Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

**Отчет по лабораторным работам**

Студент: Палазник А. В.

ФИТ 3 курс 1 группа

Преподаватель: Урбанович П. П.

Ассистент: Сазонова Д. В.

Минск 2024

# **Лабораторная работа №2**

**Тема «Элементы теории информации. Параметры и характеристики дискретных информационных систем»**

**Цель:** приобретение практических навыков расчета и анализа параметров и информативных характеристик дискретных ИС.

**Теоретические сведения**

Передача информации (данных) осуществляется между двумя абонентами, называемыми **источником сообщения (ИcС)** и **получателем сообщения (ПС**). Третьим элементом информационной системы является **канал (среда) передачи**, связывающий ИсС и ПС.

Отметим также, что и в системах с хранением информации всегда можно выделить ИcС и ПС. В данном случае каналом передачи здесь выступает устройство хранения информации (память). Например, при записи данных в ОЗУ (оперативное запоминающее устройство) компьютера в качестве ИcС и ПС может выступать процессор (соответственно при записи и чтении данных).

Таким образом, простейшая **информационная система состоит** из трех элементов: источника сообщения, канала передачи сообщения и получателя сообщения.

Отображение сообщения обеспечивается изменением какой-либо физической величины, характеризующей процесс (например, амплитуда, частота, фаза). Эта величина является **информационным параметром** сигнала (в общем случае – информационной системы).

Сигналы, как и сообщения, могут быть **непрерывными** и **дискретными**. Информационный параметр непрерывного сигнала с течением времени может принимать любые мгновенные значения в определенных пределах. Непрерывный сигнал часто называют **аналоговым**, а каналы и устройства, функционирующие на основе такого типа сигналов, – **аналоговыми**.

**Дискретный сигнал** (устройство или канал передачи) характеризуется конечным числом значений информационного параметра.

**Дискретные сообщения** состоят из последовательности дискретных знаков. Часто этот параметр принимает всего два значения (0 или 1). Сообщение или канал его передачи на основе этих двух значений сигнала называют **двоичным** или **бинарным**.

Построение сигнала по определенным правилам, обеспечивающим соответствие между сообщением и сигналом, называют **кодированием**.

Кодирование в широком смысле – *преобразование сообщения в сигнал.*

Кодирование в узком смысле – *представление исходных знаков*, называемых символами, *в другом алфавите с меньшим числом знаков*. Оно осуществляется с целью повышения надежности и преобразования сигналов к виду, удобному для передачи по каналам связи. Последний тип кодирования относится к так называемой **прикладной теории кодирования информации**, занимающейся поиском и реализацией методов и средств обнаружения несоответствий (**ошибок**) между переданным Xk и принятым Yk сообщениями.

Рассмотрим основные характеристики и параметры двоичных систем.

Важнейшая характеристика источника, получателя или канала – алфавит.

**Алфавит, А** – это общее число знаков или символов (N), используемых для генерации или передачи сообщений. Символы алфавита будем обозначать через {аi}, где 1 ≤ i ≤ N; N – мощность алфавита.

Минимальное число элементов алфавита Nmin = 2, А = {0, 1} – двоичный код. Один дискретный знак представляет собой **элементарное сообщение**, последовательность знаков – сообщение. Набор элементов алфавита, создаваемых дискретным источником сообщений, заранее, априори (до опыта) известен получателю ИсС в каждый дискретный момент времени выдает один элемент алфавита. Этот элемент сообщения является одним из символов алфавита. Понятно, что ПС заранее не известно, какой это элемент. Если обозначить вероятность выбора каждого элемента алфавита p(аi), то ∑ = = N i p ai 1 ( ) 1.

Вероятности p(аi) могут быть получены в результате анализа частотных свойств символов алфавита, если на входе такого анализатора принять документ на основе соответствующего алфавита. Причем объем документа должен быть таким, чтобы от частости (частоты) появления каждого символа в анализируемом документе можно было перейти к вероятности соответствующего события. Можно предположить, что указанному требованию будет соответствовать объем электронного документа не менее нескольких десятков килобайт.

Двоичный канал передачи информации строится на основе двоичного алфавита: А = {0, 1}. При этом канал, в котором вероятности искажения переданного 0 (принята соответственно 1; этому событию соответствует условная вероятность р(1|0)) и переданной 1 (принят соответственно 0; этому событию соответствует условная вероятность р(0|1)) равны, как и равны вероятности передачи 0 (р(0)) и 1 (р(1)), называют двоичным симметричным каналом (ДСК).

**С физической точки зрения энтропия алфавита показывает, какое количество информации приходится в среднем на один символ алфавита.**

**Практическое задание**

Рассчитать энтропию алфавитов: латиница, кириллица, бинарный. В качестве входного параметра произвольный электронный текстовый документ на основе соответствующего алфавита. Подсчитать количество информации в сообщении, состоящем из собственных фамилии, имени и отчества (на основе исходного алфавита – (а) и в кодах ASCII – (б)). Выполнить задание пункта (в) при условии, что вероятность ошибочной передачи единичного бита сообщения составляет: 0,1; 0,5; 1,0.

|  |
| --- |
| using System;  using System.IO;  using System.Collections.Generic;  namespace lab2  {  class Program  {  static void Main(string[] args)  {  // Задание 1  Console.WriteLine("Задание 1");  Console.WriteLine("Энтропия алфавитов");  Console.WriteLine("Датский: " + AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Danish));  Console.WriteLine("Подсчет вероятности появления символа: ");  AlphabetTask.CountSymbolFrequency(AlphabetTask.Alphabets.Danish);  Console.WriteLine("Казахский: " + AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Kazakh));  Console.WriteLine("Подсчет вероятности появления символа: ");  AlphabetTask.CountSymbolFrequency(AlphabetTask.Alphabets.Kazakh);  Console.WriteLine();  // Задание 2  Console.WriteLine("Задание 2");  Console.WriteLine("Бинарный: " + AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Binary));  Console.WriteLine();  // Задание 3  Console.WriteLine("Задание 3");  Console.WriteLine("К-во информации в ФИО (датский): " + (AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Danish) \* "ПалазникАрсенийВикторович".Length));  Console.WriteLine("К-во информации в ФИО (ASCII): " +  (AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Binary) \*  System.Text.Encoding.Unicode.GetBytes("ПалазникАрсенийВикторович").Length));  Console.WriteLine();  // Задание 4  Console.WriteLine("Задание 4");  Console.WriteLine("К-во информации в ФИО (ASCII, вероятность ошибки 0.1): " +  (AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Binary, 0.1f) \*  System.Text.Encoding.Unicode.GetBytes("Palaznik Arseniy Viktorovich").Length));  Console.WriteLine("К-во информации в ФИО (ASCII, вероятность ошибки 0.5): " +  (AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Binary, 0.5f) \*  System.Text.Encoding.Unicode.GetBytes("Palaznik Arseniy Viktorovich").Length));  Console.WriteLine("К-во информации в ФИО (ASCII, вероятность ошибки 1): " +  (Double.IsNaN(AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Binary, 1f)) ? 0 : AlphabetTask.CalculateEntropy(AlphabetTask.Alphabets.Binary, 1f) \*  System.Text.Encoding.Unicode.GetBytes("Palaznik Arseniy Viktorovich").Length));  }  }  } |

Листинг *1* – Код файла Program.cs

|  |
| --- |
| using System.IO;  using System.Collections.Generic;  using System;  using System.Linq;  using System.Text;  namespace lab2  {  public static class AlphabetTask  {  // Перечисление алфавитов  public enum Alphabets  {  Danish,  Kazakh,  Binary  }  // Метод для вычисления энтропии алфавита  public static double CalculateEntropy(Alphabets alphabet, float errorProbability = 0)  {  string symbols = "";  string filePath = "";  Dictionary<char, int> occurrences = new Dictionary<char, int>();  switch (alphabet)  {  case Alphabets.Danish:  symbols = "qwertyuiopasdfghjklzxcvbnmæøå";  filePath = "datch.txt";  break;  case Alphabets.Kazakh:  symbols = "аәбвгдеёжзийклмнопрстуфхцчшщъыьэюяғқңөұҺ";  filePath = "kazakh.txt";  break;  case Alphabets.Binary:  symbols = "01";  filePath = "binary.bin";  break;  }  foreach (var ch in symbols)  occurrences[ch] = 0;  using (StreamReader reader = new StreamReader(filePath, Encoding.UTF8))  {  string content = reader.ReadToEnd().ToLower();  foreach (var ch in content.Where(ch => symbols.Contains(ch)))  {  occurrences[ch]++;  }  double entropy = 0;  foreach (var ch in symbols)  {  if (occurrences[ch] > 0)  {  double probability = (double)occurrences[ch] / content.Length \* (1 - errorProbability);  entropy += probability \* Math.Log2(probability);  }  }  return -entropy;  }  }  // Метод для подсчета частоты символов  public static void CountSymbolFrequency(Alphabets alphabet)  {  string filePath = "";  string symbols = "";  Dictionary<char, int> occurrences = new Dictionary<char, int>();  switch (alphabet)  {  case Alphabets.Danish:  filePath = "datch.txt";  symbols = "qwertyuiopasdfghjklzxcvbnmæøå";  break;  case Alphabets.Kazakh:  filePath = "kazakh.txt";  symbols = "аәбвгдеёжзийклмнопрстуфхцчшщъыьэюяғқңөұҺ";  break;  }  foreach (var ch in symbols)  occurrences[ch] = 0;  string content;  using (StreamReader reader = new StreamReader(filePath, Encoding.UTF8))  {  content = reader.ReadToEnd();  }  foreach (var ch in content.Where(ch => symbols.Contains(ch)))  {  occurrences[ch]++;  }  foreach (var kvp in occurrences)  {  char symbol = kvp.Key;  int count = kvp.Value;  double probability = (double)count / content.Length;  Console.WriteLine($"Символ: {symbol}, Количество повторений: {count}, Вероятность: {probability}");  }  }  // Метод для получения количества появлений символов в строке  public static Dictionary<char, int> GetCharacterOccurrences(string text)  {  var occurrences = new Dictionary<char, int>();  foreach (char ch in text)  {  if (occurrences.ContainsKey(ch))  {  occurrences[ch]++;  }  else  {  occurrences.Add(ch, 1);  }  }  return occurrences;  }  }  } |

