        }
        MT[i] := MT[(i + m) mod n] xor xA
    }
    index := 0
}
```

Константы, используемые в алгоритме:

— $p = 624$;
— $w = 32$;
— $r = 31$;
— $q = 397$;
— $a = 2567483615$;
— $u = 11$;
— $s = 7$;
— $t = 15$;
— $l = 18$;
— $b = 2636928640$;
— $c = 4022730752$.

Параметры:

— модуль;
— начальное значение.

1.7 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. Описание алгоритма:

1. Инициализация S_i.

2. $i = 0, j = 0$.
3. Итерация алгоритма:
   а) $i = (i + 1) \mod 256$;
   б) $j = (j + S_i) \mod 256$;
   в) $Swap(S_i, S_j)$;
   г) $t = (S_i + S_j) \mod 256$;
   д) $K = S_t$.

Параметры:

— 256 начальных значений S_i.

1.8 ГПСЧ на основе RSA

Очевидным недостатком этого ГПСЧ на основе алгоритма шифрования является низкая скорость и громоздкость реализации. Описание алгоритма:

1. Инициализация чисел: $n = pq$, где p и q простые числа, числа e: $1 < e < f$, НОД(e, f) = 1, $f = (p - 1)(q - 1)$ и числа x_0 из интервала $[1, n - 1]$.
2. `For i = 1 to w do`
   а) $x_i \leftarrow x_{i-1}^e \mod n$.
   б) $z_i \leftarrow$ последний значащий бит x_i
3. Вернуть $z_1, \ldots, z_w$.

Параметры:

— n, модуль;

— e, число;

— w, длина слова;

— x_0, начальное значение.

1.9 Алгоритм Блюм-Блюма-Шуба

Описание алгоритма:

1. Инициализация числа: $n = 127 \cdot 131 = 16637$.
2. Вычислим $x_0 = x^2 \mod n$, которое будет начальным вектором.
3. `For i = 1 to w do`
   а) $x_i \leftarrow x_{i-1}^2 \mod n$.
   б) $z_i \leftarrow$ последний значащий бит x_i
4. Вернуть $z_1, \ldots, z_w$.

Параметры:

— n, модуль;

2 Преобразование ПСЧ к заданному распределению

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.

2.1 Метод генерации случайной величины

Если максимальное значение равномерного целого случайного числа X равно $(m-1)$, для генерации стандартных равномерных случайных чисел необходимо применять следующую формулу: $U = X/m$. Далее будут перечислены формулы преобразования случайных последовательностей к тому или иному распределению с учетом значения U.

2.2 Стандартное равномерное с заданным интервалом

$$Y = bU + a$$

2.3 Треугольное распределение

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

2.4 Общее экспоненциальное распределение

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

2.5 Нормальное распределение

$$Z_1 = \mu\sigma\sqrt{-2\ln(1-U_1)}\cos(2\pi U_2)$$
$$Z_2 = \mu\sigma\sqrt{-2\ln(1-U_1)}\sin(2\pi U_2)$$

2.6 Гамма распределение

Алгоритм для $c = k$ (k – целое число)

$$Y = a - b \ln\{(1 - U_1) \dots (1 - U_k)\}$$

2.7 Логнормальное распределение

$$Y = a + \exp(b - Z)$$

2.8 Логистическое распределение

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

2.9 Биномиальное распределение

```
y = binominal(x, a, b, m):
  u = U(x)
  s = 0
  k = 0
  Начало цикла:
    s = s + C(k, b) + a^k * (1 - a)^(b - k)
    if s > u:
      y = k
      Завершить
    if k < b - 1:
      k = k + 1
      Перейти к новой итерации цикла
    y = b
```

ПРИЛОЖЕНИЕ А

Код задания 1

