Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Кузбасский государственный технический университет имени Т. Ф. Горбачева»

Кафедра информационных и автоматизированных производственных систем

**Денис Евгеньевич Турчин**

# ТЕОРИЯ ИНФОРМАЦИИ

## *Лабораторный практикум*

## Электронное учебное пособие

Кемерово 2016



**Вперед→**

**УДК 621.715.2(075.8)(086.76)**

Рецензент(ы) Чичерин Иван Владимирович – кандидат технических наук, доцент, председатель учебно-методической комиссии направления 09.03.02 «Информационные системы и технологии» КузГТУ

**Турчин Денис Евгеньевич**

**Теория информации. Лабораторный практикум**: электронное учебное пособие для студентов направления подготовки 09.03.02. «Информационные системы и технологии» / Д. Е. Турчин; КузГТУ. – Электрон. дан. – Кемерово, 2016. – 1 электрон. опт. диск (2,6 Мб).

В данных методических указаниях изложены содержание лабораторных работ, порядок их выполнения, а также контрольные вопросы к ним.

**Текстовое (символьное) электронное издание**

| | |
|---|---|
| Минимальные системные требования: | Частота процессора не менее 1,0 ГГц; ОЗУ 512 Мб; 20 Гб HDD; операционная система Windows XP; CD-ROM 4-скоростной; ПО для чтения файлов PDF-формата; SVGA-совместимая видеокарта; мышь. |



Вперед→

| | |
|---|---|
| Сведения о программном обеспечении, которое использовано для создания электронного издания | MS Word |
| Сведения о технической подготовке материалов для электронного издания | Редактор З. М. Савина |
| Объем издания в единицах измерения объема носителя, занятого цифровой информацией (байт, Кб, Мб) | 2,6 Мб |
| Наименование и контактные данные юридического лица, осуществившего запись на материальный носитель | Федеральное государственное бюджетное образовательное учреждение высшего образования «Кузбасский государственный технический университет имени Т. Ф. Горбачева»<br>650000, Кемерово, ул. Весенняя, 28<br>Тел./факс: 8(3842) 58-35-84 |

Вперед→

**СОДЕРЖАНИЕ**

ПРЕДИСЛОВИЕ ..... 5
ЛАБОРАТОРНЫЕ РАБОТЫ ..... 7
1. Основы моделирования случайных величин и процессов . 7
1.1. Цель и задачи работы ..... 7
1.2. Основные теоретические сведения ..... 7
1.3. Порядок выполнения работы и варианты заданий ..... 27
1.4. Контрольные вопросы и задачи ..... 32
2. Моделирование и расчет характеристик дискретных источников сообщений ..... 33
2.1. Цель и задачи работы ..... 33
2.2. Основные теоретические сведения ..... 33
2.3. Порядок выполнения работы и варианты заданий ..... 51
2.4. Контрольные вопросы и задачи ..... 54
3. Моделирование и расчет характеристик дискретных каналов связи ..... 56
3.1. Цель и задачи работы ..... 56
3.2. Основные теоретические сведения ..... 56
3.3. Порядок выполнения работы и варианты заданий ..... 72
3.4. Контрольные вопросы и задачи ..... 74
4. Сжатие информации методом арифметического кодирования ..... 76
4.1. Цель и задачи работы ..... 76
4.2.Основные теоретические сведения ..... 76
4.3. Порядок выполнения работы и варианты заданий ..... 90
4.4. Контрольные вопросы и задачи ..... 91
5. Сжатие информации с помощью алгоритмов Лемпела-Зива ..... 93
5.1. Цель и задачи работы ..... 93
5.2. Основные теоретические сведения ..... 93
5.3. Порядок выполнения работы и варианты заданий ... 104
5.4. Контрольные вопросы и задачи ..... 105
6. Основы построения и декодирования линейных блоковых кодов ..... 107
6.1. Цель и задачи работы ..... 107
6.2. Основные теоретические сведения ..... 107

6.3. Порядок выполнения работы и варианты заданий ...122
6.4.Контрольные вопросы и задачи .....124
7. Построение и декодирование циклических кодов .....126
7.1. Цель и задачи работы .....126
7.2. Основные теоретические сведения.....126
7.3. Порядок выполнения работы и варианты заданий ...141
7.4. Контрольные вопросы и задачи .....143
8. Основы защиты информации с помощью симметричных алгоритмов шифрования.....145
8.1. Цель и задачи работы .....145
8.2. Основные теоретические сведения.....145
8.3. Порядок выполнения работы и варианты заданий ...163
8.4. Контрольные вопросы и задачи .....164
РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА.....166
ПРИЛОЖЕНИЕ.....168
П.1. Функции плотности распределения случайных величин .....168
П.2. Неприводимые полиномы над полем GF(2) .....169

## ПРЕДИСЛОВИЕ

Пособие предназначено для студентов направления подготовки бакалавра 09.03.02. «Информационные системы и технологии», изучающих дисциплину «Теория информации».

Лабораторный практикум состоит из восьми работ, посвященных решению различных задач теории информации, а также близких к ней теории кодирования и криптографии. Первая работа связана с изучением простейших методов моделирования случайных величин и процессов. Работы 2 и 3 посвящены моделированию и расчету характеристик дискретных источников сообщений и каналов связи. Работы 4 и 5 связаны с изучением методов сжатия информации. В работах 6 и 7 рассматриваются методы помехоустойчивого кодирования. Работа 8 связана с изучением симметричных алгоритмов шифрования и их реализацией на платформе .NET Framework.

Программная реализация различных алгоритмов в лабораторных работах осуществляется с помощью языка программирования C# и среды разработки MS Visual Studio. Простые расчеты, а также построение графиков и диаграмм производятся с помощью MS Excel.

Выписка из ФГОС ВПО по направлению 09.03.02. и рабочей программы дисциплины

| Код | Компетенции, формируемые при освоении дисциплины | Результаты освоения |
|---|---|---|
| Б2.В.2 | • **ОК-6** владение широкой общей подготовкой (базовыми знаниями) для решения практических задач в области информационных систем и технологий;<br>• **ОК-10** готовность использовать основные законы естественнонаучных дисциплин в профессиональной деятельности, применять методы матема- | ***Знать***:<br>• основные понятия и теоремы теории информации;<br>• модели и информационные характеристики источников сообщений и каналов связи;<br>• основные методы сжатия информации;<br>• основные виды помехоустойчивых кодов, методы их построения и декодирования; |

| Код | Компетенции, формируемые при освоении дисциплины | Результаты освоения |
|---|---|---|
| | тического анализа и моделирования, теоретического и экспериментального исследования;<br>• **ПК-12** способность разрабатывать средства реализации информационных технологий (методические, информационные, математические, алгоритмические, технические и программные);<br>• **ПК-26** готовность использовать математические методы обработки, анализа и синтеза результатов профессиональных исследований;<br>• **ПК-33** готовность обеспечивать безопасность и целостность данных информационных систем и технологий. | • основные понятия криптографии и методы шифрования информации.<br>***Уметь***:<br>• определять количество информации в сообщениях;<br>• рассчитывать информационные характеристики источников сообщений и каналов связи;<br>• использовать методы сжатия информации;<br>• использовать методы помехоустойчивого кодирования информации;<br>• шифровать информацию с помощью симметричных алгоритмов. |

# ЛАБОРАТОРНЫЕ РАБОТЫ

# 1. ОСНОВЫ МОДЕЛИРОВАНИЯ СЛУЧАЙНЫХ ВЕЛИЧИН И ПРОЦЕССОВ

## 1.1. Цель и задачи работы

Цель работы – приобрести умение моделировать случайные величины и процессы для решения задач теории информации.

Основные задачи работы:

• освоить моделирование дискретных случайных величин на основе метода суперпозиции;

• научиться моделировать марковские процессы с дискретными состояниями (цепи Маркова);

Работа рассчитана на 4 часа.

## 1.2. Основные теоретические сведения

### 1.2.1. Случайные величины и их моделирование. Метод суперпозиции

***Дискретные и непрерывные случайные величины.***

Теория информации является наукой, которая в значительной степени основывается на использовании понятий и методов теория вероятностей. В частности в теории информации для описания источников сообщений и каналов связи применяются случайные величины и процессы.

Под ***случайной величиной*** понимают величину, которая в результате опыта может принять какое-либо значение, причем неизвестно заранее, какое именно. Множество всех значений, которые случайная величина может принимать, называют ***множеством возможных значений*** этой величины.

Для исследования вероятностных свойств случайной величины необходимо знать правило, позволяющее находить вероятность того, что случайная величина примет значение из подмно-

жества ее значений. Любое такое правило называют ***законом распределения*** случайной величины. Общим законом распределения, присущим всем случайным величинам, является функция распределения.

***Функцией распределения*** (***вероятностей***) случайной величины $X$ называют функцию $F(x)$, значение которой в точке $x$ равно вероятности того, что $X < x$:

$$F(x) = P\{X < x\}.$$

Случайную величину $X$ называют ***дискретной***, если множество ее возможных значений конечно или счетно.

Для дискретной случайной величины $X$ функция распределения может быть задана с помощью ***ансамбля*** (***ряда распределения***), представляющего перечисление возможных значений $x_1$, $x_2$, … , $x_n$ величины $X$ и соответствующих им вероятностей $p_1$, $p_2$, … , $p_n$. Ансамбль величины $X$ записывается следующим образом:

$$X = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix}; \qquad \sum_{i=1}^{n} p_i = 1. \tag{1.1}$$

Случайную величину $X$ называют ***непрерывной***, если ее возможные значения непрерывно заполняют некоторый промежуток.

Для непрерывной случайной величины $X$ функция распределения $F(x)$ задается следующим образом:

$$F(x) = \int_{-\infty}^{x} f(x)dx. \tag{1.2}$$

где $f(x)$ – функция ***плотности распределения*** (***вероятностей***) величины $X$.

Функцию распределения $F(x)$ в выражении (1.2) называют ***интегральным законом распределения*** случайной величины $X$, а плотность распределения $f(x)$ – ***дифференциальным законом распределения*** той же случайной величины.

### *Моделирование случайных величин. Метод суперпозиции.*

Под ***моделированием случайной величины X*** с заданной функцией распределения $F(x)$ понимают процесс получения на компьютере ее выборочных значений $x_1$, $x_2$, … , $x_n$. Исходными

данными для моделирования служат равномерно распределенные на интервале (0, 1) случайные числа $r_1$, $r_2$, … , $r_n$, вырабатываемые генератором случайных чисел.

Для моделирования дискретных случайных величин используют ***методом суперпозиции***, который заключается в следующем.

Пусть дискретная случайная величина $X$ задается ансамблем (1.1). Отрезок [0, 1] разбивают на $n$ последовательных отрезков $\Delta_1$, $\Delta_2$, … , $\Delta_n$, длины которых соответственно равны вероятностям $p_1$, $p_2$, … , $p_n$. Разыгрывается значение величины $r \in [0, 1]$ с равномерным распределением и далее принимается

$$X = x_i, \text{ если } r \in \Delta_i.$$

❒ ***Пример* 1.1.** ***Моделирование дискретной случайной величины.***

Требуется разработать программу, которая обеспечивает моделирование дискретной случайной величины (ДСВ) $X$, описываемой ансамблем:

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ 0{,}20 & 0{,}30 & 0{,}45 & 0{,}05 \end{pmatrix}.$$

Результат работы программы должен представлять последовательность значений случайной величины $X$. Кроме того, программа должна выводить частоты появления каждого из значений ДСВ.

Программу реализуем в виде консольного приложения на языке C#. Для получения значений ДСВ добавим в приложение класс **DiscrRnd**, который представляет генератор значений величины $X$. В данном классе будут следующие открытые методы:

• **string GenerValue()** – возвращает строковое значение величины $X$, получаемое с помощью метода суперпозиции;

• **string GetFreqs()** – возвращает строку с частотами появления значений величины $X$.

Исходный код класса **DiscrRnd** представлен в листинге 1.1.

Листинг 1.1. Исходный код класса **DiscrRnd** (Генератор значений дискретной случайной величины)

```
    /// <summary>
    /// Представляет генератор значений дискретной случайной
    /// величины (ДСВ) с заданным ансамблем
    /// </summary>
    class DiscrRnd
    {
        int n;      // Общее число возможных значений ДСВ
        string[] x; // Массив значений ДСВ
        double[] p; // Массив вероятностей появления значений
        int[] f;    // Массив частот появления значений
        Random rnd; // Случайная переменная

        /// <summary>
        /// Конструктор с параметрами по умолчанию
        /// </summary>
        public DiscrRnd()
        {
            n = 4;
            x = new string[n];
            p = new double[n];
            f = new int[n];
            rnd = new Random();
            x[0] = "x1"; x[1] = "x2"; x[2] = "x3"; x[3] = "x4";
            p[0] = 0.20; p[1] = 0.30; p[2] = 0.45; p[3] = 0.05;
        }

        /// <summary>
        /// Возвращает случайное значение ДСВ
        /// </summary>
        /// <returns>Случайное значение ДСВ</returns>
        public string GenerValue()
        {
            string y = ""; // Значение на выходе генератора
            double r = rnd.NextDouble(); // Случайное число от 0 до 1
            double q = 0.0; // Нижняя граница интервала вероятностей
            for (int i = 0; i < n; i++)
            {
                if (r > q && r <= q + p[i])
                { y = x[i]; f[i]++; break; }
                q += p[i];
            }
            return y;
        }

        /// <summary>
        /// Возвращает строку с частотами появления значений ДСВ
        /// </summary>
        /// <returns>Частоты появления значений ДСВ</returns>
        public string GetFreqs()
        {
            string buf = "\nЧастоты появления значений:\n";
            for (int i = 0; i < n; i++)
            {
                buf += string.Format("x{0} - {1}\n", i + 1, f[i]);
            }
            return buf;
        }
    }
```

Исходный код метода **Main()** консольного приложения приведен в листинге 1.2.

Листинг 1.2. Исходный код метода **Main()** консольного приложения

```
10      class Program
11      {
12          static void Main(string[] args)
13          {
14              Console.Title = "Моделирование дискретной случайной величины";
15              DiscrRnd dRnd = new DiscrRnd(); // Экземпляр класса Генератор
16
17              Console.WriteLine("Введите число генерируемых значений:");
18              string buf = Console.ReadLine(); // Строка-буфер
19              int n = int.Parse(buf); // Число генерируемых значений
20              buf = "";
21
22              for (int i = 0; i < n; i++)
23              { buf += dRnd.GenerValue(); }
24
25              Console.WriteLine("\nПоследовательность значений:\n" + buf);
26              Console.WriteLine(dRnd.GetFreqs());
27
28              Console.Read();
29          }
```

Результат работы консольного приложения для введенного числа значений $N = 500$ показан на рис. 1.1.

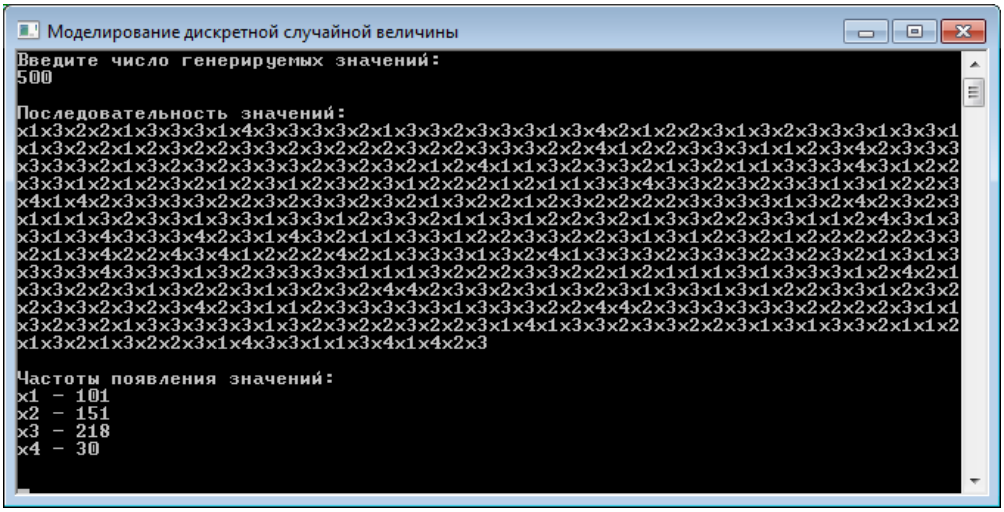


Рис. 1.1. Результат работы консольного приложения

Вычислим эмпирические вероятности появления значений случайной величины и сравним их с вероятностями ансамбля:

$p^*(x_1) = 101 / 500 = 0{,}202;$ $\quad p(x_1) = 0{,}2;$

$p^*(x_2) = 151 / 500 = 0{,}302;$ $\quad p(x_2) = 0{,}3;$

$p^*(x_3) = 218 / 500 = 0{,}436;$ $\quad p(x_3) = 0{,}45;$

$p^*(x_4) = 30 / 500 = 0{,}06;$ $\quad p(x_4) = 0{,}05.$

Таким образом, полученные значения эмпирических вероятностей является достаточно близкими к вероятностям заданного ансамбля. ❐

### 1.2.2. Случайные процессы. Понятие о цепях Маркова

***Случайные процессы.***

Обобщением случайных величин являются случайные процессы (случайные функции).

***Случайный* (*стохастический*) *процесс*** – это процесс (т. е. изменение во времени состояния некоторой системы), течение которого зависит от случая и для которого определена вероятность того или иного его течения. Чаще всего под случайным процессом понимают некоторую случайную величину $X(t)$, меняющуюся с течением времени $t$.

При описании случайных процессов используются понятия сечения и реализации случайного процесса.

Случайная величина $X(t_0)$, в которую обращается случайный процесс при $t = t_0$, называется ***сечением случайного процесса***, соответствующим данному значению аргумента $t$.

***Реализацией* (*траекторией*) *случайного процесса*** называется неслучайная функция $x(t)$, в которую превращается случайный процесс $X(t)$ в результате опыта (рис. 1.2). Например, записывая температуру воздуха в зависимости от времени в течение суток можно получить реализацию случайного процесса.

Любой случайный процесс можно рассматривать как совокупность всех его сечений или всех его реализаций.

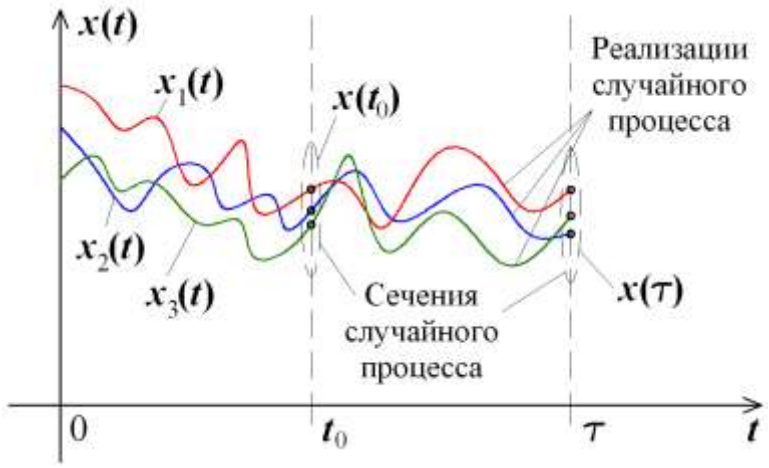


Рис. 1.2. Пример сечений и реализаций случайного процесса

Случайные процессы классифицируют по разным признакам, основными из которых являются классификации по времени $T$ и по состояниям $S$. В зависимости от множества $T$ значений времени $t$, в которые возможны переходы системы из состояния в состояние, а также множества S самих состояний все случайные процессы можно разделить на четыре класса:

• процессы с дискретными состояниями и дискретным временем;

• процессы с дискретными состояниями и непрерывным временем;

• процессы с непрерывными состояниями и дискретным временем;

• процессы с непрерывными состояниями и непрерывным временем.

Простейшим видом случайных процессов являются марковские случайные процессы, широко используемые в теории информации для описания источников сообщений и каналов связи.

***Марковский процесс***[1] – это случайный процесс, эволюция которого после любого заданного значения временного параметра $t$ не зависит от эволюции, предшествовавшей $t$, при условии, что значение процесса в этот момент фиксировано. То есть в марковском процессе будущее состояние зависит только от настоящего состояния и не зависит от «предыстории» процесса.

[1] Название дано в честь русского математика А.А. Маркова (1856 – 1922 гг.), разработавшего теорию данного класса случайных процессов.

***Понятие о цепях Маркова.***

Важным классом марковских процессов являются процессы с дискретными состояниями, также называемые цепями Маркова.

***Цепью Маркова*** называется случайный процесс, протекающий в системе с множеством состояний $S = \{s_1, s_2, \dots, s_m\}$ и обладающий следующим свойством: для каждого момента времени $t_0$ вероятность любого состояния системы в будущем ($t > t_0$) зависит только от ее состояния в настоящем ($t = t_0$) и не зависит от того когда и каким образом система пришла в это состояние.

Для анализа цепей Маркова используют ориентированные графы, называемые ***графами состояний*** (рис. 1.3). Вершинами такого графа являются состояния системы, а дугами – возможные переходы из одного состояния в другое.

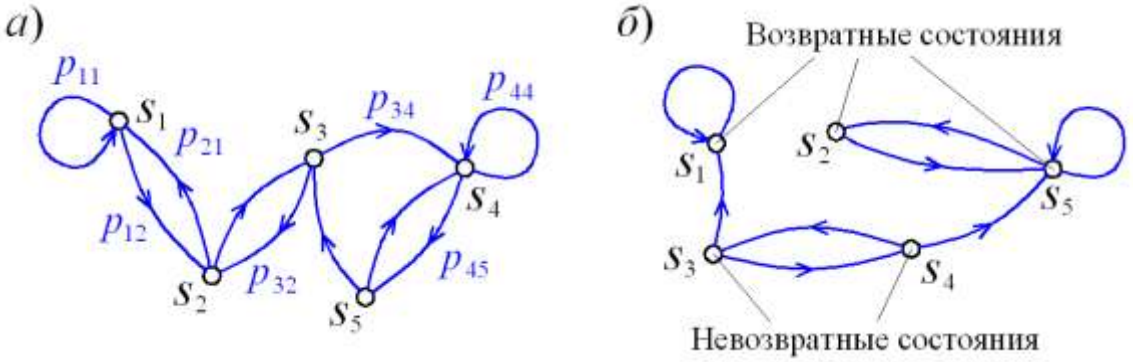


Рис. 1.3. Графы состояния марковских процессов: *а*) – эргодического; *б*) – неэргодического

Каждой дуге ($s_i$, $s_j$) графа состояний приписывается условная вероятность $p_{ij} = p(s_j \mid s_i)$ того, что система перейдет в состояние $s_j$ при условии, что ее текущим состоянием является $s_i$. Указанная вероятность $p_{ij}$ называется ***переходной вероятностью***.

Цепь Маркова может быть описана с помощью ***матрицы переходных вероятностей***, которая содержит все переходные вероятности системы и имеет следующий вид:

$$P_m = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix};$$

Сумма элементов всех элементов любой строки матрицы $P_m$ должна быть равна 1, поскольку текущее состояние системы должно обязательно перейти в какое-либо состояние:

$$\sum_{j=1}^{m} p_{ij} = \sum_{j=1}^{m} p(s_j \mid s_i) = 1 \qquad (i = 1, 2, \ldots, m).$$

Состояние $s_i$ марковкого процесса называется ***возвратным***, если вероятность возвращения в него через произвольный промежуток времени равна 1, и ***невозвратным***, если эта вероятность меньше 1.

Для теории информации практический интерес представляют только такие марковские процессы, которые являются эргодическими (обладают свойством эргодичности).

Марковский процесс называется ***эргодическим***, если система, в которой он протекает, может из каждого своего состояния перейти в любое другое состояние (непосредственно или через определенное число промежуточных состояний) (рис. 1.3 *а*). В эргодическом марковском процессе не могут присутствовать невозвратные состояния. Марковские процессы, не обладающие таким свойством, называются ***неэргодическими*** (рис. 1.3 *б*).

Для эргодического марковского процесса с течением времени (спустя большое число переходов $k$) система приходит к предельному стационарному распределению $p^*(s_1), p^*(s_2), \ldots, p^*(s_m)$, не зависящему от начального состояния:

$$\lim_{k \to \infty} \frac{k(s_j)}{k} = p^*(s_j); \;\; j = 1, 2, \ldots, m; \qquad (1.3)$$

где $k(s_j)$ – число переходов системы в состояние $s_j$ за общее число переходов $k$.

❒ ***Пример* 1.2. *Моделирование цепи Маркова.***

Требуется разработать компьютерную модель цепи Маркова с состояниями $S = \{s_1, s_2, s_3, s_4\}$ и матрицей переходных вероятностей:

| ***i, j*** | $s_1$ | $s_2$ | $s_3$ | $s_4$ |
|---|---|---|---|---|
| $s_1$ | 0,11 | 0,36 | 0,19 | 0,34 |
| $s_2$ | 0,27 | 0 | 0,45 | 0,28 |
| $s_3$ | 0,35 | 0,29 | 0,05 | 0,31 |
| $s_4$ | 0,23 | 0,48 | 0,29 | 0 |

В результате моделирования необходимо получить стационарное распределение вероятностей появления состояний $s_1$, $s_2$, $s_3$, $s_4$. Для этого требуется найти частоты появления состояний при заданном числе шагов моделирования.

Программу для моделирования создадим в форме консольного приложения на языке C#. Модель цепи Маркова реализуем в форме класса **MarkovChain**, в который будут добавлены следующие открытые методы:

• **string GenerState()**, который случайным образом генерирует и возвращает состояние на основе матрицы переходных вероятностей и предыдущего состояния;

• **string GetFreqs()**, который возвращает строку с частотами появления символов.

Исходный код класса **MarkovChain** представлен в листинге 1.3.

В методе **GenerState()** класса **MarkovChain** производится проверка принадлежности случайного числа **x** определенному интервалу переходных вероятностей. Этот интервал выбирается как строка матрицы **p[m,m]**, которая имеет номер равный значению **k** (индекс предыдущего переданного символа).

Исходный код метода **Main()** консольного приложения будет аналогичен коду из листинга 1.2.

Результат работы консольного приложения при заданной длине исходного сообщения $N = 50000$ символов показан на рис. 1.4.

Листинг 1.3. Исходный код класса **MarkovChain**

```
/// <summary>
/// Представляет цепи Маркова
/// </summary>
class MarkovChain
{
    const int m = 4; // Число состояний марковской цепи
    string[] s;      // Массив состояний
    double[,] p;     // Матрица переходных вероятностей
    int[] f;         // Массив частот появления состояний
    Random r;        // Случайное поле
    int k;           // Индекс текущего состояния

    /// <summary>
    /// Конструктор с параметрами по умолчанию
    /// </summary>
    public MarkovChain()
    {
        s = new string[m] { "s1", "s2", "s3", "s4" };
        p = new double[m, m] {{0.11, 0.36, 0.19, 0.34},
                              {0.27, 0.00, 0.45, 0.28},
                              {0.35, 0.29, 0.05, 0.31},
                              {0.23, 0.48, 0.29, 0.00}};
        f = new int[m];
        r = new Random();
        k = r.Next(0, m - 1);
    }

    /// <summary>
    /// Генерирует случайным образом состояние цепи и возвращает его
    /// </summary>
    /// <returns>Состояние</returns>
    public string GenerState()
    {
        string st = ""; // Текущее состояние цепи
        double x = r.NextDouble();
        double q = 0.0;  // Кумулятивная вероятность
        for (int i = 0; i < m; i++)
        {
            // Проверяем принадлежность x определенному
            // интервалу переходных вероятностей
            if ((x > q) && (x <= q + p[k, i]))
            {
                st = s[i]; k = i; f[i]++;
                break;
            }
            q += p[k, i];
        }
        return st;
    }

    /// <summary>
    /// Возвращает строку с частотами появления состояний
    /// </summary>
    /// <returns>Частоты появления состояний</returns>
    public string GetFreqs()
    {
        string buf = "\nЧастоты появления состояний:\n";
        for (int i = 0; i < m; i++)
        {
            buf += string.Format("s{0} - {1}\n", i + 1, f[i]);
        }
        return buf;
    }
}
```

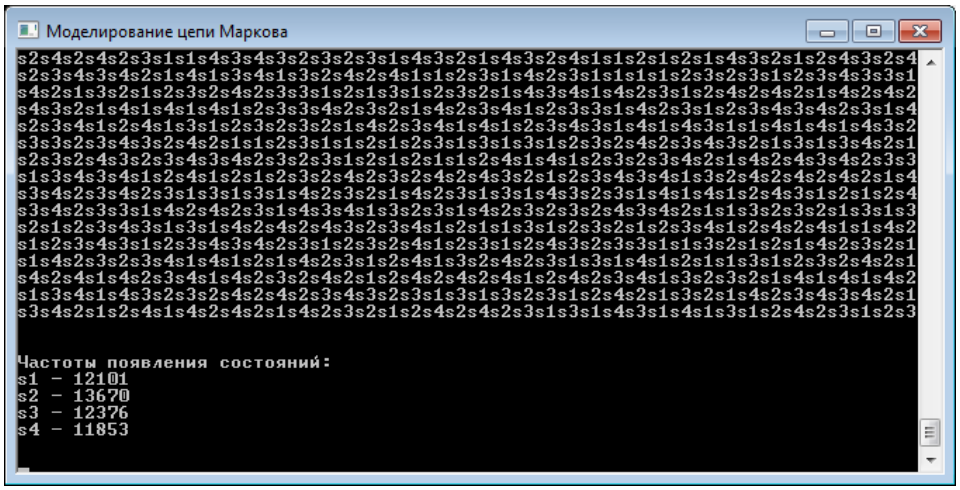


Рис. 1.4. Результат работы консольного приложения

Стационарное распределение вероятностей появления состояний будет следующим:

$p^*(s_1) = 12101 / 50000 = 0{,}242;$

$p^*(s_2) = 13670 / 50000 = 0{,}273;$

$p^*(s_3) = 12376 / 50000 = 0{,}248;$

$p^*(s_4) = 11853 / 50000 = 0{,}237;$

$\sum_{i=1}^{4} p^*(s_i) = 0{,}242 + 0{,}273 + 0{,}248 + 0{,}237 = 1.$ ❐

### 1.2.3. Метод обратных функций. Интеграция приложения на языке C# с MS Excel

***Метод обратных функций.***

Для непрерывных случайных величин наиболее эффективным аналитическим методом моделирования является ***метод обратных функций*** (метод монотонного нелинейного преобразования), который заключается в следующем.

Пусть для непрерывной случайной величины $X$ задана функция плотности распределения $f(x)$. Задача заключается в поиске требуемой функции $x = w(r)$, где $r$ – случайная величина с равномерным распределением.

В методе обратных функций качестве функции $w(r)$ выступает функция $F^{-1}(r)$ обратная интегральному закону распределения $F(x)$. Для большинства известных законов распределения задача получения функции $F^{-1}(r)$ может быть решена аналитически.

Наглядно процесс получения функции $F^{-1}(r)$ представлен на рис. 1.5, где изображены графики для произвольной плотности распределения $f(x)$, соответствующей ей функции $F(r)$, а также обратной функции $F^{-1}(r)$.

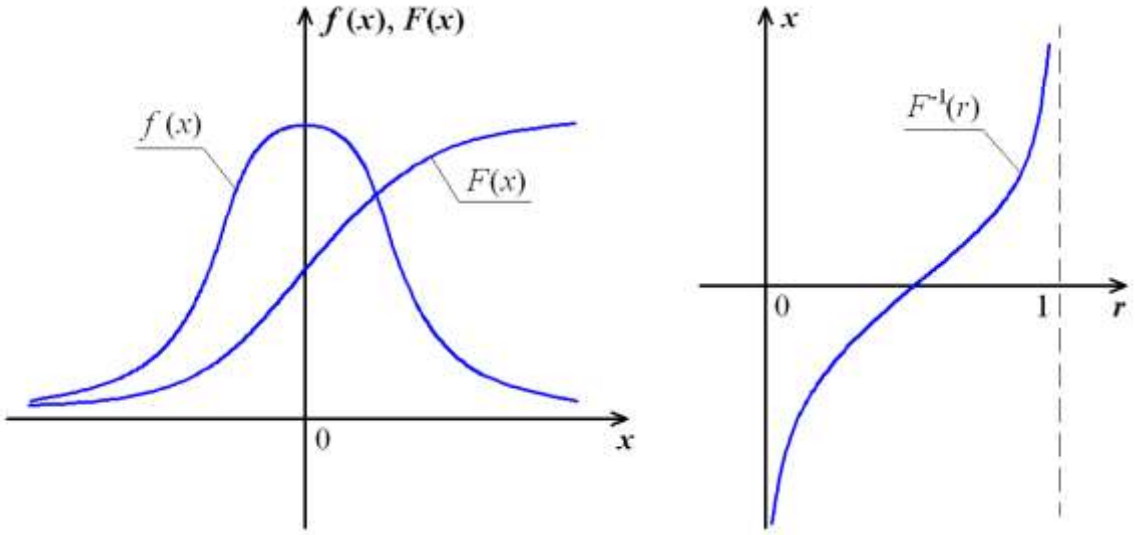


Рис. 1.5. Плотность распределения $f(x)$ и соответствующая ей функция $x = F^{-1}(r)$

Наиболее широко используемые функции плотности распределения $f(x)$ и соответствующие им функции $F^{-1}(x)$ приведены в приложении (табл. П.1).

У случайной величины $X$ с нормальным законом распределения функция $F^{-1}(r)$ не выражается через элементарные функции. В подобных случаях используются различные приближенные методы.

Набор значений величины $x$ с функцией распределения близкой к нормальной можно получить путем суммирования $n$ случайных чисел $r_i$ с равномерным законом распределения в интервале (0, 1):

$$x_j = \sum_{i=1}^{12} r_i - 6. \qquad (1.4)$$

❐ ***Пример* 1.3. *Моделирование непрерывной случайной величины*.**

Требуется разработать программу для моделирования непрерывной случайной величины (НСВ) с логарифмически-нормальным законом распределения ($b$ = 1, $c$ = 0.5). Результатом работы программы должен быть набор значений НСВ, а также частоты попадания этих значений в интервалы, на которые разделен диапазон значений НСВ.

Программа будет реализована в виде консольного приложения на языке C#. Для получения значений НСВ добавим в приложение класс **ContinRnd**, который представляет специализированный генератор случайных чисел. В этом классе создадим открытый метод **double GenerLogNormValue()**, который возвращает значение НСВ, получаемое с помощью метода обратных функций.

Исходный код класса **ContinRnd** представлен в листинге 1.4. Полями данного класса являются **xMin** и **xMax** – наименьшее и наибольшее значения случайной величины, а также **n** – число интервалов разбиения диапазона значений НСВ.

Для инициализации экземпляров класса используются два конструктора: с параметрами по умолчанию, с параметрами пользователя.

Для доступа к закрытому массиву частот **f** из других классов в класс **ContinRnd** добавлен соответствующий индексатор. Подсчет частот попадания в интервалы выполняется с помощью вспомогательного метода **CountFreqs()**.

Листинг 1.4. Исходный код класса **ContinRnd** (Генератор значений непрерывной случайной величины). Часть 1

```
    /// <summary>
    /// Представляет генератор значений непрерывной случайной
    /// величины (НСВ) с определенным законом распределения
    /// </summary>
    class ContinRnd
    {
        double xMin, xMax; // Наименьшее и наибольшее значение НСВ
        int n;             // Число интервалов разбиения диапазона значений НСВ
        int[] f;           // Массив частот попадания значений НСВ в интервалы
        Random rnd;        // Случайная переменная
        double dx;         // Ширина одного интервала

        /// <summary>
        /// Конструктор с параметрами по умолчанию
        /// </summary>
        public ContinRnd()
        {
            xMin = -5; this.xMax = 5;
            n = 10;
            f = new int[n];
            rnd = new Random();
            dx = (xMax - xMin) / n;
        }

        /// <summary>
        /// Конструктор с параметрами пользователя
        /// </summary>
        /// <param name="_xMin">Минимальное значение</param>
        /// <param name="_xMax">Максимальное значение</param>
        /// <param name="_n">Число интервалов</param>
        public ContinRnd(double _xMin, double _xMax, int _n)
        {
            xMin = _xMin; xMax = _xMax;
            n = _n;
            f = new int[_n];
            rnd = new Random();
            dx = (_xMax - _xMin) / _n;
        }

```

Листинг 1.4. Исходный код класса **ContinRnd** (Генератор значений непрерывной случайной величины). Часть 2

```
        /// <summary>
        /// Индексатор по массиву частот
        /// </summary>
        /// <param name="i">Индекс элемента в массиве</param>
        /// <returns>Элемент с заданным индексом</returns>
        public int this[int i]
        {
            get
            {
                if (i >= 0 && i < n) return (f[i]);
                else return (f[0]);
            }
            set
            {
                if (i >= 0 && i < n) f[i] = value;
            }
        }

        /// <summary>
        /// Возвращает случайное значение НСВ с логарифмически-нормальным
        /// законом распределения
        /// </summary>
        /// <param name="_b">Коэффициент b</param>
        /// <param name="_c">Коэффициент c</param>
        /// <returns>Случайное число</returns>
        public double GenerLogNormValue(double _b, double _c)
        {
            double t = 0;
            int m = 12; // Число суммируемых элементов
            for (int i = 0; i < m; i++)
            { t += rnd.NextDouble(); }
            t -= m / 2;
            // Обратная функция преобразования
            double x = Math.Exp(_c * t + _b);
            CountFreqs(x);
            return x;
        }

        /// <summary>
        /// Подсчитывает частоту попадания значения НСВ
        /// в определенный интервал
        /// </summary>
        /// <param name="_x">Случайное число</param>
        private void CountFreqs(double _x)
        {
            for (int i = 0; i < n; i++)
            {
                if ((_x > xMin + i * dx) && (_x < xMin + (i + 1) * dx))
                { f[i]++; break; }
            }
        }
    }
```

Исходный код класса **Program** консольного приложения приведен в листинге 1.5.

Листинг 1.5. Исходный код метода **Main()** консольного приложения

```
using System.Threading.Tasks;
using XL = Microsoft.Office.Interop.Excel;

namespace RandVarModel
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Title = "Моделирование непрерывной случайной величины";

            double xMin = -24;
            double xMax = 2;
            int n = 15;
            ContinRnd cRnd = new ContinRnd(xMin, xMax, n);

            Console.Write("Введите число генерируемых значений:\n");
            string buf = Console.ReadLine(); // Строка-буфер
            int m = int.Parse(buf);
            buf = "";

            double b = 1;   // Параметры логарифмически-нормального
            double c = 0.5; // распределения

            for (int i = 0; i < m; i++)
            { buf += String.Format("{0}. {1:f5}\n", i, cRnd.GenerLogNormValue(b, c)); }

            Console.WriteLine("\nПоследовательность значений:\n" + buf);

            int[] f = new int[n]; // Массив частот
            Console.WriteLine("\nЧастоты попадания в интервалы:");

            // Цикл для заполнения массива частот и их вывода на консоль
            for (int i = 0; i < n; i++)
            {
                f[i] = cRnd[i];
                Console.WriteLine("Интервал {0}: {1}", i, f[i]);
            }

            DrawGistogram(f); // Вызов метода построения гистограммы

            Console.Read();
        }
```

Результат работы консольного приложения для числа значений $m$ = 1000 показан на рис. 1.6. ❐

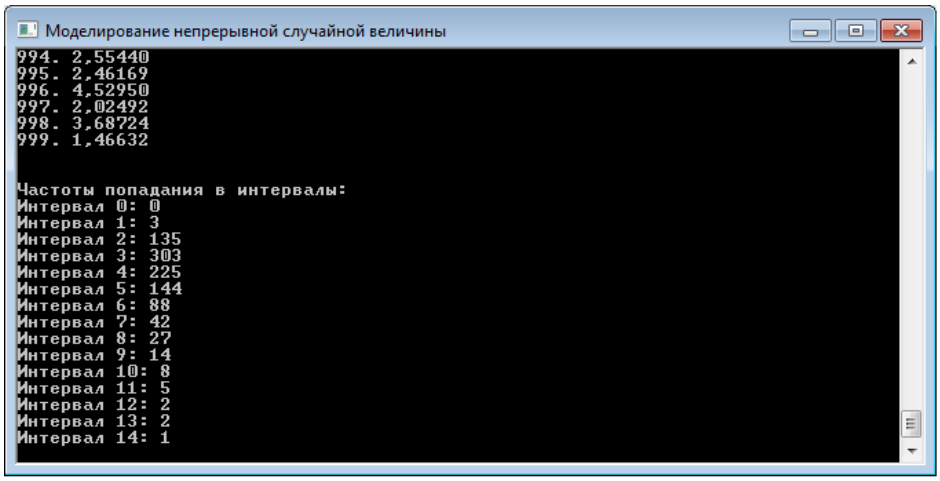


Рис. 1.6. Результат работы консольного приложения

***Интеграция приложения на C# с MS Excel. Построение диаграмм.***

Пакет приложений Microsoft Office может являться сервером OLE-объектов и его функции могут использоваться различными приложениями. В частности возможности программы на C# можно расширить путем использования средств MS Office Excel, связанных с построением диаграмм.

Чтобы добавить в текущий проект приложения возможности MS Excel, следует подключить библиотеку типов Excel. Для этого в пункте меню **Project** следует выбрать команду **Add Reference**. Затем, если на компьютере установлен пакет MS Office 2010, следует в окне **Менеджера ссылок** на вкладке **COM** выбрать ссылку на библиотеку **Microsoft Excel 14.0 Object Library** (рис. 1.X). Подключить новую библиотеку типов в текущий проект можно также через контекстное меню окна **Solution Explorer** (Обозреватель решений), щелкнув на пункте **Add Reference**.

Объектная модель MS Office Excel предоставляет работу со следующими наиболее важными классами, размещенными в пространстве имен **Microsoft.Office.Interop.Excel**:

- **Application** – представляет приложение Excel;
- **Workbook** – представляет одну рабочую книгу в приложении Excel;
- **Worksheet** – представляет отдельный лист рабочей книги;
- **Range** – представляет диапазон ячеек на листе.