Листинг *2* – Код файла AlphabetTask.cs

|  |
| --- |
| const text =    "Қыз жуынатын ббөлмеге кірді. Ол тіс пастасының қақпағын ашты. Ол тіс щеткасына тіс пастасын біраз тамызды. Ол суық суды ағызды. Ол үстіңгі тістерін тазалады, сосын тіс пастасының біраз бөлігін түкірді. Ол астыңғы тістерін тазалап, тағы да тіс пастасының біраз бөлігін түкірді. Ол тіс щеткасын жуды. Ол тіс щеткасын орнына қойды. Ол ыдысқа су құйып, аузын шайқады. Ол аузын шайып болған соң, жуынатын бөлмеден шықты. Жігіт шарап дүкеніне кірді. Ол лотерея билеттерін сататын бөлімге қарай барды. Ол СуперЛотерея формасынOO алды. Ол алты санды белгіледі. Ол оның және анасының туған күнінің нөмірі еді. Ол лотерея бөліміне жақындап қолындағы форманы берді. Сатушы форманы машинаға салды. Сатушы оған алты нөмір жазылған билетті шығарып берді. Ол сатушыға бір доллар берді. Сатушы: «Сәттілік тілеймін», - деді. Ол сатушыға алғыс білдіріп дүкеннен шықты. Ол өзінің билеті миллион доллар ұтатынына үміттенді. Жігіт іш киімін киді. Ол шалбарын киді. Ол белбеуді шалбардың ілмектерінен өткізіп алды. Ол қара шұлық киді. Сосын ол қоңыр туфли киді. Ол бауын байлады. Ол тұрды да, шкафқа барды. Ол гольф ойынына арналған қысқа жең көйлекті тауып алды. Ол ақшыл көк түсті. Жігіт оны басынан өткізіп киіп алды. Ол көйлекті шалбардың ішіне салды. Ол шалбардың замогын салып, белбеуін тақты. Ол айнаға қарады. Ол көйлектегі үш түймені тақты. Ол жағасының түзу тұрғанына көзі жетті. Ол қалтасына кілті мен әмиянын салды. Ол пәтерінен шықты.";  const alp = "аәбвгдеёжзийклмнопрстуфхцчшщъыьэюяғқңөұҺ";  for (let i = 0; i < alp.length; i++) {    const count = text.split(alp[i]).length - 1;    console.log(`${alp[i]}: ${count}`);  } |

Листинг *3* – Код файла src.js

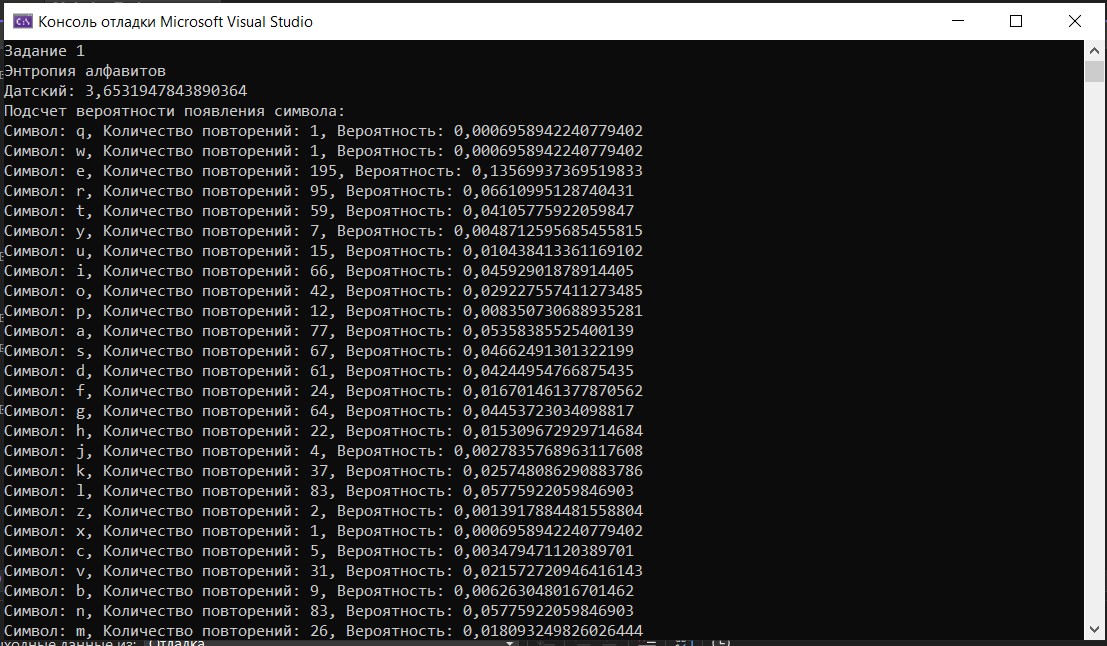
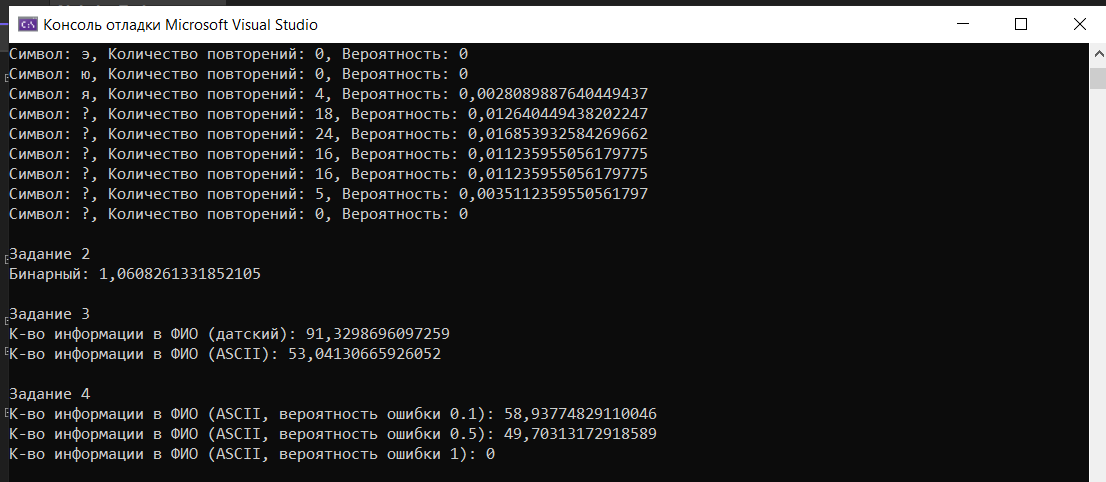
 