```
#include <cmath>
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <functional>
#include <map>
#include <bitset>
#include <boost/dynamic_bitset.hpp>
#include <numeric>


const int BIT_AMOUNT = 32;
const int UPPER_BOUND = 1024;


void print_vector(std::vector<int> some_vec)
{
    // вывести содержимое вектора - используется при тестировании
    std::cout << std::endl;
    for (int i = 0; i < some_vec.size(); i++)
    {
        std::cout << some_vec[i] << ',';
    }
    std::cout << std::endl;
}


std::vector<int> split_to_int(std::string str, std::string token)
{
    // функция конвертации строки с числами в вектор с числами
    std::vector<int> result;
    while(str.size())
    {
        int index = str.find(token);
        if(index != std::string::npos)
        {
            result.push_back(std::stoi(str.substr(0,index)));
            str = str.substr(index + token.size());
            if (str.size() == 0) result.push_back(std::stoi(str));
        }
        else
        {
            result.push_back(std::stoi(str));
            str = "";
        }
```

```
    }
    return result;
}


std::vector<std::string> split_to_str(std::string str, std::string token)
{
    // функция конвертации строки с числами в вектор со строками
    std::vector<std::string> result;
    while(str.size())
    {
        int index = str.find(token);
        if(index != std::string::npos)
        {
            result.push_back(str.substr(0,index));
            str = str.substr(index + token.size());
            if (str.size() == 0) result.push_back(str);
        }
        else
        {
            result.push_back(str);
            str = "";
        }
    }
    return result;
}


std::string help_message =
"Аргументы должны вводиться следующим образом:"
"\n /g:название_генератора - указывает на метод генерации ПСЧ, принимает "
"следующие значения: lc, add, 5p, lfsr, nfsr, mt, rc4, rsa, bbs"
"\n /i:инициализирущий_вектор - записывать в формате {x,y,z}, где x, y и z - "
"параметры генератора."
"\n /n:количество_генерируемых_чисел - целое число, определяющее количество "
"генерируемых чисел. По умолчанию имеет значение 10000."
"\n /f:имя_файла - полное имя файла, в который будут записываться данные. "
"По умолчанию имеет значение 'rnd.dat'."
"\n /h - информация о параметрах командной строки. При указании параметра /g: "
"будет также появляется дополнительная информация о параметрах генератора.\n";

std::string lc_help_message =
"Для линейного конгруэнтного генератора необходимо ввести параметры следующим "
"образом: '/i:m,a,c,last_x' - где:"
"\n m - модуль,"
"\n a - множитель,"
"\n c - приращение (инкремент),"
"\n last_x - начальное значение\n";
```

```
std::string add_help_message =
"Для аддитивного генератора необходимо ввести параметры следующим "
"образом: '/i:m,k,j,x_0,x_1,...,x_n' - где:"
"\n m - модуль,"
"\n k - некоторый младший индекс (j > k >= 1),"
"\n j - некоторый старший индекс (j > k >= 1),"
"\n x_0,x_1,...,x_n - вводимая последовательность\n";

std::string lfsr_help_message =
"Для генератора РСЛОС необходимо ввести параметры следующим "
"образом: '/i:r,c' - где:"
"\n r - начальное состояние регистра (32 числа '0' или '1' без разделителей),"
"\n c - коэффициенты многочлена (32 числа '0' или '1' без разделителей)\n";

std::string fp_help_message =
"Для пятипараметрического генератора необходимо ввести параметры следующим "
"образом: '/i:p,q_1,q_2,q_3,w,x' - где:"
"\n p,q_1,q_2,q_3 - коэффициенты пятипараметрического метода,"
"\n w - размерность чисел в битах (целое положительное число)"
"\n x - начальное значение регистра (целое положительное число)";

std::string nfsr_help_message =
"Для нелинейной комбинации РСЛОС необходимо ввести параметры следующим "
"образом: '/i:r1,r2,r3,w,c1,c2,c3' - где:"
"\n r1 - двоичное представление коэффициентов 1 (32 числа '0' или '1' без разделителей),"
"\n r2 - двоичное представление коэффициентов 2,"
"\n r3 - двоичное представление коэффициентов 3,"
"\n w - длина слова,"
"\n c1 - десятичное представление начальных состояний регистров 1,"
"\n c2 - десятичное представление начальных состояний регистров 2,"
"\n c3 - десятичное представление начальных состояний регистров 3\n";

std::string mt_help_message =
"Для метода вихря Мерсенна необходимо ввести параметры следующим "
"образом: '/i:m,x' - где:"
"\n m - модуль,"
"\n x - начальное значение\n";

std::string rc4_help_message =
"Для метода RC4 необходимо ввести параметры следующим "
"образом: '/i:x1,x2,...,x256' - где:"
"\n x1,... - первые 256 начальных значений\n";

std::string rsa_help_message =
"Для метода RSA необходимо ввести параметры следующим "
"образом: '/i:n,e,w,x' - где:"
"\n n - модуль, n = pq, где p и q - простые числа,"
```

