Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Теоретической и прикладной математики

Лабораторная работа № 5

по дисциплине «ОСНОВЫ ТЕОРИИ ИНФОРМАЦИИИ КРИПТОГРАФИИ»

ГЕНЕРИРОВАНИЕ РАВНОМЕРНО РАСПРЕДЕЛЕННЫХ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ

Факультет:	ПМИ
Группа:	ПМИ-02
Вариант:	7
Студент:	Сидоров Даниил,
	Дюков Богдан
Преподаватель:	Авдеенко Татьяна Владимировна,
	Сивак Мария Алексеевна.

Новосибирск

2026

1. Цель работы

Освоить основные алгоритмы программной генерации равномерно распределенных псевдослучайных последовательностей (РРПП).

2. Задача

Часть 1

I. Реализовать приложение, удовлетворяющее следующим требованиям.

1. Во входном файле хранятся параметры генератора и прочие входные данные, необходимые для работы программы.
2. Программа генерирует псевдослучайную равномерно распределенную последовательность с помощью заданного в варианте генератора.
3. Для сгенерированной последовательности программа вычисляет период.
4. Программа проводит проверку качества сгенерированной последовательности по критерию χ2-Пирсона.
5. Сгенерированная последовательность, а также данные о найденном периоде и о результате проверки по критерию χ2-Пирсона сохраняются в выходной файл (или файлы).

II. С помощью реализованного приложения выполнить следующие задания.

1. Протестировать правильность работы приложения при различных параметрах генератора.
2. Подобрать параметры генератора так, чтобы период генерируемой последовательности был максимальным.
3. С помощью критерия χ2-Пирсона проверить сгенерированную последовательность на равномерность.
4. Сделать выводы о проделанной работе

Вариант	Генератор	Уточнение задания
7	LFSR	Рассмотреть генераторы, основанные на многочленах: $f(x) = x^6 + x^3 + 1$, $f(x) = x^6 + x^2 + x + 1$, $f(x) = x^6 + x + 1$

3. Метод решения задачи

В данной работе реализуется программа, генерирующая псевдослучайную равномерно распределенную последовательность с помощью регистра сдвига с линейной обратной связью (LFSR или РСЛОС). РСЛОС задаётся характеристическим многочленом степени **L**.

В регистре сдвига с линейной обратной связью выделяют две части:

- собственно регистр сдвига;
- схему обратной связи, вычисляющую значение вдвигаемого бита.

Регистр состоит из функциональных ячеек памяти, в каждой из которых хранится текущее состояние (значение) одного бита. Количество **L**, называют длиной регистра. Биты (ячейки) обычно нумеруются числами i = 0, 1, … , **L**-1.

Функцией обратной связи для РСЛОС является линейная булева функция от значений всех или некоторых битов регистра. Сложение по модулю 2 является линейной булевой функцией и применяется в наших регистрах. При этом биты, являющиеся переменными функции обратной связи, называются **отводами**, а сам регистр называется **конфигурацией Фибоначчи**.

Формирование последовательности выглядит следующим образом:

1) Задается начальное состояние регистра и из многочлена находятся номера битов отвода;
2) Читается бит, расположенный в ячейке **L**-1; этот бит является очередным битом выходной последовательности;
3) С помощью функции обратной связи (операция XOR для битов отвода) вычисляется новое значение для ячейки 0, используя текущие значения ячеек;
4) Содержимое каждой ячейки регистра перемещается в следующую ячейку;
5) В ячейку 0 записывается бит, вычисленный в 3 пункте функцией обратной связи.

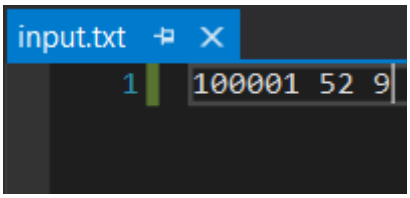
Пункты 2-5 повторяются до тех пор, пока очередное состояние регистра не вернется к исходному, а выходная последовательность формируется в порядке их генерации в РСЛОС.