Рис. *1*.*1* – Результат выполнения программы

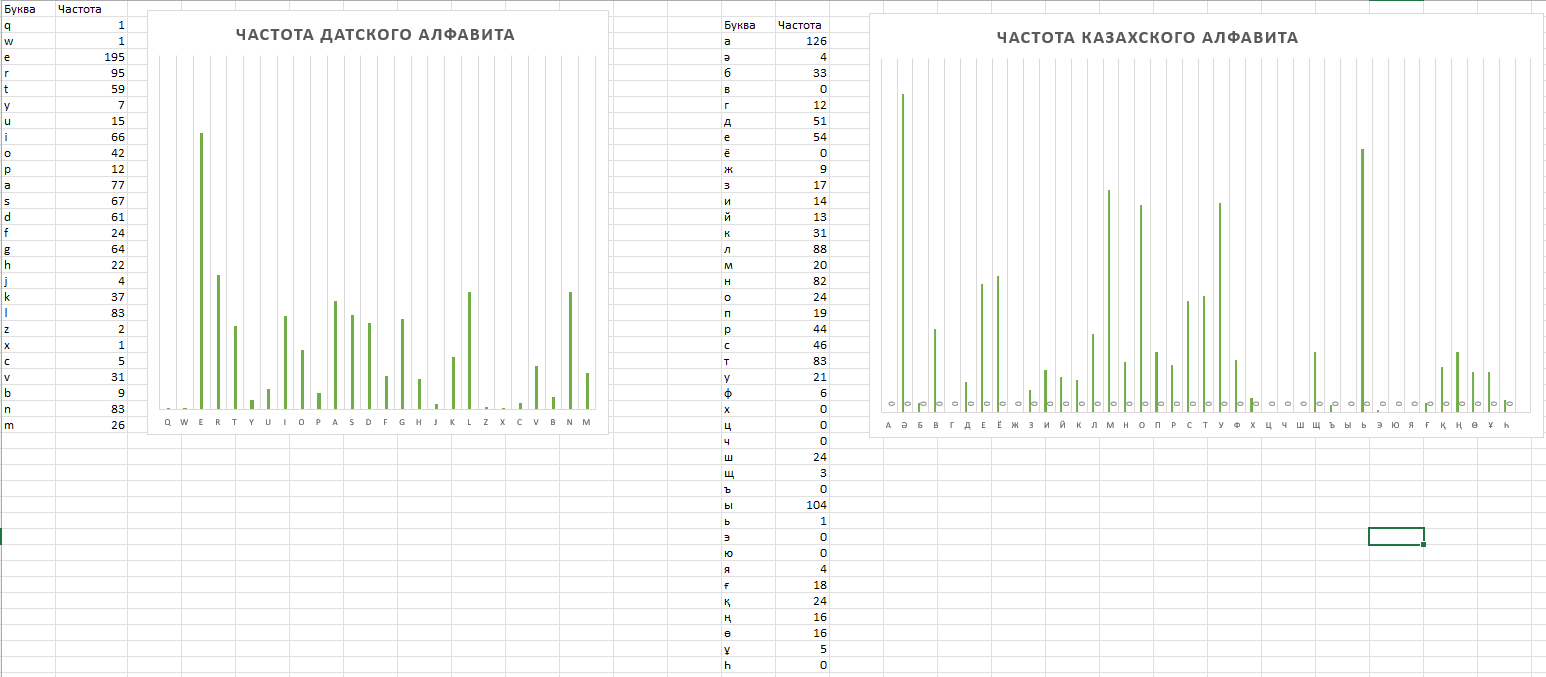


Рис. *1*.*2* – Частоты датского и казахского алфавита

**Ответы на вопросы:**

1. **Что такое алфавит источника сообщения?**

**Алфавит** – конечная совокупность символов (знаков), с помощью которых можно представить любое сообщение в ИС.

1. **Что такое мощность алфавита источника сообщения?**

**Мощность алфавита** – количество символов, составляющих алфавит

1. **Какова мощность алфавита белорусского языка?**

32

1. **Какова мощность алфавита русского языка?**

33

1. **Какова мощность алфавита «компьютерного» языка?**

256

1. **Что такое энтропия алфавита?**

Информационной характеристикой алфавита (источника сообщений на основе этого алфавита) является *энтропия*. С физической точки зрения энтропия показывает, какое количество информации (бит) приходится в среднем на один символ алфавита.

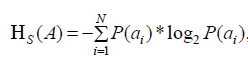
1. **От чего зависит энтропия алфавита?**

От частоты встречаемости символов и мощности алфавита.

1. **Что такое энтропия сообщения?**

Энтропия – информационная характеристика алфавита (источника сообщений на основе этого алфавита), которая показывает, какое количество информации приходится в среднем на один символ алфавита (сообщения).

1. **Записать формулу для вычисления энтропии (рисунок 1).**

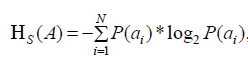


1. **Что нужно знать для вычисления энтропии алфавита?**

Частоту встречаемости каждого символа алфавита и мощность.

1. **Как рассчитываются энтропия Шеннона и энтропия Хартли? В чем принципиальное различие между этими характеристиками? Дайте толкование физического смысла энтропии.**

Энтропию алфавита А={*ai*} по К.Шеннону рассчитывают по следующей формуле, представленной на рисунке 2.



Частным случаем энтропии Шеннона является энтропия Хартли. Дополнительным условием при этом является то,что все вероятности одинаковы и постоянны для всех символов алфавита. С учетом этого формулу (2.1) можно преобразовать к виду, представленному на рисунке 3.



1. **Поясните назначение знака «минус» в формулах (2.1) и (2.4).**

Поскольку мы находим логарифмы от чисел меньших 1, то логарифм всегда будет отрицательным, поэтому мы добавляем минус перед логарифмом

1. **. Что такое избыточность алфавита и избыточность сообщений,** **сформированных в компьютерных системах? Принцип действия каких систем основан на существовании данной избыточности?**

Избыточностью алфавита называется уменьшение информационной нагрузки на один символ вследствие неравновероятности и взаимозависимости появления его символов.

Информационная избыточность характеризует относительную нагруженность алфавита.

1. **Расположите в порядке возрастания энтропии известные вам** **алфавиты.**

3,2 – белорусского

3,895 – французского

4,25 – молдавского

1. **Вычислить энтропию алфавита белорусского (русского) языка.**

3.2 бит

1. **Вычислить энтропию Шеннона бинарного алфавита, если вероятность появления в произвольном документе на основе этого алфавита одного из символов составляет 0.25, другого – 0.75; либо 0 и 1.0; либо 0.5 и 0.5.**
2. 0,811 бит
3. 0 бит
4. 1 бит
5. **Чему равна энтропия алфавита по Хартли, если мощность этого алфавита равна: а) 1 символ, б) 2 символа, в) 8 символов?**
6. 0 бит
7. 1 бит
8. 3 бит

**Вывод:** приобрел практические навыки расчета и анализа параметров и информативных характеристик дискретных ИС.

# **Лабораторная работа №3**

**Тема «Элементы теории информации. Информативность данных в различных кодировках»**

**Цель:** приобретение практических навыков трансформации данных и сопоставление энтропийных свойств используемых при этом алфавитов.

**Теоретические сведения**

Из энтропийных оценок (алфавитов и сообщений), полученных в ходе выполнения лабораторной работы № 2, мы выяснили, что энтропия зависит от статических характеристик самих алфавитов и сообщений (вспомним энтропию по Шеннону и по Хартли).

Энтропия максимальна при равномерном появлении букв на любом месте сообщения. Для характеристики источника сообщений с различным алфавитом представляет интерес сравнение фактической энтропии источника с максимально возможной. В этом смысле введено понятие избыточности источника сообщений, или избыточности алфавита.

**Избыточностью алфавита** называют уменьшение информационной нагрузки на один символ вследствие разной вероятности и взаимозависимости появления его символов в сообщениях.

В наиболее общем виде избыточность алфавита R можно оценить отношением энтропии по Хартли и по Шеннону; при этом первая рассчитывается по выражению (2.2), вторая – по формуле (2.1):



При выполнении предыдущей работы мы убедились, что формально одно и то же сообщение, но представленное на основе алфавита русского (белорусского, английского или иного) языка – с одной стороны, и представленное в кодах ASCII – с другой, будут характеризоваться различным количеством содержащейся в них информации. Эта дополнительная избыточность обусловлена переносом сообщения из одной среды в другую или, иначе говоря, кодированием символов исходного алфавита.

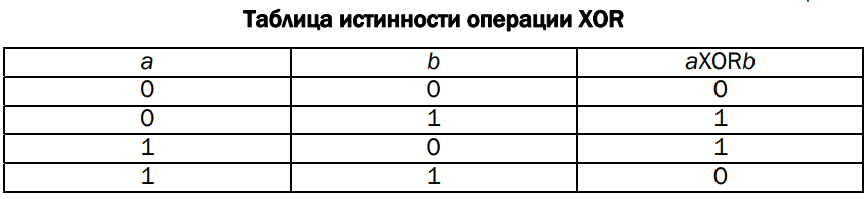
Утверждение восьмибитных кодировок (ASCII) как стандарта принесло некоторые проблемы. К этому моменту уже существовала определенная инфраструктура, использующая семибитные кодировки. Известны проблемы с «обрезанием восьмого бита» в системе электронной почты. Утверждение восьмибитного символа дало 256 различных значений, что позволило уместить в одной кодовой таблице и общепринятые символы (цифры, знаки препинания, латиницу), и символы кириллицы.

Уже созданное к тому времени и работающее программное обеспечение зачастую было приспособлено для семибитных кодировок, что приводило, например, к тому, что почтовый сервер при передаче письма обнулял старшие биты в каждом байте сообщения.

Одним из решений проблемы стала кодировка (а точнее – алгоритм) base64. В PGP алгоритм base64 используется для кодирования бинарных данных. Кодирование base64 разработано для представления произвольных последовательностей октетов в форме, позволяющей использовать строчные и прописные буквы. Используется 65-символьное подмножество набора символов US-ASCII, обеспечивающее представление одним печатным символом 6 битов данных (дополнительный 65-й символ используется для обозначения функции специальной обработки).

Kаждые 6 битов буфера, начиная с самых старших, используются как индексы строки «ABCDEFGHIJKLMNOPQRSTUV WXYZabcdefghijklmnopqrstuvwxyz0123456789+/», и ее символы, на которые указывают индексы, помещаются в выходную строку.

**Если кодируются только один или два байта, в результате получаются только первые два или три символа строки, а выходная строка дополняется двумя или одним символами «=». Это предотвращает добавление дополнительных битов к восстановленным данным.**

****

Процесс повторяется над оставшимися входными данными. Такая обработка выполняется в тех случаях, когда последняя группа входных данных содержит меньше 24 битов. Кодируемое значение всегда завершается полным квантом кодирования.