```
"\n e - случайное целое число, такое, что: 1 < e < (p - 1)(q - 1),"
"\n w - длина слова,"
"\n x - начальное значение из интервала [1, n - 1]\n";

std::string bbs_help_message =
"Для метода BBS необходимо ввести параметры следующим "
"образом: '/i:x' - где:"
"\n x - начальное значение, взаимно простое с 16637\n";

std::string other_gen_help_message = "Ошибка в названии генератора!\n";

std::string find_generate_method(std::string method)
{
    // функция для нахождения названия метода в
    // списке допустимых названий методов
    std::vector<std::string> method_list = {"lc", "add", "5p", "lfsr", "nfsr",
                                            "rc4", "rsa", "bbs", "mt"};
    for (int i = 0; i < method_list.size(); i++)
    {
        if (method == method_list[i])
        {
            return method;
        }
    }
    return "";
}


void show_help_message(std::string g="", bool h=false)
{
    // функция печати в консоли вспомогательной информации
    if (h || g == "")
    {
        std::cout << help_message << std::endl;
    }
    if (g == "lc")
    {
        std::cout << std::endl << lc_help_message << std::endl;
    }
    else if (g == "add")
    {
        std::cout << std::endl << add_help_message << std::endl;
    }
    else if (g == "lfsr")
    {
        std::cout << std::endl << lfsr_help_message << std::endl;
    }
    else if (g == "5p")
```

```
    {
        std::cout << std::endl << fp_help_message << std::endl;
    }
    else if (g == "nfsr")
    {
        std::cout << std::endl << nfsr_help_message << std::endl;
    }
    else if (g == "mt")
    {
        std::cout << std::endl << mt_help_message << std::endl;
    }
    else if (g == "rc4")
    {
        std::cout << std::endl << rc4_help_message << std::endl;
    }
    else if (g == "rsa")
    {
        std::cout << std::endl << rsa_help_message << std::endl;
    }
    else if (g == "bbs")
    {
        std::cout << std::endl << bbs_help_message << std::endl;
    }
    else if (g != "")
    {
        std::cout << std::endl << other_gen_help_message << std::endl;
    }
}


void show_progress(int i, int n)
{
    // функция для отображения в консоли статуса генерации последовательности
    if (i % (static_cast<int> (static_cast<float> (n) / 10) + 1) == 0)
    {
        std::cout << "\r" << "Генерация выполнена на " << (i * 100) / n
                  << "%" << std::flush;
    }
}


std::vector<int> get_prime_factors(int n)
{
    std::vector<int> divs;
    while (n % 2 == 0)
    {
        divs.push_back(2);
        n = n/2;
```

```
    }
    for (int i = 3; i <= sqrt(n); i = i + 2)
    {
        while (n % i == 0)
        {
            divs.push_back(i);
            n = n/i;
        }
    }
    if (n > 2)
        divs.push_back(n);

    return divs;
}

bool check_linear_params(int m, int a, int c)
{
    bool is_pass = false;

    // 1-е условие
    if (std::gcd(m, c) != 1)
    {
        return false;
    }

    // 2-е условие
    std::vector<int> m_divisors = get_prime_factors(m);
    int a_min_one = a - 1;
    for (int i = 0; i < m_divisors.size(); i++)
    {
        int p = m_divisors[i];
        if (a_min_one % p == 0)
        {
            is_pass = true;
            break;
        }
    }