Схема регистра, которая задаётся характеристическим многочленом $f(x) = x^6 + x^3 + 1$:

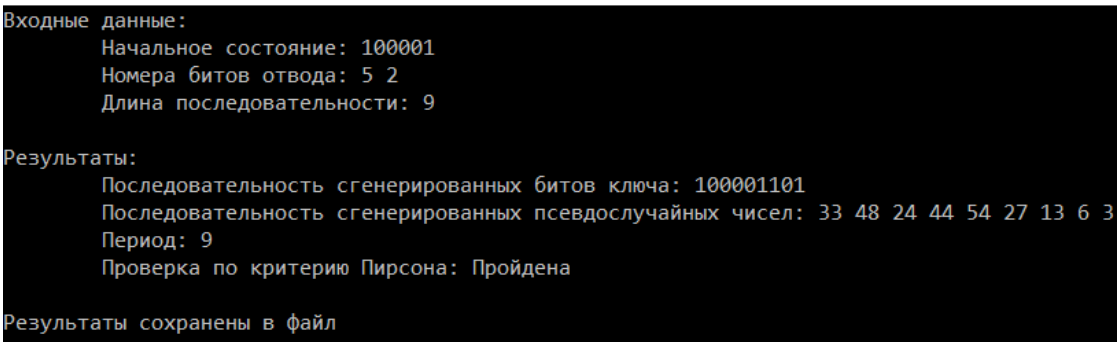
1. Разработанное программное средство

Разработанное программное средство представляет собой консольное приложение на языке C#.

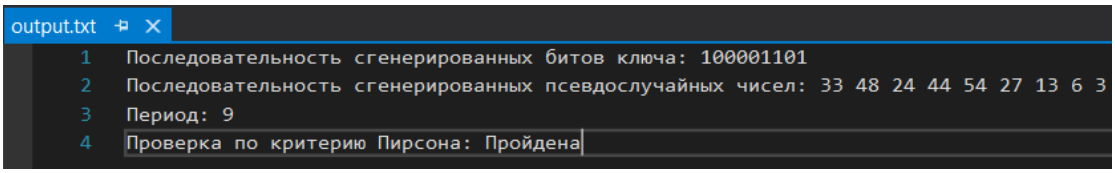
У пользователя есть возможность ввести во входном файле черед пробел начальное состояние регистра, номера битов отвода и длину сгенерированной последовательности:



Сгенерируем псевдослучайную равномерно распределенную последовательность для данных входных параметров:



Содержимое файла с результатами:



В файл с результатами записываются последовательность битов и псевдослучайных чисел до первого повтора состояния регистра с исходным, период и проверка по критерию Пирсона.

2. Исследования

Определение понятий:

- **Период регистра сдвига** - минимальная длина получаемой последовательности до начала её повторения. РСЛОС длины L имеет 2^L начальных состояний, задающих значения бит в ячейках. Из них $2^L - 1$

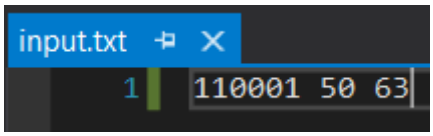
состояний - ненулевые, так что генерируемая последовательность имеет период $T \leq 2^L - 1$.

- **Критерий согласия χ2-Пирсона** используется для проверки гипотезы о том, что сгенерированная последовательность подчиняется равномерному распределению. Для этого вычисляют статистику критерия по формуле:

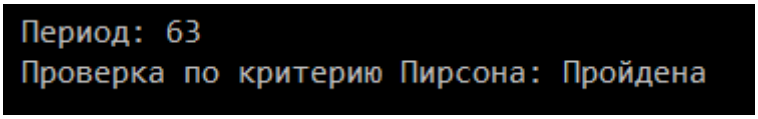
$$s^* = N \sum_{i=1}^{K} \frac{\left(\frac{v_i}{N} - P_i\right)^2}{P_i}$$

Если вычисленная статистика S* больше критического значения статистики Sкр при заданных степени свободы (r = K – 1) и уровне значимости (α = 0.05), то гипотеза отвергается, иначе нет оснований для отклонения проверяемой гипотезы.