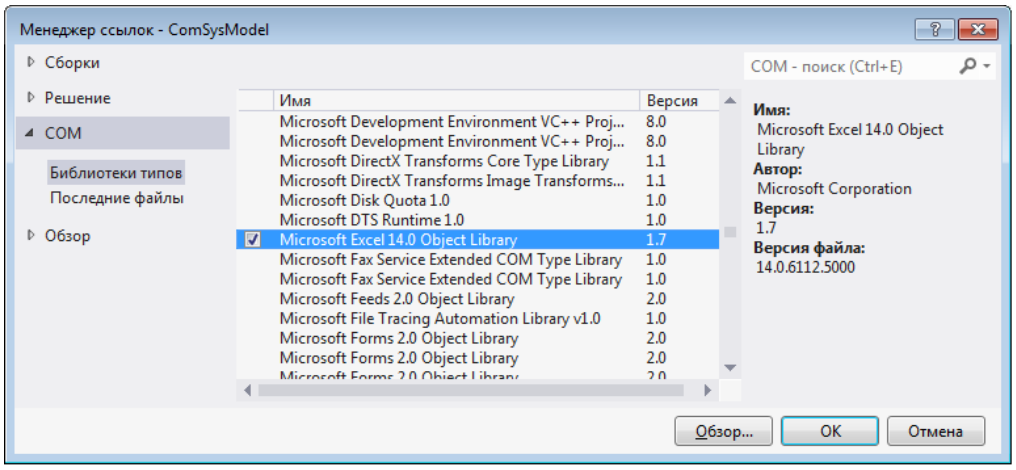


Рис. 1.7. Окно **Менеджер ссылок**

Для работы с диаграммами в приложении Excel используется коллекция **Charts**. Добавление новой диаграммы в эту коллекцию осуществляется методом **Add()**.

Настройка выбранной диаграммы осуществляется через класс **ActiveChart**, который позволяет задать тип диаграммы, оси, легенду и другие параметры.

❒ ***Пример* 1.4. *Построение диаграммы средствами MS Excel*.**

Для консольного приложения, полученного в примере 1.3, требуется разработать метод **DrawGistogram**, который передает данные о частотах в MS Excel для построения гистограммы распределения значений.

Метод **DrawGistogram** должен быть добавлен в класс **Program**, а его вызов будет выполняться из метода **Main()**. В качестве параметра, передаваемого в метод **DrawGistogram**, будет выступать целочисленный массив частот **_f**.

Для использования в проекте на C# типов библиотеки **Microsoft Excel Object Library** добавим соответствующую ссылку, а также импортируем пространство имен:

```
using XL = Microsoft.Office.Interop.Excel;
```

В качестве типа диаграммы используем **xlColumnClustered** (гистограмма). В создаваемой диаграмме должен присутствовать

заголовок, а также подписи горизонтальной и вертикальной осей («Интервалы» и «Частоты»).

Исходный код метода построения гистограммы приведен в листинге 1.6.

Листинг 1.6. Исходный код метода построения гистограммы в MS Excel

```
/// <summary>
/// Метод построения гистограммы распределения в MS Excel
/// </summary>
/// <param name="_f">Массив частот</param>
static void DrawGistogram(int[] _f)
{
    // Создание приложения Excel
    XL.Application xL1 = new XL.Application();
    xL1.Workbooks.Add(); // Добавление рабочей книги Excel
    xL1.ActiveSheet.Range["A1"].Value = "Частоты";

    string adr = ""; // Адрес ячейки в рабочей книге Excel
    int i = 2; // Номер строки в книге Excel

    // Цикл для заполнения 1-го столбца значениями частот
    foreach (int fi in _f)
    {
        adr = string.Format("A{0}", i++);
        xL1.ActiveSheet.Range[adr].Value = fi;
    }

    xL1.Charts.Add(); // Добавление диаграммы
    // Задание типа диаграммы (гистограмма)
    xL1.ActiveChart.ChartType = XL.XlChartType.xlColumnClustered;
    // Отключение легенды и задание заголовка диаграммы
    xL1.ActiveChart.HasLegend = false;
    xL1.ActiveChart.HasTitle = true;
    xL1.ActiveChart.ChartTitle.Characters.Text = "Гистограмма распределения значений";
    // Подпись оси X
    xL1.ActiveChart.Axes(XL.XlAxisType.xlCategory).HasTitle = true;
    xL1.ActiveChart.Axes(XL.XlAxisType.xlCategory).AxisTitle.Characters.Text = "Интервалы";
    // Подпись оси Y
    xL1.ActiveChart.Axes(XL.XlAxisType.xlValue).HasTitle = true;
    xL1.ActiveChart.Axes(XL.XlAxisType.xlValue).AxisTitle.Characters.Text = "Частоты";
    // Вывод диаграммы
    xL1.Visible = true;
}
```

Гистограмма, соответствующая результатам из примера 1.3, показана на рис. 1.8. ❐

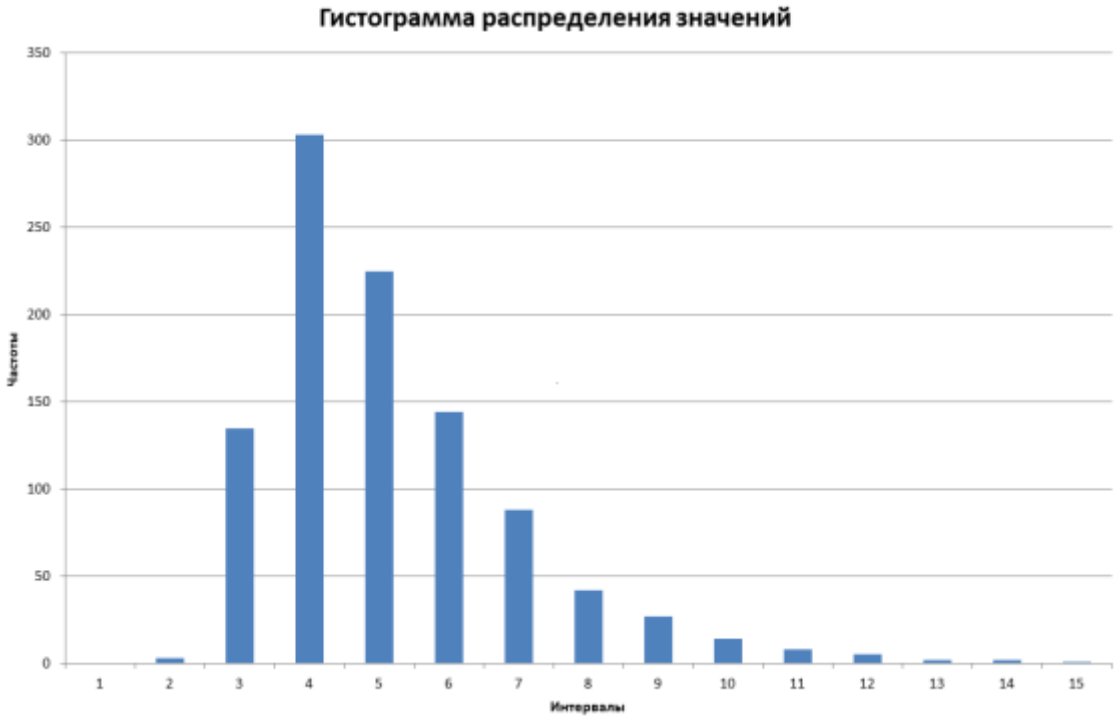


Рис. 1.8. Полученная гистограмма

## 1.3. Порядок выполнения работы и варианты заданий

***Основные этапы выполнения работы.***

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Смоделировать дискретную случайную величину (ДСВ) с заданным ансамблем (табл. 1.1). Для этого требуется разработать консольное приложение на языке C#, которое позволяет получить последовательность значений ДСВ и определить частоту появления каждого из этих значений. На основе полученных частот требуется вычислить эмпирические вероятности и сравнить их с заданными вероятностями.
3. Смоделировать цепь Маркова, заданную с помощью матрицы переходных вероятностей (табл. 1.2). Для этого необходимо разработать консольное приложение на языке C#, генерирующее последовательность состояний, через которые проходит моделируемая система. С помощью модели требуется произвести вычислительный эксперимент для нахождения стационарных веро-

ятностей. Для этого в приложении требуется задать достаточно большое число шагов моделирования (например, несколько десятков тысяч).

4. Дополнительно смоделировать непрерывную случайную величину (НСВ) с заданной функцией плотности распределения (табл. 1.3). Для этого необходимо разработать консольное приложение на языке C#, которое позволяет получить последовательность значений НСВ. Кроме того, программа должна определять частоты попадания значений в интервалы, на которые разбивается диапазон значений НСВ.

5. Частоты попадания значений НСВ в интервалы, полученные в п. 4, должны автоматически передаваться в MS Excel для построения гистограммы. Для этого в консольное приложение требуется добавить соответствующий метод. Полученную гистограмму требуется сравнить с функцией плотности вероятности.

6. Оформить и защитить отчет по лабораторной работе.

***Индивидуальные варианты заданий.***

Таблица 1.1

Вероятности появления значений $x_1$, $x_2$, … , $x_8$ дискретной случайной величины $X$

| № вар. | Вероятности для значений $x_1, x_2, \dots, x_8$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ |
| **1** | 0,05 | 0,16 | 0,21 | 0,11 | 0,1 | 0,07 | 0,1 | 0,2 |
| **2** | 0,1 | 0,25 | 0,09 | 0,1 | 0,17 | 0,06 | 0,11 | 0,12 |
| **3** | 0,08 | 0,2 | 0,07 | 0,13 | 0,14 | 0,09 | 0,05 | 0,24 |
| **4** | 0,21 | 0,12 | 0,05 | 0,08 | 0,03 | 0,14 | 0,2 | 0,17 |
| **5** | 0,13 | 0,04 | 0,19 | 0,16 | 0,18 | 0,12 | 0,13 | 0,05 |
| **6** | 0,2 | 0,15 | 0,16 | 0,04 | 0,19 | 0,05 | 0,12 | 0,09 |
| **7** | 0,14 | 0,1 | 0,04 | 0,17 | 0,12 | 0,16 | 0,05 | 0,22 |
| **8** | 0,06 | 0,21 | 0,03 | 0,23 | 0,13 | 0,12 | 0,14 | 0,08 |
| **9** | 0,08 | 0,19 | 0,22 | 0,2 | 0,09 | 0,03 | 0,17 | 0,02 |
| **10** | 0,09 | 0,18 | 0,06 | 0,12 | 0,15 | 0,1 | 0,07 | 0,23 |
| **11** | 0,22 | 0,13 | 0,12 | 0,15 | 0,07 | 0,04 | 0,19 | 0,08 |
| **12** | 0,1 | 0,24 | 0,08 | 0,07 | 0,17 | 0,05 | 0,16 | 0,13 |
| **13** | 0,04 | 0,17 | 0,15 | 0,11 | 0,1 | 0,09 | 0,13 | 0,21 |
| **14** | 0,17 | 0,05 | 0,16 | 0,22 | 0,08 | 0,19 | 0,09 | 0,04 |

| № вар. | Вероятности для значений $x_1, x_2, \dots, x_8$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $p_1$ | $p_2$ | $p_3$ | $p_4$ | $p_5$ | $p_6$ | $p_7$ | $p_8$ |
| **15** | 0,12 | 0,09 | 0,08 | 0,14 | 0,16 | 0,21 | 0,04 | 0,16 |
| **16** | 0,23 | 0,1 | 0,07 | 0,06 | 0,05 | 0,12 | 0,22 | 0,15 |
| **17** | 0,16 | 0,08 | 0,05 | 0,18 | 0,11 | 0,15 | 0,07 | 0,2 |
| **18** | 0,24 | 0,12 | 0,13 | 0,09 | 0,14 | 0,08 | 0,15 | 0,05 |
| **19** | 0,11 | 0,04 | 0,19 | 0,16 | 0,18 | 0,12 | 0,13 | 0,07 |
| **20** | 0,09 | 0,23 | 0,05 | 0,21 | 0,15 | 0,1 | 0,12 | 0,05 |
| **21** | 0,2 | 0,15 | 0,1 | 0,14 | 0,08 | 0,06 | 0,17 | 0,1 |
| **22** | 0,15 | 0,06 | 0,16 | 0,23 | 0,07 | 0,19 | 0,08 | 0,06 |
| **23** | 0,22 | 0,12 | 0,06 | 0,07 | 0,04 | 0,13 | 0,2 | 0,16 |
| **24** | 0,13 | 0,08 | 0,1 | 0,09 | 0,21 | 0,19 | 0,05 | 0,15 |

Таблица 1.2

Матрицы переходных вероятностей $p(s_j \mid s_i)$ цепи Маркова

| № вар. | Матрица переходных вероятностей $p(s_j \mid s_i)$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $s_i \backslash s_j$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ |
| **1, 9, 17** | $s_1$ | 0,14 | 0,2 | 0,02 | 0,27 | 0,09 | 0,02 | 0,1 | 0,16 |
| | $s_2$ | 0,2 | 0,05 | 0,13 | 0,08 | 0,2 | 0,13 | 0,2 | 0,01 |
| | $s_3$ | 0,06 | 0,2 | 0,16 | 0,22 | 0,12 | 0,09 | 0,05 | 0,1 |
| | $s_4$ | 0,07 | 0,18 | 0,2 | 0,04 | 0,2 | 0,09 | 0,1 | 0,12 |
| | $s_5$ | 0,09 | 0,3 | 0,08 | 0,2 | 0,01 | 0,1 | 0,07 | 0,15 |
| | $s_6$ | 0,04 | 0,1 | 0,1 | 0,05 | 0,28 | 0,07 | 0,23 | 0,13 |
| | $s_7$ | 0,15 | 0,06 | 0,07 | 0,2 | 0,2 | 0,08 | 0,2 | 0,04 |
| | $s_8$ | 0,2 | 0,03 | 0,16 | 0,1 | 0,09 | 0,1 | 0,07 | 0,25 |
| **2, 10, 18** | $s_1$ | 0,07 | 0,1 | 0,2 | 0,08 | 0,13 | 0,08 | 0,23 | 0,11 |
| | $s_2$ | 0,13 | 0,05 | 0,06 | 0,2 | 0,19 | 0,1 | 0,03 | 0,24 |
| | $s_3$ | 0,1 | 0,04 | 0,1 | 0,15 | 0,2 | 0,09 | 0,07 | 0,25 |
| | $s_4$ | 0,07 | 0,18 | 0,2 | 0,04 | 0,2 | 0,08 | 0,1 | 0,13 |
| | $s_5$ | 0,2 | 0,06 | 0,1 | 0,08 | 0,09 | 0,1 | 0,28 | 0,09 |
| | $s_6$ | 0,06 | 0,2 | 0,04 | 0,17 | 0,09 | 0,3 | 0,03 | 0,11 |
| | $s_7$ | 0,05 | 0,08 | 0,1 | 0,27 | 0,2 | 0,08 | 0,2 | 0,02 |
| | $s_8$ | 0,09 | 0,25 | 0,14 | 0,1 | 0,06 | 0,21 | 0,1 | 0,05 |
| **3, 11, 19** | $s_1$ | 0,2 | 0,2 | 0,04 | 0,15 | 0,2 | 0,02 | 0,14 | 0,05 |
| | $s_2$ | 0,1 | 0,05 | 0,08 | 0,14 | 0,19 | 0,23 | 0,2 | 0,01 |
| | $s_3$ | 0,04 | 0,1 | 0,27 | 0,24 | 0,15 | 0,1 | 0,01 | 0,09 |
| | $s_4$ | 0,09 | 0,13 | 0,2 | 0,08 | 0,2 | 0,05 | 0,15 | 0,1 |
| | $s_5$ | 0,11 | 0,22 | 0,2 | 0,07 | 0,07 | 0,2 | 0,03 | 0,1 |
| | $s_6$ | 0,05 | 0,06 | 0,09 | 0,09 | 0,2 | 0,1 | 0,2 | 0,21 |
| | $s_7$ | 0,1 | 0,05 | 0,2 | 0,08 | 0,14 | 0,17 | 0,2 | 0,06 |

| № вар. | Матрица переходных вероятностей $p(s_j\|s_i)$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | $s_i \backslash s_j$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ |
| | $s_8$ | 0,06 | 0,04 | 0,3 | 0,1 | 0,03 | 0,2 | 0,07 | 0,2 |
| **4, 12, 20** | $s_1$ | 0,08 | 0,1 | 0,07 | 0,02 | 0,3 | 0,1 | 0,13 | 0,2 |
| | $s_2$ | 0,29 | 0,1 | 0,01 | 0,08 | 0,21 | 0,17 | 0,04 | 0,1 |
| | $s_3$ | 0,06 | 0,05 | 0,1 | 0,26 | 0,1 | 0,08 | 0,07 | 0,28 |
| | $s_4$ | 0,14 | 0,09 | 0,3 | 0,02 | 0,2 | 0,09 | 0,1 | 0,06 |
| | $s_5$ | 0,2 | 0,2 | 0,07 | 0,07 | 0,01 | 0,1 | 0,2 | 0,15 |
| | $s_6$ | 0,15 | 0,2 | 0,01 | 0,06 | 0,11 | 0,2 | 0,09 | 0,18 |
| | $s_7$ | 0,02 | 0,1 | 0,28 | 0,15 | 0,09 | 0,2 | 0,1 | 0,06 |
| | $s_8$ | 0,1 | 0,07 | 0,18 | 0,09 | 0,2 | 0,21 | 0,05 | 0,1 |
| **5, 13, 21** | $s_1$ | 0,25 | 0,07 | 0,1 | 0,09 | 0,1 | 0,16 | 0,03 | 0,2 |
| | $s_2$ | 0,04 | 0,2 | 0,08 | 0,2 | 0,2 | 0,07 | 0,06 | 0,15 |
| | $s_3$ | 0,13 | 0,22 | 0,07 | 0,29 | 0,05 | 0,1 | 0,1 | 0,04 |
| | $s_4$ | 0,15 | 0,07 | 0,1 | 0,01 | 0,2 | 0,08 | 0,3 | 0,09 |
| | $s_5$ | 0,12 | 0,1 | 0,09 | 0,2 | 0,04 | 0,2 | 0,18 | 0,07 |
| | $s_6$ | 0,1 | 0,05 | 0,09 | 0,12 | 0,22 | 0,16 | 0,2 | 0,06 |
| | $s_7$ | 0,01 | 0,2 | 0,13 | 0,2 | 0,08 | 0,13 | 0,05 | 0,2 |
| | $s_8$ | 0,16 | 0,1 | 0,02 | 0,09 | 0,27 | 0,02 | 0,2 | 0,14 |
| **6, 14, 22** | $s_1$ | 0,05 | 0,1 | 0,21 | 0,06 | 0,1 | 0,14 | 0,25 | 0,09 |
| | $s_2$ | 0,02 | 0,2 | 0,08 | 0,2 | 0,27 | 0,1 | 0,08 | 0,05 |
| | $s_3$ | 0,11 | 0,03 | 0,3 | 0,09 | 0,17 | 0,04 | 0,2 | 0,06 |
| | $s_4$ | 0,09 | 0,28 | 0,1 | 0,09 | 0,08 | 0,1 | 0,06 | 0,2 |
| | $s_5$ | 0,13 | 0,1 | 0,08 | 0,2 | 0,04 | 0,2 | 0,18 | 0,07 |
| | $s_6$ | 0,25 | 0,07 | 0,09 | 0,2 | 0,15 | 0,1 | 0,04 | 0,1 |
| | $s_7$ | 0,24 | 0,03 | 0,1 | 0,19 | 0,2 | 0,06 | 0,05 | 0,13 |
| | $s_8$ | 0,11 | 0,23 | 0,08 | 0,13 | 0,08 | 0,2 | 0,1 | 0,07 |
| **7, 15, 23** | $s_1$ | 0,2 | 0,07 | 0,2 | 0,03 | 0,1 | 0,3 | 0,04 | 0,06 |
| | $s_2$ | 0,06 | 0,2 | 0,17 | 0,14 | 0,08 | 0,2 | 0,05 | 0,1 |
| | $s_3$ | 0,21 | 0,2 | 0,1 | 0,2 | 0,09 | 0,09 | 0,06 | 0,05 |
| | $s_4$ | 0,1 | 0,03 | 0,2 | 0,07 | 0,07 | 0,28 | 0,15 | 0,1 |
| | $s_5$ | 0,11 | 0,14 | 0,05 | 0,2 | 0,08 | 0,2 | 0,13 | 0,09 |
| | $s_6$ | 0,09 | 0,01 | 0,1 | 0,15 | 0,24 | 0,27 | 0,1 | 0,04 |
| | $s_7$ | 0,01 | 0,2 | 0,23 | 0,19 | 0,14 | 0,08 | 0,05 | 0,1 |
| | $s_8$ | 0,05 | 0,14 | 0,02 | 0,2 | 0,15 | 0,04 | 0,2 | 0,2 |
| **8, 16, 24** | $s_1$ | 0,1 | 0,05 | 0,21 | 0,2 | 0,09 | 0,18 | 0,07 | 0,1 |
| | $s_2$ | 0,06 | 0,1 | 0,2 | 0,09 | 0,15 | 0,28 | 0,1 | 0,02 |
| | $s_3$ | 0,18 | 0,09 | 0,2 | 0,11 | 0,06 | 0,01 | 0,2 | 0,15 |
| | $s_4$ | 0,15 | 0,2 | 0,1 | 0,01 | 0,07 | 0,07 | 0,2 | 0,2 |
| | $s_5$ | 0,06 | 0,1 | 0,09 | 0,2 | 0,02 | 0,3 | 0,09 | 0,14 |
| | $s_6$ | 0,28 | 0,07 | 0,08 | 0,1 | 0,26 | 0,1 | 0,05 | 0,06 |
| | $s_7$ | 0,1 | 0,04 | 0,17 | 0,21 | 0,08 | 0,01 | 0,1 | 0,29 |

| № вар. | Матрица переходных вероятностей $p(s_j\|s_i)$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $s_i \backslash s_j$ | $s_1$ | $s_2$ | $s_3$ | $s_4$ | $s_5$ | $s_6$ | $s_7$ | $s_8$ |
| | $s_8$ | 0,2 | 0,13 | 0,1 | 0,3 | 0,02 | 0,07 | 0,1 | 0,08 |

Таблица 1.3

Варианты задания для моделирования непрерывной случайной величины

| № вар. | Закон распределения | Параметры | Диапазон |
|---|---|---|---|
| **1** | Нормальный | $M = 0$, $\sigma = 1$ | $x_{min} = -5$, $x_{max} = 5$ |
| **2** | Экспоненциальный | $\lambda = 2$ | $x_{min} = -0.5$, $x_{max} = 3.5$ |
| **3** | Вейбулла | $b = 1$, $c = 1$ | $x_{min} = -0.5$, $x_{max} = 9$ |
| **4** | Лапласа | $a = 1.5$, $\lambda = 2$ | $x_{min} = -0.5$, $x_{max} = 0.5$ |
| **5** | Коши | $a = 2$ | $x_{min} = -15$, $x_{max} = 1$ |
| **6** | Симпсона | $a = 2.5$ | $x_{min} = 5.5$, $x_{max} = 8$ |
| **7** | Нормальный | $M = 2$, $\sigma = 1.5$ | $x_{min} = -4$, $x_{max} = 8$ |
| **8** | Экспоненциальный | $\lambda = 1.25$ | $x_{min} = -1$, $x_{max} = 6$ |
| **9** | Вейбулла | $b = 1.5$, $c = 1$ | $x_{min} = -0.5$, $x_{max} = 11$ |
| **10** | Лапласа | $a = 3$, $\lambda = 1.5$ | $x_{min} = -0.6$, $x_{max} = 0$ |
| **11** | Коши | $a = 1.5$ | $x_{min} = -10$, $x_{max} = 1$ |
| **12** | Симпсона | $a = 1.75$ | $x_{min} = 3.5$, $x_{max} = 6$ |
| **13** | Нормальный | $M = 1$,$\sigma = 2$ | $x_{min} = -7$, $x_{max} = 10$ |
| **14** | Экспоненциальный | $\lambda = 1.5$ | $x_{min} = -1$, $x_{max} = 5$ |
| **15** | Вейбулла | $b = 2$, $c = 1$ | $x_{min} = -0.5$, $x_{max} = 15$ |
| **16** | Лапласа | $a = 2.5$, $\lambda = 1$ | $x_{min} = -1$, $x_{max} = 0$ |
| **17** | Коши | $a = 1.25$ | $x_{min} = -8$, $x_{max} = 1$ |
| **18** | Симпсона | $a = 1.5$ | $x_{min} = 3$, $x_{max} = 5$ |
| **19** | Нормальный | $M = 2$,$\sigma = 1.5$ | $x_{min} = -5$, $x_{max} = 8$ |
| **20** | Экспоненциальный | $\lambda = 0.75$ | $x_{min} = -1$, $x_{max} = 8$ |
| **21** | Вейбулла | $b = 2.5$, $c = 1$ | $x_{min} = -1$, $x_{max} = 20$ |
| **22** | Лапласа | $a = 3$, $\lambda = 1.5$ | $x_{min} = -0.7$, $x_{max} = 0$ |
| **23** | Коши | $a = 1.5$ | $x_{min} = -24$, $x_{max} = 2$ |
| **24** | Симпсона | $a = 1.25$ | $x_{min} = 2.5$, $x_{max} = 4$ |

***Требования к отчёту.***

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.

3. Исходный код программы для моделирования дискретной случайной величины и результаты моделирования.

4. Исходный код программы для моделирования цепи Маркова и результаты моделирования.

5. Выводы по лабораторной работе.

## 1.4. Контрольные вопросы и задачи

***Теоретические вопросы.***

1. Что понимают под случайной величиной?
2. Что называют функцией распределения?
3. Что такое ансамбль дискретной случайной величины?
4. Каким образом задают функцию распределения непрерывной случайной величины?
5. Что понимают под моделированием случайной величины?
6. В чем заключается метод суперпозиции?
7. Что такое случайный процесс?
8. Какие выделяют виды случайных процессов?
9. Что понимают под цепью Маркова?
10. Какие состояния в цепи Маркова называют возвратными и невозвратными?
11. Какие марковские процессы называют эргодическими?
12. В чем заключается метод обратных функций?

***Практические задачи.***

1. Задан закон распределения дискретной случайной величины (табл. 1.4). Требуется смоделировать на компьютере $N$ значений случайной величины. Найти значения математического ожидания и дисперсии случайной величины.

Таблица 1.4

| **№ вар.** | **Нечётный вариант** | | | | **№ вар.** | **Чётный вариант** | | | |
|---|---|---|---|---|---|---|---|---|---|
| $x_i$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_i$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
| $p$ | 0,5 | 0,15 | 0,1 | 0,25 | $p$ | 0,35 | 0,2 | 0,05 | 0,4 |

# 2. МОДЕЛИРОВАНИЕ И РАСЧЕТ ХАРАКТЕРИСТИК ДИСКРЕТНЫХ ИСТОЧНИКОВ СООБЩЕНИЙ

## 2.1. Цель и задачи работы

Цель работы – приобрести умение моделировать дискретные источники сообщений, а также рассчитывать их информационные характеристики.

Основные задачи работы:

• научиться определить основные информационные характеристики марковских источников сообщений (информационную энтропию, избыточность и эффективность);

• освоить моделирование статистически взаимосвязанных дискретных источников сообщений;

• научиться рассчитывать информационные характеристики дискретных источников сообщений (условную энтропию, энтропию объединения и взаимную информацию).

Работа рассчитана на 4 часа.

## 2.2. Основные теоретические сведения

### 2.2.1. Дискретные источники сообщений. Характеристики марковских источников сообщений

***Дискретные источники сообщений.***

Одним из основных объектов исследования в теории информации являются ***источники сообщений***, которые представляет собой исследуемый или наблюдаемый объект, формирующий сообщения о своем состоянии. В зависимости от вида формируемых сообщений, различают дискретные и непрерывные источники сообщений.

Источник сообщений, который может в каждый момент времени случайным образом принять одно из конечного множества возможных состояний, называют ***дискретным источником сообщений***. Каждому состоянию источника соответствует условное обозначение в виде передаваемого символа.

Множество всех символов $S = \{s_1, s_2, ..., s_m\}$, доступных источнику сообщений, называют ***алфавитом*** этого источника, а общее число символов $m$ – ***объемом алфавита***.

Простейшим видом дискретных источников сообщений являются ***источники без памяти*** (с нулевой памятью), в которых текущее состояние источника не зависит от его предшествующих состояний.

Дискретный источник сообщений без памяти в общем случае характеризуется ***ансамблем*** $S$, то есть полной совокупностью состояний с вероятностями их появления, составляющими в сумме единицу:

$$S = \begin{pmatrix} s_1 & s_2 & ... & s_m \\ p(s_1) & p(s_2) & ... & p(s_m) \end{pmatrix}, \qquad \sum_{i=1}^{m} p_i = 1.$$

Энтропия $H(S)$ дискретного источника сообщений без памяти, задаваемого ансамблем $S$, может быть определена следующим образом:

$$H(S) = -\sum_{i=1}^{m} p(s_i) \log_2 p(s_i), \text{ (бит/символ);} \qquad (2.1)$$

где $p(s_i)$ – вероятность появления символа $s_i$ из алфавита объемом $m$.

Моделирование дискретного источника сообщений без памяти является идентичным моделированию дискретной случайной величины. В этом случае множество значений случайной величины является алфавитом источника.

Во многих случаях источник сообщений без памяти является грубой моделью реальных источников информации. Текущее состояние многих систем зависит от того, в каких состояниях эти системы находились ранее.

Дискретный источник сообщений называется ***источником с памятью n-го порядка***, если вероятность его перехода в определенное состояние зависит от того, через какие $n$ состояний этот источник последовательно прошел в предшествующие моменты времени.

В теории информации наиболее изученными дискретными источниками сообщений с памятью являются ***марковские ис-***

***точники***. Математическое описание марковских источников основывается на использовании такого класса случайных процессов, как марковские процессы с дискретными состояниями (цепи Маркова).

***Марковские источники сообщений и их характеристики.***

***Марковским источником*** называется дискретный источник сообщений с памятью 1-го порядка, работающий по схеме эргодической цепи Маркова. Графы состояний марковских источников, имеющих два, три и четыре состояния, показаны на рис. 2.1.

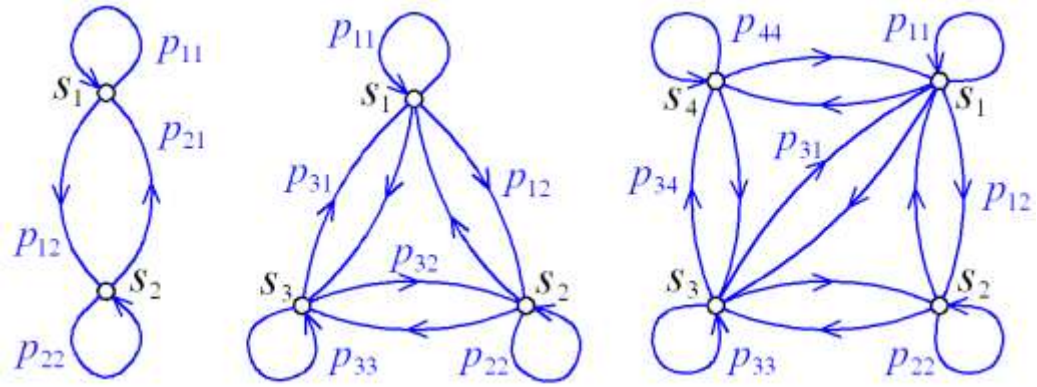


Рис. 2.1. Графы состояний простейших марковских источников

Марковский источник может быть описан с помощью ***матрицы переходных вероятностей***, которая имеет следующий вид:

$$P_m = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{bmatrix};$$

Сумма элементов всех элементов любой строки матрицы $P_m$ должна быть равна 1, поскольку текущее состояние источника должно обязательно перейти в какое-либо состояние:

$$\sum_{j=1}^{m} p_{ij} = \sum_{j=1}^{m} p(s_j \mid s_i) = 1 \qquad (i = 1, 2, \ldots, m).$$

С учетом статистической связи между парами символов энтропия марковского источника может быть найдена из формулы:

$$H_2(S) = -\sum_{i=1}^{m}\sum_{j=1}^{m} p(s_i)\,p(s_j \mid s_i)\log_2 p(s_j \mid s_i), \text{ (бит/символ);} \quad (2.2)$$

где $p(s_j \mid s_i)$ – условная вероятность появления символа $s_j$ при заданном символе $s_i$.

Для нахождения энтропии марковского источника требуется найти стационарное распределение вероятностей символов $p^*(s_i)$ этого источника.

Помимо энтропии, важными характеристиками источника сообщений, являются информационная эффективность, избыточность и производительность.

***Эффективность*** $E(S)$ источника сообщений $S$ определяется следующим образом:

$$E(S) = \frac{H(S)}{H_{\max}}; \quad (2.3)$$

где $H_{\max} = \log_2 m$ – максимально возможная для данного источника сообщений энтропия, которая достигается при равной вероятности появления символов.

***Избыточность*** $R(S)$ источника сообщений $S$ показывает относительную недогруженность информацией на символ его алфавита. Данная характеристика может быть определена следующим образом:

$$R(S) = 1 - \frac{H(S)}{H_{\max}} = 1 - E(S); \quad (2.4)$$

Кроме общей информационной избыточности существуют частные избыточности, основными из которых являются:

• Избыточность, вызванная статистической связью между символами $s_1$, $s_2$, ..., $s_m$:

$$R_p = 1 - \frac{H(S)}{H_1(S)}; \quad (2.5)$$

где $H_1(S) = -\sum_{i=1}^{m} p(s_i)\log_2 p(s_i)$ – энтропия источника сообщений при отсутствии статистической связи между символами.

• Избыточность, вызванная неэкстремальным распределением символов $s_1$, $s_2$, ..., $s_m$ (неравновероятным появлением символов в сообщениях):

$$R_{\varphi} = 1 - \frac{H_1(S)}{H_{\max}}. \qquad (2.6)$$

Общая информационная избыточность связана с частными избыточностями следующим образом:

$$R(S) = R_p + R_{\varphi} - R_p R_{\varphi}. \qquad (2.7)$$

❒ ***Пример* 2.1. *Определение информационных характеристик марковского источника сообщений.***

Требуется определить условную энтропию и избыточность дискретного источника сообщений с памятью, описанного в примере 1.2. Безусловные вероятности появления символов источника в сообщении также возьмем из примера 1.2.

Условная энтропия дискретного источника сообщений с учетом парных сочетаний символов по формуле (2.2) будет:

$H_2(S) = -$ [0,241·(0,11·log$_2$0,11 + 0,36·log$_2$0,36 +

+0,19·log$_2$0,19 +0,34·log$_2$0,34) + 0,272·(0,27·log$_2$0,27 +

+ 0,45·log$_2$0,45 + 0,28·log$_2$0,28) +0,249·(0,35·log$_2$0,35 +

+ 0,29·log$_2$0,29 + 0,05·log$_2$0,05 + 0,31·log$_2$0,31) +

+ 0,238·(0,23·log$_2$0,23 + 0,48·log$_2$0,48 + 0,29·log$_2$0,29)] =

= – [0,241·(0,350 + 0,531 + 0,455 + 0,529) + 0,272·(0,510 +

+ 0,518 + 0,514) + 0,249·(0,530 + 0,518 + 0,216 + 0,523) +

+ 0,238·(0,488 + 0,508 + 0,518) = 0,451 + 0,418 + 0,447 +

+ 0,358 = 1,674 (бит/символ).

Безусловная энтропия источника может быть определена по формуле (2.1) на основе эмпирических вероятностей из примера 1.2:

$H_1(S) = -$ (0,242·log$_2$0,242 + 0,271·log$_2$0,271 + 0,25·log$_2$0,25 +

+ 0,237·log$_2$0,237) = 1,998 (бит/символ).

Набольшая энтропия источника будет:

$$H_{max}(S) = \log_2 4 = 2 \text{ (бит/символ)}.$$

По формулам (2.3) и (2.4) найдем эффективность и избыточность источника сообщений:

$$E(S) = 1{,}674 / 2 = 0{,}837;$$

$$R(S) = 1 - 0{,}837 = 0{,}163.$$

Частные избыточности источника сообщений по формулам (2.5) и (2.6) будут:

$$R_p = 1 - 1{,}674 / 1{,}998 = 0{,}162;$$

$$R_\varphi = 1 - 1{,}998 / 2 = 0{,}001.$$

По формуле (2.7) получим следующее значение общей избыточности источника сообщений:

$$R(S) = 0{,}162 + 0{,}001 - 0{,}162 \cdot 0{,}001 = 0{,}163.$$

Полученное значение совпадает с результатом, найденным по формуле (2.4). ❐

### 2.2.2. Статистические связанные источники сообщений и их характеристики

***Статистически связанные источники сообщений.***

Между состояниями двух дискретных источников $A$ и $B$ сообщений могут существовать статистические связи. В этом случае вероятности $p(b_j|a_i)$ перехода одного источника (зависимого) в определенное состояние зависит от того, в каком состоянии находится другой источник.

Указанная статистическая зависимость может быть отображена в виде графа переходов состояний (рис. 2.2).

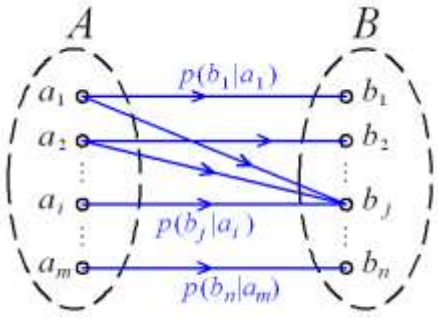


Рис. 2.2. Граф переходов состояний двух статистически взаимосвязанных источников *A* и *B*

Для задания статистической связи между состояниями источников используется матрица переходных вероятностей:

$$P_{mn} = \begin{bmatrix} p(b_1 \mid a_1) & p(b_2 \mid a_1) & \dots & p(b_n \mid a_1) \\ p(b_1 \mid a_2) & p(b_2 \mid a_2) & \dots & p(b_n \mid a_2) \\ \dots & \dots & \dots & \dots \\ p(b_1 \mid a_m) & p(b_2 \mid a_m) & \dots & p(b_n \mid a_m) \end{bmatrix}.$$

❒ ***Пример* 2.2. *Моделирование работы статистически связанных источников сообщений.***

Требуется разработать компьютерную модель дискретного источника сообщений *U*, символы которого статистически взаимосвязаны с символами в сообщениях источника *S* (пример 2.1).

Состояния источника *U* описываются следующей матрицей переходных вероятностей $\boldsymbol{p(s_j|u_i)}$:

| ***i, j*** | $\boldsymbol{u_1}$ | $\boldsymbol{u_2}$ | $\boldsymbol{u_3}$ | $\boldsymbol{u_4}$ | $\boldsymbol{u_5}$ |
|---|---|---|---|---|---|
| $\boldsymbol{s_1}$ | 0,24 | 0,06 | 0,38 | 0 | 0,32 |
| $\boldsymbol{s_2}$ | 0,09 | 0,25 | 0,15 | 0,24 | 0,27 |
| $\boldsymbol{s_3}$ | 0,18 | 0 | 0,33 | 0,19 | 0,3 |
| $\boldsymbol{s_4}$ | 0,1 | 0,25 | 0,29 | 0,36 | 0 |

Исходный код класса **SourceU** представлен в листинге 2.1.

Исходный код класса **Program** консольного приложения приведён в листинге 2.2.

Листинг 2.1. Исходный код класса **SourceU**

```
/// <summary>
/// Представляет источники сообщений (источники U), статистически
/// связанные с другими источниками (источники S)
/// </summary>
class SourceU
{
    const int mu = 5; // Объем алфавита источника сообщений U
    const int ms = 4; // Объем алфавита источника сообщений S
    string[] u;       // Алфавит источника сообщений U
    double[,] p;      // Матрица переходных вероятностей
    int[] f;          // Массив частот появления символов
    Random rnd;

    /// <summary>
    /// Конструктор с параметрами по умолчанию
    /// </summary>
    public SourceU()
    {
        u = new string[mu] { "u1", "u2", "u3", "u4", "u5" };
        p = new double[ms, mu] {{0.24, 0.06, 0.38, 0.00, 0.32},
                                {0.09, 0.25, 0.15, 0.24, 0.27},
                                {0.18, 0.00, 0.33, 0.19, 0.30},
                                {0.10, 0.25, 0.29, 0.36, 0.00}};
        f = new int[mu];
        rnd = new Random();
    }

    /// <summary>
    /// Генерирует случайным образом символ и возвращает его
    /// </summary>
    /// <param name="si">Символ источника S</param>
    /// <returns>Символ источника U</returns>
    public string GenerSymbol(string si)
    {
        string uj = ""; // Текущий символ источника U
        int i = Convert.ToInt32(si.Remove(0, 1)) - 1; // Индекс символа источника S
        double q = 0.0;  // Нижняя граница интервала вероятностей
        double r = rnd.NextDouble();
        for (int j = 0; j < mu; j++)
        {
            if ((r > q) && (r <= q + p[i, j]))
            {
                uj = u[j]; f[j]++; break;
            }
            q += p[i, j];
        }
        return uj;
    }

    /// <summary>
    /// Возвращает строку с частотами появления символов
    /// </summary>
    /// <returns>Частоты появления символов</returns>
    public string GetFreqs()
    {
        string buf = "\nЧастоты появления символов:\n";
        for (int j = 0; j < mu; j++)
        {
            buf += string.Format("{0} - {1}\n", u[j], f[j]);
        }
        return buf;
    }
}
```

Листинг 2.2. Исходный код класса **Program** консольного приложения

```
 9    class Program
10    {
11        static void Main(string[] args)
12        {
13            Console.Title = "Моделирование работы статистически связанных источников";
14            Console.WriteLine("Введите число генерируемых символов:");
15            string buf = Console.ReadLine(); // Строка-буфер
16            int n = int.Parse(buf); // Число генерируемых символов
17            buf = "";
18
19            SourceS sourceS = new SourceS();
20            SourceU sourceU = new SourceU();
21            string mes = ""; // Сообщение, выводимое в консоли
22
23            for (int i = 0; i < n; i++)
24            {
25                buf = sourceS.GenerSymbol();
26                mes += string.Format(" {0}-->", buf);
27                buf = sourceU.GenerSymbol(buf);
28                mes += string.Format("{0}; ", buf);
29            }
30
31            Console.WriteLine(mes);
32            Console.WriteLine(sourceU.GetFreqs());
33            Console.Read();
34        }
35    }
```

Результат моделирования для 50000 символов показан на рис. 2.3.