    // 3-е условие
    if (m % 4 == 0)
    {
        if (a_min_one % 4 == 0 && is_pass) return true;
        return false;
    }
    else
    {
        if (is_pass) return true;
    }
```

```
    return false;
}


int linear_congruent_method(int n, std::vector<int> params, std::string f)
{
    // реализация линейного конгруэнтного метода
    int m = params[0];
    int a = params[1];
    int c = params[2];
    int last_x = params[3];

    if (!check_linear_params(m, a, c))
    {
        std::cout << std::endl;
        std::cout << "Свойства теоремы не выполняются! ";
        std::cout << std::endl;
    }

    std::ofstream output_file;
    output_file.open(f);

    output_file << last_x << ',';
    for (int i = 0; i < n; i++)
    {
        last_x = ((a * last_x + c) % m) % UPPER_BOUND;
        output_file << last_x % UPPER_BOUND << ',';
        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


int additive_method(int n, std::vector<int> params, std::string f)
{
    // реализация аддитивного метода
    int m = params[0];
    params.erase(params.begin());
    int k = params[1];
    params.erase(params.begin());
    int j = params[2];
    params.erase(params.begin());
    int seq_size = params.size();

    std::ofstream output_file;
    output_file.open(f);
```

```
    for (int i = 0; i < seq_size; i++)
    {
        output_file << params[i] % UPPER_BOUND << ',';
    }

    for (int i = 0; i < n; i++)
    {
        int next_x = (params[seq_size - k] + params[seq_size - j]) % m;
        params.push_back(next_x % UPPER_BOUND);
        params.erase(params.begin());
        output_file << next_x % UPPER_BOUND << ',';
        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


std::bitset<BIT_AMOUNT> lfsr_iteration(std::bitset<BIT_AMOUNT> init_register,
                                       std::bitset<BIT_AMOUNT> poly_coeffs)
{
    bool current_bit = 0;
    for (int j = 0; j < BIT_AMOUNT; j++)
    {
        current_bit ^= init_register[j] * poly_coeffs[j];
    }
    init_register >>= 1;
    init_register[BIT_AMOUNT - 1] = current_bit;
    return init_register;
}


int lfsr_method(int n, std::vector<std::string> str_init, std::string f)
{
    // реализация РСЛОС метода
    std::bitset<BIT_AMOUNT> poly_coeffs(str_init[0]);
    std::bitset<BIT_AMOUNT> init_register(str_init[1]);

    std::ofstream output_file;
    output_file.open(f);
    for (int i = 0; i < n; i++)
    {
        output_file << init_register.to_ulong() % UPPER_BOUND << ',';
        init_register = lfsr_iteration(init_register, poly_coeffs);
        show_progress(i, n);
    }
    output_file.close();
    return 0;
```

```
}


int five_param_method(int n, std::vector<int> str_init, std::string f)
{
    // реализация пятипараметрического метода
    int p = str_init[0];
    int q_1 = str_init[1];
    int q_2 = str_init[2];
    int q_3 = str_init[3];
    int w = str_init[4];
    int init_x = str_init[5];

    boost::dynamic_bitset<> init_register(w, init_x);

    std::ofstream output_file;
    output_file.open(f);
    for (int i = 0; i < n; i++)
    {
        output_file << init_register.to_ulong() % UPPER_BOUND << ',';

        init_register[p] = init_register[0]
                           ^ init_register[q_3]
                           ^ init_register[q_2]
                           ^ init_register[q_1];
        init_register >>= 1;
        init_register[w - 1] = 0;

        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


template <typename Bitset>
void set_in_range(Bitset& b, unsigned value, int from, int to)
{
  for (int i = from; i < to; ++i, value >>= 1)
    b[i] = (value & 1);
}

int nfsr_method(int n, std::vector<std::string> str_init, std::string f)
{
    // реализация нелинейной комбинации РСЛОС
    std::bitset<BIT_AMOUNT> pc_1(str_init[0]);
    std::bitset<BIT_AMOUNT> pc_2(str_init[1]);
    std::bitset<BIT_AMOUNT> pc_3(str_init[2]);
```