Например, для следующих входных данных:



Проведенные исследования:



3. Код программы

```
using System;

using System.IO;

using System.Text;

using System.Threading.Tasks;

using System.Linq;


namespace LFSR

{
```

```
    class Program

    {



        // Получить период полученной последовательности битов

        private static int GetPeriod(string Value)

        {

            return Value.Length;

        }



        // Сгенерировать последовательность битов с помощью регистра сдвига с линейной
обратной связью

        private static string LFSR(uint startState, int[] feedbackPoints, int
startStateLength, ref string numbers)

        {

            string bitsResult = ""; // Строка с результирующей последовательностью битов



            uint nextState = startState; // Текущее состояние регистра



            int lastBit = startStateLength - 1; // Запоминаем номер первого бита



            while (true)

            {

                // Запоминаем последовательность чисел, полученных при переводе текущего

                // состояния регистра из 2й в 10ю систему

                numbers += Convert.ToString(nextState);

                numbers += " ";



                // Получаем крайний бит (он справа): 1001 -> 1 и сохраняем его в результат

                bitsResult += GetBit(nextState, 0);



                // Начало операции XOR, берем один из битов отвода

                int xor = GetBit(nextState, feedbackPoints[0]);



                // Продолжаем XOR с остальными отводами
```

```
                for (int i = 1; i < feedbackPoints.Length; i++)
                {
                    xor ^= GetBit(nextState, feedbackPoints[i]);
                }

                // Сдвигаем все биты вправо на 1 позицию
                nextState >>= 1;

                // В ячейку 0 записываем бит, ранее вычисленный функцией обратной связи XOR
                // Если xor равен 0, то нету смысла что-либо делать, там и так 0 записан
                if (xor == 1)
                {
                    nextState |= 1u << lastBit;
                }

                // Сверяем исходное состояние с текущим, если они совпадают, то завершаем генерация последовательности битов
                if (nextState == startState)
                {
                    return bitsResult;
                }

            }

        }

        // Получить значение конкретного бита битовой последовательности
        private static int GetBit(uint Value, int number)
        {
            return Convert.ToInt32((Value & 1 << number) != 0);
        }

        // Запись в файл
        private static async void WriteTextToFile(string text)
```

```
        {


            using (FileStream fstream = new
FileStream(@"C:\Users\Bogdan\Desktop\LFSR\LFSR\output.txt", FileMode.OpenOrCreate))

            {

                // Преобразуем строку в байты

                byte[] buffer = Encoding.Default.GetBytes(text);



                // Записываем массив байтов в файл

                await fstream.WriteAsync(buffer, 0, buffer.Length);

            }



        }



        // Чтение из файла

        private static async Task<string> ReadTextFromFile()

        {



            using (FileStream fstream =
File.OpenRead(@"C:\Users\Bogdan\Desktop\LFSR\LFSR\input.txt"))

            {

                // Выделяем массив для считывания данных из файла

                byte[] buffer = new byte[fstream.Length];



                // Считываем данные

                await fstream.ReadAsync(buffer, 0, buffer.Length);



                // Декодируем байты в строку

                string textFromFile = Encoding.Default.GetString(buffer);



                textFromFile = textFromFile.Substring(3, textFromFile.Length - 3);



                return textFromFile;

            }
```

```
        }

        // Проверка качества сгенерированной последовательности с помощью критерия согласия Пирсона
        private static bool PirsonCriteria(int[] numbers, double[] xi2)
        {
            int N = numbers.Length;
            int M = numbers.Max();
            int K = M < 10 ? M : (int)(5 * Math.Log10(N));

            double[] interval = new double[K + 1];

            for(int i = 0; i < K; i++)
            {
                interval[i] = (double)N / K * i;
            }

            interval[K] = N;

            double S = 0.0;
            double Pi = 1 / (double)K;

            for(int i = 0; i < K; i++)
            {
                int Vi = 0;

                for(int j = 0; j < K; j++)
                {

                    if(numbers[j] < interval[i + 1] && numbers[j] >= interval[i])
                    {
                        Vi++;
                    }
```

```
                }

                S += Math.Pow((double)Vi / K - Pi, 2) / Pi;
            }

            S *= K;

            if(S > xi2[K - 1])
            {
                return false;
            }
            else
            {
                return true;
            }

        }

        static void Main(string[] args)
        {
            // Критические значения статистики Skp при заданных степени свободы
            double[] xi2 = { 3.841, 5.991, 7.815, 9.488, 11.070, 12.592,
                14.067, 15.507, 16.919, 18.307, 19.675, 21.026, 22.362, 23.685, 24.996,
26.296 };

            string text = ReadTextFromFile().Result;

            // Поделили входные данные
            string[] words = text.Split(' ');

            // Проверили входные данные на корректность задания
            if (words.Length != 3 || !int.TryParse(words[0], out var number1)
```