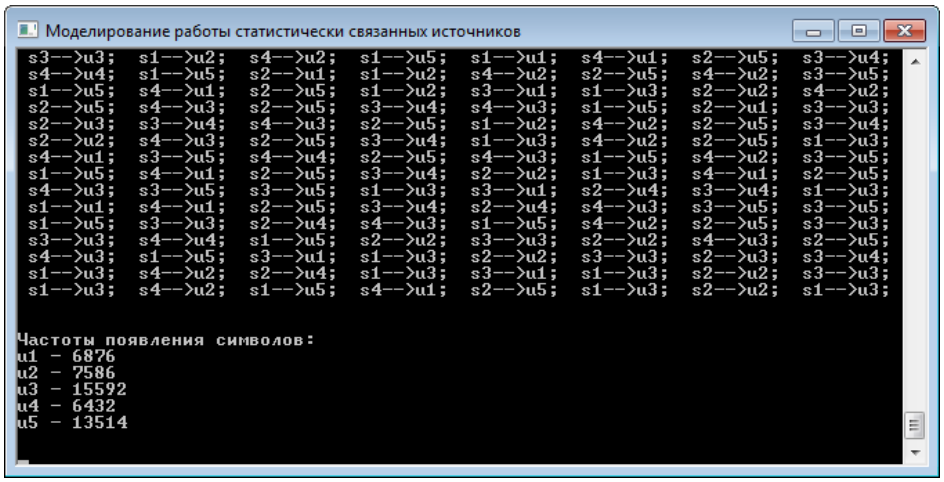


Рис. 2.3. Результат работы консольного приложения

Определим эмпирические вероятности появления символов источника $U$:

$p^*(u_1) = 6876 / 50000 = 0{,}138;$

$p^*(u_2) = 7586 / 50000 = 0{,}152;$

$p^*(u_3) = 15592 / 50000 = 0{,}311;$

$p^*(u_4) = 6432 / 50000 = 0{,}129;$

$p^*(u_5) = 13514 / 50000 = 0{,}270;$

$$\sum_{i=1}^{5} p^*(u_i) = 0{,}138 + 0{,}152 + 0{,}311 + 0{,}129 + 0{,}270 = 1.$$

Отсюда безусловная энтропия источника $U$ по формуле (2.1) будет:

$$H(U) = -(0{,}138\cdot\log_2 0{,}138 + 0{,}152\cdot\log_2 0{,}152 + 0{,}311\cdot\log_2 0{,}311 + 0{,}129\cdot\log_2 0{,}129 + 0{,}27\cdot\log_2 0{,}27) = 2{,}221 \text{ (бит/символ)}.$$

Определим наибольшую энтропию источника $U$:

$$H_{max}(U) = \log_2 5 = 2{,}232 \text{ (бит/символ)}.$$

Найдем эффективность и избыточность источника сообщений $U$:

$$E(U) = H(U) / H_{max}(U) = 2{,}221 / 2{,}232 = 0{,}995;$$

$$R(U) = 1 - E(U) = 1 - 0{,}995 = 0{,}005.$$

Таким образом, источник $U$ имеет следующие информационные характеристики: $H$ = 2,221 бит/символ, $H_{max}$ = 2,232 бит/символ, $E$ = 0,995, $R$ = 0,005. ❐

***Условная энтропия, энтропия объединения и взаимная информация дискретных источников сообщений.***

***Условная энтропия*** $H(B|A)$ источника сообщений $B$ относительно статистически связанного с ним источника $A$ характеризует степень неопределенности состояния источника $B$, при известном состоянии источника $A$.

Понятие условной энтропии в теории информации используется при определении взаимозависимости между символами кодируемого алфавита, для определения потерь при передаче ин-

формации по каналам связи, при вычислении энтропии объединения.

Условная энтропия дискретного источника сообщений $B$ относительно источника $A$ может быть определена следующим образом:

$$H(B \mid A) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p(a_i)\, p(b_j \mid a_i) \log_2 p(b_j \mid a_i) =$$

$$= -\sum_{i=1}^{m}\sum_{j=1}^{n} p(a_i, b_j) \log_2 p(b_j \mid a_i); \text{ (бит/символ).} \quad (2.8)$$

Под ***объединением*** двух источников сообщений $A$ и $B$ с возможными состояниями $a_1, a_2, \ldots, a_m$; $b_1, b_2, \ldots, b_n$ понимается сложный источник $(A,B)$, состояния которого представляют все возможные комбинации состояний $a_i$, $b_j$ источников $A$ и $B$.

$$(A,B) = \begin{pmatrix} (a_1,b_1) & (a_2,b_1) & \ldots & (a_i,b_j) & \ldots & (a_m,b_n) \\ p(a_1,b_1) & p(a_2,b_1) & \ldots & p(a_i,b_j) & \ldots & p(a_m,b_n) \end{pmatrix}.$$

где $p(a_i, b_j)$ – вероятность того, что система $(A,B)$ находится в состоянии $(a_i, b_j)$.

***Энтропия объединения*** $H(A,B)$ источников сообщений $A$ и $B$ характеризует неопределенность того, что источник $(A,B)$ находится в состоянии $(a_i,b_j)$.

Для источников сообщений $A$ и $B$ ***энтропия объединения*** (взаимная энтропия) представляет собой сумму вида:

$$H(A,B) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p(a_i,b_j) \log_2 p(a_i,b_j); \text{ (бит/два символа);} \quad (2.9)$$

где $p(a_i,b_j)$ – вероятность совместного появления в сообщениях символов $a_i$ и $b_j$ из источников $A$ и $B$.

Энтропия объединения и условная энтропия дискретных источников $A$ и $B$ при наличии статистической связи между их состояниями связаны между собой соотношениями:

$$H(A,B) = H(A) + H(B|A) = H(B) + H(A|B); \quad (2.10)$$

$$H(B|A) = H(A,B) - H(A);$$

$$H(A|B) = H(A,B) - H(B).$$

***Взаимной информацией***, содержащейся в источниках *A* и *B*, называется величина *I*(*A*,*B*), которая характеризует среднее количество информации, получаемое от одного символа источника *A* при одном переданном символе источника *B*.

Взаимную информацию можно определить как уменьшение энтропии источника *B* в результате получения сведений о состоянии источника *A* и наоборот:

$$I(A,B) = H(B) - H(B|A) = H(A) - H(A|B). \qquad (2.11)$$

Через вероятности состояний источников *A*, *B* и (*A*,*B*) взаимная информация может быть найдена следующим образом:

$$I(A,B) = \sum_{i=1}^{m}\sum_{j=1}^{n} p(a_i,b_j)\log_2 \frac{p(a_i,b_j)}{p(a_i)p(b_j)}\text{; (бит/символ).} \qquad (2.12)$$

❒ ***Пример* 2.3. *Определение информационных характеристик взаимосвязанных источников сообщений.***

Требуется определить условную энтропию *H*(*U*|*S*) источника сообщений *U* (пример 2.2) относительно источника *S* (пример 2.1), также энтропию объединения *H*(*S*,*U*) указанных двух источников, а также их взаимную информацию *I*(*S*,*U*).

На основе матрицы переходных вероятностей и стационарного распределения вероятностей источника *S* (пример 2.1) получим условную энтропию источника *U* относительно источника *S* по формуле (2.8):

$H(U|S) = -$ [0,24·log$_2$0,24 + 0,06·log$_2$0,06 +

+ 0,38·log$_2$0,38 + 0,32·log$_2$0,32) + 0,271· (0,09·log$_2$0,09 +

+ 0,25·log$_2$0,25 + 0,15·log$_2$0,15 + 0,24·log$_2$0,24 +

+ 0,27·log$_2$0,27) + 0,25· (0,18·log$_2$0,18 + 0,33·log$_2$0,33 +

+ 0,19·log$_2$0,19 + 0,3·log$_2$0,3) + 0,237· (0,1·log$_2$0,1 +

+ 0,25·log$_2$0,25 + 0,29·log$_2$0,29 + 0,36·log$_2$0,36)] =

= 0,217 + 0,187 + 0,619 + 0,192 + 0,423 = 1,638 (бит/символ).

Энтропия объединения по формуле (2.10) может быть найдена через энтропию *H*(*S*) источника *S* (пример 2.2) и условную энтропию *H*(*U*|*S*):

$H(S,U) = H(S) + H(U|S) = 1{,}674 + 1{,}638 = 3{,}312$ (бит/два символа).

По формуле (2.11) найдем взаимную информацию источников $S$ и $U$:

$I(S,U) = H(U) - H(U|S) = 2{,}221 - 1{,}638 = 0{,}583$ (бит/символ).

Таким образом, взаимосвязанные источники $S$ и $U$ имеют следующие информационные характеристики: $H(U|S) = 1{,}638$ бит/символ, $H(S,U) = 3{,}312$ бит/два символа, $I(S,U) = 0{,}583$ бит/символ. ❐

### 2.3.3. Коды, их характеристики и виды. Кодирование источников сообщений

***Понятие кода. Характеристики и виды кодов.***

Символы, выдаваемые источником сообщений, обычно подаются на вход кодера, который осуществляет процесс кодирования информации.

Под ***кодированием информации*** понимают процесс преобразования информации из формы, удобной для непосредственного ее использования, в форму, удобную для передачи, хранения или автоматической обработки. Обратный процесс называется ***декодированием***.

Кодер имеет собственный алфавит символов $C = \{c_1, c_2, ..., c_K\}$, называемый ***вторичным алфавитом*** (за первичный алфавит принимается алфавит источника сообщений). Кроме того, кодер имеет определенный алгоритм кодирования (код).

***Код –*** правило сопоставления каждому конкретному сообщению (символу первичного алфавита) строго определённой комбинации символов вторичного алфавита. Отдельная комбинация таких символов называется ***кодовым словом***. Иногда используют термины «кодовая комбинация», «кодовая последовательность» или «кодограмма».

Основными целями кодирования информации являются:

• преобразование информации в форму, пригодную для её автоматической обработки с помощью технических средств;

• эффективное использование канала связи; уменьшение стоимости передачи и хранения; уменьшение избыточности;

• защита от помех в канале связи; повышение помехоустойчивости и достоверности передачи сообщений;

• криптографическая защита информации; сокрытие смысла передаваемой информации от непосвящённых лиц (шифры).

Основные характеристики кодов:

• ***Основание кода***. Объем алфавита $K$ кодера, то есть число различных символов кода называют ***основанием кода***. Например, если $K = 2, 3, 4, \ldots$ , то код называют бинарным (двоичным), триарным (троичным), тетрарным и т. д.

• ***Длина кодового слова*** (разрядность) $L$. Число символов $L$ в кодовом слове называется ***длиной кодового слова***.

Коды, все слова которых имеют одинаковую длину, называются ***равномерными*** (***блоковыми***). Соответственно коды, в которых данное условие не выполняется, называют ***неравномерными***. Классическим примером равномерного кода является пятизначный двоичный код Бодо, используемый в телеграфной связи. К неравномерным кодам относится код Морзе.

• ***Общее число кодовых слов*** $N$ (мощность кода). Если все кодовые слов используются для кодирования сообщений, то такой код называется полным.

Для равномерного кода общее число кодовых слов определяется следующим образом:

$$N = K^L.$$

• ***Взвешенность кода*** – соответствие символов кода весовым коэффициентам системы счисления.

Если все кодовые комбинации кода соответствуют числам выбранной системы счисления, то такие коды ***называются взвешенными*** (арифметическими). Взвешенные коды широко используются для передачи и обработки числовой информации. В противном случае коды называют ***невзвешенными***. Примером взвешенного кода является натуральный двоичный код (НДК), в котором кодовые комбинации соответствуют последовательности натуральных чисел. Примером невзвешенного кода является двоичный код Грэя.

По назначению коды можно разделить на следующие группы:

• ***Коды представления данных*** обеспечивают запись данных в требуемом для автоматической обработки формате.

• ***Эффективные (оптимальные) коды*** позволяют сжать передаваемые сообщения.

• ***Помехоустойчивые коды*** обеспечивают обнаружение и исправление ошибок в полученных сообщениях.

• ***Шифры*** служат для защиты передаваемой информации от прочтения противником.

***Кодирование источников сообщений.***

Первым кодером в системе передачи сообщений является ***кодер источника сообщений***, который служит для представления сообщений от источника в наиболее компактной форме. Это нужно, чтобы максимально эффективно использовать ресурсы канала связи либо запоминающего устройства. Кодеры источника устраняют из сообщений избыточность.

***Декодер источника сообщений*** преобразует закодированное сообщение в исходную форму, пригодную для восприятия получателем.

❐ ***Пример* 2.4. *Моделирование кодера и декодера источника сообщений.***

Требуется разработать модель кодера и декодера для источника сообщений, описанного в примере 2.1.

Примем, что кодер осуществляет простое кодирование символов источников сообщений с помощью следующего двоичного кода:

$$s_1 \rightarrow \mathbf{00}; \quad s_2 \rightarrow \mathbf{01}; \quad s_3 \rightarrow \mathbf{10}; \quad s_4 \rightarrow \mathbf{11}.$$

Соответственно, декодер осуществляет обратное преобразование.

Декодер источника сообщений преобразует кодовые слова в символы источника, а также определяет число кодовых слов, которые из-за ошибок не могут быть декодированы.

В проект консольного приложения из примера 2.1 добавим классы **SourceEncoder** и **SourceDecoder**, представляющие соответственно кодер и декодер источника сообщений.

В классе **SourceEncoder** должен присутствовать метод **string SimpleEncoding(string _s)**, который для заданного символа **_s** возвращает соответствующее ему кодовое слово. В класс **SourceDecoder** добавим метод **string SimpleDecoding(string _c)**, который для полученного кодового слова **_c** возвращает требуемый символ источника.

Дополнительно в классе **SourceDecoder** реализуем метод **string GetErrorNumber()**, который возвращает число ошибок при декодировании.

Исходный код классов **SourceEncoder** и **SourceDecoder** приведен в листингах 2.3 и 2.4.

Листинг 2.3. Исходный код класса **SourceEncoder**

```
 9      /// <summary>
10      /// Представляет кодер источника сообщений
11      /// </summary>
12      class SourceEncoder
13      {
14          /// <summary>
15          /// Выполняет простое кодирование символов источника
16          /// </summary>
17          /// <param name="s">Символ источника</param>
18          /// <returns>Кодовое слово</returns>
19          public string SimpleEncoding(string _s)
20          {
21              string c = ""; // Кодовое слово
22              switch (_s)
23              {
24                  case "s1": c = "00"; break;
25                  case "s2": c = "01"; break;
26                  case "s3": c = "10"; break;
27                  case "s4": c = "11"; break;
28              }
29              return c;
30          }
31      }
```

Листинг 2.4. Исходный код класса **SourceDecoder**

```
/// <summary>
/// Представляет декодер источника сообщений
/// </summary>
class SourceDecoder
{
    int errNum; // Число ошибок декодирования

    /// <summary>
    /// Конструктор по умолчанию
    /// </summary>
    public SourceDecoder()
    { this.errNum = 0; }

    /// <summary>
    /// Выполняет простое декодирование кодовых слов
    /// </summary>
    /// <param name="_c">Кодовое слово</param>
    /// <returns>Символ источника</returns>
    public string SimpleDecoding(string _c)
    {
        string s = ""; // Символ
        switch (_c)
        {
            case "00": s = "s1"; break;
            case "01": s = "s2"; break;
            case "10": s = "s3"; break;
            case "11": s = "s4"; break;
            default:   s = "><"; errNum++; break; // Знак ошибки
        }
        return s;
    }

    /// <summary>
    /// Возвращает строку с данными об ошибках декодирования
    /// </summary>
    /// <returns>Данные об ошибках</returns>
    public string GetErrorNumber()
    {
        return string.Format("Число ошибок декодирования: " + errNum);
    }
}
```

В методе **Main()** консольного приложения выполним форматный вывод в окно консоли строк, возвращаемых методами классов **MarkovSource**, **SourceEncoder** и **SourceDecoder**.

Исходный код метода **Main()** представлен в листинге 2.5.

Листинг 2.5. Исходный код метода **Main()** консольного приложения

```
class Program
{
    static void Main(string[] args)
    {
        Console.Title = "Моделирование марковского источника";
        // Экземпляр класса марковский источник сообщений
        MarkovSource mSrc = new MarkovSource();
        // Экземпляр класса кодер источника сообщений
        SourceEncoder sEnc = new SourceEncoder();
        // Экземпляр класса декодер источника сообщений
        SourceDecoder sDec = new SourceDecoder();

        Console.WriteLine("Введите число генерируемых символов:");
        string buf = Console.ReadLine();
        int n = int.Parse(buf); // Число генерируемых символов

        string src = ""; // Сообщение на выходе источника (ИС)
        string enc = ""; // Сообщение на выходе кодера источника (КИС)
        string dec = ""; // Сообщение на выходе декодера источника (ДИС)
        buf = "|{0,5}|{1,7}|{2,7}|{3,7}|"; // Строка форматного вывода

        Console.WriteLine(buf, "№", "ИС", "КИС", "ДИС");
        for (int i = 0; i < n; i++)
        {
            src = mSrc.GenerSimbol();
            enc = sEnc.SimpleEncoding(src);
            dec = sDec.SimpleDecoding(enc);
            Console.WriteLine(buf, i + 1, src, enc, dec);
        }

        Console.WriteLine(mSrc.GetFreqs());
        Console.WriteLine(sEnc.GetErrorNumber());
        Console.WriteLine(sDec.GetErrorNumber());
        Console.Read();
    }
}
```

Результат работы консольного приложения для введенного числа символов показан на рис. 2.4. ❐

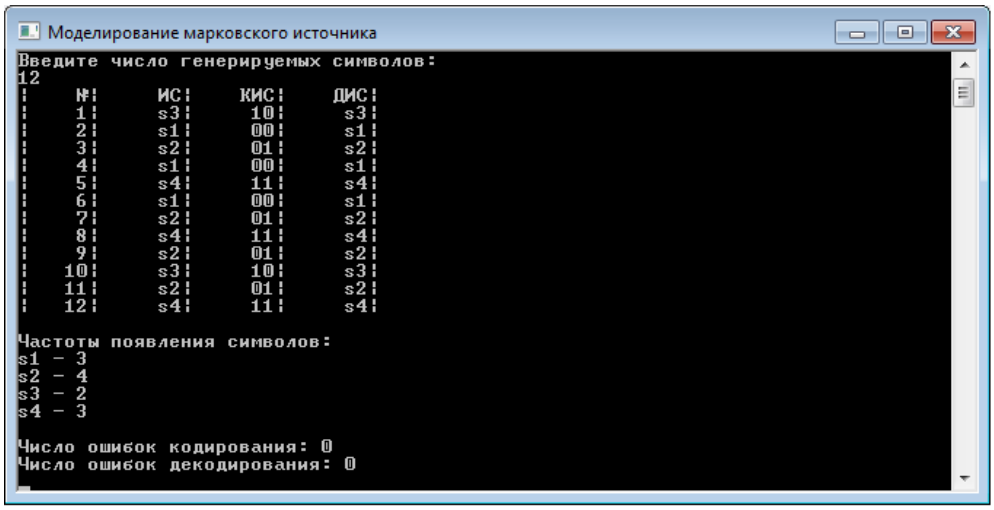


Рис. 2.4. Результат работы консольного приложения

## 2.3. Порядок выполнения работы и варианты заданий

***Основные этапы выполнения работы.***

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к практической работе.
2. Определить энтропию марковского источника сообщений $S$ на основе данных из табл. 1.1 (переходные вероятности) и результатов моделирования (стационарные вероятности состояний). Также определить эффективность $E$, частные избыточности, вызванные наличием связи между символами $R_p$ и неэкстремальным распределением символов $R_\varphi$. На основе частных избыточностей найти общую информационную избыточность $R$ источника сообщений.
3. Разработать компьютерную модель источника сообщений $U$, зависимого от марковского источника $S$. Статистическая связь между состояниями источников задана матрицей переходных вероятностей (таблица 2.1). Осуществить моделирование работы источника при достаточно большой длине сообщения (например, 50000 символов). На основе полученных частот появления символов определить их эмпирические вероятности. Найти энтропию источника $U$, его эффективность и избыточность.

4. Определить условную энтропию $H(U|S)$ источника $U$, состояния которого статистически связаны с состояниями марковского источника $S$. Кроме того, требуется найти энтропию объединения $H(U, S)$ двух источников $U$ и $S$, а также их взаимную информацию $H(U, S)$.

5. Разработать модели кодера и декодера источника сообщений. В качестве кода рекомендуется использовать равномерный двоичный код с трехразрядными кодовыми словами.

6. Оформить и защитить отчет по лабораторной работе.

***Индивидуальные варианты заданий.***

Таблица 2.1

Матрицы переходных вероятностей $\boldsymbol{p(u_j \mid s_i)}$ дискретного источника сообщений $U$ относительно источника $S$

| № вар. | Матрица переходных вероятностей $p(u_j \mid s_i)$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | $s_i \backslash u_j$ | $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ |
| **1, 7, 13, 19** | $s_1$ | 0,23 | 0,19 | 0 | 0,31 | 0,14 | 0,13 |
| | $s_2$ | 0,16 | 0,25 | 0,13 | 0,09 | 0,22 | 0,15 |
| | $s_3$ | 0,07 | 0,22 | 0,18 | 0,16 | 0,32 | 0,05 |
| | $s_4$ | 0,21 | 0,34 | 0 | 0,1 | 0,08 | 0,27 |
| | $s_5$ | 0,13 | 0,09 | 0,28 | 0,26 | 0 | 0,24 |
| | $s_6$ | 0,25 | 0,14 | 0,23 | 0,03 | 0,27 | 0,08 |
| | $s_7$ | 0,05 | 0 | 0,09 | 0,35 | 0,26 | 0,25 |
| | $s_8$ | 0,33 | 0,27 | 0,11 | 0,06 | 0,12 | 0,11 |
| **2, 8, 14, 20** | $s_1$ | 0,31 | 0,12 | 0,05 | 0,27 | 0,11 | 0,14 |
| | $s_2$ | 0,09 | 0,24 | 0,21 | 0 | 0,29 | 0,17 |
| | $s_3$ | 0,23 | 0,31 | 0,04 | 0,17 | 0,2 | 0,05 |
| | $s_4$ | 0,24 | 0 | 0,36 | 0,08 | 0,03 | 0,29 |
| | $s_5$ | 0,04 | 0,17 | 0,33 | 0 | 0,16 | 0,3 |
| | $s_6$ | 0,16 | 0,02 | 0,15 | 0,34 | 0,1 | 0,23 |
| | $s_7$ | 0,29 | 0,13 | 0,09 | 0,16 | 0,28 | 0,05 |
| | $s_8$ | 0,15 | 0,29 | 0,18 | 0 | 0,17 | 0,21 |
| **3, 9, 15, 21** | $s_1$ | 0,18 | 0,22 | 0 | 0,33 | 0,21 | 0,06 |
| | $s_2$ | 0,14 | 0,25 | 0,09 | 0,17 | 0,13 | 0,22 |
| | $s_3$ | 0,32 | 0,1 | 0,16 | 0,26 | 0,16 | 0 |
| | $s_4$ | 0,19 | 0,08 | 0,31 | 0,11 | 0,24 | 0,07 |
| | $s_5$ | 0,35 | 0,13 | 0,25 | 0 | 0,16 | 0,11 |
| | $s_6$ | 0,27 | 0,36 | 0,1 | 0,12 | 0,07 | 0,08 |
| | $s_7$ | 0,13 | 0,12 | 0,31 | 0,05 | 0,15 | 0,24 |

| № вар. | Матрица переходных вероятностей $p(u_j \mid s_i)$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | $s_i \backslash u_j$ | $u_1$ | $u_2$ | $u_3$ | $u_4$ | $u_5$ | $u_6$ |
| | $s_8$ | 0,08 | 0,15 | 0,29 | 0,11 | 0 | 0,37 |
| **4, 10, 16, 22** | $s_1$ | 0,05 | 0,14 | 0,28 | 0,16 | 0,27 | 0,1 |
| | $s_2$ | 0,3 | 0,26 | 0 | 0,12 | 0,14 | 0,18 |
| | $s_3$ | 0,23 | 0 | 0,19 | 0,07 | 0,35 | 0,16 |
| | $s_4$ | 0,12 | 0,25 | 0,06 | 0,21 | 0,13 | 0,23 |
| | $s_5$ | 0,08 | 0,31 | 0,26 | 0,1 | 0,16 | 0,09 |
| | $s_6$ | 0,32 | 0,04 | 0,3 | 0,12 | 0 | 0,22 |
| | $s_7$ | 0,17 | 0,14 | 0,16 | 0,23 | 0,24 | 0,06 |
| | $s_8$ | 0,2 | 0 | 0,24 | 0,05 | 0,36 | 0,15 |
| **5, 11, 17, 23** | $s_1$ | 0,13 | 0 | 0,3 | 0,09 | 0,25 | 0,23 |
| | $s_2$ | 0,04 | 0,26 | 0,21 | 0,33 | 0,11 | 0,05 |
| | $s_3$ | 0,35 | 0,18 | 0 | 0,24 | 0,06 | 0,17 |
| | $s_4$ | 0,22 | 0,05 | 0,19 | 0,15 | 0,32 | 0,07 |
| | $s_5$ | 0,16 | 0,23 | 0,15 | 0,28 | 0,13 | 0,05 |
| | $s_6$ | 0,2 | 0 | 0,36 | 0,22 | 0,08 | 0,14 |
| | $s_7$ | 0,06 | 0,17 | 0,25 | 0,18 | 0,24 | 0,1 |
| | $s_8$ | 0,32 | 0,08 | 0,26 | 0,19 | 0,15 | 0 |
| **6, 12, 18, 24** | $s_1$ | 0,24 | 0,09 | 0,15 | 0,27 | 0,06 | 0,19 |
| | $s_2$ | 0 | 0,35 | 0,08 | 0,16 | 0,31 | 0,1 |
| | $s_3$ | 0,23 | 0,14 | 0 | 0,34 | 0,18 | 0,11 |
| | $s_4$ | 0,07 | 0,26 | 0,28 | 0,21 | 0,15 | 0,03 |
| | $s_5$ | 0,16 | 0,18 | 0,22 | 0,17 | 0,2 | 0,07 |
| | $s_6$ | 0,3 | 0,06 | 0,34 | 0 | 0,09 | 0,21 |
| | $s_7$ | 0,19 | 0,24 | 0,16 | 0,04 | 0,25 | 0,12 |
| | $s_8$ | 0,14 | 0,19 | 0 | 0,25 | 0,07 | 0,35 |

***Требования к отчёту.***

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Расчёт энтропии, эффективности и избыточности марковского источника сообщений.
4. Исходный код программы для моделирования статистически взаимосвязанных источников сообщений, а также результаты моделирования (частоты появления символов зависимого источника).

5. Расчёт условной энтропии, энтропии объединения и взаимной информации.

6. Выводы по лабораторной работе.

## 2.4. Контрольные вопросы и задачи

***Теоретические вопросы.***

1. Что понимают под дискретным источником сообщений?
2. Какие дискретные источники сообщений называют источниками с памятью?
3. Что характеризует информационная избыточность источника сообщений?
4. Какие выделяют частные виды информационной избыточности и как они связаны с общей избыточностью источника сообщений?
5. Что характеризует и как определяется условная энтропия одного дискретного источника сообщений относительно другого?
6. В каком случае условная энтропия равна нулю?
7. Что характеризует и как определяется энтропия объединения двух дискретных источников сообщений?
8. Чему равна энтропия объединения источников сообщений при их полной статистической зависимости?
9. Как связана энтропия объединения и условная энтропия двух статистически зависимых источников сообщений?
10. Что характеризует взаимная информация источников сообщений?
11. Что понимают под кодом и кодированием?
12. Для решения каких задач осуществляют кодирование?
13. Какие коды называют равномерными и неравномерными?
14. Что называют взвешенностью кода?
15. Какие выделяют разновидности кодов по назначению?

***Практические задачи.***

1. Требуется определить энтропию источника $A = \{a_1, a_2, a_3\}$ условную энтропию $H(B|A)$ источника $B = \{b_1, b_2\}$ относительно источника $A$ и энтропию объединения $H(A,B)$ источников

*A* и *B*. Статистическая зависимость состояний источника *B* относительно состояний источника *A* задана матрицей переходных вероятностей (табл. 2.4).

Таблица 2.4

Варианты заданий для решения задачи

| **Нечётный вариант** | **Чётный вариант** |
|---|---|
| $A = \begin{pmatrix} a_1 & a_2 & a_3 \\ 0{,}5 & 0{,}3 & 0{,}2 \end{pmatrix};$ $P(b \mid a) = \begin{bmatrix} 0{,}8 & 0{,}2 \\ 0{,}4 & 0{,}6 \\ 0{,}5 & 0{,}5 \end{bmatrix}.$ | $A = \begin{pmatrix} a_1 & a_2 & a_3 \\ 0{,}1 & 0{,}6 & 0{,}3 \end{pmatrix};$ $P(b \mid a) = \begin{bmatrix} 0{,}1 & 0{,}9 \\ 0{,}3 & 0{,}7 \\ 0{,}6 & 0{,}4 \end{bmatrix}.$ |

# 3. МОДЕЛИРОВАНИЕ И РАСЧЕТ ХАРАКТЕРИСТИК ДИСКРЕТНЫХ КАНАЛОВ СВЯЗИ

## 3.1. Цель и задачи работы

Цель работы – приобрести умение моделировать дискретные каналы связи с помехами и рассчитывать информационные характеристики дискретного канала связи без памяти.

Основные задачи работы:

• освоить моделирование дискретных каналов связи без памяти, в которых присутствуют помехи;

• научиться рассчитывать информационные характеристики дискретного канала связи (информационные потери при передаче сообщений через канал, пропускную способность канала связи).

Работа рассчитана на 4 часа.

## 3.2. Основные теоретические сведения

### 3.2.1. Дискретные каналы связи. Информационные потери при передаче сообщений через канал с помехами

***Дискретные каналы связи.***

***Каналом связи*** (КС) называют совокупность средств, предназначенных для передачи сообщений через определенную физическую среду в пространстве и во времени (рис. 3.1). В частном случае в качестве физической среды может выступать носитель информации.

Каналы связи могут быть описаны в терминах множества входных сигналов $X$ и выходных сигналов $Y$.

Канал связи называется ***дискретным***, если множества $X$ и $Y$ являются конечными, и называется ***непрерывным***, если $X$ и $Y$ – непрерывные множества. Если одно из множеств $X$ и $Y$ является непрерывным, а другое – дискретным, то такой канал называется ***непрерывно-дискретным*** (дискретно-непрерывным).


Рис. 3.1. Схема канала связи

В случае дискретного канала множество $X = \{x_i\}$ называют ***входным алфавитом***, а множество $Y = \{y_j\}$ – ***выходным алфавитом***.

Наличие помех и шумов в дискретном канале связи приводит к тому, что определенный входной символ $x_i$ с вероятностями, зависящими от самого канала, может переходить в различные выходные символы $y_j$.

Дискретный канал связи называется ***каналом без памяти***, если вероятности перехода входных символов в выходные символы не зависят от ранее переданных символов.

Отобразить влияние помех и шумов на передачу сообщений через дискретный канал без памяти можно с помощью графа перехода состояний (рис. 3.2).

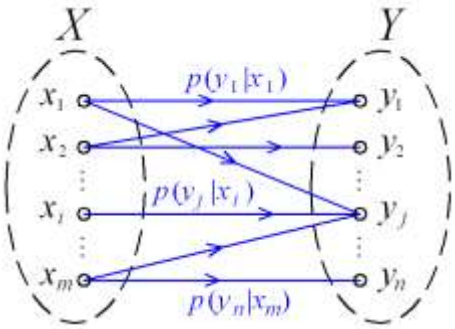


Рис. 3.2. Граф перехода состояний дискретного канала связи без памяти

Для описания дискретных каналов связи без памяти используются матрицы переходных вероятностей, которые имеют следующий вид:

$$P_{mn} = \begin{bmatrix} p(y_1 \mid x_1) & p(y_1 \mid x_2) & \dots & p(y_1 \mid x_n) \\ p(y_2 \mid x_1) & p(y_2 \mid x_2) & \dots & p(y_2 \mid x_n) \\ \dots & \dots & \dots & \dots \\ p(y_n \mid x_1) & p(y_n \mid x_2) & \dots & p(y_n \mid x_n) \end{bmatrix};$$

где $p\ (y_j \mid x_i)$ – вероятность перехода символа $x_i$ входного алфавита в символ $y_j$ выходного алфавита ($i = 1, 2, \dots, m$; $j = 1, 2, \dots, n$).

Среди простых моделей дискретных каналов связи без памяти наибольшее практическое значение имеют модели двоичных каналов (рис. 3.3).

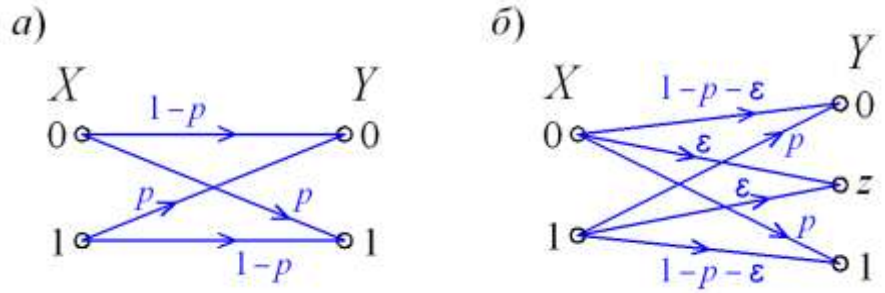


Рис. 3.3. Графы перехода состояний двоичных каналов связи: *а*) – двоичный симметричный канал; *б*) – двоичный канал со стиранием

Матрица переходных вероятностей двоичного симметричного канала связи записывается следующим образом:

$$P = \begin{bmatrix} 1-p & p \\ p & 1-p \end{bmatrix};$$

где $p$ – вероятность ошибки при передаче одного символа.

Матрица переходных вероятностей двоичного канала связи со стиранием имеет следующий вид:

$$P = \begin{bmatrix} 1-p-\varepsilon & \varepsilon & p \\ p & \varepsilon & 1-p-\varepsilon \end{bmatrix};$$

где $\varepsilon$ – вероятность стирания символа в канале связи.

❒ ***Пример* 3.1. *Моделирование двоичного канала связи со стиранием*.**

Требуется разработать компьютерную модель двоичного канала связи со стиранием, для которого вероятность ошибки при передаче одного символа составляет $p$ = 0,02, а вероятность стирания символа – $\varepsilon$ = 0,02. Полученную модель будем использовать в составе системы, полученной в примере 2.4.

Для реализации модели двоичного канала связи создадим в консольном приложении класс **BinaryChannel**. В данный класс добавим метод передачи последовательности двоичных символов **string TransmitBits(string cx)**, который в качестве параметра принимает строку **cx** (входная двоичная последовательность) и возвращает строку, учитывающую влияние помех в канале связи.

Исходный код класса **BinaryChannel** приведен в листингах 3.1 и 3.2.

Листинг 3.1. Исходный код класса **BinaryChannel** (часть 1)

```
/// <summary>
/// Представляет двоичный канал связи со стиранием (ДКС)
/// </summary>
class BinaryChannel
{
    const int m = 2; // Объём входного алфавита ДКС
    const int n = 3; // Объём выходного алфавита ДКС
    string[] x; // Входной алфавит ДКС
    string[] y; // Выходной алфавит ДКС
    double p; // Вероятность ошибки при передаче символа через ДКС
    double e; // Вероятность стирания символа при передаче через ДКС
    double[,] pM; // Матрица переходных вероятностей ДКС
    int errNum; // Число ошибок при передаче через ДКС
    Random rnd;

    /// <summary>
    /// Конструктор с параметрами по умолчанию
    /// </summary>
    public BinaryChannel()
    {
        x = new string[m] { "0", "1" };
        y = new string[n] { "0", " ", "1" };
        p = 0.02;
        e = 0.01;
        pM = new double[m, n]{{1 - p - e, e, p        },
                              {p,         e, 1 - p - e}};
        errNum = 0;
        rnd = new Random();
    }
```

Листинг 3.1. Исходный код класса **BinaryChannel** (часть 2)

```
55          /// <summary>
56          /// Возвращает переданную двоичную последовательность
57          /// </summary>
58          /// <param name="cx">Входная последовательность</param>
59          /// <returns>Выходная последовательность</returns>
60          public string TransmitBits(string cx)
61          {
62              string cy = ""; // Выходная последовательность
63              double r = rnd.NextDouble();
64              double q = 0.0; // Кумулятивная вероятность
65              int i = 0; // Индекс символа входного алфавита
66
67              for (int k = 0; k < cx.Length; k++)
68              {
69                  i = Convert.ToInt32(cx.Substring(k, 1));
70                  for (int j = 0; j < n; j++)
71                  {
72                      if ((r > q) && (r <= q + pM[i, j]))
73                      {
74                          cy += y[j];
75                          if (x[i] != y[j]) errNum++; // Подсчет ошибок
76                          break;
77                      }
78                      q += pM[i, j];
79                  }
80              }
81              return cy;
82          }
83
84          /// <summary>
85          /// Возвращает строку с данными об ошибках при передаче
86          /// </summary>
87          /// <returns>Данные об ошибках при передаче</returns>
88          public string GetErrorNumber()
89          {
90              return string.Format("Число ошибок при передаче: " + errNum);
91          }
92      }
```

В код метода **Main()** из примера 2.4 следует добавить создание экземпляра класса **BinaryChannel**, а также вызов его метода **TramsmitBits()**. Кроме того, в строке форматного вывода должно присутствовать поле подстановки для сообщения на выходе канала связи.

Результат работы консольного приложения приведен на рис. 3.4, из которого видно, что в сообщениях 13 и 17 произошли ошибки, а в сообщении 20 был потерян символ. ❐

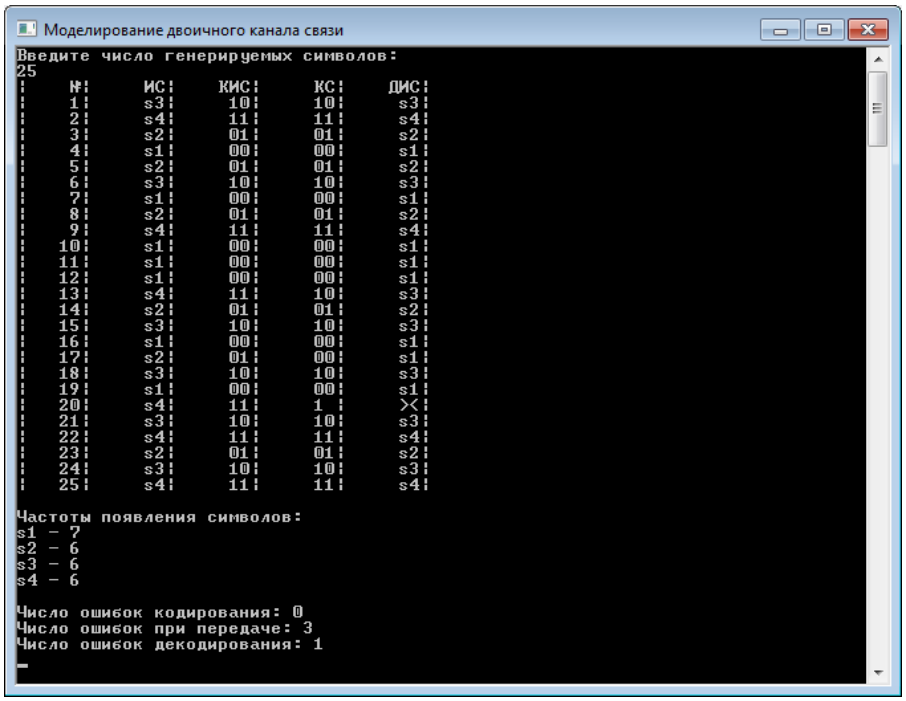


Рис. 3.4. Результат работы консольного приложения

***Информационные потери при передаче сообщений через канал связи с помехами.***

В канале связи без помех количество информации, получаемой с выхода канала связи, совпадает с количеством информации, подаваемой на его вход. То есть в этом случае передача осуществляется без потерь.

При наличии помех в канале связи нарушается соответствие между средним количеством информации, получаемой с выхода канала в единицу времени, и количеством информации на его входе за тот же период времени.

Количество переданной через канал связи информации определяется взаимной информацией:

$$I(X,Y) = H(X) - H(X|Y). \qquad (3.1)$$

Схема потоков информации через канал связи показана на рис. 3.5, где энтропия $H(X|Y)$ характеризует неопределенность состояния источника $X$ при известном состоянии источника $Y$. Данная величина определяет информационные потери при передаче одного символа через канал связи.

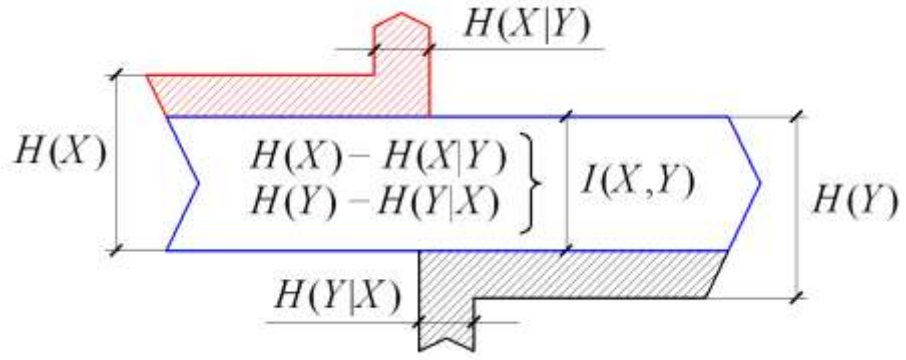


Рис. 3.5. Схема информационного потока через канал связи с помехами

Если канал полностью зашумлен, то $H(X|Y) = H(X)$ и никакой передачи информации не происходит (вся передаваемая информация теряется), то есть $I(X,Y) = 0$.

Информационные потери при передаче $L$ символов по данному каналу связи можно найти из формулы:

$$\Delta I = L \cdot H(X|Y); \qquad (3.2)$$

где $H(X|Y)$ – условная энтропия входного алфавита $X$ канала связи относительно выходного алфавита $Y$.

Отсюда среднее количество принятой информации может быть определено следующим образом:

$$I = L \cdot H(Y) - \Delta I = L[H(Y) - H(X|Y)]. \qquad (3.3)$$

❒ ***Пример* 3.2. *Расчет информационных потерь при передаче информации по каналу связи с помехами.***

Задан дискретный канал связи без памяти, в котором присутствуют помехи. Требуется найти информационные потери при передаче 500 символов через канал, а также количество принятой информации.