```
    int w = stoi(str_init[3]);
    boost::dynamic_bitset<> cur_reg(w, 0);

    std::bitset<BIT_AMOUNT> ir_1(stoi(str_init[4]));
    std::bitset<BIT_AMOUNT> ir_2(stoi(str_init[5]));
    std::bitset<BIT_AMOUNT> ir_3(stoi(str_init[6]));

    std::ofstream output_file;
    output_file.open(f);
    for (int i = 0; i < n; i++)
    {
        set_in_range(cur_reg,
                     ((ir_1 & ir_2) ^ (ir_2 & ir_3) ^ ir_3).to_ulong(),
                     0,
                     w - 1);
        output_file << cur_reg.to_ulong() % UPPER_BOUND << ',';

        ir_1 = lfsr_iteration(ir_1, pc_1);
        ir_2 = lfsr_iteration(ir_2, pc_2);
        ir_3 = lfsr_iteration(ir_3, pc_3);

        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


template<typename T>
constexpr T low_bits(T v, int bit_mnt) {
    return v & ((1 << bit_mnt) - 1);
}

void twist(int* mt, const int p,
           const int upper_mask, const int lower_mask,
           const uint32_t a, const int m, int &index)
{
    for (int i = 0; i < p; i++)
    {
        int x = (mt[i] & upper_mask) + (mt[(i + 1) % p] & lower_mask);
        int xA = x >> 1;
        if (x % 2 != 0)
        {
            xA = xA ^ a;
        }
        mt[i] = mt[(i + m) % p] ^ xA;
    }
```

```
    index = 0;
}


int mt_method(int n, std::vector<int> str_init, std::string f)
{
    // Метод вихря Мерсенна
    int mod = str_init[0];
    int x = str_init[1];

    // выставляю константные параметры вихря (согласно англ. википедии)
    const unsigned int w = BIT_AMOUNT; // количество бит (можно переделать под 64)
    const unsigned int p = 624;
    const unsigned int m = 397;
    const unsigned int r = 31;
    const uint32_t a = 0x9908B0DFUL;
    const unsigned int u = 11;
    const uint32_t d = 0xFFFFFFFFUL;
    const unsigned int s = 7;
    const uint32_t b = 0x9D2C5680UL;
    const unsigned int t = 15;
    const uint32_t c = 0xEFC60000UL;
    const unsigned int l = 18;
    const uint32_t f_par = 1812433253;

    const int lower_mask = (1 << r) - 1;
    const int upper_mask = low_bits(~lower_mask, w);

    int mt[p];
    int index = p + 1;


    // Инициализация генератора
    index = p;
    mt[0] = x;
    for (int i = 0; i < p; i++)
    {
        mt[i] = low_bits((f_par * mt[i - 1] ^ (mt[i - 1] >> (w - 2))) + i,
                         w - 1);
    }

    std::ofstream output_file;
    output_file.open(f);
    output_file << x % UPPER_BOUND << ',';
    for (int i = 0; i < n; i++)
    {
        if (index >= p)
        {
```

```
            twist(mt, p, upper_mask, lower_mask, a, m, index);
        }

        int y = mt[index];
        y ^= ((y >> u) & d);
        y ^= ((y << s) & b);
        y ^= ((y >> t) & c);
        y ^= y >> l;

        index += 1;

        output_file << (y % mod) % UPPER_BOUND << ',';
        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


int rc4_method(int n, std::vector<int> first_xs, std::string f)
{
    // Метод RC4
    std::vector<int> s_block;
    for (int i = 0; i < 256; i++)
    {
        s_block.push_back(i);
    }
    int j = 0;
    for (int i = 0; i < 256; i++)
    {
        j = (j + s_block[i] + first_xs[i]) % 256;
        std::swap(s_block[i], s_block[j]);
    }

    std::ofstream output_file;
    output_file.open(f);
    for (int i = 0; i < first_xs.size(); i++)
    {
        output_file << first_xs[i] % UPPER_BOUND << ',';
    }

    int i = 0;
    j = 0;
    for (int l = 0; l < n; l++)
    {
        i = (i + 1) % 256;
        j = (j + s_block[i]) % 256;
        std::swap(s_block[i], s_block[j]);
```