Если на входе доступно менее 24 битов, входная группа дополняется (справа) нулями до формирования целого числа 6-битных групп. Заполнение в конце данных осуществляется как раз с использованием символа «=». Поскольку входная информация base64 всегда включает целое число октетов, возможны лишь перечисленные ниже случаи:

* размер финального блока кодирования на входе кратен 24 битам, кодированный результат будет содержать целое число 4-символьных групп без заполнения символами «=»;
* размер финального блока кодирования на входе составляет 8 битов, выходной блок будет представлять 2 символа, дополненные последовательностью из двух символов заполнения «==»;
* размер финального блока кодирования на входе составляет 16 битов, выходной блок будет представлять 3 символа, дополненные символом заполнения «=».

Если а и b имеют длину более 1 бита, к примеру 1 байт, то рассматриваемая операция над ними выполняется побитово. Указанным байтам могут соответствовать символы в определенной кодировке. Положим, символу «М» (hex4d) соответствует 8-битный код 01001101 (см. табл. 3.2), а символу «а» (hex61) соответствует код 01100001, тогда операция сложения по модулю 2 этих двух бинарных кодов дает 00101100 ((hex2с), или символ «,».

**Практическое задание**

Создать собственное приложение (приветствуется!) или воспользоваться Base64-онлайн-кодировщиком, с помощью которого конвертировать произвольный документ (а) на латинице (можно использовать документ из лабораторной работы № 1) в документ (б) формата base64.

С помощью приложения, созданного в лабораторной работе № 1, получить распределение частотных свойств алфавитов по документам (а) и (б). Вычислить энтропию Хартли и Шеннона, а также избыточность алфавитов. Объяснить полученный результат.

Написать функцию, которая принимает в качестве аргументов два буфера (а и b) одинакового размера и возвращает XOR (собственная фамилия (а) и имя (b); при разной длине меньшую дополнить нулями). Входные аргументы представлять: 1) в кодах ASCII; 2) в кодах base64. Что будет результатом операции аXORbXORb?

При написании не использовать стандартные функции языка программирования. Итоговые данные сравнить с результатами использования стандартных функций языка программирования (если они есть).

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace laba3  {  internal class MainClass  {  static void Main(string[] args)  {  // Создание документа в формате Base64  CurrentLab.GenerateBase64Document();  // Создание и вывод отчета  Console.WriteLine(CurrentLab.GenerateReport());  // ASCII XOR операции  Console.Write("ASCII XOR: ");  string lastName = "Palaznik", firstName = "Arseniy";  string lastNameAscii = "", firstNameAscii = "";  foreach (char ch in lastName)  {  lastNameAscii += Convert.ToInt32(ch);  }  foreach (char ch in firstName)  {  firstNameAscii += Convert.ToInt32(ch);  }  while (lastNameAscii.Length != firstNameAscii.Length)  {  firstNameAscii += '0';  }  foreach (byte b in CurrentLab.XOR(Encoding.Unicode.GetBytes(lastNameAscii), Encoding.Unicode.GetBytes(firstNameAscii)))  {  Console.Write(b);  }  Console.WriteLine();  // Base64 XOR операции  Console.Write("Base64 XOR: ");  string firstNameBase64 = Convert.ToBase64String(Encoding.Unicode.GetBytes(firstName));  string lastNameBase64 = Convert.ToBase64String(Encoding.Unicode.GetBytes(lastName));  while (lastNameBase64.Length != firstNameBase64.Length)  {  firstNameBase64 += '0';  }  foreach (byte b in CurrentLab.XOR(Encoding.Unicode.GetBytes(lastNameBase64), Encoding.Unicode.GetBytes(firstNameBase64)))  {  Console.Write(b);  }  Console.WriteLine();  // aXORbXORb операция  Console.Write("aXORbXORb: \t");  byte[] aXORbXORbResult = CurrentLab.XOR(Encoding.Unicode.GetBytes(lastNameBase64), CurrentLab.XOR(Encoding.Unicode.GetBytes(firstNameBase64), Encoding.Unicode.GetBytes(lastNameBase64)));  foreach (byte b in aXORbXORbResult)  {  Console.Write(b);  }  Console.WriteLine();  // Вывод имени в Base64  Console.Write("a: \t\t");  foreach (byte b in Encoding.Unicode.GetBytes(firstNameBase64))  {  Console.Write(b);  }  Console.WriteLine();  }  }  } |

Листинг *1* – Код файла Program.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  namespace laba3  {  public static class PreviousLab  {  public enum AlphabetList  {  Danish,  Kazakh,  Binary,  Base64  }  public static double CalculateEntropy(AlphabetList alphabet, float errorProbability = 0)  {  string alphabetSymbols = "";  string filePath = "";  switch (alphabet)  {  case AlphabetList.Danish:  alphabetSymbols = "qwertyuiopasdfghjklzxcvbnmæøå";  filePath = "datch.txt";  break;  case AlphabetList.Kazakh:  alphabetSymbols = "аәбвгдеёжзийклмнопрстуфхцчшщъыьэюяғқңөұҺ";  filePath = "kazakh.txt";  break;  case AlphabetList.Binary:  alphabetSymbols = "01";  filePath = "binary.bin";  break;  case AlphabetList.Base64:  alphabetSymbols = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";  filePath = "base64.txt";  break;  }  Dictionary<char, int> symbolCounts = new Dictionary<char, int>();  foreach (char ch in alphabetSymbols)  {  symbolCounts[ch] = 0;  }  using (StreamReader reader = new StreamReader(filePath))  {  string content = reader.ReadToEnd().ToLower();  foreach (char ch in content)  {  if (alphabetSymbols.Contains(ch))  {  symbolCounts[ch]++;  }  else if (alphabet != AlphabetList.Base64)  {  content = content.Remove(content.IndexOf(ch), 1);  }  }  double entropy = 0;  foreach (char ch in alphabetSymbols)  {  if (symbolCounts[ch] > 0)  {  double probability = (double)symbolCounts[ch] / content.Length \* (1 - errorProbability);  entropy += probability \* Math.Log(probability, 2);  }  }  return -entropy;  }  }  }  } |

Листинг *2* – Код файла PreviousLab.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  namespace laba3  {  public static class CurrentLab  {  public static string ConvertToBase64(string filePath)  {  using (StreamReader reader = new StreamReader(filePath))  {  string content = reader.ReadToEnd();  return Convert.ToBase64String(Encoding.Unicode.GetBytes(content));  }  }  public static void GenerateBase64Document()  {  using (FileStream fileStream = new FileStream("base64.txt", FileMode.OpenOrCreate))  {  string base64Content = ConvertToBase64("datch.txt");  byte[] bytes = Encoding.Unicode.GetBytes(base64Content);  fileStream.Write(bytes, 0, bytes.Length);  }  }  public static string GenerateReport()  {  double base64Entropy = PreviousLab.CalculateEntropy(PreviousLab.AlphabetList.Base64);  double latinEntropy = PreviousLab.CalculateEntropy(PreviousLab.AlphabetList.Danish);  double base64Capacity = Math.Log("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/".Length, 2);  double latinCapacity = Math.Log("qwertyuiopasdfghjklzxcvbnm".Length, 2);  StringBuilder reportBuilder = new StringBuilder();  reportBuilder.AppendLine($"Base64 энтропия (Шеннон): {base64Entropy}");  reportBuilder.AppendLine($"Latin энтропия (Шеннон): {latinEntropy}");  reportBuilder.AppendLine($"Base64 энтропия (Чартли): {base64Capacity}");  reportBuilder.AppendLine($"Latin энтропия (Чартли): {latinCapacity}");  reportBuilder.AppendLine($"Резервирование base64: {(base64Capacity - base64Entropy) / base64Capacity \* 100}%");  reportBuilder.AppendLine($"Резервирование latin: {(latinCapacity - latinEntropy) / latinCapacity \* 100}%");  return reportBuilder.ToString();  }  public static byte[] XOR(byte[] buffer1, byte[] buffer2)  {  if (buffer1.Length != buffer2.Length) return null;  byte[] result = new byte[buffer1.Length];  for (int i = 0; i < buffer1.Length; i++)  {  result[i] = (byte)(buffer1[i] ^ buffer2[i]);  }  return result;  }  public static byte[] CustomXOR(byte[] buffer1, byte[] buffer2)  {  if (buffer1.Length != buffer2.Length) return null;  byte[] result = new byte[buffer1.Length];  for (int i = 0; i < buffer1.Length; i++)  {  string buf1Binary = Convert.ToString(buffer1[i], 2).PadLeft(8, '0');  string buf2Binary = Convert.ToString(buffer2[i], 2).PadLeft(8, '0');  string xorResult = "";  for (int j = 0; j < buf1Binary.Length; j++)  {  xorResult += buf1Binary[j] == buf2Binary[j] ? '0' : '1';  }  result[i] = Convert.ToByte(xorResult, 2);  }  return result;  }  }  } |

Листинг *2* – Код файла CurrentLab.cs

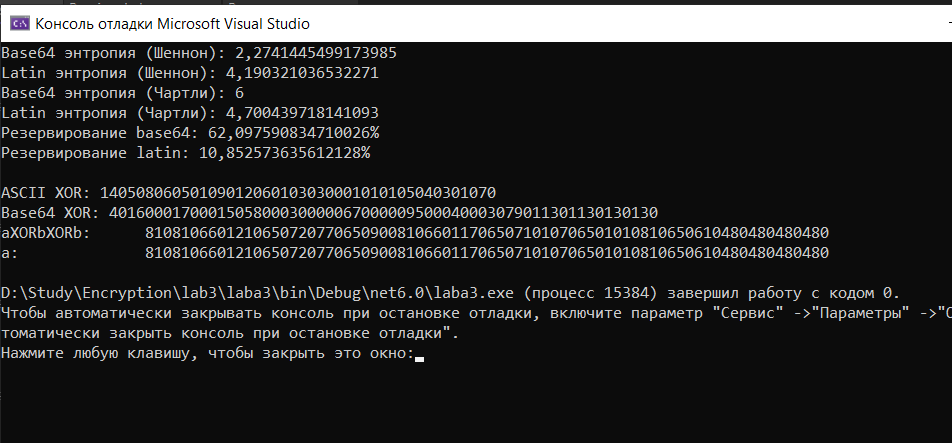


Рис. *3*.*1* – Результат выполнения программы

**Ответы на вопросы:**

1. **Что такое base64?**

Base64 буквально означает — позиционная система счисления с основанием 64. Здесь 64 — это наибольшая степень двойки (26), которая может быть представлена с использованием печатных символов ASCII.