Входным и выходным алфавитами канала связи являются множества: $X = \{x_1, x_2, x_3, x_4\}$, $Y = \{y_1, y_2, y_3, y_4\}$. Вероятности появления символов входного алфавита $X$ равны:

$$p(x_1) = 0{,}37; \quad p(x_2) = 0{,}3; \quad p(x_3) = 0{,}23; \quad p(x_4) = 0{,}1.$$

Матрица переходных вероятностей $p(y_j|x_i)$ канала связи имеет следующий вид:

$$P(Y \mid X) = \begin{bmatrix} 0{,}97 & 0{,}02 & 0{,}01 & 0 \\ 0{,}02 & 0{,}98 & 0 & 0 \\ 0 & 0{,}02 & 0{,}96 & 0{,}02 \\ 0 & 0 & 0{,}01 & 0{,}99 \end{bmatrix}.$$

Информационные потери можно найти по формуле (3.2), в которой условная энтропия $H(X|Y)$ может быть определена следующим образом:

$$H(X|Y) = H(X,Y) - H(Y);$$

где $H(X,Y)$ – энтропия объединения входного $X$ и выходного $Y$ алфавитов; $H(Y)$ – энтропия выходного алфавита $Y$.

Определим энтропию $H(Y)$ выходного алфавита $Y$:

$$H(Y) = -\sum_{j=1}^{n} p(y_j) \log_2 p(y_j);$$

где $p(y_j) = \sum_{i=1}^{m} p(x_i) p(y_j \mid x_i)$; причем $\sum_{j=1}^{n} p(y_j) = 1$.

По матрице переходных вероятностей получим:

$p(y_1) = 0{,}37 \cdot 0{,}97 + 0{,}3 \cdot 0{,}02 = 0{,}3649;$

$p(y_2) = 0{,}37 \cdot 0{,}02 + 0{,}3 \cdot 0{,}98 + 0{,}23 \cdot 0{,}02 = 0{,}306;$

$p(y_3) = 0{,}37 \cdot 0{,}01 + 0{,}23 \cdot 0{,}96 + 0{,}1 \cdot 0{,}01 = 0{,}2255;$

$p(y_4) = 0{,}23 \cdot 0{,}02 + 0{,}1 \cdot 0{,}99 = 0{,}1036;$

$$0{,}3649 + 0{,}306 + 0{,}2255 + 0{,}1036 = 1.$$

Отсюда энтропия выходного алфавита будет:

$H(Y) = -(0{,}3649 \log_2 0{,}3649 + 0{,}306 \log_2 0{,}306 +$

$+ 0{,}2255 \log_2 0{,}2255 + 0{,}1036 \log_2 0{,}1036 = 0{,}5306 + 0{,}5227 +$

$+ 0{,}4842 + 0{,}3377 = 1{,}874$ (бит/символ).

Энтропию объединения $H(X,Y)$ можно определить по формуле:

$$H(X,Y) = -\sum_{i=1}^{m} \sum_{j=1}^{n} p(x_i, y_j) \log_2 p(x_i, y_j).$$

Для нахождения энтропии объединения $H(X,Y)$ получим матрицу совместных вероятностей $P(X,Y)$, определив для этого все совместные вероятности:

$$p(x_i,y_j) = p(x_i)\cdot p(y_j|x_i);$$

$p(x_1,y_1) = 0{,}37\cdot 0{,}97 = 0{,}3589$; $p(x_1,y_2) = 0{,}37\cdot 0{,}02 = 0{,}0074$;

$p(x_1,y_3) = 0{,}37\cdot 0{,}01 = 0{,}0037$; $p(x_1,y_4) = 0{,}37\cdot 0 = 0$;

$p(x_2,y_1) = 0{,}3\cdot 0{,}02 = 0{,}006$; $p(x_2,y_2) = 0{,}3\cdot 0{,}98 = 0{,}249$;

$p(x_2,y_3) = 0{,}3\cdot 0 = 0$; $p(x_2,y_4) = 0{,}3\cdot 0 = 0$;

$p(x_3,y_1) = 0{,}23\cdot 0 = 0$; $p(x_3,y_2) = 0{,}23\cdot 0{,}02 = 0{,}0046$;

$p(x_3,y_3) = 0{,}23\cdot 0{,}96 = 0{,}2208$; $p(x_3,y_4) = 0{,}23\cdot 0{,}02 = 0{,}0046$;

$p(x_4,y_1) = 0{,}1\cdot 0 = 0$; $p(x_4,y_2) = 0{,}1\cdot 0 = 0$;

$p(x_4,y_3) = 0{,}1\cdot 0{,}01 = 0{,}001$; $p(x_4,y_4) = 0{,}1\cdot 0{,}99 = 0{,}099$.

Матрица совместных вероятностей будет иметь следующий вид:

$$P(X,Y) = \begin{bmatrix} 0{,}3589 & 0{,}0074 & 0{,}0037 & 0 \\ 0{,}006 & 0{,}249 & 0 & 0 \\ 0 & 0{,}0046 & 0{,}2208 & 0{,}0046 \\ 0 & 0 & 0{,}001 & 0{,}099 \end{bmatrix}.$$

Отсюда энтропия объединения будет:

$H(X,Y) = -(0{,}3589\cdot\log_2 0{,}3589 + 0{,}0074\cdot\log_2 0{,}0074 +$

$+ 0{,}0037\cdot\log_2 0{,}0037 + 0{,}006\cdot\log_2 0{,}006 + 0{,}249\cdot\log_2 0{,}249 +$

$+ 0{,}0046\cdot\log_2 0{,}0046 + 0{,}2208\cdot\log_2 0{,}2208 + 0{,}0046\cdot\log_2 0{,}0046 +$

$+ 0{,}001\cdot\log_2 0{,}001 + 0{,}099\cdot\log_2 0{,}099) = 2{,}069$ (бит/два симв.).

Найдем условную энтропию входного алфавита относительно выходного:

$H(X|Y) = 2{,}069 - 1{,}874 = 0{,}196$ (бит/символ).

Таким образом, информационные потери при передаче заданного числа символов через канал связи будут:

$\Delta I = 500\cdot 0{,}196 = 97{,}76$ (бит).

Отсюда количество принятой информации по формуле (3.3) будет:

$I = 500 \cdot 1{,}87 - 97{,}76 = 839{,}11$ (бит). ❐

### 3.2.2. Пропускная способность дискретного канала связи. Кодеры и декодеры канала связи

***Пропускная способность дискретного канала связи.***

Для описания возможностей канала по передаче информации используют понятие скорости передачи (технической и информационной).

Под ***технической скоростью передачи*** $V_T$ (скорость манипуляции) понимают число элементарных сигналов (символов) передаваемых по каналу в единицу времени. Техническая скорость зависит от свойств линии связи и быстродействия аппаратуры канала и определяется следующим образом:

$$V_T = 1 / \tau_{ср}, \text{ (бод)} \qquad (3.4)$$

где $\tau_{ср}$ – среднее время передачи одного символа через канал связи.

Под ***информационной скоростью*** $V$ (скоростью передачи информации) понимают среднее количество информации, которое передается по каналу связи в единицу времени.

При известной технической скорости $V_T$ скорость передачи информации по каналу связи определяется соотношением:

$$V = V_T \cdot I(X,Y), \text{ (бит/с)} \qquad (3.5)$$

где $I(X,Y)$ – средняя взаимная информация, переносимая одним символом.

***Пропускной способностью*** $C$ дискретного канала связи называется максимальное значение скорости передачи информации по этому каналу:

$$C = \max_{\bar{p}(x)} \{V\} = \max_{\bar{p}(x)} \{V_T \cdot I(X,Y)\}, \text{ (бит/с)} \qquad (3.6)$$

Пропускная способность канала связи при отсутствии помех будет:

$$C = V_T \cdot \log_2 m. \qquad (3.7)$$

❒ ***Пример* 3.3. *Расчет пропускной способности дискретного канала связи с помехами.***

Требуется определить пропускную способность дискретного канала связи, описанного в примере 3.2. Примем, что каждый символ сообщения передается через данный канал в среднем за $2 \cdot 10^{-4}$ с.

По формуле (3.3) техническая скорость передачи будет:

$$V_T = 1 / 0{,}0002 = 5000. \text{ (бод)}$$

Для нахождения пропускной способности канала связи по формуле (3.6) необходимо определить взаимную информацию $I(X,Y)$ входного $X$ и выходного $Y$ алфавитов:

$$I(X,Y) = H(X) - H(X|Y);$$

$$I(X,Y) = H(Y) - H(Y|X).$$

В примере 3.2. было получено, что $H(Y) = 1{,}874$ бит/символ и $H(X|Y) = 0{,}196$ бит/символ.

Энтропия $H(X)$ входного алфавита $X$ может быть найдена следующим образом:

$$H(X) = -\sum_{i=1}^{m} p(x_i) \log_2 p(x_i);$$

$H(X) = -(0{,}37 \cdot \log_2 0{,}37 + 0{,}3 \cdot \log_2 0{,}3 + 0{,}23 \cdot \log_2 0{,}23 +$

$+ 0{,}1 \cdot \log_2 0{,}1) = 0{,}531 + 0{,}521 + 0{,}488 + 0{,}332 =$

$= 1{,}872$ (бит/символ).

Определим условную энтропию $H(Y|X)$ выходного алфавита $Y$ относительно входного $X$ по формуле:

$$H(Y \mid X) = -\sum_{i=1}^{m} \sum_{j=1}^{n} p(x_i)\, p(y_j \mid x_i) \log_2 p(y_j \mid x_i);$$

$H(Y|X) = -[0{,}37 \cdot (0{,}97 \cdot \log_2 0{,}97 + 0{,}02 \cdot \log_2 0{,}02 +$

$+ 0{,}01 \cdot \log_2 0{,}01) + 0{,}3 \cdot (0{,}02 \cdot \log_2 0{,}02 + 0{,}98 \cdot \log_2 0{,}98) +$

$+ 0{,}23 \cdot (0{,}02 \cdot \log_2 0{,}02 + 0{,}96 \cdot \log_2 0{,}96) + 0{,}01 \cdot (0{,}02 \cdot \log_2 0{,}02 +$

$+ 0{,}96 \cdot \log_2 0{,}96) + 0{,}1 \cdot (0{,}01 \cdot \log_2 0{,}01 + 0{,}99 \cdot \log_2 0{,}99) =$

$= 0{,}37(0{,}043 + 0{,}113 + 0{,}066) + 0{,}3\ (0{,}113 + 0{,}029) +$

+ 0,23 (0,113 + 0,057) + 0,01 (0,066 + 0,014) = 0,082 +

+ 0,042 + 0,065 + 0,008 = 0,198 (бит/символ).

Отсюда получим взаимную информацию:

$I(X,Y)$ = 1,872 – 0,196 = 1,676 (бит/символ);

$I(X,Y)$ = 1,874 – 0,198 = 1,676 (бит/символ).

Таким образом, пропускная способность канала связи будет:

$C = 5000 \cdot 1{,}676 = 8381$ (бит/с). ❐

***Кодер и декодер канала связи. Код с битом четности.***

При наличии помех в канале связи обязательными частями системы передачи сообщений являются кодер и декодер канала связи.

***Кодер канала связи*** служит для представления сообщений в форме, обеспечивающей их защиту от помех при передаче по каналу либо возможных искажений при хранении информации.

Для защиты информации от помех кодер канала преобразует сообщения в ***помехоустойчивый код***, в котором к символам, несущим полезную информацию (***информационные символы***), добавлены дополнительные символы (***проверочные символы***). Значения проверочных символов помехоустойчивого кода определяются путем проведения математических операций над информационными символами.

***Декодер канала связи*** обнаруживает и исправляет ошибки в сообщениях на выходе канала, используя свойства помехоустойчивого кода. Обнаружение ошибки происходит, если декодер канала сигнализирует об отличии принятого кодового слова от переданного. Ошибка может быть исправлена, если декодер канала указывает позицию и правильное значение искаженного символа кодового слова.

Простейшим примером помехоустойчивого кода является двоичный код с битом чётности (с проверкой на чётность).

***Код с битом чётности*** – код, в котором к информационным символам, добавляется символ, принимающий значение 0 или 1 так, чтобы общее число единиц в кодовом слове было чётным. Например, к кодовому слову 10101101 следует добавить 1,

чтобы получить 110101101. Если общее число единиц в полученном кодовом слове нечётное, то в слове присутствует ошибка.

Код с битом чётности позволяет обнаружить наличие ошибки, но не позволяет исправить, поскольку неизвестно в каком знаке она произошла. Кроме того, может произойти серия ошибок, которая переведет одну разрешенную комбинацию в другую. Например, вместо 110101101 будет получено 110011101. В этом случае код с битом чётности ошибку не обнаружит.

❒ ***Пример* 3.4. *Моделирование системы передачи сообщений.***

Требуется разработать модель системы передачи сообщений, включающей такие элементы, как источник, кодер и декодер источника, канал, кодер и декодер канала.

Для решения этой задачи в приложение, полученное в примере 3.1, добавим классы **ChannelEncoder** и **ChannelDecoder**, соответствующие кодеру и декодеру канала связи. Примем, что кодер и декодер канала связи используют в своей работе код с битом чётности.

В качестве канала связи будет использовать двоичный симметричный канал. Для этого в модели канала, полученной в примере 3.1, зададим вероятность стирания символа $\varepsilon = 0$.

Структура моделируемой системы с указанием имен классов и передаваемых строковых переменных представлена на рис. 3.6.

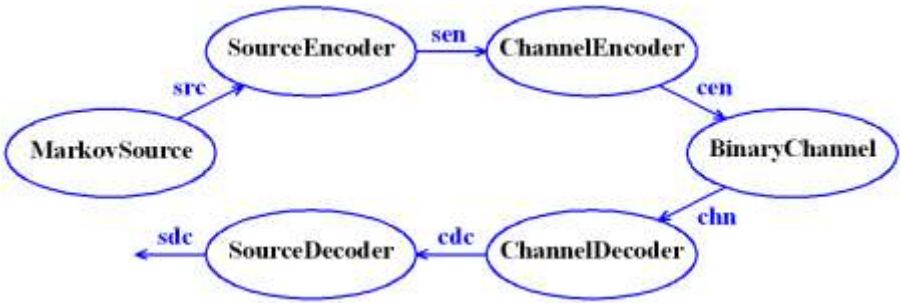


Рис. 3.6. Элементы моделируемой системы передачи сообщений

Для реализации помехоустойчивого кодирования в класс **ChannelEncoder** добавим метод **string ParityBitEncoding(string**

**cx)**, который принимает значение входной двоичной последовательности **cx**, а возвращает кодовое слово с битом четности.

Для декодирования в классе **ChannelDecoder** будет присутствовать метод **string ParityBitEncoding(string cx)**, в котором строковый параметр **cx** соответствует кодовому слову с битом четности. Данный метод должен возвращать двоичную последовательность без бита четности.

Исходный код классов **ChannelEncoder** и **ChannelDecoder** приведен в листингах 3.2 и 3.3.

Листинг 3.2. Исходный код класса **ChannelEncoder**

```
 9      /// <summary>
10      /// Представлет кодер канала связи (ККС)
11      /// </summary>
12      class ChannelEncoder
13      {
14          /// <summary>
15          /// Выполняет кодирование с битом чётности
16          /// </summary>
17          /// <param name="cx">Входная последовательность</param>
18          /// <returns>Кодовое слово с битом чётности</returns>
19          public string ParityBitEncoding(string cx)
20          {
21              int n = cx.Length; // Длина входной последовательности
22              int u = 0; // Число единиц во входной последовательности
23              for (int i = 0; i < n; i++)
24              {
25                  // Проверить, является ли символ единицей
26                  if (cx.Substring(i, 1) == "1") u++;
27              }
28              // Проверить, является ли число единиц четным,
29              if (u % 2 == 0) cx += "0"; // добавить бит чётности
30              else cx += "1";
31
32              return cx;
33          }
34      }
```

Листинг 3.3. Исходный код класса **ChannelDecoder**

```
/// <summary>
/// Представляет декодер канала связи
/// </summary>
class ChannelDecoder
{
    int errNum; // Число ошибок при декодировании

    /// <summary>
    /// Конструктор по умолчанию
    /// </summary>
    public ChannelDecoder()
    { this.errNum = 0; }

    /// <summary>
    /// Декодирует кодовые слова с битом четности
    /// </summary>
    /// <param name="cx">Входная последовательность</param>
    /// <returns>Выходная последовательность</returns>
    public string ParityBitDecoding(string cx)
    {
        int n = cx.Length; // Длина входной последовательности
        int u = 0; // Число единиц во входной последовательности
        for (int i = 0; i < n; i++)
        {
            if (cx.Substring(i, 1) == "1") u++;
        }
        // Проверить, является ли число единиц четным
        if (u % 2 == 0)
        {
            cx = cx.Remove(n - 1, 1); // Удалить бит чётности
        }
        else
        {
            errNum++;
            cx = "111"; // Ошибочная последовательность
        }

        return cx;
    }

    /// <summary>
    /// Возвращает строку с данными об ошибках декодирования
    /// </summary>
    /// <returns>Данные об ошибках</returns>
    public string GetErrorNumber()
    {
        return string.Format("Число обнаруженных ошибок: " + errNum);
    }
}
```

Исходный код класса **Program** консольного приложения представлен в листинге 3.4.

Листинг 3.4. Исходный код класса **Program** консольного приложения

```
class Program
{
    static void Main(string[] args)
    {
        Console.Title = "Моделирование системы передачи сообщений";
        // Экземпляр класса марковский источник сообщений (ИС)
        MarkovSource mSrc = new MarkovSource();
        // Экземпляр класса кодер источника сообщений (КИС)
        SourceEncoder sEnc = new SourceEncoder();
        // Экземпляр класса кодер канала связи (ККС)
        ChannelEncoder cEnc = new ChannelEncoder();
        // Экземпляр класса двоичный канал связи (КС)
        BinaryChannel bChn = new BinaryChannel();
        // Экземпляр класса декодер канала связи (ДКС)
        ChannelDecoder cDec = new ChannelDecoder();
        // Экземпляр класса декодер источника сообщений (ДИС)
        SourceDecoder sDec = new SourceDecoder();

        Console.WriteLine("Введите число генерируемых символов:");
        string buf = Console.ReadLine();
        int n = int.Parse(buf); // Число генерируемых символов

        string src = ""; // Сообщение на выходе ИС
        string sen = ""; // Сообщение на выходе КИС
        string cen = ""; // Сообщение на выходе ККС
        string chn = ""; // Сообщение на выходе КС
        string cdc = ""; // Сообщение на выходе ДКС
        string sdc = ""; // Сообщение на выходе ДИС
        // Строка форматного вывода
        buf = "|{0,5}|{1,7}|{2,7}|{3,7}|{4,7}|{5,7}|{6,7}|";

        Console.WriteLine("\nСообщения на выходе элементов системы:");
        Console.WriteLine(buf, "№", "ИС", "КИС", "ККС", "КС", "ДКС", "ДИС");
        for (int i = 0; i < n; i++)
        {
            src = mSrc.GenerSimbol();
            sen = sEnc.SimpleEncoding(src);
            cen = cEnc.ParityBitEncoding(sen);
            chn = bChn.TransmitBits(cen);
            cdc = cDec.ParityBitDecoding(chn);
            sdc = sDec.SimpleDecoding(cdc);
            Console.WriteLine(buf, i + 1, src, sen, cen, chn, cdc, sdc);
        }

        Console.WriteLine(mSrc.GetFreqs());
        Console.WriteLine(bChn.GetErrorNumber());
        Console.WriteLine(cDec.GetErrorNumber());
        Console.WriteLine(sDec.GetErrorNumber());
        Console.Read();
    }
}
```

Результат работы консольного приложения для введенного числа символов показан на рис. 3.6. Из результата видно, что де-

кодер канала связи обнаружил одиночную ошибку в сообщении 9, но пропустил двойную ошибку в сообщении 4. ❒

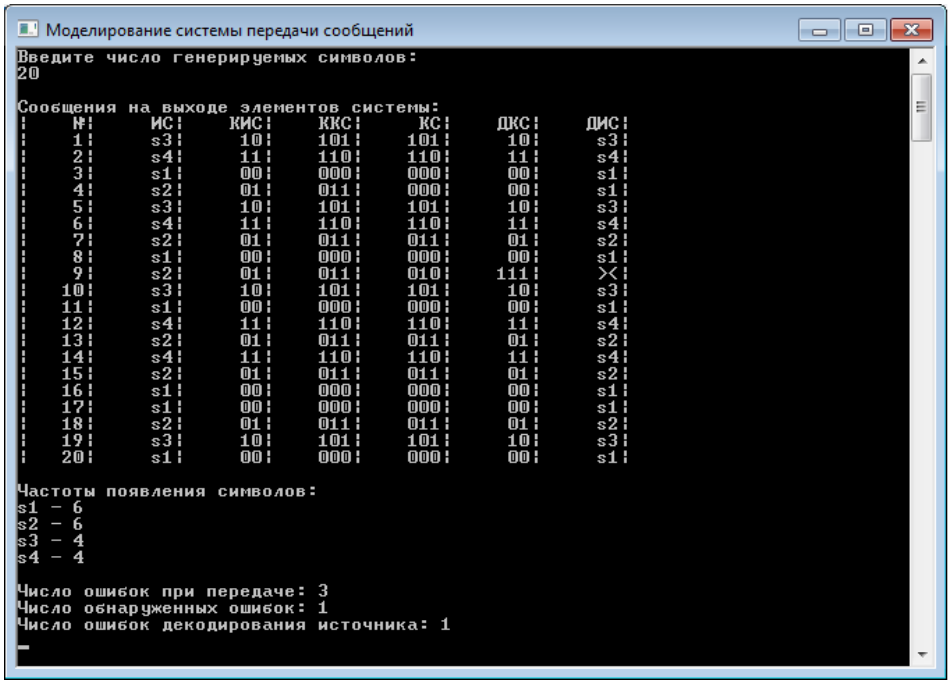


Рис. 3.6. Результат работы консольного приложения

## 3.3. Порядок выполнения работы и варианты заданий

***Основные этапы выполнения работы.***

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать модель двоичного канала связи со стиранием (с параметрами: $p$ = 0,02, $\varepsilon$ = 0.01). Добавить полученную модель к системе из работы №2. Для построенной модели системы передачи сообщений определить число ошибок в полученном сообщении при разных значениях вероятностей $p$ и $\varepsilon$.
3. На основе заданного распределения вероятностей символов входного алфавита $X$ (таблица 3.1) и матрицы переходных вероятностей $p(y|x)$ канала связи (таблица 3.2) найти информационные потери при передаче 500 символов, а также количество принятой информации.

4. Используя описание дискретного канала связи без памяти (таблицы 3.1 и 3.2) определить его пропускную способность $C$. Среднее время передачи одного символа принять $\tau = 1 \cdot 10^{-4}$ с.

5. Разработать модели кодера и декодера двоичного канала связи. В качестве помехоустойчивого кода принять код с битом четности. Добавить полученные модели элементов к системе из п. 2. Промоделировать полную систему передачи сообщений и рассмотреть влияние ошибок в канале связи на работу декодеров.

6. Оформить и защитить отчет по лабораторной работе.

***Индивидуальные варианты заданий.***

Таблица 3.1

Вероятности появления символов входного алфавита

| № вар. | Входной алфавит X | | | | № вар. | Входной алфавит X | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
| **1** | 0,23 | 0,27 | 0,39 | 0,11 | **13** | 0,24 | 0,16 | 0,28 | 0,32 |
| **2** | 0,21 | 0,32 | 0,09 | 0,38 | **14** | 0,15 | 0,41 | 0,35 | 0,09 |
| **3** | 0,18 | 0,42 | 0,25 | 0,15 | **15** | 0,25 | 0,35 | 0,09 | 0,31 |
| **4** | 0,08 | 0,26 | 0,32 | 0,34 | **16** | 0,40 | 0,17 | 0,20 | 0,23 |
| **5** | 0,37 | 0,13 | 0,09 | 0,41 | **17** | 0,40 | 0,10 | 0,17 | 0,33 |
| **6** | 0,16 | 0,31 | 0,24 | 0,29 | **18** | 0,36 | 0,08 | 0,14 | 0,42 |
| **7** | 0,10 | 0,40 | 0,31 | 0,19 | **19** | 0,19 | 0,31 | 0,08 | 0,42 |
| **8** | 0,24 | 0,27 | 0,16 | 0,33 | **20** | 0,34 | 0,18 | 0,26 | 0,22 |
| **9** | 0,15 | 0,25 | 0,42 | 0,18 | **21** | 0,31 | 0,19 | 0,41 | 0,09 |
| **10** | 0,17 | 0,42 | 0,33 | 0,08 | **22** | 0,37 | 0,15 | 0,23 | 0,25 |
| **11** | 0,40 | 0,30 | 0,08 | 0,22 | **23** | 0,22 | 0,08 | 0,29 | 0,41 |
| **12** | 0,26 | 0,31 | 0,24 | 0,19 | **24** | 0,12 | 0,30 | 0,18 | 0,40 |

Таблица 3.2

Матрицы переходных вероятностей $p(y_j|x_i)$ канала связи

| № вар. | X | Выходной алфавит Y | | | | № вар. | X | Выходной алфавит Y | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $y_1$ | $y_2$ | $y_3$ | $y_4$ | | | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
| **1, 9, 17** | $x_1$ | 0,95 | 0,03 | 0,02 | 0 | **5, 13, 20** | $x_1$ | 0,98 | 0,02 | 0 | 0 |
| | $x_2$ | 0,02 | 0,97 | 0,01 | 0 | | $x_2$ | 0,03 | 0,96 | 0,01 | 0 |
| | $x_3$ | 0 | 0 | 0,99 | 0,01 | | $x_3$ | 0 | 0,02 | 0,94 | 0,04 |
| | $x_4$ | 0 | 0,02 | 0,04 | 0,94 | | $x_4$ | 0 | 0,01 | 0,02 | 0,97 |
| **2, 10,** | $x_1$ | 0,97 | 0,02 | 0,01 | 0 | **6, 14,** | $x_1$ | 0,94 | 0,03 | 0,02 | 0,01 |
| | $x_2$ | 0,02 | 0,98 | 0 | 0 | | $x_2$ | 0,01 | 0,99 | 0 | 0 |

| № вар. | X | Выходной алфавит Y | | | | № вар. | X | Выходной алфавит Y | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $y_1$ | $y_2$ | $y_3$ | $y_4$ | | | $y_1$ | $y_2$ | $y_3$ | $y_4$ |
| **18** | $x_3$ | 0 | 0,02 | 0,95 | 0,03 | **22** | $x_3$ | 0 | 0,03 | 0,96 | 0,01 |
| | $x_4$ | 0 | 0 | 0,01 | 0,99 | | $x_4$ | 0 | 0,01 | 0,04 | 0,95 |
| **3, 11, 19** | $x_1$ | 0,96 | 0,03 | 0,01 | 0 | **7, 15, 23** | $x_1$ | 0,99 | 0,01 | 0 | 0 |
| | $x_2$ | 0,03 | 0,95 | 0,02 | 0 | | $x_2$ | 0,02 | 0,94 | 0,03 | 0,01 |
| | $x_3$ | 0 | 0,01 | 0,97 | 0,02 | | $x_3$ | 0 | 0,02 | 0,98 | 0 |
| | $x_4$ | 0 | 0 | 0,02 | 0,98 | | $x_4$ | 0 | 0,01 | 0,03 | 0,96 |
| **4, 12, 20** | $x_1$ | 0,98 | 0 | 0,01 | 0,01 | **8, 16, 24** | $x_1$ | 0,95 | 0,02 | 0 | 0,03 |
| | $x_2$ | 0,02 | 0,95 | 0 | 0,03 | | $x_2$ | 0,01 | 0,98 | 0,01 | 0 |
| | $x_3$ | 0 | 0,03 | 0,96 | 0,01 | | $x_3$ | 0 | 0,03 | 0,96 | 0,01 |
| | $x_4$ | 0 | 0,01 | 0 | 0,99 | | $x_4$ | 0,01 | 0 | 0,02 | 0,97 |

***Требования к отчёту.***

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код программы для моделирования двоичного канала связи со стиранием, а также результаты моделирования (число ошибок при передаче заданного числа символов по каналу связи).
4. Расчёт информационных потерь при передаче через канал связи с помехами.
5. Расчёт пропускной способности канала связи.
6. Выводы по лабораторной работе.

## 3.4. Контрольные вопросы и задачи

***Теоретические вопросы.***

1. Что понимают под каналом связи?
2. Какие каналы связи называют дискретными?
3. Что понимают под дискретным каналом без памяти?
4. Как строится и что характеризует матрица переходных вероятностей канала связи без памяти?
5. Как описываются двоичный симметричный канал связи и двоичный канал связи со стиранием?
6. Как определяются информационные потери при передаче сообщений по каналу связи с помехами?

7. Как определяют скорость передачи информации по дискретному каналу связи?

8. Что характеризует пропускная способность канала связи?

9. В каком случае пропускная способность канала связи равна нулю?

10. Как определяется пропускная способность дискретного канала при отсутствии помех?

11. Как определяется пропускная способность дискретного канала с помехами?

12. Для чего предназначен кодер канала связи?

13. В чем заключается помехоустойчивое кодирование?

14. Каково назначение декодера канала связи?

15. Как строится и работает код с битом чётности?

***Практические задачи.***

Задан дискретный канал связи без памяти, в котором присутствуют помехи. Модель канала связи представлена ансамблем входного алфавита $\boldsymbol{X}$ и матрицей переходных вероятностей $\boldsymbol{P}(\boldsymbol{Y}|\boldsymbol{X})$ (табл. 3.3).

1. Требуется определить информационные потери $\Delta I$ при передаче 1000 символов через заданный канал связи, а также общее количество полученной информации.

2. Требуется определить пропускную способность $C$ канала связи, если среднее время передачи одного символа $\tau_{ср}$ составляет 0,0005 с.

Таблица 3.3

Варианты заданий для решения задач

| **Нечётный вариант** | **Чётный вариант** |
|---|---|
| $X = \begin{pmatrix} x_1 & x_2 & x_3 \\ 0{,}15 & 0{,}6 & 0{,}25 \end{pmatrix};$ | $X = \begin{pmatrix} x_1 & x_2 & x_3 \\ 0{,}45 & 0{,}35 & 0{,}2 \end{pmatrix};$ |
| $P(Y \mid X) = \begin{bmatrix} 0{,}85 & 0 & 0{,}15 \\ 0{,}1 & 0{,}9 & 0 \\ 0 & 0{,}05 & 0{,}95 \end{bmatrix}.$ | $P(Y \mid X) = \begin{bmatrix} 0{,}95 & 0 & 0{,}05 \\ 0{,}1 & 0{,}9 & 0 \\ 0 & 0{,}15 & 0{,}85 \end{bmatrix}.$ |

# 4. СЖАТИЕ ИНФОРМАЦИИ МЕТОДОМ АРИФМЕТИЧЕСКОГО КОДИРОВАНИЯ

## 4.1. Цель и задачи работы

Цель работы – приобрести умение сжимать информацию с помощью метода арифметического кодирования, а также умение декодировать арифметический код.

Основные задачи работы:

• освоить сжатие последовательностей символов методом арифметического кодирования;

• научиться декодировать арифметический код.

Работа рассчитана на 4 часа.

## 4.2. Основные теоретические сведения

### 4.2.1. Метод арифметического кодирования. Декодирование арифметического кода

***Описание метода арифметического кодирования.***

***Сжатием информации*** называется операция, в результате которой данному коду или сообщению ставится более короткий код или сообщение.

Целью сжатия информации является ускорение и удешевление процессов обработки, хранения и передачи информации.

***Арифметическое кодирование*** представляет собой метод сжатия информации, в котором кодируемая последовательность символов представляется в виде дробного числа $x$, принимающего значения из интервала $0 < x < 1$.

Пусть задан дискретный источник сообщений $[S, p(s_i)]$ с алфавитом $S = \{s_1, \ldots, s_m\}$ и распределением вероятностей $p(s_i)$. На выходе источника задана последовательность символов

$$\bar{s} = (s_{i_1}, s_{i_2}, ..., s_{i_k}, ..., s_{i_M});$$

где $i$ – порядковый номер символа в алфавите; $k$ – порядковый номер символа в последовательности.

Для заданной последовательности $\bar{s}$ требуется получить кодовое слово:

$$\bar{x} = (x_{j_1}, x_{j_2}, ..., x_{j_L});$$

где $x_j$ = 0 или 1.

Определим величины, называемые ***кумулятивными вероятностями*** символов $s_i$:

$$q(s_1) = 0; \quad q(s_2) = p(s_1); \quad \ldots \quad q(s_m) = \sum_{i=1}^{m-1} p_i \, .$$

В рекурсивной форме данные вероятности можно записать следующим образом:

$$q(s_i) = q(s_{i-1}) + p(s_{i-1}). \qquad (4.1)$$

Определим величины $F_k$ и $G_k$, представляющие собой ***нижнюю границу*** и ***длину интервала вероятностей*** на шаге $k$:

$$F_k = F_{k-1} + q(s_i) \cdot G_{k-1}; \qquad (4.2)$$

$$G_k = p(s_i) \cdot G_{k-1}. \qquad (4.3)$$

При $k$ = 0 (начальный шаг алгоритма) принимают $F$=0, $G$=1.

Искомое кодовое слово $\bar{x}$ определяется как $L$знаков после запятой в двоичной записи числа$F_M + G_M/2$, где $F_M G_M$ –нижняя граница интервала вероятностей и его длина, соответствующие последнему ($M$-му) символу последовательности:

$$\text{bin}[F_M + G_M/2] = 0, x_{j_1} x_{j_2} ... x_{j_L} \rightarrow \bar{x} = (x_{j_1}, x_{j_2}, ..., x_{j_L}) \qquad (4.4)$$

Таким образом, метод арифметического кодирования заключается в последовательном нахождении величин $q(s_i)$, $F_k$, $G_k$, для $i$ = 2, 3, … , $m$ и $k$ = 1, 2,… , $M$, пока не будут определены значения $F_M$ и $G_M$. На основе последних значений $F_M$ и $G_M$ определяется искомое кодовое слово.

Длина $L$ кодового слова $\bar{x}$, обеспечивающая однозначное декодирование последовательности символов определяется следующим образом:

$$L = \lceil -\log_2 G_M \rceil + 1. \qquad (4.5)$$

где $\lceil ... \rceil$ – округление числа, стоящего в скобках, до целого в большую сторону.

Для пояснения метода арифметического кодирования можно прибегнуть к его графической интерпретации. В этом случае интервал [0, 1) вероятностей появления символов рассматривается в виде отрезка длиной 1. Символам на данном отрезке будут соответствовать непересекающиеся полуинтервалы с длинами, равными вероятностям появления символов (рис 4.1). Кумулятивные вероятности соответствуют началам этих полуинтервалов. Эти точки идентифицируют сообщения источника.

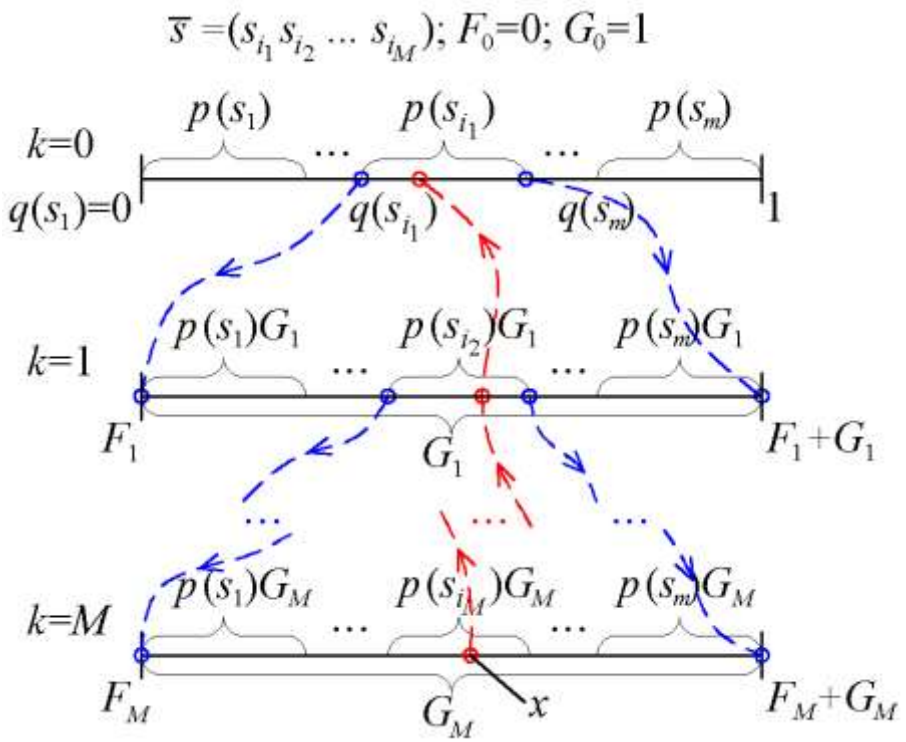


Рис. 4.1. Графическая интерпретация арифметического кодирования

На каждом шаге алгоритма производится пересчет границ всего отрезка, в котором будет расположено число, соответствующее кодовому слову. Число $F_k$ в этом случае является начальной точкой отрезка на шаге $k$, а число $G_k$ является длиной данного отрезка.

❒ ***Пример* 4.1. *Арифметическое кодирование заданной последовательности символов.***

Пусть задан дискретный источник сообщений с алфавитом $S=\{s_1, s_2, s_3\}$и распределением вероятностей $p(s_1)=0{,}1$, $p(s_2)=0{,}6$, $p(s_3)=0{,}3$. Произведем арифметическое кодирование последовательности $\bar{s} = (s_2 s_3 s_2 s_1 s_2)$ длиной $M=5$. Процесс арифметического кодирования последовательности $\bar{s}$ представлен в таблице 4.1.

Таблица 4.1.

Арифметическое кодирование последовательности

| **Шаг $k$** | $s_i$ | $\bar{s}_{i_k}$ | $p(s_i)$ | $q(s_i)$ | $F(\bar{s}_{i_k})$ | $G(\bar{s}_{i_k})$ |
|---|---|---|---|---|---|---|
| 0 | – | – | – | – | 0 | 1 |
| 1 | $s_2$ | $s_2$ | 0,6 | 0,1 | 0,1 | 0,6 |
| 2 | $s_3$ | $s_2 s_3$ | 0,3 | 0,7 | 0,52 | 0,18 |
| 3 | $s_2$ | $s_2 s_3 s_2$ | 0,6 | 0,1 | 0,538 | 0,108 |
| 4 | $s_1$ | $s_2 s_3 s_2 s_1$ | 0,1 | 0 | 0,538 | 0,0108 |
| 5 | $s_2$ | $s_2 s_3 s_2 s_1 s_2$ | 0,6 | 0,1 | 0,53908 | 0,00648 |

Длина кодового слова будет

$$L = \lceil -\log_2 0{,}00648 \rceil + 1 = \lceil 7{,}2698 \rceil + 1 = 9,$$

то есть после запятой в кодовом слове должно быть не меньше 9 знаков.

Отсюда кодовое слово запишется следующим образом:

$x = (0{,}53908 + 0{,}00648/2)_{10} = (0{,}54232)_{10} =$

$= 0{,}100010101_2 \rightarrow \bar{x} = 100010101.$

Графическая интерпретация арифметического кодирования показана на рис. 5.2. ❐

***Декодирование арифметического кода.***

Декодирование полученного арифметического кода заключается в выполнении следующих действий.

На нулевом шаге ($k = 0$) принимаем $q(s_{M+1}) = 1$, $F_k = 0$, $G_k = 1$. Последовательно (от $i = 1$ до $m$) для символов $s_i$ на каждом шаге производим проверку выполнения условия:

$$F_k + q(s_i) \cdot G_k < x.$$

Если для символа $s_i$ данное условие не выполнятся, то в качестве символа декодируемой последовательности на данном шаге принимаем $s_{i-1}$. В обратном случае (условие выполняется для всех$i$) принимаем символ $s_m$.

Для найденного символа определяют величины $F_k$ и $G_k$ по тем же формулам, что и при кодировании.

Алгоритм продолжают до тех пор, пока не будет найден последний символ ($k = M$) декодируемой последовательности $\bar{s}$.

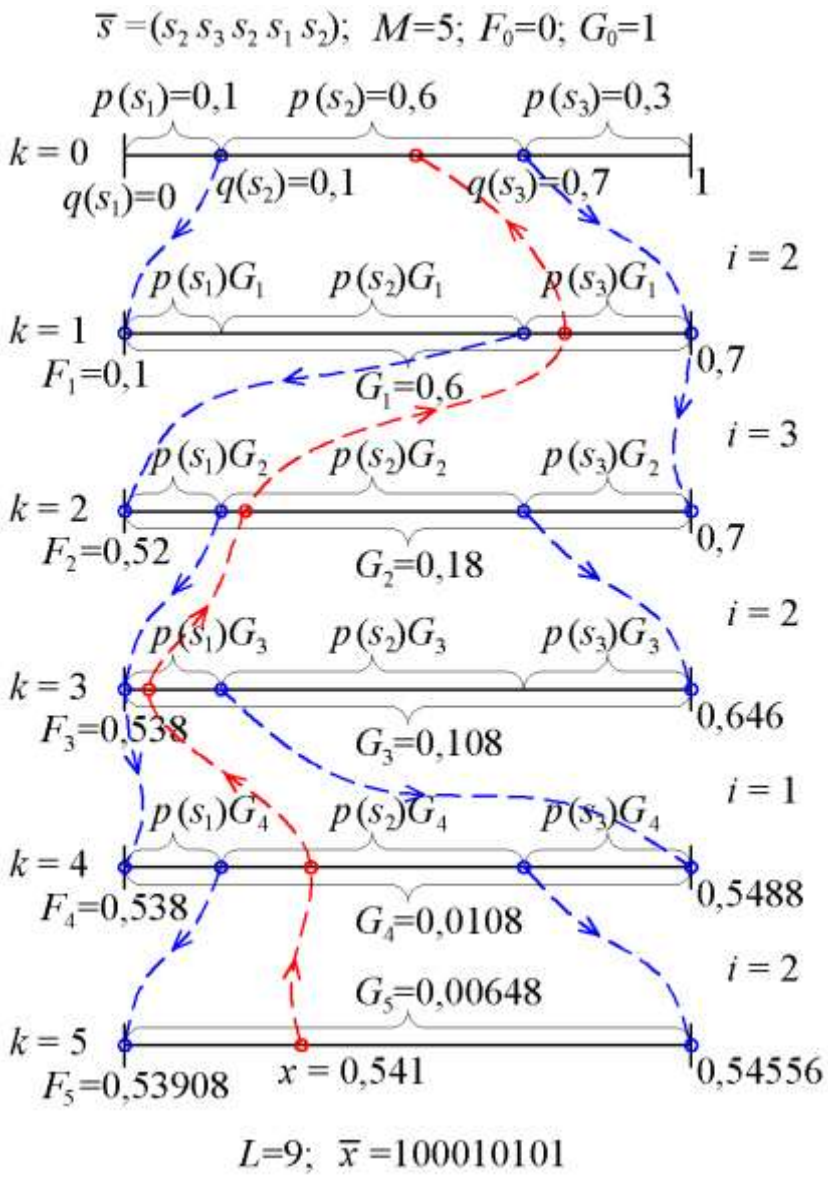


Рис. 4.2. Графическая интерпретация арифметического кодирования последовательности $\bar{s}=(s_2s_3s_2s_1s_2)$

❒ ***Пример* 4.2. *Декодирование арифметического кода.***

Требуется выполнить декодирование арифметического кода $\bar{x}$ = 100010101, полученного в примере 4.1.