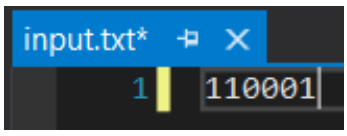
```
               || !int.TryParse(words[1], out var number2) || !int.TryParse(words[2], out var
number3) || number1 == 0)

            {

                Console.WriteLine("Ошибка! Неверно заданы входные данные.");

                Console.ReadKey();

                return;

            }



            int[] array = new int[words[1].Length];



            for (int i = 0; i < words[1].Length; i++)

            {

                array[i] = Math.Abs(Convert.ToInt32(words[1][i].ToString()) - words[0].Length
+ 1);

            }



            string numbersResult = "";

            var lfsr = LFSR(Convert.ToUInt32(words[0], 2), array, words[0].Length, ref
numbersResult);



            int[] numbers = numbersResult.Remove(numbersResult.Length - 1, 1).Split('
').Select(x => int.Parse(x)).ToArray();



            Console.WriteLine("Входные данные: ");

            Console.WriteLine("\tНачальное состояние: " + words[0]);

            Console.Write("\tНомера битов отвода: ");



            for (int i = 0; i < words[1].Length; i++)

                Console.Write(words[1][i] + " ");



            Console.WriteLine("\n\tДлина последовательности: " + words[2]);



            Console.WriteLine("\nРезультаты: ");

            Console.Write("\tПоследовательность сгенерированных битов ключа: ");
```

```
            for (int i = 0; i < Convert.ToInt32(words[2].ToString()); i++)
            {
                Console.Write(lfsr[i % lfsr.Length]);

                if (i % lfsr.Length == lfsr.Length - 1)
                    Console.Write(" ");
            }

            Console.Write("\n\tПоследовательность сгенерированных псевдослучайных чисел: ");

            for (int i = 0; i < Convert.ToInt32(words[2].ToString()); i++)
            {
                Console.Write(numbers[i % lfsr.Length] + " ");

                if (i % lfsr.Length == lfsr.Length - 1)
                    Console.Write(" ");
            }

            var pirson = PirsonCriteria(numbers, xi2) ? "Пройдена" : "Не пройдена";

            Console.WriteLine("\n\tПериод: " + GetPeriod(lfsr));
            Console.WriteLine("\tПроверка по критерию Пирсона: " + pirson);

            WriteTextToFile("Последовательность сгенерированных битов ключа: " + lfsr
                 + "\nПоследовательность сгенерированных псевдослучайных чисел: " +
numbersResult
                 + "\nПериод: " + GetPeriod(lfsr) + "\nПроверка по критерию Пирсона: " +
pirson);

            Console.WriteLine("\n\nРезультаты сохранены в файл");
            Console.ReadKey();
        }
    }
}
```

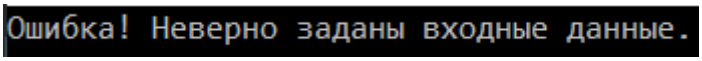
4. Тесты

Простейший тест был продемонстрирован ранее в разделе “Разработанное программное средство”.