**Base64** - это группа схожих binary-to-text encoding схем, которые представляют двоичные данные в ASCII-формате методом перевода в radix-64 представление.

Кодирование Base64 широко используется в случаях, когда требуется перекодировать двоичные данные для передачи по каналу приспособленному для передачи текстовых данных.

. Используется 65-символьное подмножество набора символов US-ASCII, обеспечивающее представление одним печатным символом 6 битов данных (дополнительный 65-й символ используется для обозначения функции специальной обработки).

**Избыточностью** алфавита называется уменьшение информационной нагрузки на один символ вследствие неравновероятности и взаимозависимости появления его символов.

Информационная избыточность характеризует относительную нагруженность алфавита

1. **Как проверить, была ли определенная строка символов закодирована в base64?**

 Мы можем использовать следующее регулярное выражение, чтобы проверить, закодирована ли строка в base64 или нет:

**^([A-Za-z0-9+/]{4})\*([A-Za-z0-9+/]{3}=|[A-Za-z0-9+/]{2}==)?$**

В кодировке base64 набор символов - [A-Z, a-z, 0-9 и + /]. Если остальная длина меньше 4, строка заполняется символами '='.

^([A-Za-z0-9+/]{4})\* означает, что строка начинается с 0 или более групп base64.

([A-Za-z0-9+/]{4}|[A-Za-z0-9+/]{3}=|[A-Za-z0-9+/]{2}==)$ означает, что строка заканчивается одной из трех форм: [A-Za-z0-9+/]{4}, [A-Za-z0-9+/]{3}= или [A-Za-z0-9+/]{2}==.

1. **Как с помощью base64 проверить подлинность вводимых данных в форму пароля и логина?**

С помощью валидатора.

1. **Охарактеризовать энтропийные свойства алфавитов в проанализированных форматах данных.**

*Энтропия алфавита - это количество информации, приходящееся на один символ.* Другими словами, это информационная нагрузка, которую несет один символ алфавита.

1. **Объяснить результат операции аXORbXORb. Где может найти применение такая операция?**

Эта операция называется также cложением по модулю 2, логическим сложением, исключающим «ИЛИ», строгой дизъюнкций, поразрядным дополнением.

1. **Как будут выглядеть строки: efd8b295a633908a3c0828b2 faea8766 4d72cde3aaa0 после их конвертации в base64?**

ZWZkOGIyOTVhNjMzOTA4YTNjMDgyOGIyIGZhZWE4NzY2IDRkNzJjZGUzYWFhMA==

**Выводы:** приобрел практические навыки трансформации данных и сопоставление энтропийных свойств используемых при этом алфавитов.

# **Лабораторная работа №4**

**Тема «Избыточное кодирование данных в информационных системах. Код Хемминга»**

**Цель:** приобретение практических навыков кодирования/декодирования двоичных данных при использовании кода Хемминга.

**Теоретические сведения**

**Надежность системы** – характеристика способности программного, аппаратного, аппаратно-программного средства выполнить при определенных условиях требуемые функции в течение конкретного периода времени.

**Достоверность работы системы (устройства**) – свойство, характеризующее истинность конечного (выходного) результата работы (выполнения программы), определяемое способностью средств контроля фиксировать правильность или ошибочность работы.

**Избыточное кодирование —** вид кодирования, использующий избыточное количество информации с целью последующего контроля целостности данных при записи/воспроизведении информации или при её передаче по линиям связи.

**Код Хемминга** - это алгоритм, который позволяет закодировать какое-либо информационное сообщение определённым образом и после передачи (например по сети) определить появилась ли какая-то ошибка в этом сообщении (к примеру из-за помех) и, при возможности, восстановить это сообщение.

**Ошибка устройства** – неправильное значение сигнала (бита – в цифровом устройстве) на внешних выходах устройства или отдельного его узла, вызванное технической неисправностью, или воздействующими на него помехами (преднамеренными либо непреднамеренными), или иным способом.

**Ошибка программы** – проявляется в не соответствующем реальному (требуемому) промежуточном или конечном значении(результате) вследствие неправильно запрограммированного алгоритма или неправильно составленной программы.

*Надежность является* комплексным свойством, включающим в себя единичные свойства: безотказность, ремонтопригодность, сохраняемость, долговечность.

**Безотказность** – это свойство технического объекта непрерывно сохранять работоспособное состояние в течение некоторого времени (или наработки). Наработка, как правило, измеряется в единицах времени.

**Ремонтопригодность** – это свойство технического объекта, заключающееся в приспособленности к поддержанию и восстановлению работоспособного состояния путем технического обслуживания, ремонта (или с помощью дополнительных, избыточных технических средств, функционирующих параллельно с объектом).

Код Хемминга относится к классу линейных блочных кодов.

**Линейные блочные коды** – это класс кодов с контролем четности, которые можно описать парой чисел (п, k).

Для формирования r проверочных символов (кодирования), т. е. вычисления проверочного слова Xr, используется *порождающая матрица G:* совокупность базисных векторов будем далее записывать в виде матрицы G размерностью k×n с единичной подматрицей (I) в первых k строках и столбцах:



Более *точно матрица G называется* порождающей матрицей линейного корректирующего кода в приведенно-ступенчатой форме. Кодовые слова являются линейными комбинациями строк матрицы G (кроме слова, состоящего из нулевых символов).

Кодирование заключается в умножении вектора сообщения Хk длиной k на порождающую матрицу по правилам матричного умножения (все операции выполняются по модулю 2). Очевидно, что при этом первые k символов кодового слова равны соответствующим символам сообщения, а последние r символов образуются как линейные комбинации первых [5, 9].

Для всякой порождающей матрицы G существует матрица Н размерности r×n, задающая базис нулевого пространства кода и удовлетворяющая равенству



Справедливо также



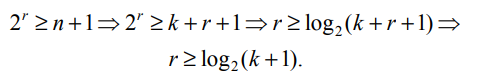
В последнем выражении символ «T» означает транспонирование, а X n = 1x , 2 x , …, xn. Матрица Н, называемая проверочной, равна



Количество r избыточных (проверочных) символов кодового слова определяется из следующей простой логической цепи рассуждений. Общее число всех возможных комбинаций 2r должно удовлетворять неравенству



в силу того, что



Присутствие цифры «1» в приведенных выражениях соотносит ее с нулевым вектор-столбцом, который в матрице не используется.

Результат умножения сообщения на выходе канала передачи (Yn) или (что равнозначно) сообщения, считываемого из памяти, на проверочную матрицу (Н) **называется синдромом (вектором ошибки) S:**



где Yn = y1, y2, …, yn – принятый вектор (сообщение на выходе канала), полученный после передачи либо считывания из памяти. Вектор Yn обычно представляют в следующем виде:



где Еn = е1, е2, …, еn – вектор ошибки.

**Синдром** – это результат проверки четности, выполняемой над сообщением Yn для определения его принадлежности заданному набору кодовых слов. При положительном результате проверки синдром S равен 0, т. е. Yn = Хn. Если Yn содержит ошибки, которые можно исправить, то синдром имеет определенное ненулевое значение, что позволяет обнаружить и исправить конкретную ошибочную комбинацию.