Процесс декодирования представлен в таблице 4.2.

Таблица 4.2

Декодирование арифметического кода $\bar{x}$ = 100010101

| Шаг | $F_k$ | $G_k$ | Гипо- | $q(s_i)$ | Проверка | Реше- | $p(s_i)$ |
|---|---|---|---|---|---|---|---|

| $k$ | | | теза $s_i$ | | $F_k+q_i G_k<x$ | ние $s_i$ | |
|---|---|---|---|---|---|---|---|
| 0 | $\overline{x} = 100010101 \rightarrow x = 0{,}100010101_2 = 0{,}541_{10}$ | | | | | | |
| 1 | 0 | 1 | $s_1$ | 0 | $0 < x$ | $s_2$ | 0,6 |
| | | | $s_2$ | 0,1 | $\mathbf{0{,}1} < x$ | | |
| | | | $s_3$ | 0,7 | $0{,}7 > x$ | | |
| 2 | 0,1 | 0,6 | $s_1$ | 0 | $0{,}1 < x$ | $s_3$ | 0,3 |
| | | | $s_2$ | 0,1 | $0{,}16 < x$ | | |
| | | | $s_3$ | 0,7 | $\mathbf{0{,}52} < x$ | | |
| 3 | 0,52 | 0,18 | $s_1$ | 0 | $0{,}52 < x$ | $s_2$ | 0,6 |
| | | | $s_2$ | 0,1 | $\mathbf{0{,}538} < x$ | | |
| | | | $s_3$ | 0,7 | $0{,}646 > x$ | | |
| 4 | 0,538 | 0,108 | $s_1$ | 0 | $\mathbf{0{,}538} < x$ | $s_1$ | 0,1 |
| | | | $s_2$ | 0,1 | $0{,}5488 > x$ | | |
| 5 | 0,538 | 0,0108 | $s_1$ | 0 | $0{,}538 < x$ | $s_2$ | 0,6 |
| | | | $s_2$ | 0,1 | $\mathbf{0{,}53908} < x$ | | |
| | | | $s_3$ | 0,7 | $0{,}5450 > x$ | | |

Отсюда последовательность символов, полученная путем декодирования арифметического кода, будет $s_2 s_3 s_2 s_1 s_2$, что полностью совпадает с данными из примера 4.1. ❐

### 4.2.2. Метод адаптивного арифметического кодирования

***Адаптивное арифметическое кодирование***

В случае адаптивного арифметического кодирования кодеру при поступлении на его вход очередного сообщения не доступна информация о сообщениях, которые появятся в будущем. Поэтому способ кодирования текущего сообще-

ния зависит только от того, какими были предыдущие сообщения.

Вместо вероятностей появления символов $s_1,\ldots, s_m$ при адаптивном арифметическом кодировании используются оценочные вероятности:

$$\tilde{p}_M(s_i) = \frac{\tau_M(s_i)+1}{M+m};$$

где $\tau_M(s_i)$ – число появлений символа $s_i$ в последовательности длины $M$; $m$ – объем алфавита.

❒ ***Пример* 4.3. *Программная реализация метода арифметического кодирования*.**

Требуется разработать программную реализацию метода арифметического кодирования для последовательностей символов заданной длины, а также декодирование арифметического кода.

Добавим в класс **SourceS** (пример 2.1) метод **string GenerBlock(int _d)**, который возвращает последовательности символов заданной длины **_d**. В этом методе отдельные символы будут добавляться в последовательность путем многократного вызова метода **GenerSimbol()**.

Исходный код метода **GenerBlock()** приведён в листинге 4.1.

Листинг 4.1. Исходный код метода **GenerBlock()** класса **SourceS**

```
59          /// <summary>
60          /// Возвращает случайную последовательность символов
61          /// </summary>
62          /// <param name="_d">Длина последовательности</param>
63          /// <returns>Последовательность символов</returns>
64          public string GenerBlock(int _d)
65          {
66              string sd = ""; // Выходная последовательность символов
67              for (int i = 0; i < _d; i++)
68              {
69                  sd += GenerSymbol();
70              }
71              return sd;
72          }
```

В класс **SourceEncoder** добавим массив вероятностей **pe**, а также метод **string ArifmEncoding(string sn)**.

В конструкторе класса **SourceEncoder** элементы массива **pe** инициализируем значениями вероятностей, полученных в примере 2.1.

Исходный код модифицированного класса **SourceEncoder** приведён в листингах 4.2 и 4.3.

Листинг 4.2. Исходный код класса **SourceEncoder** (Часть 1)

```
/// <summary>
/// Представляет кодер источника сообщений
/// </summary>
class SourceEncoder
{
    const int m = 4; // Объём алфавита источника сообщений
    double[] pe;     // Оценочные вероятности символов источника

    /// <summary>
    /// Конструктор с параметрами по умолчанию
    /// </summary>
    public SourceEncoder()
    {
        pe = new double[m];
        pe[0] = 0.242; pe[1] = 0.271; pe[2] = 0.250; pe[3] = 0.237;
    }

    /// <summary>
    /// Возвращает кодовое слово арифметического кода
    /// </summary>
    /// <param name="sn">Последовательность символов</param>
    /// <returns>Арифметический код</returns>
    public string ArifmEncoding(string sn)
    {
        int n = sn.Length / 2; // Число символов в последовательности
        int[] ix = new int[n]; // Массив индексов символов
        // Цикл для заполнения массива ix
        for (int k = 0; k < n; k++)
        {
            // Выделение символа из входной последовательности
            switch (sn.Substring(k * 2, 2))
            {
                case "s1": ix[k] = 0; break;
                case "s2": ix[k] = 1; break;
                case "s3": ix[k] = 2; break;
                case "s4": ix[k] = 3; break;
            }
        }
        double qi = 0; // Куммулятивная вероятность символа si
        double fk = 0; // Нижняя граница интервала вероятностей
        double gk = 1; // Длина интевала вероятностей
        // Цикл для вычисления нижней границы и длины интервала
        for (int k = 0; k < n; k++)
        {
            // Цикл для определения куммулятивной вероятности
            for (int i = 0; i < ix[k]; i++) qi += pe[i];
            fk += qi * gk;
            gk *= pe[ix[k]];
            qi = 0;
        }
        double x = fk + gk / 2; // Числовое значение арифмет. кода
        // Длина арифметического кода
        int d = (int)Math.Round(-Math.Log(gk, 2)) + 1;
        return BinaryTransform(x, d);
    }
```

Листинг 4.3. Исходный код класса **SourceEncoder** (Часть 2)

```
65          /// <summary>
66          /// Переводит десятичную дробь в двоичный код
67          /// </summary>
68          /// <param name="x">Десятичное дробное число</param>
69          /// <param name="d">Длина двоичного числа</param>
70          /// <returns>Двоичное число</returns>
71          private string BinaryTransform(double _x, int _d)
72          {
73              string b = ""; // Двоичное число
74              for (int j = 0; j < _d; j++)
75              {
76                  _x *= 2;
77                  if (_x >= 1)
78                  { b += "1"; _x -= 1; }
79                  else b += "0";
80              }
81              return b;
82          }
```

В класс **SourceDecoder** требуется добавить строковый массив **s**, содержащий алфавит символов источника, а также массив вероятностей **pe**. Кроме того, в класс необходимо добавить метод **string ArifmDecoding(string a, int d)**, выполняющий декодирование слова арифметического кода **a** в исходную последовательность символов длиной **d**.

Исходный код модифицированного класса **SourceDecoder** представлен в листингах 4.4 и 4.5.

Листинг 4.4. Исходный код класса **SourceDecoder** (Часть 1)

```
9   /// <summary>
10  /// Представляет декодер источника сообщений
11  /// </summary>
12  class SourceDecoder
13  {
14      const int m = 4; // Объём алфавита источника сообщений
15      string[] s;      // Алфавит источника сообщений
16      double[] pe;     // Оценочные вероятности символов
17      int errNum;      // Число ошибок декодирования
18
19      /// <summary>
20      /// Конструктор с параметрами по умолчанию
21      /// </summary>
22      public SourceDecoder()
23      {
24          s = new string[m] { "s1", "s2", "s3", "s4" };
25          pe = new double[m];
26          pe[0] = 0.242; pe[1] = 0.271; pe[2] = 0.250; pe[3] = 0.237;
27          errNum = 0;
28      }
```

Листинг 4.5. Исходный код класса **SourceDecoder** (Часть 2)

```
30          /// <summary>
31          /// Декодирует кодовое слово арифметического кода
32          /// </summary>
33          /// <param name="_a">Слово арифметического кода</param>
34          /// <param name="_d">Длина декодир. последовательности</param>
35          /// <returns>Декодированная последовательность</returns>
36          public string ArifmDecoding(string _a, int _d)
37          {
38              double[] q = new double[m]; // Массив куммулятивных вероятностей
39              q[0] = 0.0;
40              // Цикл для заполнения массива q
41              for (int i = 1; i < m; i++)
42              { q[i] = q[i - 1] + pe[i - 1]; }
43
44              // Десятичное число, соответствующее _a
45              double x = DecimalTransform(_a);
46
47              string sd = "";  // Декодированная последовательность
48              string sk = "";  // Произвольный символ на шаге k
49              double fk = 0.0; // Нижняя граница интервала вероятностей
50              double gk = 1.0; // Длина интервала вероятностей
51              bool u = false;  // Результат проверки
52              int ik = 0;      // Индекс символа на шаге k
53              // Цикл для получения последовательности sn
54              for (int k = 0; k < _d; k++)
55              {
56                  // Цикл для определения символа на шаге k
57                  for (int i = 0; i < m; i++)
58                  {
59                      if (fk + q[i] * gk > x)
60                      {
61                          sk = s[i - 1];
62                          ik = i - 1;
63                          u = true; break;
64                      }
65                  }
66                  if (u == false) // Выбор последнего символа алфавита
67                  {
68                      sk = s[m - 1];
69                      ik = m - 1;
70                  }
71                  sd += sk;
72                  fk += q[ik] * gk;
73                  gk *= pe[ik];
74                  u = false;
75              }
76              return sd;
77          }
78
79          /// <summary>
80          /// Переводит двоичный код в десятичное число
81          /// </summary>
82          /// <param name="_b">Двоичное кодовое слово</param>
83          /// <returns>Десятичное дробное число</returns>
84          private double DecimalTransform(string _b)
85          {
86              int n = _b.Length; // Длина кодового слова
87              int xj = 0; // Значение (0 или 1) символа кодового слова
88              double x = 0.0; // Десятичное дробное число
89              for (int j = 0; j < n; j++)
90              {
91                  xj = int.Parse(_b.Substring(j, 1));
92                  x += xj * Math.Pow(2, -j - 1);
93              }
94              return x;
95          }
```

В методе **Main()** консольного приложения требуется обеспечить форматный вывод на консоль исходной последовательности, кодового слова и декодированной последовательности на каждом шаге.

Исходный код класса **Program** консольного приложения приведён в листинге 4.6.

Листинг 4.6. Исходный код класса **Program** консольного приложения

```
 9    class Program
10    {
11        static void Main(string[] args)
12        {
13            Console.Title = "Арифметическое кодирование";
14            // Экземпляр класса марковский источник сообщений (ИС)
15            SourceS mSrc = new SourceS();
16            // Экземпляр класса кодер источника сообщений (КИС)
17            SourceEncoder sEnc = new SourceEncoder();
18            // Экземпляр класса декодер источника сообщений (ДИС)
19            SourceDecoder sDec = new SourceDecoder();
20
21            Console.WriteLine("Введите число последовательностей символов:");
22            string buf = Console.ReadLine();
23            int n = int.Parse(buf); // Число последовательностей
24            Console.WriteLine("Введите длину последовательности символов:");
25            buf = Console.ReadLine();
26            int d = int.Parse(buf); // Длина одной последовательности
27
28            string src = ""; // Сообщение на выходе ИС
29            string sen = ""; // Сообщение на выходе КИС
30            string sdc = ""; // Сообщение на выходе ДИС
31            // Строка форматного вывода
32            buf = "|{0,5}|{1,20}|{2,20}|{3,20}|";
33
34            Console.WriteLine("\nСообщения на выходе элементов системы:");
35            Console.WriteLine(buf, "№", "ИС", "КИС", "ДИС");
36            for (int i = 0; i < n; i++)
37            {
38                src = mSrc.GenerBlock(d);
39                sen = sEnc.ArifmEncoding(src);
40                sdc = sDec.ArifmDecoding(sen, d);
41                Console.WriteLine(buf, i + 1, src, sen, sdc);
42            }
43
44            Console.WriteLine(mSrc.GetFreqs());
45            Console.WriteLine(sDec.GetErrorNumber());
46            Console.Read();
47        }
48    }
```

Результат работы консольного приложения при числе последовательностей $n = 7$ и длине каждой последовательности $M = 8$ показан на рис. 4.3. ❒

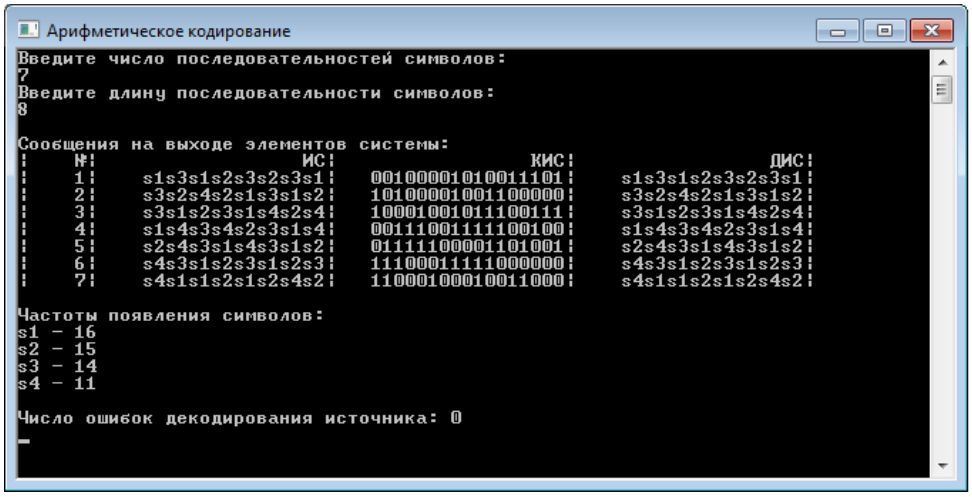


Рис. 4.3. Результат работы консольного приложения

## 4.3. Порядок выполнения работы и варианты заданий

***Основные этапы выполнения работы.***

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Произвести арифметическое кодирование заданной последовательности символов (табл. 4.3). Результат кодирования представить в виде двоичного кодового слова.
3. Осуществить декодирование полученного арифметического кода. Построить графическую интерпретацию арифметического кодирования.
4. Программно реализовать методы построения и декодирования арифметического кода. Продемонстрировать процесс арифметического кодирования и последующего декодирования.
5. Оформить и защитить отчет по лабораторной работе.

***Индивидуальные варианты заданий.***

Таблица 4.3

Кодируемая последовательность символов и вероятности их появления

| **Вар.** | $\bar{S}$ | $p(s_i)$ | **Вар.** | $\bar{S}$ | $p(s_i)$ |
|---|---|---|---|---|---|
| **1** | $s_2s_4s_1s_2s_3s_1s_4$ | $p(s_1) = 0{,}1$ | **13** | $s_3s_1s_2s_3s_1s_3s_4$ | $p(s_1) = 0{,}3$ |
| **2** | $s_4s_2s_1s_3s_2s_4s_1$ | $p(s_2) = 0{,}4$ | **14** | $s_1s_3s_1s_2s_3s_4s_3$ | $p(s_2) = 0{,}2$ |
| **3** | $s_1s_4s_3s_2s_4s_1s_2$ | $p(s_3) = 0{,}2$ | **15** | $s_4s_3s_1s_3s_1s_2s_3$ | $p(s_3) = 0{,}4$ |
| **4** | $s_4s_1s_2s_3s_1s_2s_4$ | $p(s_4) = 0{,}3$ | **16** | $s_3s_4s_3s_1s_3s_2s_1$ | $p(s_4) = 0{,}1$ |
| **5** | $s_1s_3s_1s_4s_3s_2s_1$ | $p(s_1) = 0{,}4$ | **17** | $s_2s_3s_1s_2s_4s_4s_2$ | $p(s_1) = 0{,}2$ |
| **6** | $s_3s_1s_4s_1s_2s_1s_3$ | $p(s_2) = 0{,}1$ | **18** | $s_4s_2s_2s_3s_1s_2s_4$ | $p(s_2) = 0{,}4$ |
| **7** | $s_4s_3s_1s_2s_1s_3s_1$ | $p(s_3) = 0{,}3$ | **19** | $s_4s_3s_2s_4s_2s_2s_1$ | $p(s_3) = 0{,}1$ |
| **8** | $s_1s_4s_2s_1s_3s_1s_3$ | $p(s_4) = 0{,}2$ | **20** | $s_3s_2s_4s_4s_2s_1s_2$ | $p(s_4) = 0{,}3$ |
| **9** | $s_4s_2s_4s_1s_3s_4s_2$ | $p(s_1) = 0{,}2$ | **21** | $s_2s_1s_3s_2s_3s_3s_4$ | $p(s_1) = 0{,}1$ |
| **10** | $s_2s_4s_1s_4s_2s_4s_3$ | $p(s_2) = 0{,}3$ | **22** | $s_3s_2s_2s_3s_4s_3s_1$ | $p(s_2) = 0{,}3$ |
| **11** | $s_1s_4s_2s_3s_4s_2s_4$ | $p(s_3) = 0{,}1$ | **23** | $s_1s_2s_3s_3s_2s_4s_3$ | $p(s_3) = 0{,}4$ |
| **12** | $s_4s_1s_4s_2s_4s_3s_2$ | $p(s_4) = 0{,}4$ | **24** | $s_3s_1s_3s_4s_2s_2s_3$ | $p(s_4) = 0{,}2$ |

***Требования к отчёту***

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Таблица с выполнением шагов арифметического кодирования. Полученное десятичное число и его двоичное представление.
4. Таблица с выполнением декодирования арифметического кода. Графическое представление кодирования и декодирования.
5. Исходные коды классов кодер и декодер источника сообщений.
6. Скриншот окна консольного приложения с результатами моделирования системы передачи сообщений.

## 4.4. Контрольные вопросы и задачи

***Теоретические вопросы.***

1. Что понимают под сжатием информации?

2. В какую форму преобразуется последовательность символов в методе арифметического кодирования?

3. Что понимают под кумулятивными вероятностями символов?

4. Как в методе арифметического кодирования определяются значения величин $F_k$ и $G_k$?

5. Что представляют собой величины $p(s_i)$, $q(s_i)$, $F_k$, $G_k$ в графической интерпретации арифметического кодирования?

6. Как находят значение числа, соответствующего арифметическому коду?

7. В чем заключается декодирование арифметического кода?

***Практические задачи***

Задан источник сообщений без памяти $S$ с алфавитом $s_1$, $s_2$, $s_3$ и распределением вероятностей

$p(s_1) = 0{,}25$; $p(s_2) = 0{,}45$; $p(s_3) = 0{,}3$.

Требуется:

• произвести арифметическое кодирование последовательности символов $\boldsymbol{s_1 s_2 s_3 s_2}$;

• декодировать полученный арифметический код.

# 5. СЖАТИЕ ИНФОРМАЦИИ С ПОМОЩЬЮ АЛГОРИТМОВ ЛЕМПЕЛА-ЗИВА

## 5.1. Цель и задачи работы

Цель работы – приобрести умение производить сжатие информации с помощью алгоритмов Лемпела-Зива (LZ77 и LZ78).

Основные задачи работы:

- освоить сжатие данных с помощью алгоритма LZ77;
- научиться сжимать данные при помощи алгоритма LZ78.

Работа рассчитана на 4 часа.

## 5.2. Основные теоретические сведения

### 5.2.1. Словарные методы сжатия информации. Алгоритм LZ77

***Словарные методы сжатия информации***

Статистические методы сжатия (метод Хаффмана, метод арифметического кодирования и др.) используют статистическую модель источника, в которой точно указаны вероятности появления символов этого источника. Степень сжатия информации с помощью таких методов напрямую зависит от того, насколько адекватна полученная модель.

Если объем алфавита очень велик (например, в качестве символов алфавита рассматриваются не отдельные буквы, а пары и тройки букв), то хранение и модификация информации о каждом символе потребует значительных затрат памяти и времени. В этих случаях достаточно эффективными оказываются простые словарные методы.

***Словарные методы*** выбирают некоторые последовательности символов, называемые ***фразами,*** и сохраняют их в специальном словаре. С помощью словаря выполняется кодирование всех последовательностей в виде меток.

Словарь, используемый в данных методах, может быть двух видов:

• ***статический словарь***, который является постоянным; иногда в такой словарь могут добавляться новые последовательности, но удаление не производится;

• ***динамический словарь***, в который слова добавляются по мере поступления из сообщения; кроме того, могут удаляться старые слова, поскольку поиск по большому словарю выполняется медленно.

Наиболее известными и широко применяемыми словарными методами сжатия являются ***алгоритмы Лемпела-Зива*** (***LZ-алгоритмы***)[2]. Основная идея алгоритмов Лемпела-Зива состоит в замене появления фрагмента в сжимаемых данных ссылкой на предыдущее появление этого фрагмента.

Наиболее известные модификации алгоритмов Лемпела-Зива показаны на рис. 5.1.

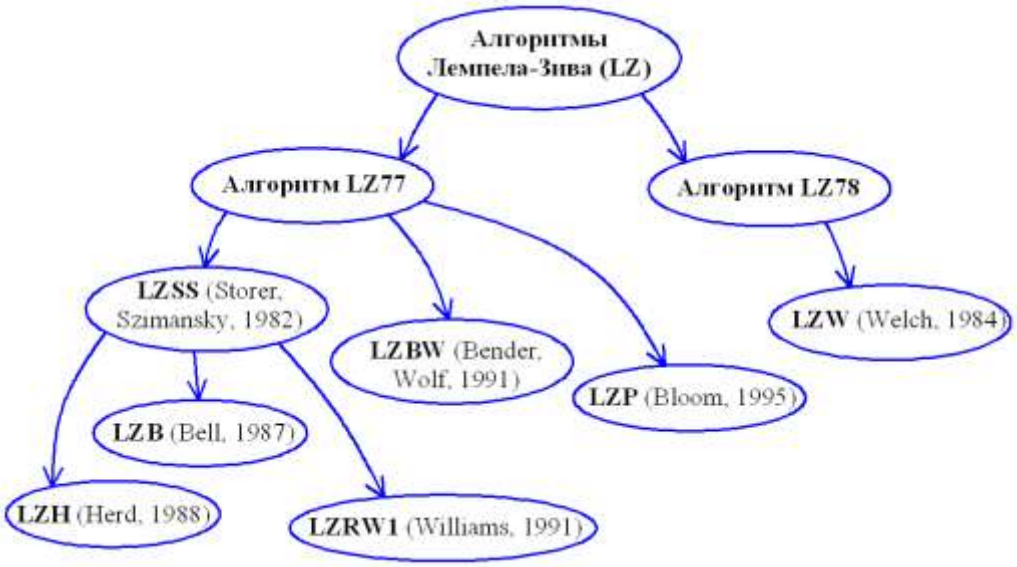


Рис. 5.1. Наиболее известные модификации LZ-алгоритмов

### *Алгоритм LZ77.*

***Алгоритм* LZ77** (LZ1) является родоначальником семейства словарных алгоритмов – так называемых ***алгоритмов со скользящим окном***. Его описание было опубликовано в 1977 г. В данном алгоритме используется скользящее по сообщению окно, которое состоит из двух неравных частей (рис. 5.2):

[2] Название дано в честь Абрахама Лемпела (*англ. Abraham Lempel*) и Якоба Зива (*англ. Jakob Ziv*), которые совместно разработали алгоритмы LZ77 и LZ78.

• ***словарь*** (*window*), являющийся большей по размеру частью длины ***w*** (обычно несколько килобайт) и включающий уже просмотренную часть сообщения;

• ***буфер*** (*lookahead buffer*), представляющий значительно меньшую по размеру часть длины ***b*** (не более ста байт) и включающий еще не просмотренную часть сообщения, которая предшествует словарю.

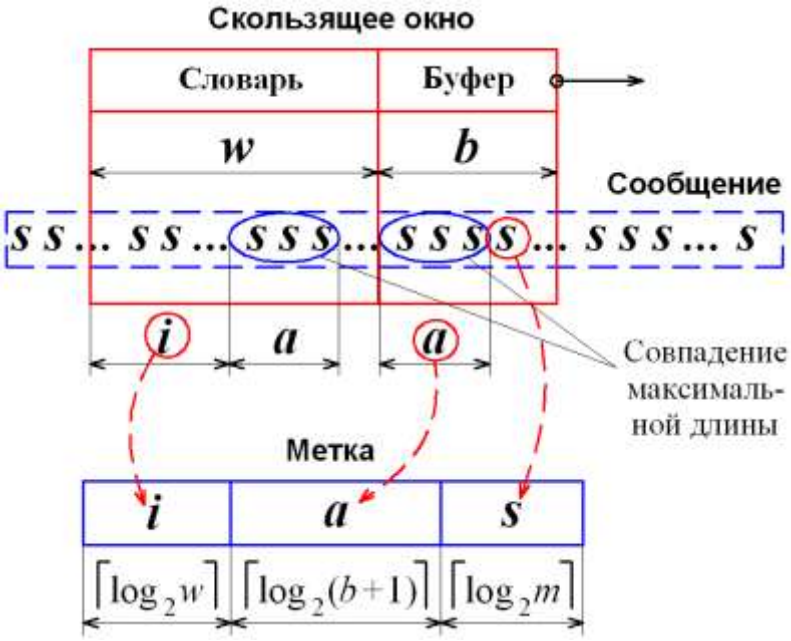


Рис. 5.2. Схема работы скользящего окна в алгоритме LZ77

Алгоритм ищет в словаре фразы максимальной длины, совпадающие с началом содержимого буфера. Если это удается, то будет выдана пара значений, соответствующая позиции фразы в словаре и длине фразы, иначе выводится первый символ буфера.

Кодер метода LZ77 выдает кодовые слова (метки, англ. *pointers*), состоящие из трех полей:

• ***i*** – ***смещение*** в словаре относительно его начала до фразы, совпадающей с началом буфера ($i \le w - 1$); размер поля «смещение» принимают равным $\lceil \log_2 w \rceil$, где $w$ – размер словаря;

• ***a*** – ***длина*** совпадающей фразы ($a \le b$); поле «длина» имеет размер $\lceil \log_2 (b+1) \rceil$, где $b$ – длина буфера;

• ***s*** – первый ***символ*** буфера, следующий за совпадающей фразой; размер поля «символ» равен $\lceil \log_2 m \rceil$, где $m$ – объём алфавита источника сообщений.

Отсюда общая длина двоичного кодового слова, соответствующего метке, будет:

$$L = \lceil \log_2 w \rceil + \lceil \log_2 (b+1) \rceil + \lceil \log_2 m \rceil. \qquad (5.1)$$

В форме псевдокода кодирование по методу LZ77 можно описать следующим образом:

```
словарь пуст;
записать в буфер начало сообщения;
пока (буфер не пуст)
    найти в словаре фразу максимальной длины,
    совпадающую с началом буфера;
    если (длина_фразы > 0) то
        получить смещение фразы от начала словаря;
    иначе смещение = 0;
    сформировать метку (смещение, длина_фразы,
    первый символ буфера после совпадающей фразы);
    сдвинуть содержимое словаря и буфера влево на
    величину длина_фразы + 1;
```

Декодер метода LZ77 значительно проще кодера. Он создает такой же словарь, как и кодер. Декодер читает метки и использует смещение для нахождения текстовой строки в словаре.

Текстовое описание декодирования по LZ77 имеет следующий вид:

```
словарь пуст;
от j = 1 до число_меток
    считать поля i, a, s метки[j];
    найти в словаре фразу p длиной a символов,
    смещенную на i от начала словаря;
    добавить к фразе p символ s, то есть p + s;
    удалить a + 1 символов из начала словаря;
```

записать *p* + *s* в конец *словаря*;

Алгоритмы со скользящим окном характеризуются сильной несимметричностью по времени – кодирование значительно медленнее декодирования, поскольку при сжатии много времени тратится на поиск фраз.

Достоинства LZ77:

• хорошее сжатие сообщения при близком расположении похожих фрагментов;

• высокая скорость декодирования.

Недостатки LZ77:

• если похожие фрагменты сообщения расположены далеко друг от друга, то алгоритм не обеспечит хорошее сжатие;

• увеличение размера буфера приводит к увеличению размеров меток, что уменьшает коэффициент сжатия;

• с ростом размеров словаря скорость кодирования пропорционально замедляется;

• кодирование одиночных символов очень неэффективно.

❒ ***Пример* 5.1. *Кодирование и декодирование с помощью алгоритма* LZ77.**

Требуется с помощью алгоритма LZ77 выполнить кодирование сообщения «**кран_крась_красной_краской**» (26 символов), а также декодировать полученный код.

Процесс кодирования сообщения представлен в табл. 5.1.

Таблица 5.1

Кодирование с помощью алгоритма LZ77

| **Шаг** | **Словарь (*w* = 20)** | **Буфер (*b* = 5)** | **Совп.** | **Поля метки** | | |
|---|---|---|---|---|---|---|
| | | | | ***i*** | ***a*** | ***s*** |
| 0 | "********************" | "****" | - | - | - | - |
| 1 | "********************" | "кран_" | - | 0 | 0 | 'к' |
| 2 | "*******************к" | "ран_к" | - | 0 | 0 | 'р' |
| 3 | "******************кр" | "ан_кр" | - | 0 | 0 | 'а' |
| 4 | "*****************кра" | "н_кра" | - | 0 | 0 | 'н' |
| 5 | "****************кран" | "_крас" | - | 0 | 0 | '_' |
| 6 | "***************кран_" | "крась" | кра | 15 | 3 | 'с' |
| 7 | "**********кран_крас" | "ь_кра" | - | 0 | 0 | 'ь' |

| Шаг | Словарь ($w$ = 20) | Буфер ($b$ = 5) | Совп. | Поля метки | | |
|---|---|---|---|---|---|---|
| | | | | $i$ | $a$ | $s$ |
| 8 | "**********кран_крась" | "_крас" | _крас | 14 | 5 | '' |
| 9 | "*****кран_крась_крас" | "ной_к" | н | 8 | 1 | 'о' |
| 10 | "***кран_крась_красно" | "й_кра" | - | 0 | 0 | 'й' |
| 11 | "**кран_крась_красной" | "_крас" | _крас | 6 | 5 | '' |
| 12 | "н_крась_красной_крас" | "кой**" | к | 2 | 1 | 'о' |
| 13 | "крась_красной_краско" | "й****" | й | 12 | 1 | '*' |
| 14 | "ась_красной_краской*" | "*****" | - | - | - | - |

Таким образом, при кодировании сообщения были получены следующие метки (всего 13 шт.):

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **1** | **(0, 0, 'к')** | **2** | **(0, 0, 'р')** | **3** | **(0, 0, 'а')** | **4** | **(0, 0, 'н')** |
| **5** | **(0, 0, '_')** | **6** | **(15, 3, 'с')** | **7** | **(0, 0, 'ь')** | **8** | **(14, 5, '')** |
| **9** | **(8, 1, 'о')** | **10** | **(0, 0, '_')** | **11** | **(6, 5, '')** | **12** | **(2, 1, 'о')** |
| **13** | **(12, 1, '*')** | **14** | – | **15** | – | **16** | – |

Для записи символов используем кодировку ASCII, содержащую 256 символов, каждый из которых занимает 8 бит.

Определим по формуле (5.1) длину одного двоичного кодового слова:

$$L = \lceil \log_2 20 \rceil + \lceil \log_2 (5+1) \rceil + \lceil \log_2 256 \rceil = 5 + 3 + 8 = 16 \text{ (бит)}$$

Отсюда объём сжатых данных по методу LZ77 будет:

$$V_{out} = n \cdot L = 13 \cdot 16 = 208 \text{ (бит)}$$

Процесс декодирования полученных меток приведён в таблице 5.2.

Таблица 5.2

Декодирование с помощью алгоритма LZ77

| Шаг | Метка | | | Печать | Словарь ($w$ = 20) |
|---|---|---|---|---|---|
| | $i$ | $a$ | $s$ | | |
| 0 | - | - | - | "" | "********************" |
| 1 | 0 | 0 | 'к' | "к" | "*******************к" |
| 2 | 0 | 0 | 'р' | "р" | "******************кр" |
| 3 | 0 | 0 | 'а' | "а" | "*****************кра" |
| 4 | 0 | 0 | 'н' | "н" | "****************кран" |

| Шаг | Метка | | | Печать | Словарь ($w$ = 20) |
|---|---|---|---|---|---|
| | $i$ | $a$ | $s$ | | |
| 5 | 0 | 0 | '_' | "_" | "**********кран_" |
| 6 | 15 | 3 | 'с' | "крас" | "**********кран_крас" |
| 7 | 0 | 0 | 'ь' | "ь" | "**********кран_крась" |
| 8 | 14 | 5 | '' | "_крас" | "*****кран_крась_крас" |
| 9 | 8 | 1 | 'о' | "но" | "***кран_крась_красно" |
| 10 | 0 | 0 | 'й' | "й" | "**кран_крась_красной" |
| 11 | 6 | 5 | '' | "_крас" | "н_крась_красной_крас" |
| 12 | 2 | 1 | 'о' | "ко" | "крась_красной_краско" |
| 13 | 12 | 1 | '*' | "й*" | "ась_красной_краской*" |

Таким образом, в результате декодирования было получено исходное сообщение «**кран_крась_красной_краской**». ❐

### 5.2.2. Алгоритм LZ78. Особенности алгоритма LZW

***Алгоритм LZ78***

Алгоритм LZ78 (LZ2) был опубликован в 1978 г. и впоследствии стал родоначальником семейства словарных методов LZ78.

***Алгоритм LZ78*** в отличие от LZ77 не использует скользящее окно, а генерирует по определённым правилам словарь из уже просмотренных фрагментов сообщения.

На каждом шаге в словарь вставляется новая фраза, которая представляет собой объединение одной из фраз словаря, имеющей самое длинное совпадение со строкой сообщения, и символа ***s***. Символ ***s*** является символом, следующим за строкой, для которой найдена совпадающая фраза.

В начале обработки словарь пуст. Далее, теоретически, словарь может расти бесконечно, то есть на его рост сам алгоритм не накладывает ограничений. На практике при достижении определенного объема занимаемой памяти словарь должен очищаться полностью или частично.

В отличие от алгоритма LZ77 в словаре не может быть одинаковых фраз.

При кодировании по LZ78 фрагментам сообщения присваиваются кодовые слова (метки), которые содержат следующие поля:

• ***i*** – ***индекс*** самой длинной фразы в словаре, которая совпадает с фрагментом сообщения ($i \le w - 1$); размер поля принимают равным $\lceil \log_2 w \rceil$, где $w$ – размер словаря;

• ***s*** – ***символ,*** следующий за строкой, для которой в словаре найдена совпадающая фраза максимальной длины; размер данного поля равен $\lceil \log_2 m \rceil$, где $m$ – объём алфавита источника сообщений.

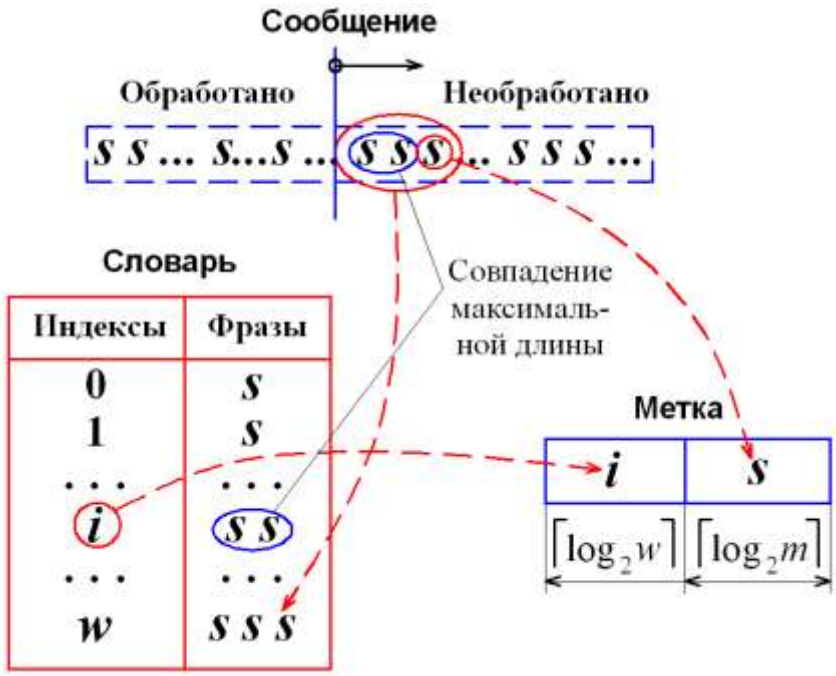


Рис. 5.3. Схема работы словаря в алгоритме LZ78

Длина двоичного кодового слова, соответствующего метке, будет:

$$L = \lceil \log_2 w \rceil + \lceil \log_2 m \rceil. \qquad (5.2)$$

Текстовое описание кодирования по алгоритму LZ78 может иметь следующий вид:

```
словарь пуст;
пока (сообщение не пусто)
    найти самую длинную совпадающую фразу p
    в словаре и в начале сообщения;
    определить индекс i фразы p в словаре и символ s,
```

```
следующий за фразой в сообщении;
записать i и s в очередную метку;
определить длину a фразы p + s;
записать p + s в словарь;
удалить a первых символов в сообщении;
```

Декодирование по методу LZ78 в отличие от LZ77 осуществляется в порядке обратном кодированию.

В начале декодирования словарь пуст. На каждом шаге производится считывание очередной метки (***i***, ***s***), которая по индексу ***i*** указывает на фразу ***p***, уже присутствующую в словаре. На выходе получают строку ***p*** + ***s***, которая образована объединением фразы ***p*** словаря и символа ***s*** метки. Строка ***p*** + ***s*** добавляется в словарь. После декодирования словарь выглядит так же, как и после кодирования.

Декодирование по методу LZ78 можно описать следующим образом:

```
словарь пуст;
от j = 1 до число_меток
    считать поля i и s метки[j];
    найти по индексу i в словаре фразу p;
    добавить к фразе p символ s и записать p + s
    в сообщение;
    записать p + s в конец словаря;
```

Достоинства LZ78:

• обеспечивает хорошее сжатие текстов и других однородных данных;

• более высокая скорость кодирования, чем для LZ77;

• простой в реализации.

Недостатки LZ78:

• меньшая степень сжатия по сравнению с LZ77;

• скорость декодирования ниже, чем у LZ77.

❒ ***Пример* 5.2. *Кодирование и декодирование сообщения с помощью алгоритма* LZ78.**

Требуется с помощью алгоритма LZ78 провести кодирование фразы «**кран_крась_красной_краской**», а также декодировать полученный код.

Процесс кодирования сообщения представлен в табл. 5.3.

Таблица 5.3

Кодирование с помощью алгоритма LZ78

| **Шаг (позиция фразы в словаре)** | **Словарь** | **Поля метки** | |
|---|---|---|---|
| | | ***i*** | ***s*** |
| 0 | "" | - | - |
| 1 | "к" | 0 | 'к' |
| 2 | "р" | 0 | 'р' |
| 3 | "а" | 0 | 'а' |
| 4 | "н" | 0 | 'н' |
| 5 | "_" | 0 | '_' |
| 6 | "кр" | 1 | 'р' |
| 7 | "ас" | 3 | 'с' |
| 8 | "ь" | 0 | 'ь' |
| 9 | "_к" | 5 | 'к' |
| 10 | "ра" | 2 | 'а' |
| 11 | "с" | 0 | 'с' |
| 12 | "но" | 4 | 'о' |
| 13 | "й" | 0 | 'й' |
| 14 | "_кр" | 9 | 'р' |
| 15 | "аск" | 7 | 'к' |
| 16 | "о" | 0 | 'о' |
| 17 | "й*" | 13 | '*' |

При кодировании сообщения по алгоритму LZ78 были получены следующие метки (всего 17 шт.):

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **(0, 'к')** | **2** | **(0, 'р')** | **3** | **(0, 'а')** | **4** | **(0, 'н')** | **5** | **(0, '_')** |
| **6** | **(1, 'р')** | **7** | **(3, 'с')** | **8** | **(0, 'ь')** | **9** | **(5, 'к')** | **10** | **(2, 'а')** |
| **11** | **(0, 'с')** | **12** | **(4, 'о')** | **13** | **(0, 'й')** | **14** | **(9, 'р')** | **15** | **(7, 'к')** |
| **16** | **(0, 'о')** | **17** | **(13, '*')** | **18** | – | **19** | – | **20** | – |

Размер словаря равен общему числу фраз, то есть $w = 18$. Объём алфавита для кодировки ASCII будет $m = 256$. Отсюда длина одного двоичного кодового слова по формуле (5.2) будет:

$$L = \lceil \log_2 18 \rceil + \lceil \log_2 256 \rceil = 5 + 8 = 13 \text{ (бит)}$$

Отсюда объём сжатых данных по методу LZ78 будет:

$$V = n \cdot L = 17 \cdot 13 = 221 \text{ (бит)}$$

Полученный результат (221 бит) является большим, чем при кодировании по алгоритму LZ77 (208 бит).