```
        int t = (s_block[i] + s_block[j]) % 256;
        int k = s_block[t];

        output_file << k % UPPER_BOUND << ',';
        show_progress(i, n);
    }

    output_file.close();
    return 0;
}


int module_power(int x, int y, int mod)
{
    // возведение в степень по модулю
    if (y == 0)
    {
        return 1;
    }
    int temp = module_power(x, y / 2, mod) % mod;
    temp = (temp * temp) % mod;
    if (y % 2 == 1)
    {
        temp = (temp * x) % mod;
    }
    return temp;
}


int rsa_method(int n, std::vector<int> str_init, std::string f)
{
    // Метод RSA
    int mod = str_init[0];
    int e = str_init[1];
    int l = str_init[2];
    int x = str_init[3];

    std::ofstream output_file;
    output_file.open(f);

    output_file << x % UPPER_BOUND << ',';
    for (int i = 0; i < n; i++)
    {
        std::string z_seq = "";
        for (int j = 0; j < l; j++)
        {
            x = module_power(x, e, mod);
            int cur_z = x % 2;
```

```
            z_seq.append(std::to_string(cur_z));
        }
        output_file << std::stoi(z_seq, nullptr, 2) % UPPER_BOUND << ',';
        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


int bbs_method(int n, std::vector<int> str_init, std::string f)
{
    // Метод Блюма-Блюма-Шуба
    int x = str_init[0];
    int prime_number = 16637;

    if (std::gcd(x, prime_number) != 1)
    {
        std::cout << std::endl;
        std::cout << "Генерация отменена - введенный x не взаимно простой с 16637! "
                     "Измените входные параметры.";
        return 1;
    }

    int l = 10;
    std::ofstream output_file;
    output_file.open(f);
    output_file << x % UPPER_BOUND << ',';
    for (int i = 0; i < n; i++)
    {
        std::string z_seq = "";
        for (int j = 0; j < l; j++)
        {
            x = module_power(x, 2, prime_number);
            int cur_z = x % 2;
            z_seq.append(std::to_string(cur_z));
        }
        output_file << std::stoi(z_seq, nullptr, 2) % UPPER_BOUND << ',';
        show_progress(i, n);
    }
    output_file.close();
    return 0;
}


void parse_args(std::string &g, std::vector<int> &init, int &n, std::string &f,
                bool &h, std::vector<std::string> &str_init, int arg_amount,
                char** args)
```

```
{
    // функция для парсинга параметров
    std::vector<std::pair<std::string, int>> default_args = {{"/g:", 0},
        {"/i:", 1}, {"/n:", 2}, {"/f:", 3}, {"/h", 4}};
    int default_args_amount = default_args.size();
    f = "rnd.dat";
    n = 10000;
    std::string pre_init;
    try
    {
        for (int i = 1; i < arg_amount; ++i)
        {
            std::string cur_arg = args[i];
            for (int j = 0; j < default_args_amount; j++)
            {
                if (cur_arg.find(default_args[j].first) != std::string::npos)
                {
                    switch (default_args[j].second)
                    {
                    case 0:
                        cur_arg.erase(0, 3);
                        g = cur_arg;
                        break;
                    case 1:
                        cur_arg.erase(0, 3);
                        pre_init = cur_arg;
                        break;
                    case 2:
                        cur_arg.erase(0, 3);
                        n = std::stoi(cur_arg);
                        break;
                    case 3:
                        cur_arg.erase(0, 3);
                        f = cur_arg;
                        break;
                    case 4:
                        h = true;
                        break;
                    }
                }
            }
        }
        if (g == "lfsr" || g == "nfsr")
        {
            str_init = split_to_str(pre_init, ",");
        }
        else
        {
```