**Важно запомнить, что в силу выражений (4.3)–(4.7) ненулевой синдром всегда равен сумме по модулю 2 тех векторстолбцов матрицы Н, номера которых соответствуют номерам ошибочных битов в слове Yn.**

**При этом нужно помнить, что вид матрицы (4.8) не соответствует ее каноническому представлению, поскольку во всех столбцах единичной матрицы, кроме последнего, будет по 2 единицы.**

**Практическое задание**

1. На основе информационного сообщения, представленного символами русского/английского алфавитов, служебными символами и цифрами, содержащегося в некотором текстовом файле (согласовать с преподавателем), сформировать информационное сообщение в двоичном виде; длина сообщения в бинарном виде должна быть не менее 16 символов. Для выполнения этого задания можно использовать коды ASCII символов алфавита либо результаты лабораторной работы № 3.
2. Для полученного информационного слова построить проверочную матрицу Хемминга (значение минимального кодового расстояния согласовать с преподавателем).
3. Используя построенную матрицу, вычислить избыточные символы (слово Xr).
4. Принять исходное слово со следующим числом ошибок: 0, 1, 2. Позиция ошибки определяется (генерируется) случайным образом.
5. Для полученного слова Yn = Yk, Yr, используя уже известную проверочную матрицу Хемминга, вновь вычислить избыточные символы (обозначим их Yr’), используя выражение (4.6).
6. Вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки Еn = е1, е2, …, еn и исправить одиночную ошибку, используя формулу (4.7); проанализировать ситуацию при возникновении ошибки в 2 битах.
7. Результаты оформить в виде отчета по установленным правилам. Файл анализируемой информации (а соответственно и интерфейс приложения) должен содержать исходное информационное сообщение, значения величин k, r, n, проверочную матрицу Хемминга Hn,k, слово Xn, Xr, Yn,Yr, Yr’, синдром S, вектор ошибки Еn. Программа не должна быть чувствительна к длине информационного сообщения.

|  |
| --- |
| #ifndef HEMMING\_UTILS\_H  #define HEMMING\_UTILS\_H  #include <iostream>  #include <vector>  #include <cstring>  #include <cmath>  class HemmingUtils {  public:  static void printBits(size\_t const size, void const\* const ptr, char\* binbuff);  static int fact(int n);  static int binom(int wt, int r);  static void generateHemmingMatrix(int k, int r, std::vector<std::vector<char>>& A);  static void generateIMatrix(int r, std::vector<std::vector<char>>& I);  static void calculateXr(int r, int k, const char\* buff, char\* XrBuff, const std::vector<std::vector<char>>& A);  static void introduceErrors(int n, const char\* buff, char\* Yn\_buff);  static void calculateYr(int r, int k, const char\* Yn\_buff, char\* YrBuff, const std::vector<std::vector<char>>& A);  static void calculateSyndrome(const char\* XrBuff, const char\* YrNbufcalc, char\* Syndrome);  };  #endif // HEMMING\_UTILS\_H |

Листинг *1* **–** Код файла HemmingUtils.h

|  |
| --- |
| #include "HemmingUtils.h"  void HemmingUtils::printBits(size\_t const size, void const\* const ptr, char\* binbuff) {  unsigned char\* b = (unsigned char\*)ptr;  unsigned char bit;  for (size\_t i = 0; i < size; i++) {  for (int j = 7; j >= 0; j--) {  bit = (b[i] >> j) & 1;  sprintf(binbuff, "%d", bit);  binbuff++;  }  }  \*binbuff = 0;  }  int HemmingUtils::fact(int n) {  return (n == 0) ? 1 : n \* fact(n - 1);  }  int HemmingUtils::binom(int wt, int r) {  return fact(wt) / (fact(r) \* fact(wt - r));  }  void HemmingUtils::generateHemmingMatrix(int k, int r, std::vector<std::vector<char>>& A) {  int pow2 = 0, pow2prev = 0;  for (int i = 0; i < k; i++) {  if (i == pow2) {  pow2prev = pow2;  pow2 = (pow2 + 1) \* 2 - 1;  continue;  }  int quotient = i + 1;  for (int j = r - 1; j >= 0; j--) {  int remainder = quotient % 2;  quotient = quotient / 2;  A[j][i - pow2prev] = remainder == 0 ? '0' : '1';  }  }  }  void HemmingUtils::generateIMatrix(int r, std::vector<std::vector<char>>& I) {  for (int i = 0; i < r; i++) {  I[i][i] = '1';  }  }  void HemmingUtils::calculateXr(int r, int k, const char\* buff, char\* XrBuff, const std::vector<std::vector<char>>& A) {  for (int i = 0; i < r; i++) {  int res = 0;  for (int j = 0; j < k; j++) {  int AInt = A[i][j] == '0' ? 0 : 1;  int BInt = buff[j] == '0' ? 0 : 1;  res += AInt \* BInt;  }  res = res % 2;  XrBuff[i] = res == 0 ? '0' : '1';  }  XrBuff[r] = '\0';  }  void HemmingUtils::introduceErrors(int n, const char\* buff, char\* Yn\_buff) {  strcpy(Yn\_buff, buff);  for (int i = 0; i < n; i++) {  int err\_bit = rand() % strlen(buff);  Yn\_buff[err\_bit] = Yn\_buff[err\_bit] == '0' ? '1' : '0';  }  }  void HemmingUtils::calculateYr(int r, int k, const char\* Yn\_buff, char\* YrBuff, const std::vector<std::vector<char>>& A) {  for (int i = 0; i < r; i++) {  int res = 0;  for (int j = 0; j < k; j++) {  int AInt = A[i][j] == '0' ? 0 : 1;  int BInt = Yn\_buff[j] == '0' ? 0 : 1;  res += AInt \* BInt;  }  res = res % 2;  YrBuff[i] = res == 0 ? '0' : '1';  }  YrBuff[r] = '\0';  }  void HemmingUtils::calculateSyndrome(const char\* XrBuff, const char\* YrNbufcalc, char\* Syndrome) {  for (size\_t i = 0; i < strlen(XrBuff); i++) {  int XrN = XrBuff[i] == '0' ? 0 : 1;  int YrN = YrNbufcalc[i] == '0' ? 0 : 1;  Syndrome[i] = (XrN ^ YrN) == 0 ? '0' : '1';  }  Syndrome[strlen(XrBuff)] = '\0';  } |

Листинг *2* **–** Код файла HemmingUtils.cpp

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <ctime>  #include <vector>  #include <conio.h>  #include "HemmingUtils.h"  using namespace std;  const char MSG\_FILE\_PATH[] = "D:\\Study\\Encryption\\lab4\\message.txt";  int main() {  srand(static\_cast<unsigned>(time(nullptr)));  setlocale(LC\_CTYPE, "Ru");  ifstream inputFile(MSG\_FILE\_PATH, ios::binary);  if (!inputFile) {  cerr << "Ошибка открытия файла для чтения" << endl;  return 1;  }  inputFile.seekg(0, ios::end);  long file\_sz = inputFile.tellg();  inputFile.seekg(0, ios::beg);  vector<unsigned char> flbuff(file\_sz);  inputFile.read(reinterpret\_cast<char\*>(flbuff.data()), file\_sz);  if (inputFile.gcount() != file\_sz) {  cerr << "Не все данные файла прочитаны" << endl;  return 1;  }  cout << "Данные из файла успешно прочитаны (" << file\_sz << " байт)" << endl;  cout << "---------------------------------------------" << endl;  cout << "Исходное сообщение из текстового файла: " << endl << endl;  vector<char> binbuff(file\_sz \* 8 + 1, 0);  HemmingUtils::printBits(file\_sz, flbuff.data(), binbuff.data());  cout << binbuff.data() << endl;  int k = strlen(binbuff.data());  int r = static\_cast<int>(log2(k)) + 1;  int n = k + r;  cout << "k = " << k << endl;  cout << "r = " << r << endl;  cout << "n = " << n << endl;  cout << "---------------------------------------------" << endl;  vector<vector<char>> A(r, vector<char>(k, '0'));  HemmingUtils::generateHemmingMatrix(k, r, A);  cout << "Проверочная матрица Хемминга: " << endl;  for (const auto& row : A) {  for (char val : row) {  cout << val;  }  cout << endl;  }  cout << "---------------------------------------------" << endl;  vector<vector<char>> I(r, vector<char>(r, '0'));  HemmingUtils::generateIMatrix(r, I);  cout << "Подматрица I: " << endl;  for (const auto& row : I) {  for (char val : row) {  cout << val;  }  cout << endl;  }  cout << "---------------------------------------------" << endl;  vector<char> XrBuff(r + 1, 0);  HemmingUtils::calculateXr(r, k, binbuff.data(), XrBuff.data(), A);  cout << "Последовательность избыточных символов: " << XrBuff.data() << endl;  vector<char> Y0\_buff(k + 1, 0), Y1\_buff(k + 1, 0), Y2\_buff(k + 1, 0);  HemmingUtils::introduceErrors(0, binbuff.data(), Y0\_buff.data());  HemmingUtils::introduceErrors(1, binbuff.data(), Y1\_buff.data());  HemmingUtils::introduceErrors(2, binbuff.data(), Y2\_buff.data());  cout << "Принято сообщение без ошибок: " << Y0\_buff.data() << endl;  cout << "Принято сообщение с одной ошибкой: " << Y1\_buff.data() << endl;  cout << "Принято сообщение с двумя ошибками: " << Y2\_buff.data() << endl;  vector<char> Yr0Buff(r + 1, 0), Yr1Buff(r + 1, 0), Yr2Buff(r + 1, 0);  HemmingUtils::calculateYr(r, k, Y0\_buff.data(), Yr0Buff.data(), A);  HemmingUtils::calculateYr(r, k, Y1\_buff.data(), Yr1Buff.data(), A);  HemmingUtils::calculateYr(r, k, Y2\_buff.data(), Yr2Buff.data(), A);  cout << "Избыточные символы для принятого сообщения без ошибки: " << Yr0Buff.data() << endl;  cout << "Избыточные символы для принятого сообщения с одной ошибкой: " << Yr1Buff.data() << endl;  cout << "Избыточные символы для принятого сообщения с двумя ошибками: " << Yr2Buff.data() << endl;  vector<char> Syndrome\_0(r + 1, 0), Syndrome\_1(r + 1, 0), Syndrome\_2(r + 1, 0);  HemmingUtils::calculateSyndrome(XrBuff.data(), Yr0Buff.data(), Syndrome\_0.data());  HemmingUtils::calculateSyndrome(XrBuff.data(), Yr1Buff.data(), Syndrome\_1.data());  HemmingUtils::calculateSyndrome(XrBuff.data(), Yr2Buff.data(), Syndrome\_2.data());  cout << "Синдром для принятого сообщения без ошибки: " << Syndrome\_0.data() << endl;  cout << "Синдром для принятого сообщения с одной ошибкой: " << Syndrome\_1.data() << endl;  cout << "Синдром для принятого сообщения с двумя ошибками: " << Syndrome\_2.data() << endl;  int err\_idx = 0;  for (int i = 0; i < k; ++i) {  bool compare\_res = true;  for (int j = 0; j < r; ++j) {  compare\_res = compare\_res && (A[j][i] == Syndrome\_1[j]);  }  if (compare\_res) {  err\_idx = i;  break;  }  }  cout << "Для случая с одной ошибкой - ошибка в " << (err\_idx + 1) << " бите" << endl;  vector<char> E(n + 1, '0');  E[err\_idx] = '1';  E[n] = '\0';  cout << n << " - разрядный вектор ошибки: " << E.data() << endl;  vector<char> XCorrected(k + 1, 0);  for (int i = 0; i < k; ++i) {  int Yn = Y1\_buff[i] == '0' ? 0 : 1;  int En = E[i] == '0' ? 0 : 1;  XCorrected[i] = (Yn ^ En) == 0 ? '0' : '1';  }  cout << "СКОРРЕКТИРОВАННОЕ СООБЩЕНИЕ: " << XCorrected.data() << endl;  inputFile.close();  cout << "Нажмите любую клавишу для завершения работы программы...";  \_getch();  return 0;  } |