Таким образом, кодирование заданного сообщения по алгоритму LZ77 обеспечивает меньший объём сжатых данных, чем по алгоритму LZ78.

Процесс декодирования полученных меток приведён в таблице 5.4.

Таблица 5.4

Декодирование по алгоритму LZ78

| **Шаг (позиция фразы в словаре)** | **Поля метки** | | **Печать (словарь *w* = 18)** |
|---|---|---|---|
| | ***i*** | ***s*** | |
| 0 | - | - | "" |
| 1 | 0 | 'к' | "к" |
| 2 | 0 | 'р' | "р" |
| 3 | 0 | 'а' | "а" |
| 4 | 0 | 'н' | "н" |
| 5 | 0 | '_' | "_" |
| 6 | 1 | 'р' | "кр" |
| 7 | 3 | 'с' | "ас" |
| 8 | 0 | 'ь' | "ь" |
| 9 | 5 | 'к' | "_к" |
| 10 | 2 | 'а' | "ра" |
| 11 | 0 | 'с' | "с" |
| 12 | 4 | 'о' | "но" |
| 13 | 0 | 'й' | "й" |
| 14 | 9 | 'р' | "_кр" |
| 15 | 7 | 'к' | "аск" |
| 16 | 0 | 'о' | "о" |
| 17 | 13 | '*' | "й*" |

В результате декодирования получилось исходное сообщение «**кран_крась_красной_краской**». ❐

## 5.3. Порядок выполнения работы и варианты заданий

### *Основные этапы выполнения работы*

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Произвести кодирование заданного сообщения (табл. 5.5) с помощью алгоритма LZ77 при известном размере словаря ***w*** и буфера ***b***. Вычислить длину в битах для сжатого сообщения. Выполнить декодирование полученных кодовых слов.
3. Выполнить аналогичные п. 2 действия с помощью алгоритма LZ78. Сравнить длины сжатого сообщения, полученные при кодировании по алгоритмам LZ77 и LZ78.
4. Дополнительно требуется реализовать на языке C# предложенный LZ-алгоритм (таблица 5.5).
5. Оформить и защитить отчет по лабораторной работе.

### *Индивидуальные варианты заданий*

Таблица 5.5

Варианты заданий для сжатия информации с помощью LZ-алгоритмов

| **№ вар.** | **Сообщения (34 симв.)** | ***w*** | ***b*** | **LZ** |
|---|---|---|---|---|
| 1 | ты_коты_коток_током_ком_моток_моты | 30 | 6 | LZSS |
| 2 | он_они_кони_окно_канон_канкан_кино | 28 | 7 | LZW |
| 3 | мы_кумы_робы_ром_коробы_бором_кубы | 30 | 5 | LZSS |
| 4 | вы_оковы_око_коровы_кров_ковры_ров | 30 | 6 | LZW |
| 5 | да_дар_арка_радар_драка_кара_дакар | 28 | 7 | LZSS |
| 6 | из_изба_базар_бриз_зараз_баба_база | 30 | 5 | LZW |
| 7 | кол_пол_прокол_клок_про_порок_клоп | 30 | 6 | LZSS |
| 8 | ода_вода_овод_подо_довод_по_повод_ | 28 | 7 | LZW |
| 9 | яр_ярко_арка_доярка_кадр_до_корка_ | 30 | 5 | LZSS |
| 10 | год_гора_город_рог_род_огород_рога | 30 | 6 | LZW |
| 11 | на_она_с_окна_век_весна_сосна_сна_ | 28 | 7 | LZSS |

| № вар. | Сообщения (34 симв.) | *w* | *b* | LZ |
|---|---|---|---|---|
| 12 | лег_легче_челка_качалка_легка_елка | 30 | 5 | LZW |
| 13 | там_сам_тесак_масса_так_асс_месса_ | 30 | 6 | LZSS |
| 14 | ем_нем_инием_нес_синим_с_ним_синее | 28 | 7 | LZW |
| 15 | но_оно_окно_икона_конник_она_кино_ | 30 | 5 | LZSS |
| 16 | до_дома_мадам_мода_ам_дама_мама_да | 30 | 6 | LZW |
| 17 | под_пока_ковка_окоп_копка_поковка_ | 28 | 7 | LZSS |
| 18 | над_народ_родня_няня_ядро_наряд_яд | 30 | 5 | LZW |
| 19 | ум_кум_кулак_мула_лак_мука_макака_ | 30 | 6 | LZSS |
| 20 | нас_надо_радон_рано_донна_раса_сад | 28 | 7 | LZW |
| 21 | ел_елка_белка_калека_кабала_балка_ | 30 | 5 | LZSS |
| 22 | час_часы_чары_с_рысь_сыры_сыч_ась_ | 30 | 6 | LZW |
| 23 | лом_лоб_болон_мол_он_облом_моно_но | 28 | 7 | LZSS |
| 24 | не_вне_соне_внес_вон_весне_во_сове | 30 | 5 | LZW |

***Требования к отчёту.***

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Описание кодирования и декодирования заданного сообщения с помощью алгоритма LZ77.
4. Выполнение кодирования и декодирования сообщения с помощью алгоритма LZ78.
5. Выводы по лабораторной работе.

## 5.4. Контрольные вопросы и задачи

***Теоретические вопросы.***

1. В чём заключаются основные особенности словарных методов сжатия информации?
2. Какие виды словарей применяются в словарных методах сжатия?
3. На какие группы разделяют семейство алгоритмов Лемпела-Зива?
4. Из каких частей состоит скользящее окно в алгоритме LZ77?
5. Какие поля включает в себя кодовое слово, получаемое в алгоритме LZ77?

6. Каковы основные достоинства и недостатки алгоритма LZ77?

7. Каким образом формируется словарь при сжатии по алгоритму LZ78?

8. Из каких полей состоит метка при кодировании по алгоритму LZ78?

9. В чём заключаются достоинства и недостатки алгоритма LZ78?

***Практические задачи***

1. Задано сообщение (табл. 5.X). Требуется сжать сообщение с помощью алгоритма LZ77.

• произвести кодирование сообщения с помощью алгоритма LZ77 при размере словаря **w** и буфера **b**;

• определить объём сжатых данных;

• декодировать полученные кодовые слова.

Таблица 5.6

Варианты заданий для сжатия данных с помощью LZ77

| **Вариант** | **Сжимаемое сообщение** | **w** | **b** |
|---|---|---|---|
| **Нечётный** | `баба_баран_барабан_` | 16 | 4 |
| **Чётный** | `лама_хам_махала_хлам` | 18 | 5 |

# 6. ОСНОВЫ ПОСТРОЕНИЯ И ДЕКОДИРОВАНИЯ ЛИНЕЙНЫХ БЛОКОВЫХ КОДОВ

## 6.1. Цель и задачи работы

Цель работы – приобрести умение строить и декодировать линейные блоковые коды для обнаружения и исправления ошибок в передаваемых сообщениях.

Основные задачи работы:

• научиться строить линейные блоковые коды с помощью порождающих матриц;

• освоить декодирование линейных блоковых кодов по синдрому.

Работа рассчитана на 4 часа.

## 6.2. Основные теоретические сведения

### 6.2.1. Особенности линейных блоковых кодов. Порождающие матрицы

***Блоковые корректирующие коды***

***Блоковыми*** называют помехоустойчивые коды, в которых процедура кодирования заключается в разбиении входной последовательности информационных символов на блоки, содержащие $m$ символов. Каждому информационному блоку длиной $m$ сопоставляется $k$ проверочных символов. Полученное кодовое слово из $n = m + k$ символов называют кодовым блоком.

Число несовпадающих позиций в двух кодовых словах $\bar{x}$ и $\bar{y}$ называется ***расстоянием Хэмминга*** $d(\bar{x}, \bar{y})$ между этими словами.

Важной характеристикой корректирующего блокового кода $C$ является ***кодовое расстояние***, которое принимается равным наименьшему расстоянию Хэмминга между словами данного кода:

$$d(C) = \min\{\, d(\bar{x}, \bar{y}) :\ \bar{x}, \bar{y} \in C; \bar{x} \neq \bar{y}\}. \qquad (6.1)$$

Для блоковых кодов справедливы следующие утверждения:

• Для того чтобы блоковый код $C$ позволял обнаруживать все комбинации из $t$ или менее ошибок, необходимо и достаточно, чтобы его кодовое расстояние было равно $d(C) = t + 1$.

• Для того чтобы блоковый код $C$ позволял исправлять все комбинации из $t$ или менее ошибок необходимо и достаточно, чтобы его кодовое расстояние было равно $d(C) = 2t + 1$.

Для практических расчетов при определении числа проверочных символов $k$ в коде с кодовым расстоянием $d(C) = 3$ используют следующие формулы:

если известна длина полного кодового слова $n$, то

$$k = \lceil \log_2(n+1) \rceil; \qquad (6.2)$$

если при расчетах удобнее исходить из заданного числа информационных символов $m$, то

$$k = \lceil \log_2\{(m+1) + \lceil \log_2(m+1) \rceil\} \rceil; \qquad (6.3)$$

где $\lceil ... \rceil$ – округление числа, стоящего в скобках, до целого в большую сторону.

Для блоковых кодов с $d(C) = 4$

$$k \geq 1 + \log_2(n+1); \qquad (6.4)$$

или

$$k \geq 1 + \lceil \log_2\{(m+1) + \log_2(m+1)\} \rceil. \qquad (6.5)$$

***Особенности линейных блоковых кодов. Оценка Хэмминга.***

Самый большой класс блоковых корректирующих кодов составляют ***линейные коды***, у которых значения проверочных символов определяются при проведении линейных операций над информационными символами.

К ***линейным операциям*** относятся сложение символов и их умножение на постоянное число. Для двоичных линейных кодов в качестве линейной операции используют ***сложение по модулю 2*** (обозначается $\oplus$), которое описывается следующими правилами:

$$0 \oplus 0 = 0;\ \ 0 \oplus 1 = 1;\ \ 1 \oplus 0 = 1;\ \ 1 \oplus 1 = 0.$$

Кодовые слова линейного кода $C$ называют ***кодовыми векторами*** и обозначают $\bar{x}$, $\bar{y}$, $\bar{z}$, … Кодовый вектор $\bar{0}$, состоящий из одних нулей, называется ***нулевым вектором***.

Число единиц в двоичном кодовом векторе $\bar{x}$ называется ***весом кодового вектора*** и обозначается через $w(\bar{x})$.

Расстояние Хэмминга $d(\bar{x}, \bar{y})$ между кодовыми векторами $\bar{x}$ и $\bar{y}$ равно весу разности этих векторов:

$$\bar{z} = \bar{x} - \bar{y};\ \bar{x} \neq \bar{y};\ d(\bar{x}, \bar{y}) = w(\bar{z}).$$

Кодовое расстояние линейного кода $C$ равно минимальному весу его ненулевых кодовых векторов:

$$d(C) = \min\{ w(\bar{x})\};\ \bar{x} \in C;\ \bar{x} \neq \bar{0}.$$

Для изучения свойств линейных блоковых кодов используется $n$-мерное линейное пространство $X_n$, между любой парой векторов $\bar{x}$ и $\bar{y}$ которого можно задать расстояние Хэмминга $d(\bar{x}, \bar{y})$. Указанное пространство называют пространством ***Хэмминга***.

Между параметрами $n$, $m$, $d$ получены соотношения, называемые оценками числа элементов кода с заданным кодовым расстоянием в пространстве Хэмминга.

Одной из наиболее известных оценок является ***оценка Хэмминга***, которая определяется следующим образом: если суще-

ствует ($n$, $m$)-код с основанием $K$ и кодовым расстоянием $d = 2t + 1$, то

$$\sum_{i=1}^{t} C_n^i (K-1)^i \le K^{n-m}; \qquad (6.6)$$

где $C_n^i$ – числа сочетаний из $n$ элементов по $i$, определяемое по известной формуле из комбинаторики:

$$C_n^i = \frac{n!}{i!(n-i)!}.$$

Для двоичных кодов ($K = 2$) оценка Хэмминга будет иметь следующий вид:

$$\sum_{i=1}^{t} C_n^i \le 2^{n-m}. \qquad (6.7)$$

Линейные блоковые коды, для которых выполняется оценка Хэмминга, называются ***совершенными*** (***плотно упакованными***) ***кодами***. Такие коды обнаруживают и исправляют максимальное число ошибок при минимальном числе проверочных символов.

***Порождающие матрицы линейных блоковых кодов.***

Линейные блоковые коды задают при помощи ***порождающих*** матриц, размерность которых равна $m \times n$:

$$G_{m,n} = \left[E_m \mid P_{m,k}\right] = \left[\begin{array}{cccc|ccc} 1 & 0 & \dots & 0 & p_{11} & p_{12} & \dots p_{1k} \\ 0 & 1 & \dots & 0 & p_{21} & p_{22} & \dots p_{2k} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & p_{m1} & p_{m2} & \dots p_{mk} \end{array}\right]; \qquad (6.8)$$

где $E_m$ – единичная матрица размера $m$; $P_{m,k}$ – матрица-дополнение, состоящая из проверочных символов $p_{ij}$.

Для кодов с $d = 2$ порождающая матрица имеет следующий вид

$$G_{m,n} = \left[\begin{array}{cccc|c} 1 & 0 & \ldots & 0 & 1 \\ 0 & 1 & \ldots & 0 & 1 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & 1 \end{array}\right]. \qquad (6.9)$$

Во всех кодовых векторах, построенных при помощи такой матрицы, будет четное число единиц.

Для кодов с $d \geq 3$ вид матрицы $P$ зависит от конкретных требований к порождаемому коду. Этими требованиями могут быть либо минимум проверочных символов, либо максимальная простота кодирующей/декодирующей аппаратуры.

Чем больше вес строк матрицы $P$, тем ближе порождаемый код к совершенному. С другой стороны, число единиц в матрице $P$ определяет число сумматоров по модулю 2 в кодере и декодере канала связи. То есть, чем больше единиц в матрице $P$, тем сложнее аппаратура кодирования-декодирования.

При построении совершенных кодов с $d \geq 3$ последовательно используются строки матрицы $P$ с весом $W_P = k, k-1, k-2, \ldots, d-1$. Таким образом, вес $W_P$ каждой строки матрицы $P$ должен удовлетворять условию:

$$W_P \geq d - 1. \qquad (6.10)$$

При построении кодов с максимально простыми кодерами и декодерами каналов связи в матрице $P$ последовательно выбираются строки весом $W_P = 2, 3, \ldots, k$.

Если число комбинаций строк матрицы $P$, удовлетворяющих условию $W_P \geq d - 1$, больше $m$, то в первом случае (получение совершенного кода) не используют комбинации с наименьшим весом, а во втором (максимальная простота аппаратуры) – с наибольшим.

Матрицу $P$ следует строить таким образом, чтобы число единиц в столбцах данной матрицы было по возможности одинаковым. От числа единиц в столбце матрицы $P$ зависит число проверок, производимых при исправлении ошибок.

В общем виде формулы для определения значений проверочных символов могут быть записаны следующим образом:

$$p_j = \sum_{i=1}^{m} p_{ij} x_i; \quad (j = 1, 2, \ldots, k); \qquad (6.11)$$

где $p_{ij}$ – элементы матрицы-дополнения $P$.

Для двоичных кодов значения проверочных символов $p_j$ находят путем суммирования по модулю 2 тех строк матрицы $P$, номера которых совпадают с номерами информационных символов, значение которых равно единице.

❒ ***Пример* 6.1. *Кодирование с помощью порождающей матрицы линейного блокового кода.***

Пусть требуется построить порождающую матрицу для линейного блокового кода, способного исправлять одиночную ошибку при передаче информационных векторов из 7 символов (например, 0100111, 1101100, 1011010).

Получим порождающую матрицу в приведенной форме:

$$G_{m,n} = \left[ E_m | P_{m,k} \right].$$

Поскольку число исправляемых ошибок $t = 1$, то кодовое расстояние для линейного блокового кода будет

$$d = 2t + 1; \; d = 2 \cdot 1 + 1 = 3.$$

Так как длина информационных векторов $m = 7$, то число строк порождающей матрицы линейного блокового кода должно быть равно 7. Число столбцов порождающей матрицы равно длине кодовых векторов

$$n = m + k;$$

где $k$ – число проверочных символов, которое при $d = 3$ может быть найдено по формуле (3.3):

$$k = \left\lceil \log_2 \{(7+1) + \left\lceil \log_2 (7+1) \right\rceil\} \right\rceil = \left\lceil \log_2 (8+3) \right\rceil = 4.$$

Отсюда получим

$$n = 7 + 4 = 11.$$

Таким образом, искомый линейный блоковый код является (11, 7)-кодом.

Поскольку вес каждой строки матрицы-дополнения $P$ должен быть $W_P \geq d - 1$, то в качестве строк матрицы $P$ примем четырехзначные двоичные комбинации с числом единиц $W_P \geq 2$. Комбинации выберем так, чтобы число единиц в столбцах матрицы $P$ было одинаковым. Заданным требованиям удовлетворяет следующий набор: 1111, 1110, 1101, 1011, 0111, 0110, 1001.

Окончательный вид порождающей матрицы будет:

$$G_{7;11} = \left[\begin{array}{ccccccc|cccc} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right].$$

Получим для заданных информационных векторов 0100111, 1101100, 1011010 кодовые вектора линейного (11,7)-кода. Для этого найдем значения проверочных символов для каждого информационного вектора путем суммирования по модулю 2 тех строк матрицы $P$, номера которых совпадают с номерами разрядов, содержащих единицы в информационных векторах:

| 1) 0100111: | 2) 1101100: | 3) 1011010: |
|---|---|---|
| 1110 | 1111 | 1111 |
| $\oplus$ 0111 | $\oplus$ 1110 | $\oplus$ 1101 |
| 0110 | 1011 | 1011 |
| 1001 | 0111 | 0110 |
| 0110 | 1101 | 1111 |

Отсюда искомые кодовые векторы будут:

$\bar{x}_1 = 0100111\mathbf{0110}$; $\bar{x}_2 = 1101100\mathbf{1101}$; $\bar{x}_3 = 1011010\mathbf{1111}$. ❐

## 6.2.2. Декодирование линейных блоковых кодов

***Проверочные матрицы линейных блоковых кодов***

Для линейного блокового кода *C* с порождающей матрицей $G_{m,n} = \left[E_m | P_{m,k}\right]$ ***проверочной матрицей*** будет являться матрица:

$$H_{k,n} = \left[P_{m,k}^T | \, E_k\right]. \qquad (6.12)$$

Число проверок равно числу *k* проверочных символов кода.

В общем виде систему проверок можно записать следующим образом:

$$p_j \oplus \sum_{i=1}^{m} p_{ij} a_i = S_j; \qquad j = 1, 2, \ldots, k. \qquad (6.13)$$

В результате осуществления проверок образуется проверочный вектор $\overline{S} = [S_1 S_2 ... S_j ... S_k]$, называемый ***синдромом***. Если все разряды синдрома равны нулю, то принятый кодовый вектор считается безошибочным. Если хотя бы один из разрядов синдрома содержит единицу, то принятый кодовый вектор содержит ошибку.

Исправление одиночной ошибки осуществляется по виду синдрома, так как каждому ошибочному разряду принятого кодового вектора соответствует один единственный синдром. Вид синдрома может быть определен при помощи проверочной матрицы *H*. Столбцы матрицы *H* представляют собой синдром для соответствующего ошибочного разряда кодового вектора.

❒ ***Пример* 6.2.** ***Декодирование линейного блокового кода с исправлением одиночных ошибок.***

Запишем систему проверок для найденной матрицы *P*:

$$p_j \oplus \sum_{i=1}^{m} p_{ij} x_i = S_j; \; (j = 1, 2, ..., k).$$

Система проверок будет иметь следующий вид:

$$\begin{cases} p_1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_7 = S_1, \\ p_2 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 = S_2, \\ p_3 \oplus x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_6 = S_3, \\ p_4 \oplus x_1 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_7 = S_4. \end{cases}$$

Для того чтобы знать, какая комбинация синдрома $\bar{S}$ будет соответствовать ошибке в определенном разряде принятого кодового вектора, построим проверочную матрицу линейного блокового кода:

$$H_{k,n} = \left[ P^T_{m,k} \middle| E_k \right].$$

Транспонированная матрица-дополнение $P$ будет выглядеть следующим образом:

$$P^T_{7;4} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Отсюда проверочная матрица линейного блокового кода будет:

$$H_{4;11} = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & p_1 & p_2 & p_3 & p_4 \end{matrix} \\ \left[\begin{array}{ccccccc|cccc} 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}.$$

Предположим, что в переданных кодовых векторах произошли следующие ошибки (выделены полужирным синим шрифтом):

$\bar{x}_1 = 01001\mathbf{0}10110$; $\bar{x}_2 = 11\mathbf{1}11001101$; $\bar{x}_3 = 10110101\mathbf{0}1$.

Найдем согласно полученной системе проверок для каждого переданного кодового вектора синдром $\bar{S} = [S_1 S_2 ... S_k]$:

• Для кодового вектора $\bar{x}_1$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

$$\begin{cases} 0\oplus 0\oplus 1\oplus 0\oplus 0\oplus 1=0, \\ 1\oplus 0\oplus 1\oplus 0\oplus 1\oplus 0=1, \\ 1\oplus 0\oplus 1\oplus 0\oplus 1\oplus 0=1, \\ 0\oplus 0\oplus 0\oplus 0\oplus 1\oplus 1=0. \end{cases}$$

Синдром 0110 показывает, что значение символа $x_6$ следует заменить на противоположное.

• Для кодового вектора $\bar{x}_2$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

$$\begin{cases} 1\oplus 1\oplus 1\oplus 1\oplus 1\oplus 0=1, \\ 1\oplus 1\oplus 1\oplus 1\oplus 1\oplus 0=1, \\ 0\oplus 1\oplus 1\oplus 1\oplus 1\oplus 0=0, \\ 1\oplus 1\oplus 1\oplus 1\oplus 1\oplus 0=1. \end{cases}$$

Синдром 1101 показывает, что ошибка произошла в символе $x_3$.

• Для кодового вектора $\bar{x}_3$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |

$$\begin{cases} 1\oplus 1\oplus 0\oplus 1\oplus 1\oplus 0=0, \\ 1\oplus 1\oplus 0\oplus 1\oplus 0\oplus 1=0, \\ 0\oplus 1\oplus 0\oplus 1\oplus 0\oplus 1=1, \\ 1\oplus 1\oplus 1\oplus 1\oplus 0\oplus 0=0. \end{cases}$$

Синдром 0010 показывает, что значение символа $p_3$ является ошибочным. ❒

### *Стандартные таблицы декодирования*

В общем случае для исправления заданного числа ошибок $t$ с помощью линейных блоковых кодов используют ***стандартные таблицы декодирования*** (таблицы соответствия смежных клас-

сов), содержащие все возможные значения принятых из канала векторов $\bar{z}$. Данные таблицы организованы таким образом, чтобы мог быть найден ближайший к $\bar{z}$ переданный кодовый вектор $\bar{y}$ (табл. 6.1).

Таблица 6.1

Стандартная таблица для декодирования линейного блокового кода

| $\bar{s}_j$ | $\bar{e}_j \quad \bar{y}_i$ | $\bar{y}_1$ | $\bar{y}_2$ | … | $\bar{y}_N$ |
|---|---|---|---|---|---|
| $\bar{s}_1$ | $\bar{e}_1$ | $\bar{y}_1 + \bar{e}_1$ | $\bar{y}_2 + \bar{e}_1$ | … | $\bar{y}_N + \bar{e}_1$ |
| $\bar{s}_2$ | $\bar{e}_2$ | $\bar{y}_1 + \bar{e}_2$ | $\bar{y}_2 + \bar{e}_2$ | … | $\bar{y}_N + \bar{e}_2$ |
| … | … | … | … | … | … |
| $\bar{s}_K$ | $\bar{e}_K$ | $\bar{y}_1 + \bar{e}_K$ | $\bar{y}_2 + \bar{e}_K$ | … | $\bar{y}_N + \bar{e}_K$ |

В первой строке таблицы располагаются все кодовые векторы $\bar{y}_i$, число которых составляет $N = 2^m$. В первом столбце второй строки размещается вектор ошибки $\bar{e}_1$, вес которого равен 1.

Остальные ячейки второй строки заполняются векторами, полученными в результате суммирования по модулю 2 вектора $\bar{e}_1$ с вектором $\bar{y}_i$, расположенным в соответствующем столбце первой строки. В первом столбце третьей строки записывается вектор $\bar{e}_2$, вес которого также равен 1, однако, если вектор $\bar{e}_1$ содержит единицу в первом разряде, то $\bar{e}_2$ – во втором. В остальные ячейки третьей строки записывают суммы $\bar{x}_i$ и $\bar{e}_2$.

Аналогично поступают до тех пор, пока не будут просуммированы с векторами $\bar{y}_i$ все векторы $\bar{e}_j$, весом 1, с единицами в каждом из $n$разрядов. Затем суммируются по модулю 2 векторы $\bar{e}_j$ весом 2, с последовательным перекрытием всех возможных разрядов. Вес вектора $\bar{e}_j$ определяет число исправляемых ошибок. Число векторов $\bar{e}_j$ определяется возможным числом неповторяющихся синдромов и равно $K = 2^k - 1$ (нулевая комбинация говорит об отсутствии ошибки). Условие неповторяемости син-

дрома позволяет по его виду определять соответствующий ему вектор $\bar{e}_j$.

По виду синдрома $\bar{s}_j$ принятый кодовый вектор может быть отнесен к определенной строке таблицы соответствия (смежному классу). Принятый кодовый вектор сравнивается с векторами, записанными в данную строку и в случае совпадения в каком-либо из столбцов выбирается вектор (истинный), расположенный в первой строке данного столбца.

Векторы ошибок $\bar{e}_j$ не должны быть равны ни одному из кодовых векторов.

❒ ***Пример* 6.3.** ***Декодирование линейного блокового кода с помощью стандартной таблицы декодирования.***

Построим для линейного блокового кода из примеров 6.1 и 6.2 фрагмент стандартной таблицы декодирования, позволяющей исправить максимально возможное число двойных ошибок.

Подобная задача не представляет практического интереса, поскольку построенный линейный блоковый код изначально предназначался для исправления одиночных ошибок. Однако эта задача на простом примере иллюстрирует идею исправления ошибок кратностью больше 1.

Столбцы стандартной таблицы декодирования будут соответствовать трём полученным кодовым векторам. Число строк данной таблицы будет равно числу векторов ошибок:

$$K = 2^k - 1 = 2^4 - 1 = 15.$$

Результат построения фрагмента стандартной таблицы декодирования приведён в табл. 6.2.

Таблица 6.2

Фрагмент стандартной таблицы декодирования линейного блокового кода

| $j$ | Вектор ошибки $\bar{e}_j$ | Кодовые векторы $\bar{y}_i$ | | |
|---|---|---|---|---|
| | | $\bar{y}_1$ | $\bar{y}_2$ | $\bar{y}_3$ |
| | | **01001110110** | **11011001101** | **10110101111** |

| | | | | |
|---|---|---|---|---|
| 1 | 10000000000 | ----- | ----- | ----- |
| 2 | 01000000000 | ----- | ----- | ----- |
| 3 | 00100000000 | ----- | ----- | ----- |
| 4 | 00010000000 | ----- | ----- | ----- |
| 5 | 00001000000 | ----- | ----- | ----- |
| 6 | 00000100000 | ----- | ----- | ----- |
| 7 | 00000010000 | ----- | ----- | ----- |
| 8 | 00000001000 | ----- | ----- | ----- |
| 9 | 00000000100 | ----- | ----- | ----- |
| 10 | 00000000010 | ----- | ----- | ----- |
| 11 | 00000000001 | ----- | ----- | ----- |
| **12** | **11000000000** | 10001110110 | 00011001101 | 01110101111 |
| **13** | **10100000000** | 11101110110 | 01111001101 | 00010101111 |
| **14** | **10010000000** | **11011110110** | 01001001101 | 00100101111 |
| **15** | **01100000000** | 00101110110 | 10111001101 | 11010101111 |

Как видно из табл. 6.2, полученный линейный блоковый код обладает недостаточной корректирующей способностью для исправления любых двойных ошибок в кодовых векторах и позволяет исправить лишь 4 класса данных ошибок.

Найдём синдромы, соответствующие двойным ошибкам. Для этого достаточно произвести проверки для кодовых векторов любого столбца таблицы. Для примера возьмём столбец $\bar{y}_2$, содержащий следующие векторы с двойными ошибками:

$$\bar{y}_2 \oplus \bar{e}_{12} = 00011001101; \quad \bar{y}_2 \oplus \bar{e}_{13} = 01111001101;$$

$$\bar{y}_2 \oplus \bar{e}_{14} = 01001001101; \quad \bar{y}_2 \oplus \bar{e}_{15} = 10111001101.$$

Отсюда синдромы по системе проверок из примера 6.2 будут:

• Для вектора $\bar{y}_2 \oplus \bar{e}_{12}$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

$$\begin{cases} 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0, \\ 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0, \\ 0 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 0, \\ 1 \oplus 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1. \end{cases}$$

• Для вектора $\bar{y}_2 \oplus \bar{e}_{13}$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

$$\begin{cases} 1\oplus 0\oplus 1\oplus 1\oplus 1\oplus 0 = 0, \\ 1\oplus 0\oplus 1\oplus 1\oplus 1\oplus 0 = 0, \\ 0\oplus 0\oplus 1\oplus 1\oplus 1\oplus 0 = 1, \\ 1\oplus 0\oplus 1\oplus 1\oplus 1\oplus 0 = 0. \end{cases}$$

• Для вектора $\bar{y}_2 \oplus \bar{e}_{14}$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

$$\begin{cases} 1\oplus 0\oplus 1\oplus 0\oplus 0\oplus 0 = 0, \\ 1\oplus 0\oplus 1\oplus 0\oplus 1\oplus 0 = 1, \\ 0\oplus 0\oplus 1\oplus 0\oplus 1\oplus 0 = 0, \\ 1\oplus 0\oplus 0\oplus 0\oplus 1\oplus 0 = 0. \end{cases}$$

• Для вектора $\bar{y}_2 \oplus \bar{e}_{15}$:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |

$$\begin{cases} 1\oplus 1\oplus 0\oplus 1\oplus 1\oplus 0 = 0, \\ 1\oplus 1\oplus 0\oplus 1\oplus 1\oplus 0 = 0, \\ 0\oplus 1\oplus 0\oplus 1\oplus 1\oplus 0 = 1, \\ 1\oplus 1\oplus 1\oplus 1\oplus 1\oplus 0 = 1. \end{cases}$$

Таким образом, получены следующие синдромы, соответствующие выбранным векторам ошибок:

$\bar{s}_{12}$ = 0001; $\bar{e}_{12}$ = 11000000000;
$\bar{s}_{13}$ = 0010; $\bar{e}_{13}$ = 10100000000;
$\bar{s}_{14}$ = 0100; $\bar{e}_{14}$ = 10010000000;
$\bar{s}_{15}$ = 0011; $\bar{e}_{15}$ = 01100000000.

Выполним процесс исправления двойной ошибки $\bar{e}_{14}$ в векторе $\bar{y}_1$ на основе стандартной таблицы декодирования и найденных синдромов. Вектор с выбранной двойной ошибкой будет:

$$\bar{y}_1 \oplus \bar{e}_{14} = \mathbf{1}10\mathbf{1}1110110.$$

Найдём для данного вектора по системе проверок из примера 6.2 синдром:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $p_1$ | $p_2$ | $p_3$ | $p_4$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 |

$$\begin{cases} 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0, \\ 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 1, \\ 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0, \\ 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0. \end{cases}$$

Синдром 0100 соответствует ошибке $\bar{e}_{14}$ = 10010000000. По таблице 6.2 находим в строке 14 принятое кодовое слово в столбце, соответствующем кодовому вектору $\bar{y}_1$.

Следовательно, принятому вектору 11011110110 соответствует кодовый вектор $\bar{y}_1$ = 01001110110. ❒

## 6.3. Порядок выполнения работы и варианты заданий

***Основные этапы выполнения работы***

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к практической работе.
2. Построить порождающую матрицу *G* для линейного блокового кода, способного обнаруживать и исправлять одиночную ошибку. Для заданных последовательностей информационных символов (табл. 6.3) получить кодовые вектора линейного блокового кода.
3. На основе матрицы-дополнения *P* к порождающей матрице *G* получить систему проверок для нахождения синдрома. Построить проверочную матрицу линейного блокового кода. Показать процесс исправления одиночной ошибки в произвольном символе полученных кодовых векторов на основе синдрома и проверочной матрицы.
4. Построить стандартную таблицу декодирования линейного блокового кода, позволяющую исправить максимально возможное число двойных ошибок. Найти для выбранных двойных ошибок синдромы. Показать процесс исправления двойной ошибки в одном из кодовых векторов с помощью построенной таблицы.
5. Оформить и защитить отчет по практической работе.

***Индивидуальные варианты заданий.***

Таблица 6.3

Варианты заданий для построения и декодирования линейных блоковых кодов

| **Вар.** | **Информационные векторы** | | |
|---|---|---|---|
| **1** | 10111010 | 11010101 | 10101011 |
| **2** | 00010100 | 00001101 | 00100010 |
| **3** | 10010100 | 00110001 | 10011101 |
| **4** | 00011110 | 01100000 | 00100010 |
| **5** | 11100011 | 00011111 | 10011101 |
| **6** | 01001010 | 00110010 | 01000100 |

| Вар. | Информационные векторы | | |
|---|---|---|---|
| **7** | 10001011 | 00001101 | 11010001 |
| **8** | 00010101 | 10001110 | 01111001 |
| **9** | 11011110 | 10110001 | 10100001 |
| **10** | 01000010 | 10011110 | 01011101 |
| **11** | 11000011 | 11000001 | 10011101 |
| **12** | 00011010 | 11010100 | 00101010 |
| **13** | 10000111 | 11000110 | 10010001 |
| **14** | 00001110 | 10011100 | 00010011 |
| **15** | 10011100 | 00110101 | 11011101 |
| **16** | 00101100 | 00101000 | 11000100 |
| **17** | 00010111 | 11101110 | 00010001 |
| **18** | 10110100 | 01011001 | 11000100 |
| **19** | 00010011 | 10000110 | 11100011 |
| **20** | 00101110 | 00100100 | 01101000 |
| **21** | 10101101 | 11010000 | 01100011 |
| **22** | 10000110 | 00100100 | 00011000 |
| **23** | 00000101 | 11100100 | 01111011 |
| **24** | 00000110 | 00100110 | 01010101 |

### *Требования к отчёту*

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Описание построения линейного блокового кода и кодирования информационных векторов с помощью порождающей матрицы.
4. Описание декодирования кодовых векторов по синдрому с исправлением одиночной ошибки.
5. Описание процесса декодирования кодовых векторов с исправлением двойной ошибки с помощью стандартной таблицы декодирования.
6. Выводы по лабораторной работе.

## 6.4.Контрольные вопросы и задачи

### *Теоретические вопросы*

1. Какие помехоустойчивые коды называют блоковыми?
2. Что такое расстояние Хэмминга?
3. Что называют кодовым расстоянием?
4. В чём заключается основная особенность линейных кодов?
5. Как определяют расстояние Хэмминга и кодовое расстояние для линейных блоковых кодов?
6. Что определяет оценка Хэмминга?
7. Какие коды называют совершенными?
8. Как с помощью порождающей матрицей линейного блокового кода осуществляется кодирование информационных слов?
9. Какими соображениями руководствуются при построении матрицы-дополнения для порождающей матрицы линейного блокового кода?
10. Как с помощью проверочной матрицы линейного блокового кода можно определить принадлежность кодового вектора данному коду?
11. Что понимают под синдромом при декодировании линейных блоковых кодов?
12. Каким образом строится стандартная таблица декодирования линейного блокового кода?
13. Как по стандартной таблице декодирования выполняется исправление ошибок в принятых кодовых векторах?

### *Практические задачи*

1. Построить и декодировать линейный блоковый код.

Задан информационный вектор:

- $\boldsymbol{x}$ = **10101** (нечётный вариант);
- $\boldsymbol{x}$ = **01010** (чётный вариант).

Требуется:

• построить порождающую и проверочную матрицы линейного блокового кода, позволяющего исправить одиночную ошибку;

• перекодировать заданный информационный вектор ***x*** в вектор ***y*** линейного блокового кода;

• задать одиночную ошибку в векторе ***y*** и произвести его декодирование по синдрому с исправлением ошибки.

# 7. ПОСТРОЕНИЕ И ДЕКОДИРОВАНИЕ ЦИКЛИЧЕСКИХ КОДОВ

## 7.1. Цель и задачи работы

Цель работы – приобрести умение строить циклические коды, а также декодировать данные коды с обнаружением и исправлением ошибок в сообщениях.

Основные задачи работы:

• освоить построение циклических кодов с помощью порождающих полиномов и матриц;

• научиться декодировать циклические коды с помощью алгоритма Меггита.

Работа рассчитана на 4 часа.

## 7.2. Основные теоретические сведения

### 7.2.1. Математическое описание циклических кодов. Построение циклических кодов

***Понятие циклических кодов. Порождающие полиномы.***

Циклические коды составляют большую группу наиболее широко используемых на практике линейных блоковых кодов. Их основное свойство, давшее им название, состоит в том, что каждый вектор, получаемый из исходного кодового вектора путём циклической перестановки его символов, также является разрешённым кодовым вектором.

Сдвиг осуществляется справа налево, причем крайний левый символ каждый раз переносится в конец комбинации. Например, если $\bar{u} = [100110]$ является кодовым вектором циклического кода $C$, то $\bar{v} = [010011]$ также является вектором того же кода.

При описании циклических кодов $n$-разрядные кодовые вектора представляются в виде ***кодовых полиномов*** (многочленов) фиктивной переменной $x$:

$$[c_{n-1}\ldots c_1 c_0] \Leftrightarrow C_{n-1}(x) = c_{n-1}x^{n-1} + \ldots + c_1 x + c_0 = \sum_{i=0}^{n-1} c_i x^i . \quad (7.1)$$

Показатели степени $i$ переменной $x$ соответствуют номерам разрядов (начиная с нулевого), а коэффициентами $c_i$ при $x$ в общем случае являются элементы поля $GF(K)$, где $K$ – основание кода.

Например, двоичной кодовый вектор $\bar{c}$ = [1001101] циклического (7,4)-кода может быть представлен следующим полиномом:

$$C_6(x) = 1x^6 + 0x^5 + 0x^4 + 1x^3 + 1x^2 + 0x^1 + 1x^0 =$$

$$= x^6 + x^3 + x^2 + 1 \Leftrightarrow 1001101.$$

В теории циклических кодов для множества кодовых полиномов вводятся операции сложения, умножения и деления.

Идея построения циклических кодов базируется на использовании неприводимых в поле $GF(K)$ многочленов.

***Неприводимыми полиномами*** в поле $GF(K)$ называются полиномы, которые не могут быть представлены в виде произведения полиномов низших степеней с коэффициентами из $GF(K)$. Они так же, как простые числа не могут быть представлены произведениями других чисел. То есть неприводимые полиномы делятся без остатка только на себя или на единицу.

Примерами неприводимых полиномов в поле $GF(2)$ являются:

- $x^3 + x + 1 \Leftrightarrow 1011$;
- $x^5 + x^2 + 1 \Leftrightarrow 100101$;
- $x^8 + x^5 + x^3 + x + 1 \Leftrightarrow 100101011$.

Неприводимые полиномы в теории циклических кодов играют такую же роль, как порождающие матрицы для линейных блоковых кодов.

***Порождающим полиномом*** циклического $(n,m)$-кода $C$ называется неприводимый полином $G_k(x)$ степени $k = n - m$, при умножении на который информационного полинома $F_{m-1}(x)$ получается полином циклического кода:

$$C_{n-1}(x) = F_{m-1}(x) \cdot G_k(x). \qquad (7.1)$$

Любой полином $C_{n-1}(x)$ циклического кода делится на порождающий полином $G_k(x)$ без остатка. Если при делении полученного при передаче полинома на порождающий полином будет получен остаток, то в принятом полиноме присутствует ошибка.

Получающиеся в результате умножения кодовые полиномы не содержат информационные символы в явном виде. После исправления ошибок такие полиномы для выделения информационных символов приходится делить на порождающий полином кода.

В ***систематических циклических кодах*** $m$ символов, соответствующих старшим степеням кодового полинома, отводят под информационные символы, а под проверочные символы отводят $k = n - m$ символов, соответствующих младшим степеням.

Для получения систематического циклического ($n$, $m$)-кода, применяется следующая процедура кодирования:

1. Информационный полином $F_{m-1}(x)$ умножают на $x^k$, где $k = n - m$:

$$A_{n-1}(x) = F_{m-1}(x) \cdot x^k.$$

2. Произведение $A_{n-1}(x)$ делят на порождающий полином $G_k(x)$, получая при этом частное $Q_{m-1}(x)$ и остаток $R_{k-1}(x)$:

$$A_{n-1}(x) / G_k(x) = Q_{m-1}(x) + R_{k-1}(x) / G_k(x).$$

3. Остаток $R_{k-1}(x)$ прибавляют к $A_{n-1}(x)$, получая кодовый полином систематического циклического кода:

$$C_{n-1}(x) = A_{n-1}(x) + R_{k-1}(x).$$

❒ ***Пример 7.1. Получение кодовых полиномов циклического кода с помощью порождающего полинома.***

Требуется перекодировать информационные векторы 1001, 1110, 0101 в кодовые полиномы циклического кода, исправляющего одиночные ошибки. Для кодирования будем использовать порождающий полином $G(x)$ циклического кода. Дополнительно получим кодовые полиномы систематического циклического кода.

• ***Выбор порождающего полинома циклического кода***.