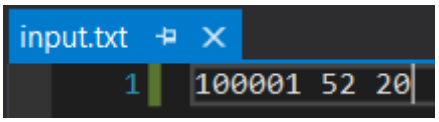
1) Рассмотрим ситуации с ошибочным вводом. Если на вход подаются следующие входные данные:



То результатом работы программы будет:

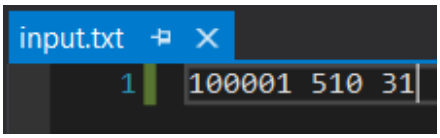


2) Проведем тесты при различных заданиях РСЛОС.
Для многочлена $f(x) = x^6 + x^3 + 1$ имеем:




```
Входные данные:
        Начальное состояние: 100001
        Номера битов отвода: 5 2
        Длина последовательности: 20

Результаты:
        Последовательность сгенерированных битов ключа: 100001101 100001101 10
        Последовательность сгенерированных псевдослучайных чисел: 33 48 24 44 54 27 13 6 3  33 48 24 44 54 27 13 6 3  33
 48
        Период: 9
        Проверка по критерию Пирсона: Пройдена


Результаты сохранены в файл
```

Для многочлена $f(x) = x^6 + x^2 + x + 1$ имеем:




```
Входные данные:
        Начальное состояние: 100001
        Номера битов отвода: 5 1 0
        Длина последовательности: 31

Результаты:
        Последовательность сгенерированных битов ключа: 1000010110101000111011111001001
        Последовательность сгенерированных псевдослучайных чисел: 33 16 40 52 26 45 22 43 21 10 5 34 49 56 28 46 55 59 6
1 62 31 15 39 19 9 36 50 25 12 6 3
        Период: 31
        Проверка по критерию Пирсона: Пройдена


Результаты сохранены в файл
```

Для многочлена $f(x) = x^6 + x + 1$ имеем:


```
Входные данные:
        Начальное состояние: 100001
        Номера битов отвода: 5 0
        Длина последовательности: 63

Результаты:
        Последовательность сгенерированных битов ключа: 100001000001111110101011001101110110100100111000101111001010001

        Последовательность сгенерированных псевдослучайных чисел: 33 16 8 4 2 1 32 48 56 60 62 63 31 47 23 43 21 42 53 2
6 13 38 51 25 44 54 59 29 46 55 27 45 22 11 37 18 9 36 50 57 28 14 7 35 17 40 52 58 61 30 15 39 19 41 20 10 5 34 49 24 1
2 6 3
        Период: 63
        Проверка по критерию Пирсона: Пройдена


Результаты сохранены в файл
```

3) Подберем параметры генератора так, чтобы период генерируемой последовательности был максимальным.

   Для того чтобы конкретный LFSR имел максимальный период (то есть $T = 2^L - 1$, где L – степень характеристического многочлена), соответствующий многочлен должен быть примитивным. В общем случае не существует простого способа нахождения примитивных многочленов данной степени, проще выбирать многочлен случайным образом и проверять, является ли он примитивным. Ещё один метод заключается в использовании готовых таблиц, в которых приведены номера отводных последовательностей, обеспечивающих максимальный период генератора.

   Многочлен $f(x) = x^6 + x + 1$, исходя из прошлого теста, является примитивным и дает максимальный период генерируемой последовательности, равный $2^6 - 1 = 63$

5. Вывод

В ходе проведения лабораторной работы мы освоили основные алгоритмы программной генерации равномерно распределенных псевдослучайных последовательностей, а именно генерацию случайных чисел на основе регистров сдвига с линейной обратной связью.

Использование LFSR для создания потоковых криптосистем предполагает уязвимость, связанную с линейным характером генерируемой последовательности. Для защиты от атак следует увеличивать размер используемого LFSR или использовать более сложные схемы генерации ключа.