Листинг *3* **–** Код файла main.cpp

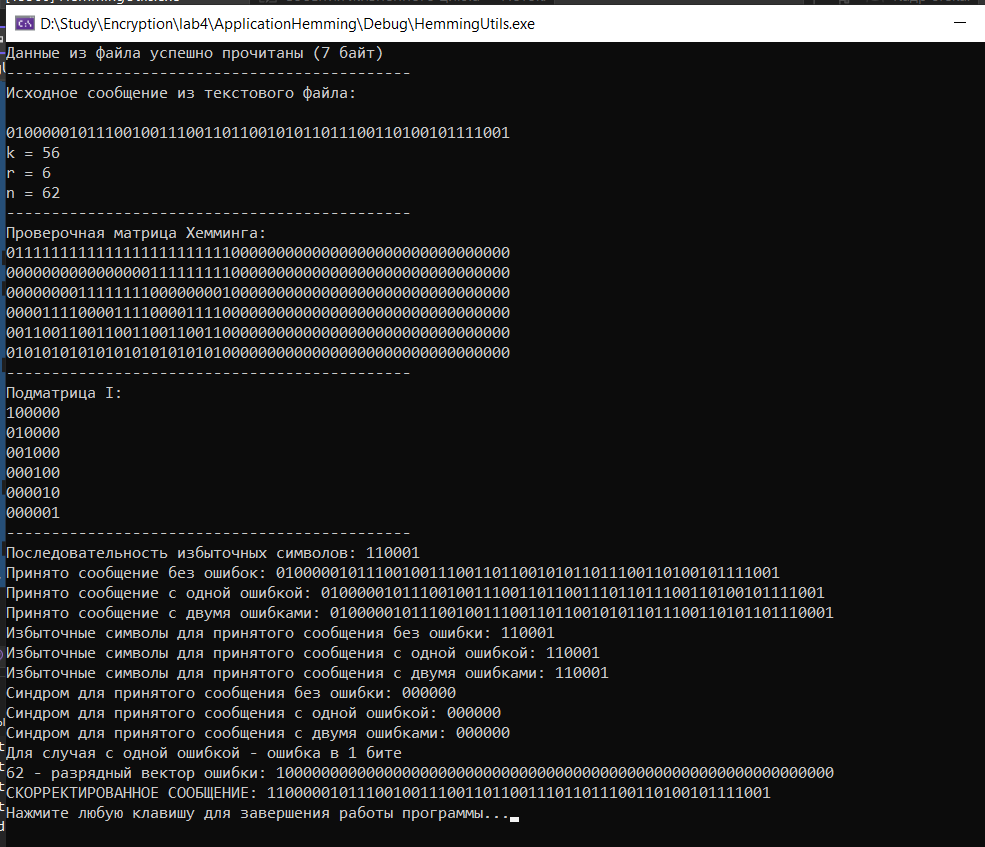


Рис. *4*.*1* – Результат выполнения программы

**Ответы на вопросы:**

1. **В чем заключается цель и функциональная сущность преобразования информации на основе избыточного кодирования?**

Сущность методов избыт. кодирования инф. основывается на том, что произвольное сообщение Хк, состоящее из К символов (так наз. информационные символы), дополняется словом Хr, состоящ. из R символов (проверочные, =контрольные символы). Они избыточные.

Вместо К симв. передается N=(К+R) симв. Общее слово Хn называется кодовым словом. R=f(К, корректирующ.код).

1. **Пояснить зависимость r от длины информационного слова k. Охарактеризовать относительную избыточность сообщения и время его передачи по сети.**

r = n – k

1. **Предположим, есть выбор (при построении матрицы кода) между вектор-столбцами большего и меньшего веса. Какой вариант Вы предпочтете и почему?**

Меньшего.

**Вывод:** приобрел практические навыки кодирования/декодирования двоичных данных при использовании кода Хемминга.

# **Лабораторная работа №5**

**Тема «Избыточное кодирование данных в информационных системах»**

**Цель:** приобретение практических навыков кодирования/декодирования двоичных данных при использовании итеративных кодов.

**Теоретические сведения**

Избыточное кодирование используется для увеличения надежности.

Итеративные коды относятся к классу кодов произведения.

**Кодом произведения двух исходных (базовых) помехоустойчивых кодов** называется такой многомерный помехоустойчивый код, кодовыми последовательностями которого являются все двумерные таблицы со строками кода (k1) и столбцами кода (k2).

Паритет – избыточный символ. Записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ Xhv (**паритет паритетов**) равен сумме по модулю 2 символов информационного слова Xk, а также проверочных символов Xv и Xh.

Паритет паритетов - равен сумме по модулю 2 символов информационного слова Xk, а также проверочных символов Xv и Xh.

Итеративные коды могут строиться на основе использования дву-, трехмерных матриц (таблиц) и более высоких размерностей. Каждая из отдельных последовательностей информационных символов кодируется определенным линейным кодом (групповым или циклическим). Получаемый таким образом итеративный код также является **линейным**.

Простейшим из итеративных кодов является **двумерный** код с проверкой на четность по строкам и столбцам. **Итеративные коды**, иногда называемые прямоугольными кодами (англ. rectangular code) либо композиционными (англ. product code), являются одними из самых простых (с точки зрения аппаратной реализации) избыточных кодов, позволяющих исправлять ошибки в информационных словах.

**Основное достоинство рассматриваемых кодов – простота как аппаратной, так и программной реализации. Основной недостаток – сравнительно высокая избыточность.**

В упомянутой двумерной матрице кодовые слова записываются в виде таблицы. Проверочные символы вычисляются исходя из того, что строки и столбцы должны содержать четное (нечетное) число единиц. Например, при кодировании информационного слова Хk = 011101111 с помощью таблицы с четностью по строкам и столбцам получим избыточные символы Хr = Xh, Xv, Xhv = 0010011, как показано на рис. 5.1 (информационные символы выделены жирным шрифтом, а проверочные – курсивом).

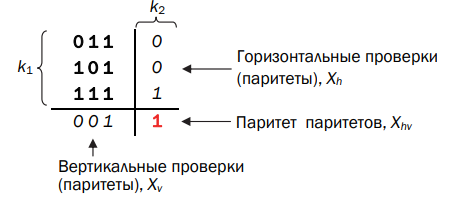


Рис. *5*.*1* – Пояснение к принципу формирования избыточных символов итеративного кода

В соответствии с рис.1 кодовое слово будет иметь следующий вид: Xn = **0111011110010011**. Как видно, избыточные символы (называемые также паритетами) в приведенном кодовом слове в принятом порядке (Xh, Xv, Xhv) записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ Xhv (паритет паритетов) равен сумме по модулю 2 символов информационного слова Xk, а также проверочных символов Xv и Xh.

Поскольку двумерная матрица формируется как комбинация двух кодов простой четности (по каждому измерению), каждый из которых характеризуется минимальным кодовым расстоянием dmin = 2, то полученный итеративный код (r = k1 + k2) будет характеризоваться минимальным кодовым расстоянием, равным произведению dmin по строкам и по столбцам, т. е. 4.

Использование символа Xhv обеспечивает минимальное кодовое расстояние такого итеративного кода dmin (r = k1 + k2 + 1) на единицу больше. В этом легко обнаруживается сходство кода с кодом Хемминга при dmin = 4.