Код, исправляющий одиночные ошибки, должен обеспечивать между комбинациями минимальное кодовое расстояние $d = 3$. Число проверочных символов $k$ при известном числе информационных символов $m = 4$ по формуле (6.3) будет:

$$k = \lceil \log_2 \{(4+1) + \lceil \log_2 (4+1) \rceil\} \rceil = \lceil \log_2 (5+3) \rceil = 3.$$

Из табл. П.2 приложения требуется выбрать полином, степень которого равна 3, корректирующая способность $t = 1$. Отсюда требуемый полином будет 1101 или

$$G_3(x) = x^3 + x^2 + 1.$$

В форме полиномов информационные векторы запишутся следующим образом:

$$1001 \Leftrightarrow x^3 + 1;$$
$$1110 \Leftrightarrow x^3 + x^2 + x;$$
$$0101 \Leftrightarrow x^2 + 1.$$

***• Получение кодовых полиномов несистематического циклического кода.***

Для выполнения кодирования умножим порождающий полином $G_3(x)$ на информационные полиномы:

$$\begin{array}{r} x^3 + x^2 + 1 \\ \times \quad x^3 + 1 \\ \hline x^3 + x^2 + 1 \\ \oplus \quad x^6 + x^5 + x^3 \\ \hline x^6 + x^5 + x^2 + 1 \end{array} \qquad \begin{array}{r} x^3 + x^2 + 1 \\ \times \quad x^3 + x^2 + x \\ \hline x^4 + x^3 + x \\ \oplus \; x^5 + x^4 + x^2 \\ \hline x^6 + x^5 + x^3 \\ \hline x^6 + x^2 + x \end{array} \qquad \begin{array}{r} x^3 + x^2 + 1 \\ \times \quad x^2 + 1 \\ \hline x^3 + x^2 + 1 \\ \oplus \quad x^5 + x^4 + x^2 \\ \hline x^5 + x^4 + x^3 + 1 \end{array}$$

Полученным кодовым полиномам будут соответствовать следующие кодовые векторы:

$$x^6 + x^5 + x^2 + 1 \Leftrightarrow 1100101;$$
$$x^6 + x^2 + x \Leftrightarrow 1000110;$$
$$x^5 + x^4 + x^3 + 1 \Leftrightarrow 0111001.$$

В векторной форме умножение полиномов будет иметь следующий вид:

$$\begin{array}{r} 1101 \\ \times\ 1001 \\ \hline 1101 \\ \oplus\ 1101 \\ \hline 1100101 \end{array} \qquad \begin{array}{r} 1101 \\ \times\ 1110 \\ \hline 1101 \\ \oplus\ 1101 \\ \hline 1000110 \end{array} \qquad \begin{array}{r} 1101 \\ \times\ 0101 \\ \hline 1101 \\ \oplus\ 1101 \\ \hline 0111001 \end{array}$$

Таким образом, при кодировании были получены кодовые полиномы: **1100101**, **1000110** и **0111001**.

• ***Получение кодовых полиномов систематического циклического кода.***

Для получения систематических кодовых полиномов требуется найти значения $k$ проверочных символов, которые будут добавлены справа к символам информационных векторов.

Выполним умножение информационных полиномов 1001, 1110, 0101 на одночлен той же степени по модулю $x^7 - 1$, то есть умножим их на $x^3$, что эквивалентно приписыванию трёх нулей справа. Осуществляется эта процедура для того, чтобы впоследствии вместо этих нулей можно было записать проверочные символы.

Умножив информационные полиномы на $x^3$, получим:

$[x^3(x^3 + 1)] \bmod (x^7 - 1) = x^6 + x^3 \Leftrightarrow 1001000;$

$[x^3(x^3 + x^2 + x)] \bmod (x^7 - 1) = x^6 + x^5 + x^4 \Leftrightarrow 1110000;$

$[x^3(x^2 + x)] \bmod (x^7 - 1) = x^5 + x^3 \Leftrightarrow 0101000.$

Найдём остатки от деления полученных произведений на порождающий полином $G_3(x)$. Эти остатки определяют значения проверочных символов, приписываемых к символам информационных полиномов справа.

Процесс нахождения остатков в полиномиальной и векторной формах будет иметь следующий вид:

• Для 1001000:

$$\begin{array}{l|l}
\oplus\ x^6+x^3 & \underline{x^3+x^2+1} \\
\underline{x^6+x^5+x^3} & x^3+x^2+x+1 \\
\oplus\ x^5 & \\
\underline{x^5+x^4+x^2} & \\
\oplus\ x^4+x^2 & \\
\underline{x^4+x^3+x} & \\
\oplus\ x^3+x^2+x+1 & \\
\underline{x^3+x^2+1} & \\
x+1 &
\end{array}
\qquad
\begin{array}{l|l}
\oplus\ 1001000 & \underline{1101} \\
\underline{1101} & 1111 \\
\oplus\ 1000 & \\
\underline{1101} & \\
\oplus\ 1010 & \\
\underline{1101} & \\
\oplus\ 1110 & \\
\underline{1101} & \\
11 &
\end{array}$$

Остаток: $x + 1 \Leftrightarrow 011$.

• Для 1110000:

$$\begin{array}{l|l}
\oplus\ x^6+x^5+x^4 & \underline{x^3+x^2+1} \\
\underline{x^6+x^5+x^3} & x^3+x \\
\oplus\ x^4+x^3 & \\
\underline{x^4+x^3+x} & \\
x &
\end{array}
\qquad
\begin{array}{l|l}
\oplus\ 1110000 & \underline{1101} \\
\underline{1101} & 1010 \\
\oplus\ 1100 & \\
\underline{1101} & \\
10 &
\end{array}$$

Остаток: $x \Leftrightarrow 010$.

• Для 0101000:

$$\begin{array}{l|l}
\oplus\ x^5+x^3 & \underline{x^3+x^2+1} \\
\underline{x^5+x^4+x^2} & x^2+x \\
\oplus\ x^4+x^3+x^2 & \\
\underline{x^4+x^3+x} & \\
x^2+x &
\end{array}
\qquad
\begin{array}{l|l}
\oplus\ 0101000 & \underline{1101} \\
\underline{1101} & 0110 \\
\oplus\ 1110 & \\
\underline{1101} & \\
110 &
\end{array}$$

Остаток: $x^2 + x \Leftrightarrow 110$.

Отсюда кодовые полиномы систематического циклического кода будут: 1001**011**, 1110**010**, 0101**110**. ❐

***Порождающие матрицы циклических кодов.***

Поскольку циклические коды являются разновидностью линейных блоковых кодов, то для их построения можно использовать порождающие матрицы.

Все строки порождающей матрицы $G_{m,n}$ циклического $(n,m)$-кода могут быть получены циклическим вправо сдвигом одного вектора $\bar{g}_k = [g_k, g_{k-1}, \ldots, g_1, g_0]$, символы которого соответствуют коэффициентам порождающего полинома $G_k(x)$.

Порождающая матрица $G_{m,n}$ циклического кода в общем случае имеет следующий вид:

$$G_{m,n} = \begin{bmatrix} g_k & g_{k-1} & \ldots & g_1 & g_0 & 0 & \ldots & 0 & 0 \\ 0 & g_k & \ldots & g_2 & g_1 & g_0 & \ldots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 0 & g_k & g_{k-1} & \ldots & g_1 & g_0 \end{bmatrix}.$$

Для построения систематического циклического кода требуется использовать порождающие матрицы в приведённой форме:

$$G_{m,n} = \left[E_m \mid P_{m,k}\right] = \left[\begin{array}{cccc|ccc} 1 & 0 & \ldots & 0 & p_{11} & p_{12} & \ldots & p_{1k} \\ 0 & 1 & \ldots & 0 & p_{21} & p_{22} & \ldots & p_{2k} \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & 1 & p_{m1} & p_{m2} & \ldots & p_{mk} \end{array}\right].$$

Порождающая матрица в приведённой форме может быть получена из матрицы циклического кода путём выполнения эквивалентных преобразований: перестановка любых двух строк; сложение какой-либо строки с другой строкой.

❒ ***Пример 7.2. Получение кодовых полиномов циклического кода с помощью порождающих матриц.***

Требуется, используя метод порождающих матриц, построить циклический код, обеспечивающий исправление одиночных ошибок при передаче информационных векторов 1001, 1110, 0101. Дополнительно получим порождающую матрицу в приве-

дённой форме и с её помощью перекодируем информационные векторы в систематический циклический код.

Поскольку в исходном коде $m = 4$, то порождающая матрица циклического кода должна содержать 4 строки. В примере 7.1 было получено, что число проверочных символов $k = 3$, поэтому число столбцов порождающей матрицы будет $n = 4 + 3 = 7$.

***• Кодирование с помощью порождающей матрицы несистематического циклического кода.***

Порождающая матрица может быть получена путем умножения строк единичной матрицы размером 4 на порождающий полином $x^3 + x^2 + 1$:

$$1000 \times 1101 = 1101000;$$
$$0100 \times 1101 = 0110100;$$
$$0010 \times 1101 = 0011010;$$
$$0001 \times 1101 = 0001101.$$

Отсюда порождающая матрица циклического кода будет иметь следующий вид:

$$G_{4;7} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Строки порождающей матрицы являются первыми четырьмя комбинациями искомого кода. Остальные кодовые комбинации могут быть найдены путем суммирования по модулю 2 строк порождающей матрицы.

Для кодирования циклическим кодом информационных векторов 1001, 1110, 0101, умножим векторы на порождающую матрицу.

1). **1001**:

$$\begin{array}{r} 1101000 \\ \oplus\ 0001101 \\ \hline 1100101 \end{array}$$

2). **1110**:

$$\begin{array}{r} 1101000 \\ \oplus\ 0110100 \\ \hline 0011010 \\ \hline 1000110 \end{array}$$

3). **0101**:

$$\begin{array}{r} 0110100 \\ \oplus\ 0001101 \\ \hline 0111001 \end{array}$$

Таким образом, при кодировании были получены кодовые полиномы: **1100101**, **1000110** и **0111001**, что совпадает с результатом из примера 7.1.

• ***Кодирование с помощью порождающей матрицы систематического циклического кода.***

Для построения систематического циклического кода получим порождающую матрицу в приведённой форме. Для этого выполним следующие эквивалентные преобразования: 1) заменим первую строку на её сумму со второй и третьей строкой; 2) заменим вторую строку на её сумму с третьей и четвёртой строкой; 3) заменим третью строку на её сумму с четвёртой строкой:

$$\begin{array}{lll} 1). & 2). & 3). \\ \begin{array}{r} 1101000 \\ \oplus\, 0110100 \\ \underline{0011010} \\ 1000110 \end{array} & \begin{array}{r} 0110100 \\ \oplus\, 0011010 \\ \underline{0001101} \\ 0100011 \end{array} & \begin{array}{r} 0011010 \\ \oplus\, \underline{0001101} \\ 0010111 \end{array} \end{array}$$

Отсюда порождающая матрица в приведённой форме будет:

$$G_{4;7} = [E_4 \mid P_{4;3}] = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right].$$

Выполним кодирование заданных информационных векторов с помощью полученной порождающей матрицы:

$$\begin{array}{lll} 1). \mathbf{1001} & 2). \mathbf{1110} & 3). \mathbf{0101} \\ \begin{array}{r} 110 \\ \oplus\, \underline{101} \\ 011 \end{array} & \begin{array}{r} 110 \\ \oplus\, 011 \\ \underline{111} \\ 010 \end{array} & \begin{array}{r} 011 \\ \oplus\, \underline{101} \\ 110 \end{array} \end{array}$$

Отсюда кодовые векторы систематического циклического кода будут: 1001**011**, 1110**010**, 0101**110**, что совпадает с результатом, полученным в примере 7.1. ❐

### 7.2.2. Декодирование циклических кодов, исправляющих одиночные ошибки

***Декодирование с исправлением одиночных ошибок***

Ошибки в циклических кодах обнаруживаются и исправляются при помощи остатков от деления полученного полнома на порождающий полином. Остатки от деления являются опознавателями ошибок, но не указывают непосредственно на место ошибки в циклическом коде.

Идея исправления ошибок основывается на том, что ошибочный кодовый полином после определенного числа циклических сдвигов подгоняется под остаток таким образом, что в сумме с остатком он дает исправленный полином. Остаток при этом представляет собой разницу между искажёнными и правильными символами.

Подгоняют ошибочный полином до тех пор, пока число единиц в остатке $w$ не будет меньше или равно максимальному числу ошибок $t$, исправляемых данным кодом.

Процесс исправления одиночной ошибки с помощью двоичного циклического кода включает следующие шаги:

1. Делят принятый кодовый полином $D_{n-1}(x)$ на порождающий полином $G_k(x)$:

$$D_{n-1}(x) / G_k(x) = Q_{m-1}(x) + S_{k-1}(x) / G_k(x).$$

где $Q_{m-1}(x)$ – частное; $S_{k-1}(x)$ – остаток.

Если $S_{k-1}(x) = 0$, то принятый полином не содержит ошибки. Если $S_{k-1}(x) \neq 0$, то в полиноме присутствует ошибка и требуется перейти к пункту 2.

2. Определяют вес $w$ полученного остатка $S_{k-1}(x)$ путем подсчета количества единиц в остатке. Если $w \leq t$, где $t$ – максимальное число исправляемых кодом ошибок (корректирующая способность), то принятый полином $D_{n-1}(x)$ складывается по модулю 2 с полученным остатком $S_{k-1}(x)$. Сумма даст исправленный полином. Если $w > t$, то переходят к пункту 3.

3. Производят циклический сдвиг принятого полинома $F(x)$ на один разряд влево. Комбинацию, полученную в результате циклического сдвига, делят на порождающий полином $G_k(x)$.

4. Повторяют пункты 2 и 3 до тех пор, пока не будет выполнено условие $w \le t$. Запоминают число сдвигов $r$ полинома.

5. Полином, полученный в результате последнего циклического сдвига, складывают с остатком от деления этого полнома на порождающий полином $G_k(x)$.

6. Производят циклический сдвиг полинома, полученного в результате суммирования последнего делимого с последним остатком вправо на $r$ разрядов. В результате будет получена исправленная кодовая комбинация $C_{m-1}(x)$.

❒ ***Пример* 7.3. *Декодирование кодовых полиномов циклического кода с исправлением одиночной ошибки.***

Требуется показать процесс исправления одиночной ошибки при декодировании кодовых полиномов **1100101**, **1000110** и **0111001**, полученных в примерах 7.1 и 7.2.

Порождающий полином циклического кода имеет следующий вид: $G_3(x) = x^3 + x^2 + 1$.

Декодирование будет производится с помощью алгоритма Меггита.

Пусть в передаваемых кодовых полиномах произошли следующие одиночные ошибки:

$$1\mathbf{0}00101;\ 100\mathbf{1}110;\ 0111000\mathbf{0}.$$

Выполним декодирование каждого из полиномов:

• ***Декодирование полинома* 1000101**.

Разделим принятые полиномы на $G_3(x)$ в полиномиальной и векторной формах:

$$\begin{array}{l|l} x^6 + x^2 + 1 & \underline{x^3 + x^2 + 1} \\ \underline{x^6 + x^5 + x^3} & x^3 + x^2 + x \\ x^5 + x^3 + x^2 + 1 & \\ \underline{x^5 + x^4 + x^2} & \\ x^4 + x^3 + 1 & \\ \underline{x^4 + x^3 + x} & \\ x + 1 & \end{array} \qquad \begin{array}{l|l} 1000101 & \underline{1101} \\ \underline{1101} & 1110 \\ 1011 & \\ \underline{1101} & \\ 1100 & \\ \underline{1101} & \\ 11 & \end{array}$$

Остаток $S(x) = x + 1 \Leftrightarrow 011$; вес остатка $w = 2$.

Поскольку $w > t$, то сдвинем влево на один разряд полученный полином, умножив его на $x$ по модулю $x^7 - 1$:

$[x(x^6 + x^2 + 1)] \bmod (x^7 - 1) = x^3 + x + 1 \Leftrightarrow 0001011.$

Разделим полином $x^3 + x + 1$ на $G(x)$.

$$\begin{array}{l|l} x^3 + x + 1 & \underline{x^3 + x^2 + 1} \\ \underline{x^3 + x^2 + 1} & 1 \\ x^2 + x & \end{array} \qquad \begin{array}{l|l} 0001011 & \underline{1101} \\ \underline{1101} & 0001 \\ 110 & \end{array}$$

Остаток $S(x) = x^2 + x \Leftrightarrow 011$; вес остатка $w = 2$.

Поскольку $w > t$, то снова сдвигаем полином влево на один разряд:

$x(x^3 + x + 1) \bmod (x^7 - 1) = x^4 + x^2 + x \Leftrightarrow 0010110.$

Разделим полином $x^4 + x^2 + x$ на $G(x)$.

$$\begin{array}{l|l} x^4 + x^2 + x & \underline{x^3 + x^2 + 1} \\ \underline{x^4 + x^3 + x} & x + 1 \\ x^3 + x^2 & \\ \underline{x^3 + x^2 + 1} & \\ 1 & \end{array} \qquad \begin{array}{l|l} 0010110 & \underline{1101} \\ \underline{1101} & 0011 \\ 1100 & \\ \underline{1101} & \\ 1 & \end{array}$$

Остаток $S(x) = 1 \Leftrightarrow 001$; вес остатка $w = 1$.

Поскольку $w = t$, то сложим полученный полином по модулю 2 с остатком:

$$\begin{array}{r} 0010110 \\ \underline{\quad 001} \\ 0010111 \end{array} \quad \Leftrightarrow \quad x^4 + x^2 + x + 1.$$

Так как число сдвигов влево $r = 2$, то выполним сдвиг суммы на два разряда вправо, разделив её на $x^2$ по модулю $x^7 - 1$:

$[(x^4 + x^2 + x + 1) / x^2] \bmod (x^7 - 1) = x^6 + x^5 + x^2 + 1 \Leftrightarrow 1100101.$

Результат совпадает с кодовым полиномом **1100101**.

Для завершения декодирования разделим кодовый полином 1100101 на $G(x)$:

$$\begin{array}{l|l} x^6+x^5+x^2+1 & \underline{x^3+x^2+1} \\ \underline{x^6+x^5+x^3} & x^3+1 \\ x^3+x^2+1 & \\ \underline{x^3+x^3+1} & \\ 0 & \end{array} \qquad \begin{array}{l|l} 1100101 & \underline{1101} \\ \underline{1101} & 1001 \\ 1101 & \\ \underline{1101} & \\ 0 & \end{array}$$

Таким образом, был получен исходный информационный вектор **1001**.

• ***Декодирование полинома*** **1001110**:

Разделим полученный полином на порождающий полином $G(x)$:

$$\begin{array}{l|l} x^6+x^3+x^2+x & \underline{x^3+x^2+1} \\ \underline{x^6+x^5+x^3} & x^3+x^2+x+1 \\ x^5+x^2+x & \\ \underline{x^5+x^4+x^2} & \\ x^4+x & \\ \underline{x^4+x^3+x} & \\ x^3 & \\ \underline{x^3+x^2+1} & \\ x^2+1 & \end{array} \qquad \begin{array}{l|l} 1001110 & \underline{1101} \\ \underline{1101} & 1111 \\ 1001 & \\ \underline{1101} & \\ 1001 & \\ \underline{1101} & \\ 1000 & \\ \underline{1101} & \\ 101 & \end{array}$$

Остаток $S(x) = x^2 + 1 \Leftrightarrow 101$; вес остатка $w = 2$.

Поскольку $w > t$, то требуется сдвинуть полученный полином влево на один разряд, умножив его на $x$ по модулю $x^7 - 1$:

$[x(x^6 + x^3 + x^2 + x)] \bmod (x^7 - 1) = x^4 + x^3 + x^2 + 1 \Leftrightarrow 0011101.$

Разделим полином $x^4 + x^3 + x^2 + 1$ на $G(x)$.

$$\begin{array}{l|l} \oplus\ x^4+x^3+x^2+1 & \underline{x^3+x^2+1} \\ \underline{x^4+x^3+x\phantom{+1}} & x \\ x^2+x+1 & \end{array} \qquad \begin{array}{l|l} \oplus\ 0011101 & \underline{1101} \\ \underline{\ \ 1101\ \ \ } & 0010 \\ 111 & \end{array}$$

Остаток $S(x) = x^2 + x + 1 \Leftrightarrow 111$; вес остатка $w = 3$.

Поскольку $w > t$, то необходимо ещё раз сдвинуть полином влево на один разряд:

$x(x^4 + x^3 + x^2 + 1) \bmod (x^7 - 1) = x^5 + x^4 + x^3 + x \Leftrightarrow 0111010.$

Разделим полином $x^5 + x^4 + x^3 + x$ на $G(x)$:

$$\begin{array}{l|l} \oplus\ x^5+x^4+x^3+x & \underline{x^3+x^2+1} \\ \underline{x^5+x^4+x^2\phantom{+x}} & x^2+1 \\ \oplus\ x^3+x^2+x & \\ \underline{x^3+x^2+1} & \\ x+1 & \end{array} \qquad \begin{array}{l|l} \oplus\ 0111010 & \underline{1101} \\ \underline{\ \ 1101\ \ \ } & 0110 \\ \oplus\ 1110 & \\ \underline{\ \ 1101\ \ \ } & \\ 11 & \end{array}$$

Остаток $S(x) = x + 1 \Leftrightarrow 011$; вес остатка $w = 2$.

Поскольку $w > t$, то снова требуется сдвинуть полином влево на один разряд:

$[x(x^5 + x^4 + x^3 + x)] \bmod (x^7 - 1) = x^6 + x^5 + x^4 + x^2 \Leftrightarrow 1110100.$

Разделим полином $x^6 + x^5 + x^4 + x^2$ на $G(x)$:

$$\begin{array}{l|l} \oplus\ x^6+x^5+x^4+x^2 & \underline{x^3+x^2+1} \\ \underline{x^6+x^5+x^3\phantom{+x}} & x^3+x \\ \oplus\ x^4+x^3+x^2 & \\ \underline{x^4+x^3+x\phantom{+1}} & \\ x^2+x & \end{array} \qquad \begin{array}{l|l} \oplus\ 1110100 & \underline{1101} \\ \underline{\ \ 1101\ \ \ } & 1010 \\ \oplus\ 1110 & \\ \underline{\ \ 1101\ \ \ } & \\ 11 & \end{array}$$

Остаток $S(x) = x^2 + x \Leftrightarrow 110$; вес остатка $w = 2$.

Поскольку $w > t$, то ещё раз сдвигаем полином влево на один разряд:

$[x(x^6 + x^5 + x^4 + x^2)] \bmod (x^7 - 1) = x^6 + x^5 + x^3 + 1 \Leftrightarrow 1101001.$

Разделим полином $x^6 + x^5 + x^3 + 1$ на $G(x)$:

$$\begin{array}{l|l} x^6+x^5+x^3+1 & \underline{x^3+x^2+1} \\ \underline{x^6+x^5+x^3} & x^3 \\ 1 & \end{array} \qquad \begin{array}{l|l} 1101001 & \underline{1101} \\ \underline{1101} & 1000 \\ 1 & \end{array}$$

Остаток $S(x) = 1 \Leftrightarrow 001$; вес остатка $w = 1$.

Поскольку $w = t$, то сложим полином $x^6 + x^5 + x^3 + 1$ по модулю 2 с остатком:

$$\begin{array}{r} 1101001 \\ \underline{\quad\;\; 001} \\ 1101000 \end{array} \quad \Leftrightarrow \quad x^6 + x^5 + x^3.$$

Число сдвигов влево $r = 4$, поэтому сдвинем полученную сумму на 4 разряда вправо, разделив её на $x^4$ по модулю $x^7 - 1$:

$[(x^6 + x^5 + x^3) / x^4] \bmod (x^7 - 1) = x^6 + x^2 + x \Leftrightarrow 1000110.$

Результат совпадает с кодовым полиномом **1000110**.

Завершим декодирование, разделив 1000110 на $G(x)$:

$$\begin{array}{l|l} x^6+x^2+x & \underline{x^3+x^2+1} \\ \underline{x^6+x^5+x^3} & x^3+x^2+x \\ x^5+x^3+x^2+x & \\ \underline{x^5+x^4+x^2} & \\ x^4+x^3+x & \\ \underline{x^4+x^3+x} & \\ 0 & \end{array} \qquad \begin{array}{l|l} 1000110 & \underline{1101} \\ \underline{1101} & 1110 \\ 1011 & \\ \underline{1101} & \\ 1101 & \\ \underline{1101} & \\ 0 & \end{array}$$

Получен исходный информационный вектор **1110**.

• ***Декодирование полинома*** **0111000**:

$$\begin{array}{l} \oplus \begin{array}{l|l} x^5+x^4+x^3 & \underline{x^3+x^2+1} \\ \underline{x^5+x^4+x^2} & x^2+1 \end{array} \\ \oplus \begin{array}{l} x^3+x^2 \\ \underline{x^3+x^2+1} \end{array} \\ 1 \end{array} \qquad \begin{array}{l} \oplus \begin{array}{l|l} 0111000 & \underline{1101} \\ \underline{1101\phantom{000}} & 0101 \end{array} \\ \oplus \begin{array}{r} 1100 \\ \underline{1101} \end{array} \\ 1 \end{array}$$

Остаток $S(x) = 1 \Leftrightarrow 001$; вес остатка $w = 1$.

Поскольку $w = t$, то для исправления ошибки сложим полученный полином с остатком:

$$\begin{array}{r} \oplus 0111000 \\ \underline{\qquad 001} \\ 0111001 \end{array} \qquad \Leftrightarrow \quad x^5 + x^4 + x^3 + 1.$$

Полученный результат совпадает с исходным кодовым полиномом **0111001**.

Найдём исходный информационный вектор путём деления 0111001 на $G(x)$:

$$\begin{array}{l} \oplus \begin{array}{l|l} x^5+x^4+x^3+1 & \underline{x^3+x^2+1} \\ \underline{x^5+x^4+x^2\phantom{+1}} & x^2+1 \end{array} \\ \oplus \begin{array}{l} x^3+x^2+1 \\ \underline{x^3+x^2+1} \end{array} \\ 0 \end{array} \qquad \begin{array}{l} \oplus \begin{array}{l|l} 0111001 & \underline{1101} \\ \underline{1101\phantom{001}} & 0101 \end{array} \\ \oplus \begin{array}{r} 1101 \\ \underline{1101} \end{array} \\ 0 \end{array}$$

Следовательно, в результате декодирования получается информационный вектор **0101**. ❐

## 7.3. Порядок выполнения работы и варианты заданий

### *Основные этапы выполнения работы*

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.

2. Для заданных информационных векторов (табл. 7.1) и корректирующей способности $t = 1$ выбрать порождающий полином циклического кода (табл. П.2). С помощью порождающего полинома осуществить кодирование информационных векторов в циклический код. Перекодировать информационные векторы в систематический циклический код.

3. Построить порождающую матрицу циклического кода. С помощью порождающей матрицы произвести кодирование информационных векторов в циклический код. Получить порождающую матрицу в приведённой форме. Выполнить кодирование информационных векторов в систематический циклический код.

4. Задать в полученных кодовых полиномах несистематического циклического кода одиночные ошибки. Используя алгоритм Меггита, произвести декодирование полиномов с исправлением ошибок.

5. Оформить и защитить отчет по лабораторной работе.

### *Индивидуальные варианты заданий*

Таблица 7.1

Варианты заданий для построения и декодирования циклических кодов

| **Вар.** | **Информационные векторы** | | |
|---|---|---|---|
| **1** | 00110 | 10111 | 10101 |
| **2** | 11010 | 10100 | 11011 |
| **3** | 10100 | 11101 | 01010 |
| **4** | 00101 | 11001 | 11101 |
| **5** | 00111 | 01100 | 10111 |
| **6** | 11000 | 10111 | 01011 |
| **7** | 10010 | 01101 | 11110 |
| **8** | 01011 | 01111 | 10001 |
| **9** | 10101 | 10010 | 01111 |
| **10** | 10111 | 01010 | 10101 |
| **11** | 00110 | 11110 | 01011 |
| **12** | 01101 | 01001 | 11011 |
| **13** | 10010 | 10101 | 11101 |
| **14** | 10110 | 11011 | 01001 |
| **15** | 01110 | 01010 | 10111 |
| **16** | 10001 | 11101 | 01101 |

| Вар. | Информационные векторы | | |
|---|---|---|---|
| **17** | 01100 | 01010 | 10111 |
| **18** | 10101 | 01111 | 01100 |
| **19** | 11101 | 11001 | 10010 |
| **20** | 11001 | 00110 | 11110 |
| **21** | 01010 | 11101 | 00110 |
| **22** | 11110 | 10100 | 10110 |
| **23** | 10111 | 10001 | 01110 |
| **24** | 01011 | 01010 | 11110 |

***Требования к отчёту.***

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Описание кодирования информационных векторов с помощью порождающего полинома в систематический циклический код.
4. Описание кодирования информационных векторов с помощью порождающей матрицы систематического циклического кода.
5. Описание декодирования полученных кодовых полиномов систематического циклического кода с исправлением одиночной ошибки с помощью алгоритма Меггита.
6. Выводы по лабораторной работе.

## 7.4. Контрольные вопросы и задачи

***Теоретические вопросы***

1. В чем заключаются основные особенности циклических кодов?
2. Какие полиномы называют неприводимыми в поле $GF(q)$?
3. Что понимают под порождающими полиномами циклических кодов?
4. Как получают кодовые полиномы циклического кода с помощью порождающего полинома?

5. В чём заключаются особенности получения кодовых полиномов систематического циклического кода?

6. Каким образом строится порождающая матрица циклического кода?

7. При каком условии считают, что полученный кодовый полином не содержит ошибок?

8. Как в полученном кодовом полиноме осуществляется исправление ошибок?

***Практические задачи***

1. Построить и декодировать циклический код.

Задан информационный вектор:

- $\boldsymbol{x}$ = **10101** (нечётный вариант);
- $\boldsymbol{x}$ = **01010** (чётный вариант).

Требуется:

• используя порождающий полином $G_3(\boldsymbol{x}) = \boldsymbol{x}^3 + \boldsymbol{x}^2 + 1$ перекодировать информационный вектор x в кодовый полином циклического (7, 4)-кода;

• задать одиночную ошибку в кодовом полиноме и произвести его декодирование с исправлением ошибки, используя алгоритм Меггита.

# 8. ОСНОВЫ ЗАЩИТЫ ИНФОРМАЦИИ С ПОМОЩЬЮ СИММЕТРИЧНЫХ АЛГОРИТМОВ ШИФРОВАНИЯ

## 8.1. Цель и задачи работы

Цель работы – приобрести выполнять криптографическую защиту информацию на платформе .NET Framework с помощью таких симметричных алгоритмов шифрования, как DES, 3DES, RC2 и Rijndael (AES).

Основные задачи работы:

• освоить основные особенности работы симметричных алгоритмов шифрования;

• научиться использовать симметричные алгоритмы шифрования на платформе .NET Framework.

Работа рассчитана на 6 часов.

## 8.2. Основные теоретические сведения

### 8.2.1. Особенности симметричных шифров. Режимы работы блочных алгоритмов шифрования

***Особенности симметричных шифров. Блочные алгоритмы шифрования***

Традиционный подход к задаче криптографической защиты информации заключается в использовании симметричных шифров.

***Симметричными*** называют шифры, которые используют один и тот же ключ, как для шифрования, так и для расшифрования информации. В частном случае ключи шифрования и расшифрования могут различаться, но при наличии одного ключа можно легко вычислить другой. Поскольку единственный ключ полностью обеспечивает секретность шифра, то он должен храниться в тайне.

В противоположность этому в асимметричных шифрах используется пара ключей, между которыми существует математическая связь, однако вычислить один ключ по другому ключу крайне сложно.

При передаче секретной информации её защищают с помощью криптографических систем смешанного типа. С помощью асимметричных алгоритмов решается задача распределения ключей для симметричных шифров. А затем все передаваемые данные шифруются с помощью симметричного шифра.

Симметричные алгоритмы шифрования подразделяется на два вида:

• ***Блочные алгоритмы шифрования***, которые делят открытый текст на отдельные блоки, как правило, одинакового размера и оперируют с каждым из них с целью получения последовательности блоков шифрованного текста.

• ***Поточные алгоритмы шифрования***, которые производят обработку данных посимвольно или отдельными байтами без разделения на блоки. Поточные шифры, в отличие от блочных, обладают памятью предыдущего состояния шифра.

Блочные алгоритмы обладают большей стойкостью по сравнению с поточными алгоритмами. Кроме того, блочные алгоритмы являются более универсальными, поскольку имеют режимы, в которых они работают как поточные алгоритмы.

Современные блочные шифры строятся на основе многократного применения простых криптографических преобразований, к которым относятся перестановки и замены. Такой подход позволяет значительно повысить криптографическую стойкость и обеспечить два основных свойства, присущих стойким шифрам: рассеивание и перемешивание.

***Рассеиванием*** (*diffusion*) называется влияние любого знака открытого текста на знаки шифротекста. Рассеивание позволяет скрыть влияние статистических свойств открытого текста на свойства шифротекста. При использовании рассеивания малейшее изменение открытого текста должно вызывать значительные изменения шифротекста.

Под ***перемешиванием*** (*confusion*) понимают использование таких преобразований, которые обеспечивают полную зависимость шифротекста от ключа. Перемешивание усложняет восстановление взаимосвязи статистических свойств открытого и шифрованного текстов при неизвестном ключе.

Современными примерами симметричных алгоритмов шифрования являются: DES, 3DES, ГОСТ 28147-89, Blowfish, RC2, Rijndael.

Алгоритм ***DES*** (*Data Encription Standart* – стандарт шифрования данных) был предложен в 1977 г. в качестве стандарта шифрования данных для защиты важной, но несекретной информации в государственных и коммерческих организациях США.

Шифр DES имеет длину блока исходных данных равную 64 битам и ключ длиной 56 бит. Процесс шифрования состоит из начальной перестановки битов входного блока, шестнадцати циклов шифрования и конечной перестановке битов.

В настоящее время алгоритм DES уже не обеспечивает достаточно надёжную защиту ценной информации, но может быть полезен в различных коммерческих приложениях.

В качестве более надёжной альтернативы DES используется алгоритм «тройной» DES – ***3DES*** (*Triple DES, TDES*), который основывается на трёхкратном применении к каждому блоку открытого текста алгоритма DES с тремя разными ключами. При этом длина ключа увеличивается втрое, но также втрое увеличивается время, затрачиваемое на обработку.

Каждый блок открытого текста шифруется первым ключом, полученный результат расшифровывается вторым ключом, и, наконец, блок шифруют третьим ключом. Расшифрование осуществляется в обратном порядке.

Официальным приемником DES, известным под названием ***AES*** (Advanced Encryption Standard – усовершенствованный стандарт шифрования) является алгоритм ***Rijndael*** (произносится «райн дал»). В отличие от DES, где размер ключа фиксирован и равняется 56 бит, в AES могут использоваться ключи размером 128, 192 и 256 бит. Кроме того, вместо фиксированного 64-битового блока данных в DES, алгоритм AES может работать с блоками данных размерами 128, 192 и 256 бит.

Ещё одним широко известным симметричным алгоритмом шифрования является алгоритм ***RC2***, который шифрует данные блоками по 64 бита с использованием ключей переменного размера: от 8 до 1024 бит. Алгоритм RC2 в настоящее время является стандартом для криптографической защиты сообщений электронной почты.

### *Режимы работы блочных алгоритмов шифрования*

Один и тот же блочный алгоритм может применяться для шифрования в различных режимах. Каждый режим шифрования имеет свои достоинства и недостатки, поэтому выбор режима зависит от конкретной ситуации.

Современные блочные алгоритмы поддерживают следующие режимы работы:

- электронная шифровальная книга (Electronic Codebook – ECB);
- сцепление шифрованных блоков (Cipher Block Chaining – CBC);
- шифрованная обратная связь (Cipher Feedback – CFB)
- обратная связь по выходу (Ouput Feedback – OFB).
- проскальзывание шифрованного текста (Cipher Text Stealing – CTS).

Режим ECB является наиболее простым режимом работы, в котором обработка очередного блока производится независимо от других блоков. Данный режим позволяет произвольно шифровать записи в файле или в базе данных. Кроме того, данный режим позволяет обрабатывать блоки параллельно, что увеличивает скорость работы алгоритма.

Недостатком режима ECB является низкая криптографическая стойкость. Вследствие большой избыточности в открытом тексте возможны повторения блоков. Это приводит к тому, что одинаковые блоки открытого текста будут представлены одинаковыми блоками шифротекста, что упрощает восстановление блоков открытого текста по шифротексту.

Схема работы алгоритма в режиме ECB показана на рис. 8.1

Шифрование в режиме ECB описывается следующим образом:

$$C_j = E_k(M_j);$$

где $M_j$ и $C_j$ – $j$-ый блок открытого и шифрованного текста.

Расшифрование в режиме ECB имеет следующий вид:

$$M_j = D_k(C_j).$$

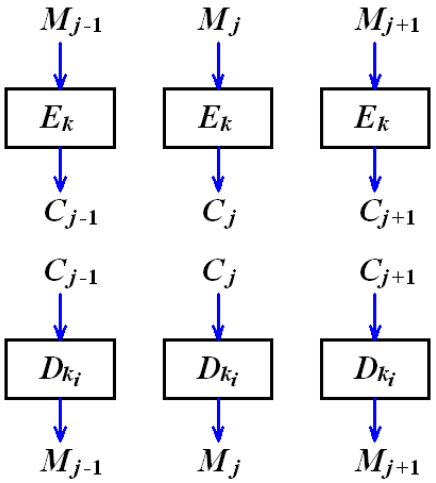


Рис. 8.1. Режим электронной шифровальной книги (ECB)

В режиме CBC каждый блок открытого текста суммируется по модулю 2 с предыдущим зашифрованным блоком. Самый первый блок открытого текста ($M_0$) суммируется по модулю 2 со случайным вектором, называемым ***вектором инициализации*** (IV). Схема работы алгоритма в режиме CBC показана на рис. 8.2.

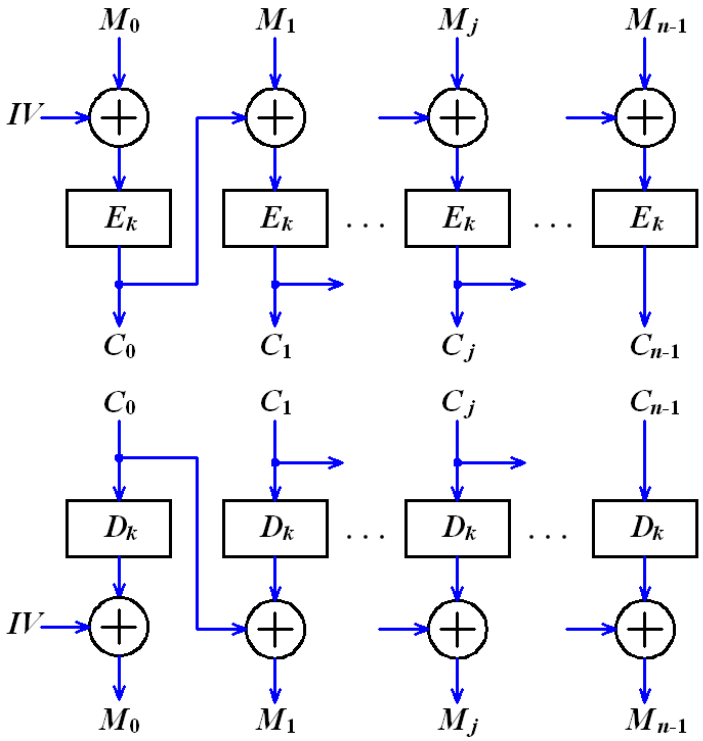


Рис. 8.2. Режим сцепления шифрованных блоков (CBC)

Шифрование в режиме CBC:

$$C_0 = E_k(M_0 \oplus IV);$$

$$C_j = E_k(M_j \oplus C_{j-1}).$$

Расшифрование в режиме CBC:

$$M_0 = D_k(C_0) \oplus IV;$$

$$M_j = D_k(C_j) \oplus C_{j-1}.$$

Режим CFB позволяет оперировать блоками открытого текста, меньшими стандартного блока, используемого в режимах ECB и CBC. При этом алгоритм работает как потоковый шифр.

Схема работы алгоритма в режиме CFB показана на рис. 8.3, где PC – регистр сдвига, выполняющий за один проход сдвиг влево на 8 бит. Сложение по модулю 2 выполняется для 8 крайних левых битов из PC.

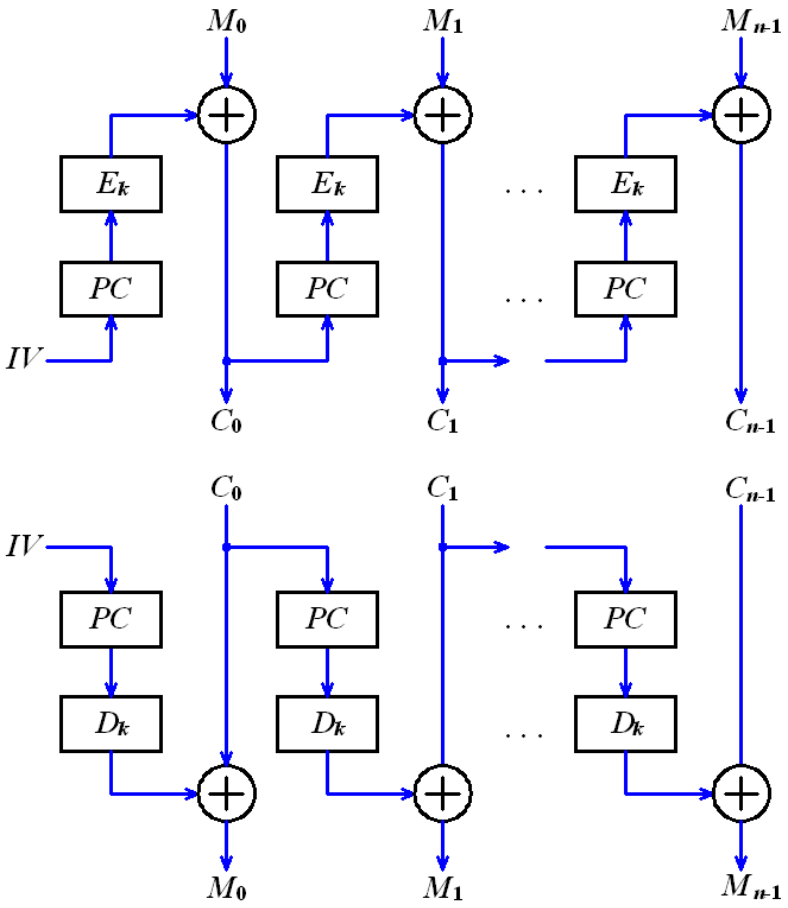


Рис. 8.3. Режим шифрованной обратной связи (CFB)

Шифрование в режиме CFB:

$$C_0 = M_0 \oplus E_k(IV);$$

$$C_j = M_j \oplus E_k(C_{j-1}).$$

Расшифрование в режиме CFB:

$$M_0 = C_0 \oplus D_k(IV);$$

$$M_j = C_j \oplus D_k(C_{j-1}).$$

Отличие режима OFB от CFB заключается в том, что в режиме OFB биты попадают в на вход сдвигового регистра перед операцией сложения по модулю 2.