```
            init = split_to_int(pre_init, ",");
        }
    }
    catch(const std::exception& e)
    {
        show_help_message(g, h);
    }
}


int main(int argc, char** argv)
{
    setlocale(LC_ALL, "Russian");
    std::string g;
    std::vector<int> init;
    std::vector<std::string> str_init;
    int n;
    std::string f;
    bool h = false;

    parse_args(g, init, n, f, h, str_init, argc, argv);

    g = find_generate_method(g);

    if (g != "" && (!init.empty() || !str_init.empty()))
    {
        int generation_status = 0;
        if (h)
        {
            show_help_message(g, h);
        }
        std::cout << "Генерация выполнена на ";
        if (g == "lc")
        {
            generation_status = linear_congruent_method(n, init, f);
        }
        else if (g == "add")
        {
            generation_status = additive_method(n, init, f);
        }
        else if (g == "lfsr")
        {
            generation_status = lfsr_method(n, str_init, f);
        }
        else if (g == "5p")
        {
            generation_status = five_param_method(n, init, f);
        }
```

```
        else if (g == "nfsr")
        {
            generation_status = nfsr_method(n, str_init, f);
        }
        else if (g == "mt")
        {
            generation_status = mt_method(n, init, f);
        }
        else if (g == "rc4")
        {
            generation_status = rc4_method(n, init, f);
        }
        else if (g == "rsa")
        {
            generation_status = rsa_method(n, init, f);
        }
        else if (g == "bbs")
        {
            generation_status = bbs_method(n, init, f);
        }
        if (!generation_status)
        {
            std::cout << "\r" << "Генерация выполнена на " << 100 << "%" << std::flush
                << std::endl;
            std::cout << "Результат работы генератора сохранен в " + f + "\n";
        }
    }
    else
    {
        show_help_message(g, h);
    }
}
```

ПРИЛОЖЕНИЕ Б
Код задания 2

```
import math
import os
import sys
import textwrap
import argparse


class ArgumentParser(argparse.ArgumentParser):
    def __init__(self, *args, **kwargs):
        super(ArgumentParser, self).__init__(*args, **kwargs)
        self.program = {key: kwargs[key] for key in kwargs }
        self.options = []

    def add_argument(self, *args, **kwargs):
        super(ArgumentParser, self).add_argument(*args, **kwargs)
        option = {}
        option["flags"] = [item for item in args]
        for key in kwargs:
            option[key] = kwargs[key]
        self.options.append(option)

    def print_help(self):
        wrapper = textwrap.TextWrapper(width=100)
        if "usage" in self.program:
            print("Usage: %s" % self.program["usage"])
        else:
            usage = []
            for option in self.options:
                usage += ["[%s %s]" % (item, option["metavar"])
                            if "metavar" in option
                            else "[%s %s]" % (item, option["dest"].upper())
                                 if "dest" in option
                                 else " [%s]" % item for item in option["flags"]]
            wrapper.initial_indent = "Usage: %s" % os.path.basename(sys.argv[0])
            wrapper.subsequent_indent = len(wrapper.initial_indent) * " "
            output = str.join(" ", usage)
            output = wrapper.fill(output)
            print(output)
        print()

        if "description" in self.program:
            print(self.program["description"])
            print()

        print("Аргументы:")
        maxlen = 0
```

```
        for option in self.options:
            option["flags2"] = str.join(", ", [" %s %s" % (item, option["metavar"])
                                               if "metavar" in option
                                               else "%s %s" % (item, option["dest"].upper())
                                                    if "dest" in option
                                                    else item for item in option["flags"] ])
            if len(option["flags2"]) > maxlen:
                maxlen = len(option["flags2"])
        for option in self.options:
            template = "  %-" + str(maxlen) + "s  "
            wrapper.initial_indent = template % option["flags2"]
            wrapper.subsequent_indent = len(wrapper.initial_indent) * " "
            if "help" in option and "default" in option:
                output = option["help"]
                output += " (по умолчанию: '%s')" % option["default"] if isinstance(option["default"]
                output = wrapper.fill(output)
            elif "help" in option:
                output = option["help"]
                output = wrapper.fill(output)
            elif "default" in option:
                output = "По умолчанию: '%s'" % option["default"] if isinstance(option["default"], s
                output = wrapper.fill(output)
            else:
                output = wrapper.initial_indent
            print(output)


def standard_uniform_distribution_with_interval(seq, m, a, b):
    return [b * (elem / m) + a for elem in seq]