### 8.2.2. Криптография в .NET Framework. Класс SymmetricAlgorithm. Криптографические потоки

***Криптография в .NET Framework. Класс SymmetricAlgorithm***

Система .NET Framework включает набор криптографических сервисов, расширяющих аналогичные сервисы Windows через программный интерфейс Cryptography API (CAPI).

Пространство имен **System.Security.Cryptography** открывает доступ к различным криптографическим сервисам, с помощью которых приложения могут шифровать данные, используя симметричные и асимметричные алгоритмы, могут обеспечивать целостность данных, а также обрабатывать цифровые подписи и сертификаты.

К числу поддерживаемых симметричных алгоритмов относятся DES, TripleDES, RC2, Rijndael и AES. Иерархия классов, представляющих симметричные алгоритмы шифрования, показана на рис. 8.4.

Классы .NET Framework, реализующие симметричные алгоритмы, являются производными от абстрактного класса **SymmetricAlgogithm**, который содержит набор виртуальных свойств и методов.

Класс **SymmetricAlgogithm** обладает следующими основными свойствами:

• **Key** – возвращает или задаёт секретный ключ для использования симметричным алгоритмом при шифровании или расшифровании; представляет собой массив значений типа **byte**;

• **IV** – возвращает или задаёт вектор инициализации для симметричного алгоритма, что требуется в режиме CBC; является массивом значений типа **byte**;

• **CipherMode** – возвращает или задаёт режим шифрования для симметричного алгоритма;

• **PaddingMode** – возвращает или задаёт тип заполнения блоков для симметричного алгоритма.

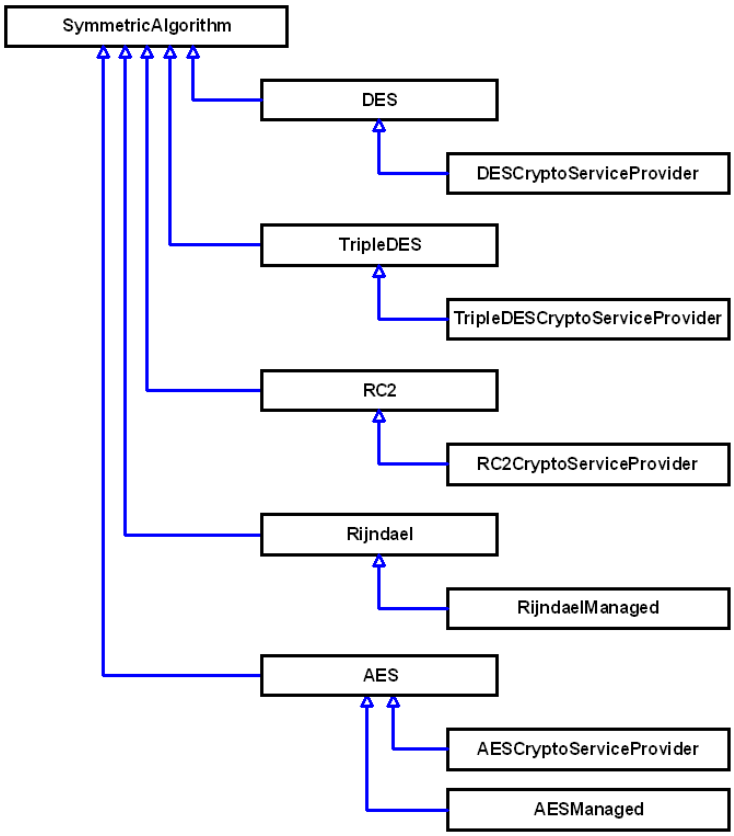


Рис. 8.4. Иерархия классов, представляющих симметричные алгоритмы

Класс **SymmetricAlgogithm** имеет следующие основные методы:

• **CreateEncryptor()** – создаёт объекта **encryptor** на основе интерфейса **ICryptoTransform**, используя заданный ключ и вектор инициализации для шифрования;

• **CreateDecryptor()** – создаёт объекта **decryptor** на основе интерфейса **ICryptoTransform**, используя заданный ключ и вектор инициализации для расшифрования;

• **Clear()** – освобождает ресурсы, занятые симметричным алгоритмом.

Производными от **SymmetricAlgogithm** являются следующие абстрактные классы:

• **DES** – представляет алгоритм DES;

• **TripleDES** – представляет алгоритм 3DES;

• **RC2** – представляет алгоритм RC2;

• **Rijndael** – представляет алгоритм Rijndael;

• **Aes** – представляет алгоритм AES.

Для работы с классами, представляющими симметричные алгоритмы шифрования, предусмотрены следующие запечатанные классы:

• **DESCryptoServiceProvider** – класс, предоставляющий доступ к реализации алгоритма DES, через интерфейс Cryptography API (CAPI);

• **TripleDESCryptoServiceProvider** – класс, предоставляющий доступ к алгоритму 3DES через CAPI;

• **RC2CryptoServiceProvider** – класс, предоставляющий доступ к алгоритму RC2 через CAPI;

• **RijndaelManaged** – управляемый класс, реализующий алгоритм Rijndael; этот класс не является оболочкой CAPI;

• **AesCryptoServiceProvider** – класс, предоставляющий доступ к алгоритму AES через CAPI;

• **AesManaged** – управляемый класс, реализующий алгоритм AES.

Используя указанную модель наследования классов, можно легко добавить новый алгоритм или новую реализацию существующего алгоритма.

### *Криптографические потоки*

Среда CLR использует поточно-ориентированный подход для реализации алгоритмов симметричного шифрования. Основой такого подхода является класс **CryptoStream**, производный от класса **System.IO.Stream**.

Благодаря использованию потоков, можно сцеплять вместе различные объекты (например, за объектом, реализующим хэширование, поставить объект, реализующий симметричное шифрование) и выполнять ряд операций над данными без использования промежуточных хранилищ данных.

Конструктор класса **CryptoStream** имеет следующий вид:

```
public CryptoStream(
    Stream stream,
    ICryptoTransform transform,
    CryptoStreamMode mode);
```

Класс **CryptoStream** может быть инициализирован с помощью любого класса, производного от класса **Stream**, включая **FileStream**, **MemoryStream** и **NetworkStream**. С помощью этих классов можно легко осуществлять симметричное шифрование различных потоковых объектов.

Кроме того, класс **CryptoStream** инициализируется с помощью класса, реализующего интерфейс **ICryptoTransform** (криптографический алгоритм), а также перечисления **CryptoStreamMode**, определяющего разрешенный тип доступа к **CryptoStream** (запись **Write** или чтение **Read**).

При передаче значения **CryptoStreamMode.Write** создаётся объект **CryptoStream**, который будет шифровать все записываемые в него данные. Передавая признак режима **CryptoStreamMode.Read**, мы создаём объект **CryptoStream**, расшифровывающий все записываемые в него данные.

❒ ***Пример* 8.1. *Разработка программы для шифрования и расшифрования текстовых файлов с помощью алгоритма DES.***

Требуется разработать программу, позволяющую шифровать и расшифровывать текстовые файлы с помощью симметричного алгоритма шифрования DES. В качестве режима шифрования будем использовать CBC, а в качестве типа заполнения – PKCS7. Ключ и вектор инициализации будут автоматически генерироваться при запуске программы.

В программе должна быть реализована функция сохранения секретного ключа и IV в текстовом файле, а также функция загрузки ключа и IV из файла.

Создадим решение, в котором будет присутствовать проект библиотеки классов.

Для выполнения шифрования и расшифрования с помощью DES, а также для задания параметров шифра разработаем вспомогательный класс **CipherDES**, который поместим в библиотеку классов.

Исходный код класса **CipherDES** приведён в листинге 8.1.

Листинг 8.1. Исходный код класса **CipherDES** (часть 1)

```
using System.IO;
// Импорт пространства имён, содержащего классы для криптографической защиты информации
using System.Security.Cryptography;

namespace SymmAlgorLib
{
    /// <summary>
    /// Представляет шифр DES
    /// </summary>
    public class CipherDES
    {
        DES des;    // Алгоритм DES
        byte[] key; // Ключ
        byte[] iv;  // Вектор инициализации
        CipherMode cMode;  // Режим шифрования
        PaddingMode pMode; // Тип заполнения блоков

        /// <summary>
        /// Конструктор по умолчанию
        /// </summary>
        public CipherDES()
        {
            des = new DESCryptoServiceProvider();
            // Генерирование ключа и вектора инициализации
            des.GenerateKey();
            des.GenerateIV();
            // Сохранение ключа и вектора инициализации
            key = des.Key;
            iv = des.IV;
            cMode = CipherMode.CBC;
            pMode = PaddingMode.PKCS7;
            des.Mode = cMode;
            des.Padding = pMode;
        }
```

Листинг 8.1. Исходный код класса **CipherDES** (часть 2)

```
42          /// <summary>
43          /// Возвращает ключ в виде строки
44          /// </summary>
45          public string Key
46          {
47              get
48              {
49                  // Строка для записи байтов в HEX-формате
50                  string hex = "";
51                  foreach (byte b in key)
52                  {
53                      hex += string.Format("{0:X2} ", b);
54                  }
55                  return hex;
56              }
57          }
58
59          /// <summary>
60          /// Возвращает вектор инициализации в виде строки
61          /// </summary>
62          public string IV
63          {
64              get
65              {
66                  // Строка для записи байтов в HEX-формате
67                  string hex = "";
68                  foreach (byte b in iv)
69                  {
70                      hex += string.Format("{0:X2} ", b);
71                  }
72                  return hex;
73              }
74          }
75
76          /// <summary>
77          /// Генерирует и сохраняет ключ
78          /// </summary>
79          public void GenerKey()
80          {
81              des.GenerateKey();
82              key = des.Key;
83          }
84
85          /// <summary>
86          /// Генерирует и сохраняет вектор инициализации
87          /// </summary>
88          public void GenerIV()
89          {
90              des.GenerateIV();
91              iv = des.IV;
92          }
```

Листинг 8.1. Исходный код класса **CipherDES** (часть 3)

```
94          /// <summary>
95          /// Шифровать с помощью алгоритма DES
96          /// </summary>
97          /// <param name="plainText">Открытый текст</param>
98          /// <returns>Шифротекст</returns>
99          public string Encrypt(string plainText)
100         {
101             // Получение массива байтов из открытого текста
102             byte[] plainBytes = Encoding.UTF8.GetBytes(plainText);
103
104             // Поток входных данных
105             MemoryStream msIn = new MemoryStream();
106             // Поток данных для шифрования
107             CryptoStream csIn = new CryptoStream(msIn, des.CreateEncryptor(key, iv),
108                 CryptoStreamMode.Write);
109             // Записать открытые данные в поток для шифрования
110             csIn.Write(plainBytes, 0, plainBytes.Length);
111             csIn.Close(); // Закрыть поток для ширования
112             // Получение строки шифротекста из массива байт
113             string cipherText = Convert.ToBase64String(msIn.ToArray());
114             msIn.Close(); // Закрыть поток входных данных
115
116             return cipherText;
117         }
118
119         /// <summary>
120         /// Расшифровать с помощью алгоритма DES
121         /// </summary>
122         /// <param name="cipherText">Шифротекст</param>
123         /// <returns>Открытый текст</returns>
124         public string Decrypt(string cipherText)
125         {
126             // Получение массива байтов из шифротекста
127             byte[] cipherBytes = Convert.FromBase64String(cipherText);
128
129             // Поток выходных данных
130             MemoryStream msOut = new MemoryStream(cipherBytes);
131             // Поток данных для расшифрования
132             CryptoStream csOut = new CryptoStream(msOut, des.CreateDecryptor(key, iv),
133                 CryptoStreamMode.Read);
134             // Считать поток для расшифрования
135             StreamReader sr = new StreamReader(csOut);
136             // Получение строки открытого текста
137             string plainText = sr.ReadToEnd();
138
139             return plainText;
140         }
```

Листинг 8.1. Исходный код класса **CipherDES** (часть 4)

```
142         /// <summary>
143         /// Сохраняет ключ в текстовом файле
144         /// </summary>
145         /// <param name="uri">URI файла</param>
146         public void SaveKey(string uri)
147         {
148             StreamWriter sw = new StreamWriter(uri);
149             sw.Write(Convert.ToBase64String(key));
150             sw.Close();
151         }
152 
153         /// <summary>
154         /// Сохраняет IV в текстовом файле
155         /// </summary>
156         /// <param name="uri">URI файла</param>
157         public void SaveIV(string uri)
158         {
159             StreamWriter sw = new StreamWriter(uri);
160             sw.Write(Convert.ToBase64String(iv));
161             sw.Close();
162         }
163 
164         /// <summary>
165         /// Загружает ключ из текстового файла
166         /// </summary>
167         /// <param name="uri">URI файла</param>
168         public void LoadKey(string uri)
169         {
170             StreamReader sr = new StreamReader(uri);
171             key = Convert.FromBase64String(sr.ReadToEnd());
172             sr.Close();
173         }
174 
175         /// <summary>
176         /// Загружает IV из текстового файла
177         /// </summary>
178         /// <param name="uri">URI файла</param>
179         public void LoadIV(string uri)
180         {
181             StreamReader sr = new StreamReader(uri);
182             iv = Convert.FromBase64String(sr.ReadToEnd());
183             sr.Close();
184         }
185     }
186 }
```

Добавим в решение проект приложения Windows Forms, содержащий одну экранную форму **Form1**.

Для создания интерфейса пользователя разместим на форме **Form1** ряд элементов управления:

- **textBoxKey** – текстовое поле, в котором будет записан ключ;
- **textBoxIV** – текстовое поле, содержащее IV;

• **textBoxText** – текстовое поле, в котором будет выводиться текст файла;

• **menuStrip1** – главное меню, содержащее пункты **Файл** (Открыть, Сохранить как, Выход) и **Операция** (Зашифровать, Расшифровать, Сгенерировать ключ и IV, Сохранить ключ и IV, Загрузить ключ и IV).

Для открытия и сохранения файлов в диалоговом режиме добавим на форму два элемента управления диалогом с пользователем: **openFileDialod1** и **saveFileDialog1**.

Общий вид интерфейса приложения, полученный в конструкторе форм, показан на рис. 8.5.

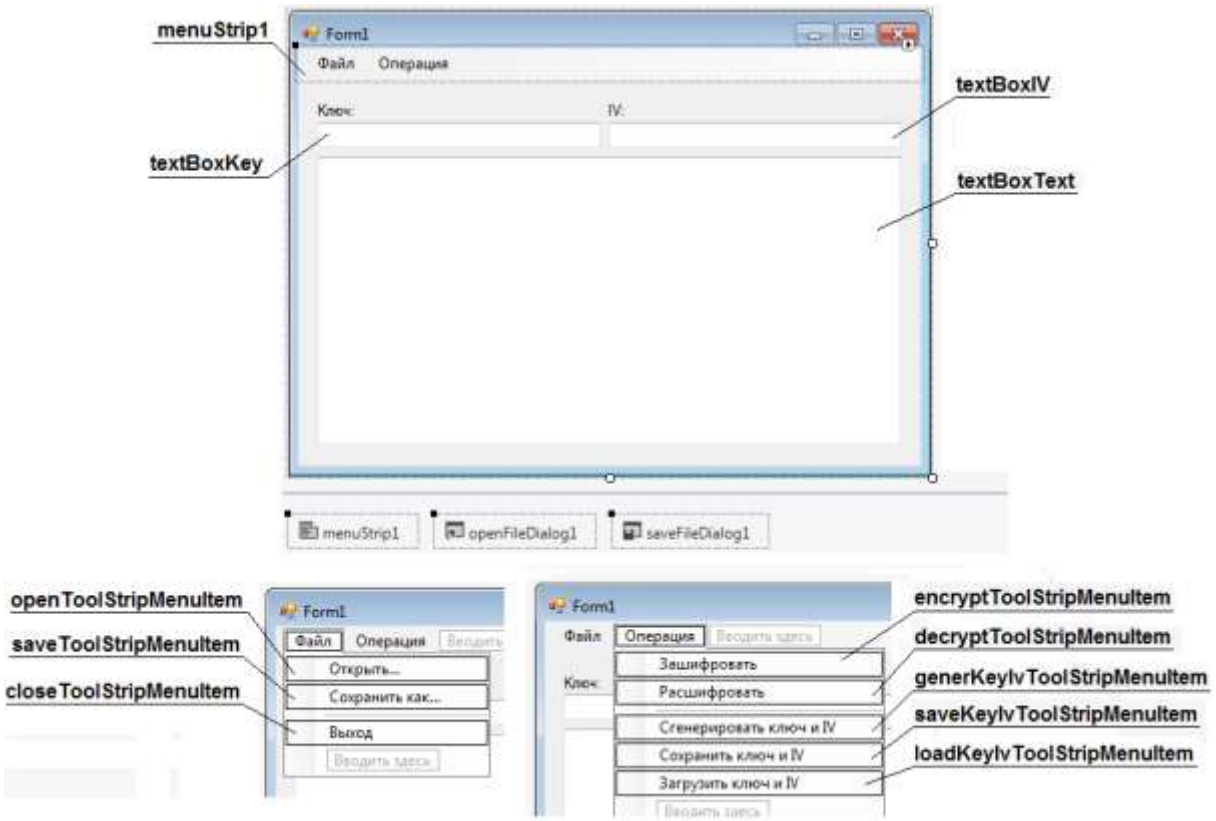


Рис. 8.5. Интерфейс пользователя приложения Windows Forms

Исходный код класса **Form1** приведён в листинге 8.2.

Листинг 8.2. Исходный код класса **Form1** (часть 1)

```
15    public partial class Form1 : Form
16    {
17        CipherDES des; // Шифр DES
18        string keyUri; // URI файла с ключом
19        string ivUri;  // URI файла с IV
20
21        public Form1()
22        {
23            InitializeComponent();
24            des = new CipherDES();
25            keyUri = @"H:\CryptoKey\key.txt";
26            ivUri = @"H:\CryptoKey\iv.txt";
27        }
28
29        private void Form1_Load(object sender, EventArgs e)
30        {
31            this.Text = "Работа с симметричным алгоритмом шифрования DES";
32            textBoxKey.Enabled = false;
33            textBoxKey.Text = des.Key;
34            textBoxIV.Enabled = false;
35            textBoxIV.Text = des.IV;
36            textBoxText.ScrollBars = ScrollBars.Vertical;
37            openFileDialog1.FileName = null;
38            openFileDialog1.Filter = "Текстовые файлы (*.txt)|*.txt|Все файлы (*.*)|*.*";
39            saveFileDialog1.Filter = "Текстовые файлы (*.txt)|*.txt|Все файлы (*.*)|*.*";
40        }
41
42        // Метод-обработчик события "Нажатие на пункт Открыть главного меню"
43        private void openToolStripMenuItem_Click(object sender, EventArgs e)
44        {
45            openFileDialog1.ShowDialog();
46            if (openFileDialog1.FileName == null) return;
47            try
48            {
49                StreamReader sr = new StreamReader(openFileDialog1.FileName,
50                    Encoding.GetEncoding(1251));
51                textBoxText.Text = sr.ReadToEnd();
52                sr.Close();
53            }
54            catch (Exception exc)
55            {
56                MessageBox.Show(exc.Message, "Ошибка",
57                    MessageBoxButtons.OK, MessageBoxIcon.Error);
58            }
59        }
```

Листинг 8.2. Исходный код класса **Form1** (часть 2)

```
42          // Метод-обработчик события "Нажатие на пункт Открыть главного меню"
43          private void openToolStripMenuItem_Click(object sender, EventArgs e)
44          {
45              openFileDialog1.ShowDialog();
46              if (openFileDialog1.FileName == null) return;
47              try
48              {
49                  StreamReader sr = new StreamReader(openFileDialog1.FileName,
50                      Encoding.GetEncoding(1251));
51                  textBoxText.Text = sr.ReadToEnd();
52                  sr.Close();
53              }
54              catch (Exception exc)
55              {
56                  MessageBox.Show(exc.Message, "Ошибка",
57                      MessageBoxButtons.OK, MessageBoxIcon.Error);
58              }
59          }
60
61          // Метод-обработчик события "нажатие на пункт Сохранить_как главного меню
62          private void saveToolStripMenuItem_Click(object sender, EventArgs e)
63          {
64              saveFileDialog1.FileName = openFileDialog1.FileName;
65              StreamWriter sw;
66              if (saveFileDialog1.ShowDialog() == DialogResult.OK)
67              {
68                  try
69                  {
70                      sw = new StreamWriter(saveFileDialog1.FileName, false,
71                          Encoding.GetEncoding(1251));
72                      sw.Write(textBoxText.Text);
73                      sw.Close();
74                  }
75                  catch (Exception exc)
76                  {
77                      MessageBox.Show(exc.Message, "Ошибка",
78                          MessageBoxButtons.OK, MessageBoxIcon.Error);
79                  }
80              }
81          }
82
83          // Метод-обработчик события "нажатие на пункт Выход главного меню"
84          private void closeToolStripMenuItem_Click(object sender, EventArgs e)
85          {
86              this.Close();
87          }
```

Листинг 8.2. Исходный код класса **Form1** (часть 3)

```
89          // Метод-обработчик события "Нажатие на пункт Зашифровать главного меню"
90          private void encryptToolStripMenuItem_Click(object sender, EventArgs e)
91          {
92              textBoxText.Text = des.Encrypt(textBoxText.Text);
93          }
94
95          // Метод-обработчик события "Нажатие на пункт Расшифровать главного меню"
96          private void decryptToolStripMenuItem_Click(object sender, EventArgs e)
97          {
98              try
99              {
100                 textBoxText.Text = des.Decrypt(textBoxText.Text);
101             }
102             catch (Exception exc)
103             {
104                 MessageBox.Show(exc.Message, "Ошибка",
105                     MessageBoxButtons.OK, MessageBoxIcon.Error);
106             }
107         }
108
109         // Метод-обработчик события "Нажатие на пункт Сгенер_ключ_и_IV главного меню"
110         private void generKeyIvToolStripMenuItem_Click(object sender, EventArgs e)
111         {
112             des.GenerKey();
113             des.GenerIV();
114             textBoxKey.Text = des.Key;
115             textBoxIV.Text = des.IV;
116         }
117
118         // Метод-обработчик события "Нажатие на пункт Сохранить_ключ_и_IV главного меню"
119         private void saveKeyIvToolStripMenuItem_Click(object sender, EventArgs e)
120         {
121             des.SaveKey(keyUri);
122             des.SaveIV(ivUri);
123         }
124
125         // Метод-обработчик события "Нажатие на пункт Загрузить_ключ_и_IV главного меню"
126         private void loadKeyIvToolStripMenuItem_Click(object sender, EventArgs e)
127         {
128             des.LoadKey(keyUri);
129             des.LoadIV(ivUri);
130             textBoxKey.Text = des.Key;
131             textBoxIV.Text = des.IV;
132         }
133     }
```

Работающее приложение Windows Forms при выполнении шифрования и расшифрования текстового файла показано на рис. 8.6. ❒

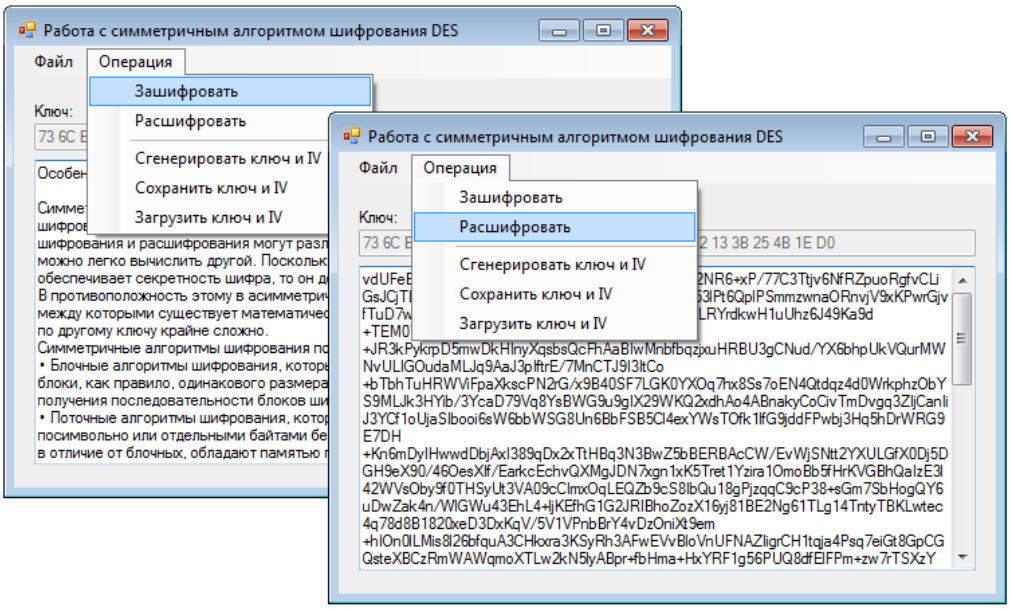


Рис. 8.6. Работа приложения Windows Forms

## 8.3. Порядок выполнения работы и варианты заданий

### *Основные этапы выполнения работы*

Данная лабораторная работа предполагает выполнение следующих этапов:

1. Изучить методические указания к лабораторной работе.
2. Разработать класс на языке C#, который содержит методы, обеспечивающие шифрование и расшифрование с помощью заданного симметричного алгоритма (табл. 8.1) при указанном режиме работы алгоритма и типе заполнения. Кроме того, класс должен обеспечивать операции для работы с ключом и вектором инициализации (IV): генерирование, сохранение и восстановление.
3. Разработать приложение Windows Forms, позволяющее шифровать и расшифровывать текстовые файлы с помощью разработанного класса. Приложение должно обеспечивать работу с текстовыми файлами в диалоговом режиме (открытие, редактирование и сохранение), а также их шифрование и расшифрование. Кроме того, в приложении требуется реализовать возможность генерирования ключа и IV, сохранения ключа и IV в текстовом файле, а также загрузку сохранённого ключа и IV из текстового файла.

4. Оформить и защитить отчет по лабораторной работе.

### *Индивидуальные варианты заданий*

Таблица 8.1

Варианты заданий для работы с симметричными алгоритмами шифрования

| № вар. | Алгоритм шифрования | Режим работы | Тип заполнения |
|---|---|---|---|
| **1, 13** | 3DES | CBC | PKCS7 |
| **2, 14** | Rijndael | ECB | ISO10126 |
| **3, 15** | AES | CFB | ANSIX923 |
| **4, 16** | RC2 | CBC | ISO10126 |
| **5, 17** | 3DES | ECB | ANSIX923 |
| **6, 18** | Rijndael | CFB | PKCS7 |
| **7, 19** | AES | CBC | ANSIX923 |
| **8, 20** | RC2 | ECB | PKCS7 |
| **9, 21** | 3DES | CFB | ISO10126 |
| **10, 22** | Rijndael | CBC | PKCS7 |
| **11, 23** | AES | ECB | ISO10126 |
| **12, 24** | RC2 | CFB | ANSIX923 |

### *Требования к отчёту.*

Отчёт по лабораторной работе должен содержать следующие пункты:

1. Титульный лист с указанием названия работы.
2. Цель и задачи работы.
3. Исходный код класса на языке C#, обеспечивающего шифрование и расшифрование с помощью заданного симметричного алгоритма.
4. Исходный код приложения Windows Forms, позволяющее шифровать и расшифровывать текстовые файлы.

## 8.4. Контрольные вопросы и задачи

### *Теоретические вопросы*

1. Какие шифры называют симметричными?
2. В чём заключается различие между блочными и поточными алгоритмами шифрования?

3. Что понимают под рассеиванием и перемешиванием в блочных шифрах?

4. Какие режимы работы поддерживаются блочными алгоритмами шифрования?

5. Каковы особенности работы блочного шифра в режиме электронной шифровальной книги (ECB)?

6. В чём заключаются особенности режима сцепления шифрованных блоков (CBC)?

7. Как работает блочный алгоритм в режиме шифрованной обратной связи?

8. Что понимают под вектором инициализации (IV)?

9. В каком пространстве имен библиотеки .NET Framework содержатся классы для криптографических преобразований?

10. Какие симметричные алгоритмы поддерживаются библиотекой классов .NET Framework?

11. Как осуществляется генерирование и установка значения ключа в классе **SymmetricAlgorithm**?

12. Каким образом задаётся режим шифрования и тип заполнения блока?

13. Для чего используется класс **CryptoStream**?

14. Какие параметры требуется передать в конструктор класса **CryptoStream**?

15. Как осуществляется выбор между операциями шифрования и расшифрования в **CryptoStream**?

***Практические задачи.***

1. Требуется разработать консольное приложение, которое выполняет шифрование введённой пользователем строки с помощью заданного алгоритма шифрования (табл. 8.2). Ключ и вектор инициализации генерируются случайным образом.

Таблица 8.2

Варианты заданий

| **Вариант** | **Алгоритм шифрования** | **Режим работы** | **Тип заполнения** |
|---|---|---|---|
| **Нечётный** | RC2 | CFB | PKCS7 |
| **Чётный** | 3DES | CBC | ISO10126 |

## РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

***Печатные издания.***

1. Аграновский А. В. Практическая криптография: алгоритмы и их программирование / А. В. Аграновский, Р. А. Хади. – М.: СОЛОН-Пресс, 2009. – 256 с.

2. Основы криптографии: учеб. пособие / А. П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А. В. Черемушкин. – 2-е изд., испр. и доп. – М.: Гелиос АРВ, 2002. – 480 с.

3. Васильев, К. К. Математическое моделирование систем связи: учебное пособие / К. К. Васильев, М. Н. Служивый. – Ульяновск: УлГТУ, 2008. – 170 с.

4. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, Ратушняк А., М. Смирнов, В. Юкин. – М.: ДИАЛОГ–МИФИ, 2003. – 284 с.

5. Вернер, М. Основы кодирования: учебник для ВУЗов. – М.: Техносфера, 2006. – 288 с.

6. Дмитриев, В. И. Прикладная теория информации. – М.: Высш. шк., 1989. – 332 с.

7. Духин, А. А. Теория информации. – М.: Гелиос АРВ, 2007. – 248 с.

8. Кудряшов, Б. Д. Теория информации. – СПб.: Питер, 2009. – 188 с.

9. Лидовский, В. В. Теория информации: учеб. пособие. – М.: Компания Спутник +, 2004. – 111 с.

10. Морелос-Сарагоса, Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение: пер. с англ. – М.: Техносфера, 2005. – 320 с.

11. Орлов, В. А. Теория информации в упражнениях и задачах: учеб. пособие для втузов / В. А. Орлов, Л. И. Филиппов. – М.: Высш. школа, 1976. – 136 с.
http://opac.mpei.ru/notices/index/IdNotice:91046/Source:default

12. Осипян, В. О. Криптография в задачах и упражнениях / В. О. Осипян, К. В. Осипян. – М.: Гелиос АРВ, 2004. – 144 с.

13. Панин, В. В. Основы теории информации: учеб. пособие для студентов вузов. – 3-е изд., испр. – М.: БИНОМ. Лаборатория знаний, 2009.

14. Першин, В. Т. Основы современной радиоэлектроники: учебное пособие / В. Т. Першин. – Ростов н/Д: Феникс, 2009. – 541 с.

15. Сэломон, Д. Сжатие данных, изображения и звука. – М.: Техносфера, 2004. – 368 с.

16. Торстейнсон, П. Криптография и безопасность в технологии .NET / П. Торстейнсон, Г. А. Ганеш; пер. с англ. – М.: БИНОМ. Лаборатория знаний, 2013. – 497 с.

17. Хэмминг, Р. В. Теория кодирования и теория информации: Пер. с англ. – М.: Радио и связь, 1983. – 176 с.

18. Цымбал, В. П. Задачник по теории информации и кодирования. – Изд. 2-е – М: ЛЕНАНД, 2014. – 280 с.

19. Цымбал, В. П. Теория информации и кодирование: учебник. – 4-е изд., перераб. и доп. – Киев: Вища школа, 1992. – 263 с.

***Интернет-ресурсы.***

20. www.intuit.ru/department/calculate/infotheory/ – учебный курс «Основы теории информации и криптографии», автор: В. В. Лидовский

21. Статья «Теория информации» в энциклопедии «Кругосвет»

22. www.compression.ru – сайт, посвященный технологиям сжатия данных

23. http://kunegin.narod.ru/ref3/code/index.htm – реферат по проблеме кодирования сообщений с исправлением ошибок

24. Статья «Обнаружение и исправление ошибок» в Википедии

25. Статья «Криптография» в Википедии

# ПРИЛОЖЕНИЕ

## П.1. Функции плотности распределения случайных величин

Таблица П.1

Функции плотности распределения $f(x)$ и соответствующие им обратные функции $F^{-1}(r)$

| № | Законы распределения и функции плотности распределения | Обратная функция |
|---|---|---|
| 1 | Равномерный $f(x)=\begin{cases}0, & (x<a)\cup(x>b)\\ 1/(b-a), & a<x<b\end{cases}$ | $x=a+r(b-a)$ |
| 2 | Экспоненциальный $f(x)=\begin{cases}0, & x<0\\ \lambda\exp(-\lambda x), & x\ge 0\end{cases}$ | $x=-\frac{1}{\lambda}\ln r$ |
| 3 | Нормальный $f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-M(X))^2}{2\sigma^2}\right)$ | $x=\sigma\sqrt{-2\ln r_1}\sin(2\pi r_2)+M$ |
| 4 | Логарифмически-нормальный $f(x)=\frac{1}{\sigma x\sqrt{2\pi}}\exp\left(-\frac{(\ln x-a)^2}{2c^c}\right)$ | $x=\exp(ct+b)$, $t=\sum_{i=1}^{12} r_i-6$ |
| 5 | Вейбулла $f(x)=\frac{cx^{c-1}}{b^c}\exp\left(-\left(\frac{x}{b}\right)^2\right)$ | $x=-b(\ln r)^{1/c}$ |
| 6 | Лапласа $f(x)=a\exp(-\lambda\lvert x\rvert),\ \lambda>0$ | $x=-\frac{1}{\lambda}\ln\left(\frac{\lambda}{a}\left(\frac{2a}{\lambda}-r\right)\right)$ |
| 7 | Коши $f(x)=\frac{a}{1+x^2}$ | $x=tg\left(\frac{r}{a}-\frac{\pi}{2}\right)$ |

| № | Законы распределения и функции плотности распределения | Обратная функция |
|---|---|---|
| 8 | Симпсона $f(x)=\frac{1}{a}\left(1-\frac{\lvert x\rvert}{a}\right),\ 0<x<a$ | $x=a+a\sqrt{4-2r}$ |
| 9 | Прямоугольного треугольника $f(x)=\frac{2}{a}\left(1-\frac{x}{a}\right),\ 0<x<a$ | $x=\frac{2/a+\sqrt{4/a^2-4r/a^2}}{2/a^2}$ |
| 10 | Параболы $f(x)=ax^2,\ 0<x<a$ | $x=\sqrt[3]{\frac{3r}{a}}$ |

В таблице П.1 параметры $r$, $r_1$, $r_2$ – случайные переменные с равномерным законом распределения.

## П.2. Неприводимые полиномы над полем GF(2)

Таблица П.2.

Неприводимые полиномы над полем *GF*(2)

| Степень | Полиномиальная форма | Векторная форма | *d* | *r* | *t* |
|---|---|---|---|---|---|
| **1** | $x+1$ | 11 | 2 | 1 | 0 |
| **2** | $x^2+x+1$ | 111 | 3 | 2 | 1 |
| **3** | $x^3+x+1$ | 1011 | 3 | 2 | 1 |
| | $x^3+x^2+1$ | 1101 | 3 | 2 | 1 |
| **4** | $x^4+x+1$ | 10011 | 3 | 2 | 1 |
| | $x^4+x^2+1$ | 10101 | 3 | 2 | 1 |
| **5** | $x^5+x^2+1$ | 100101 | 3 | 2 | 1 |
| | $x^5+x^3+1$ | 101001 | 3 | 2 | 1 |
| | $x^5+x^3+x^2+x+1$ | 101111 | 5 | 3 | 1 |
| | $x^5+x^4+x^2+x+1$ | 110111 | 5 | 3 | 1 |
| | $x^5+x^4+x^3+x+1$ | 111011 | 5 | 3 | 1 |
| | $x^5+x^4+x^3+x^2+1$ | 111101 | 5 | 3 | 1 |
| **6** | $x^6+x+1$ | 1000011 | 3 | 2 | 1 |
| | $x^6+x^3+1$ | 1001001 | 3 | 2 | 1 |
| | $x^6+x^5+1$ | 1100001 | 3 | 2 | 1 |
| | $x^6+x^4+x^2+x+1$ | 1010111 | 5 | 3 | 1 |
| | $x^6+x^4+x^3+x+1$ | 1011011 | 5 | 3 | 1 |
| | $x^6+x^5+x^2+x+1$ | 1100111 | 5 | 3 | 1 |

| Степень | Полиномиальная форма | Векторная форма | *d* | *r* | *t* |
|---|---|---|---|---|---|
| | $x^6 + x^5 + x^3 + x^2 + 1$ | 1101101 | 5 | 3 | 1 |
| | $x^6 + x^5 + x^4 + x + 1$ | 1110011 | 5 | 3 | 1 |
| | $x^6 + x^5 + x^4 + x^2 + 1$ | 1110101 | 5 | 3 | 1 |
| **7** | $x^7 + x + 1$ | 10000011 | 3 | 2 | 1 |
| | $x^7 + x^3 + 1$ | 10001001 | 3 | 2 | 1 |
| | $x^7 + x^4 + 1$ | 10010001 | 3 | 2 | 1 |
| | $x^7 + x^6 + 1$ | 11000001 | 3 | 2 | 1 |
| | $x^7 + x^3 + x^2 + x + 1$ | 10001111 | 5 | 3 | 1 |
| | $x^7 + x^4 + x^3 + x^2 + 1$ | 10011101 | 5 | 3 | 1 |
| | $x^7 + x^5 + x^2 + x + 1$ | 10100111 | 5 | 3 | 1 |
| | $x^7 + x^5 + x^3 + x + 1$ | 10101011 | 5 | 3 | 1 |
| | $x^7 + x^5 + x^4 + x^3 + 1$ | 10111001 | 5 | 3 | 1 |
| | $x^7 + x^6 + x^3 + x + 1$ | 11001011 | 5 | 3 | 1 |
| | $x^7 + x^6 + x^4 + x + 1$ | 11010011 | 5 | 3 | 1 |
| | $x^7 + x^6 + x^4 + x^2 + 1$ | 11010101 | 5 | 3 | 1 |
| | $x^7 + x^6 + x^5 + x^2 + 1$ | 11100101 | 5 | 3 | 1 |
| | $x^7 + x^6 + x^5 + x^4 + 1$ | 11110001 | 5 | 3 | 1 |
| | $x^7 + x^6 + x^5 + x^3 + x^2 + 1$ | 11101101 | 6 | 3 | 1 |
| | $x^7 + x^5 + x^4 + x^3 + x^2 + x + 1$ | 10111111 | 7 | 3 | 1 |
| | $x^7 + x^6 + x^5 + x^4 + x^2 + x + 1$ | 11110111 | 7 | 3 | 1 |
| | $x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + 1$ | 11111101 | 7 | 3 | 1 |
| **8** | $x^8 + x^4 + x^3 + x + 1$ | 100011011 | 5 | 3 | 1 |
| | $x^8 + x^4 + x^3 + x^2 + 1$ | 100011101 | 5 | 3 | 1 |
| | $x^8 + x^5 + x^3 + x + 1$ | 100101011 | 5 | 3 | 1 |
| | $x^8 + x^5 + x^3 + x^2 + 1$ | 100101101 | 5 | 3 | 1 |
| | $x^8 + x^5 + x^4 + x^3 + 1$ | 100111001 | 5 | 3 | 1 |
| | $x^8 + x^6 + x^3 + x^2 + 1$ | 101001101 | 5 | 3 | 1 |
| | $x^8 + x^6 + x^5 + x + 1$ | 101100011 | 5 | 3 | 1 |
| | $x^8 + x^6 + x^5 + x^2 + 1$ | 101100101 | 5 | 3 | 1 |
| | $x^8 + x^6 + x^5 + x^3 + 1$ | 101101001 | 5 | 3 | 1 |
| | $x^8 + x^6 + x^5 + x^4 + 1$ | 101110001 | 5 | 3 | 1 |
| | $x^8 + x^7 + x^2 + x + 1$ | 110000111 | 5 | 3 | 1 |
| | $x^8 + x^7 + x^3 + x + 1$ | 110001011 | 5 | 3 | 1 |
| | $x^8 + x^7 + x^3 + x^2 + 1$ | 110001101 | 5 | 3 | 1 |
| | $x^8 + x^7 + x^2 + x + 1$ | 110000111 | 5 | 3 | 1 |
| | $x^8 + x^6 + x^5 + x^2 + 1$ | 101100101 | 5 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x + 1$ | 111000011 | 5 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^3 + x^2 + 1$ | 111001101 | 6 | 3 | 1 |
| | $x^8 + x^5 + x^4 + x^3 + x^2 + x + 1$ | 100111111 | 7 | 3 | 1 |
| | $x^8 + x^6 + x^4 + x^3 + x^2 + x + 1$ | 101011111 | 7 | 3 | 1 |
| | $x^8 + x^6 + x^5 + x^4 + x^2 + x + 1$ | 101110111 | 7 | 3 | 1 |

| Степень | Полиномиальная форма | Векторная форма | *d* | *r* | *t* |
|---|---|---|---|---|---|
| | $x^8 + x^6 + x^5 + x^4 + x^3 + x + 1$ | 101111011 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^4 + x^3 + x^2 + x + 1$ | 110011111 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^5 + x^4 + x^3 + x^2 + 1$ | 110111101 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^4 + x^2 + x + 1$ | 111010111 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^4 + x^3 + x^2 + 1$ | 111011101 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^5 + x^2 + x + 1$ | 111100111 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^5 + x^4 + x + 1$ | 111110011 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^5 + x^4 + x^2 + 1$ | 111110101 | 7 | 3 | 1 |
| | $x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + 1$ | 111111001 | 7 | 3 | 1 |