def triangle_distribution(seq, m, a, b):
    new_seq = []
    for i in range(0, len(seq) - 1, 2):
        new_seq.append(a + b * ((seq[i] / m) + (seq[i + 1] / m) - 1))
    return new_seq


def general_exponential_distribution(seq, m, a, b):
    return [-b * math.log(elem / m) + a for elem in seq]


def normal_distribution(seq, m, a, b):
    new_seq = []
    for i in range(0, len(seq) - 1, 2):
        new_seq.append(a + b *
                       math.sqrt(-2 * math.log(1 - (seq[i] / m))) *
                       math.cos(2 * math.pi * (seq[i + 1] / m)))
```

```
            new_seq.append(a + b *
                           math.sqrt(-2 * math.log(1 - (seq[i] / m))) *
                           math.sin(2 * math.pi * (seq[i + 1] / m)))
    return new_seq


def gamma_distribution(seq, m, a, b, c):
    new_seq = []
    for i in range(0, len(seq), c):
        mult_values = 1
        for j in range(c):
            mult_values *= 1 - (seq[i + j] / m)
        new_seq.append(a - b * math.log(mult_values))
    return new_seq


def lognormal_distribution(seq, m, a, b):
    norm_seq = normal_distribution(seq, m, a, b)
    return [a + math.exp(b - elem) for elem in norm_seq]


def logistic_distribution(seq, m, a, b):
    return [a + b * math.log((elem / m) / (1 - elem / m)) for elem in seq]


def binom_coef(n, k):
    if k > n:
        return 0
    if not k or n == k:
        return 1
    return binom_coef(n - 1, k - 1) + binom_coef(n - 1, k)


def binomial_distribution(seq, m, a, b):
    new_seq = []
    for i in range(len(seq)):
        u = seq[i] / m
        s = 0
        k = 0
        while True:
            s += binom_coef(b, k) * pow(a, k) * pow(1 - a, b - k)
            if s > u:
                new_seq.append(k)
                break
            if k < (b - 1):
                k += 1
                continue
            new_seq.append(b)
```

```
    return new_seq


METHOD_NAME_TO_FUNC = {"st": standard_uniform_distribution_with_interval,
                       "tr": triangle_distribution,
                       "ex": general_exponential_distribution,
                       "nr": normal_distribution,
                       "ln": lognormal_distribution,
                       "ls": logistic_distribution,
                       "bi": binomial_distribution}


def main(args):
    # получение числовой последовательности из файла
    file_name = args.fn
    with open(file_name) as f:
        lines = f.readlines()
    file_content = "".join(lines)
    num_seq_str = file_content.split(",")
    num_seq = [int(num) for num in num_seq_str if len(num)]

    # получение максимального элемента последовательности
    max_elem = max(num_seq)
    m = max_elem + 1
    method_name = args.dist

    # преобразование к заданному распределению
    if method_name != "gm":
        processed_seq = METHOD_NAME_TO_FUNC[method_name](num_seq, m,
                                                         args.p1, args.p2)
    else:
        processed_seq = gamma_distribution(num_seq, m, args.p1, args.p2,
                                           args.p3)

    f = open(f"distr-{method_name}.dat", "w")
    f.write(",".join([str(elem) for elem in processed_seq]))
    f.close()


if __name__ == "__main__":
    parser = ArgumentParser(description="Программа приводит последовательность чисел из входного фай
    parser.add_argument("-f", "--fn", type=str, help="имя файла с входной последовательностью")
    parser.add_argument("-d", "--dist", type=str, help="код распределения для преобразования последо
    parser.add_argument("-p1", "-a", type=float, help="1-й параметр для генерации ПСЧ заданного распр
    parser.add_argument("-p2", "-b", type=int, help="2-й параметр для генерации ПСЧ заданного распре
    parser.add_argument("-p3", "-c", type=int, default=None, help="3-й параметр для генерации ПСЧ га
    parser.add_argument("-h", "--help", action="help", help="выводит текст помощи")
    args = parser.parse_args()
```

```
main(args)
```