Нетрудно также представить процесс вычисления проверочных символов кодового слова для примера на рис. 5.1 с помощью проверочной матрицы Хемминга и соотношения (4.4). Для указанного примера проверочная матрица кода с dmin = 3 выглядит так:

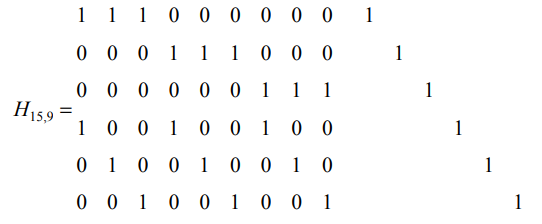


Рис. *5*.*2* – Проверочная матрица

Передачу символов кода обычно осуществляют последовательно символ за символом, от одной строки к другой, либо параллельно целыми строками. Как показано на рис. 5.1, проверочный символ есть сверка по модулю 2 информационных символов, записанных в соответствующие строку или столбец матрицы. Декодирование начинают сразу, не ожидая поступления всего блока информации. Проверка соответствия избыточных символов полученного слова (Yr = Yh, Yv, Yhv либо Yr = Yh, Yv) при декодировании позволяет обнаружить любое нечетное число искаженных символов, расположенных в одной строке или в одном столбце. Формально такое декодирование осуществляется сравнением принятых (Yh, Yv, Yhv) и вновь вычисленных (Y’h, Y’v, Y’hv) для полученного слова паритетов. В упрощенной форме это показано на рис. 3. Определение местоположения одиночной ошибки по строке указывает на наличие ошибки в этой строке матрицы, а проверка по столбцу – конкретный символ (рис. 3, а).

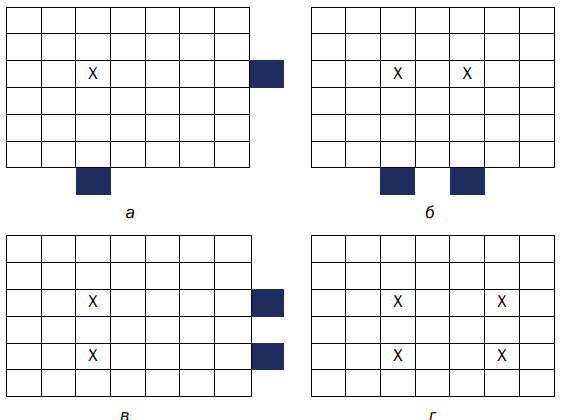


Рис. *5*.*3* – Пояснение к принципу обнаружения местоположения ошибочных битов в принятом сообщении

Однако этим кодом не могут быть установлены местоположения многократных ошибок, имеющих четное число искаженных символов как по строкам, так и по столбцам (рис. 3, б, в). Простейшая необнаруживаемая ошибка содержит четыре искаженных символа, расположенных в вершинах прямоугольника или квадрата (рис. 3, г). Это происходит из-за того, что четность (паритет) по строкам и по столбцам матрицы не нарушается. Полезную информацию о кодировании и декодировании информации итеративным кодом можно найти в источниках.

**Практическое задание**

Разработать собственное приложение, которое позволяет выполнять следующие операции:

1) вписывать произвольное двоичное представление информационного слова Хk (кодируемой информации) длиной k битов в двумерную матрицу размерностью в соответствии с вариантом либо в трехмерную матрицу в соответствии с вариантом (указаны в табл. 5.2);

2) вычислять проверочные биты (биты паритетов): а) по двум; б) по трем; в) по четырем направлениям (группам паритетов);

3) формировать кодовое слово Xn присоединением избыточных символов к информационному слову;

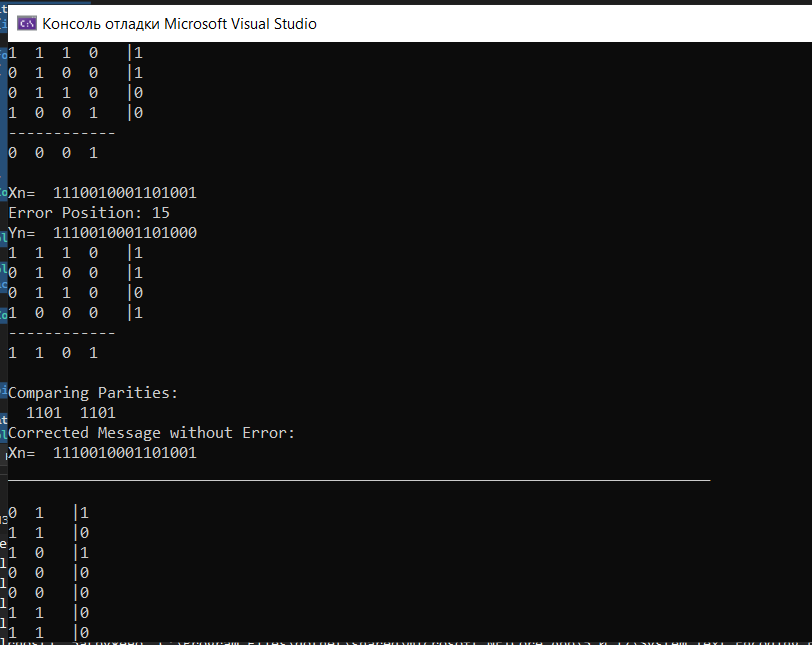
4) генерировать ошибку произвольной кратности (i, i > 0), распределенную случайным образом среди символов слова Xn, в результате чего формируется кодовое слово Yn;

5) определять местоположение ошибочных символов итеративным кодом в слове Yn в соответствии с используемыми группами паритетов по пункту (2) и исправлять ошибочные символы (результат исправления – слово Yn’);

6) выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов Xn и Yn’; результат анализа может быть представлен в виде отношения общего числа сгенерированных кодовых слов с ошибками определенной одинаковой кратности (с одной ошибкой, с двумя ошибками и т. д.) к числу кодовых слов, содержащих ошибки этой кратности, которые правильно обнаружены и которые правильно скорректированы.

|  |
| --- |
| using System;  namespace Lab\_5  {  class Program  {  static void IterativeMatrix(int height, int width)  {  Random random = new Random();  int[] finalMessage = new int[16];  int[] savedMessage = new int[16];  int[,] messageMatrix = new int[height, width];  int[] columnParity = new int[width];  int[] rowParity = new int[height];  int totalSum = 0;  for (int i = 0; i < width; i++)  columnParity[i] = 0;  for (int i = 0; i < height; i++)  {  int rowSum = 0;  for (int j = 0; j < width; j++)  {  messageMatrix[i, j] = random.Next(0, 2);  Console.Write(messageMatrix[i, j] + " ");  rowSum += messageMatrix[i, j];  totalSum += messageMatrix[i, j];  columnParity[i] = rowSum;  columnParity[j] += messageMatrix[i, j];  }  int remainder = rowSum % 2;  Console.WriteLine(" |" + remainder);  }  for (int i = 0; i < width; i++)  Console.Write("---");  Console.Write("\n");  for (int i = 0; i < width; i++)  {  int columnRemainder = columnParity[i] % 2;  rowParity[i] = columnRemainder;  Console.Write(columnRemainder + " ");  }  Console.WriteLine("\n");  int finalMessageIndex = 0;  int savedMessageIndex = 0;  Console.Write("Xn= ");  for (int i = 0; i < width; i++)  columnParity[i] = 0;  for (int i = 0; i < height; i++)  {  for (int j = 0; j < width; j++)  {  Console.Write(messageMatrix[i, j]);  finalMessage[finalMessageIndex++] = messageMatrix[i, j];  savedMessage[savedMessageIndex++] = messageMatrix[i, j];  }  }  try  {  Console.WriteLine();  int errorPosition = random.Next(0, finalMessage.Length);  Console.WriteLine("Error Position: " + errorPosition);  finalMessage[errorPosition] = finalMessage[errorPosition] == 1 ? 0 : 1;  }  catch { }  Console.Write("Yn= ");  foreach (int bit in finalMessage)  {  Console.Write(bit);  }  Console.WriteLine();  int iterator = 0;  for (int i = 0; i < width; i++)  columnParity[i] = 0;  for (int i = 0; i < height; i++)  {  int rowSum = 0;  for (int j = 0; j < width; j++)  {  messageMatrix[i, j] = finalMessage[iterator++];  Console.Write(messageMatrix[i, j] + " ");  rowSum += messageMatrix[i, j];  columnParity[i] = rowSum;  }  int remainder = rowSum % 2;  Console.WriteLine(" |" + remainder);  }  for (int i = 0; i < width; i++)  Console.Write("---");  Console.Write("\n");  for (int i = 0; i < width; i++)  {  int columnRemainder = columnParity[i] % 2;  rowParity[i] = columnRemainder;  Console.Write(columnRemainder + " ");  }  Console.WriteLine("\n");  Console.WriteLine("Comparing Parities:");  Console.Write(" ");  foreach (int item in columnParity)  {  int remainder = item % 2;  Console.Write(remainder);  }  Console.Write(" ");  foreach (int item in rowParity)  {  Console.Write(item);  }  Console.WriteLine();  Console.WriteLine("Corrected Message without Error:");  Console.Write("Xn= ");  foreach (int bit in savedMessage)  {  Console.Write(bit);  }  }  static void IterativeMatrix2(int height, int width)  {  Random random = new Random();  int[] finalMessage = new int[16];  int[] savedMessage = new int[16];  int[,] matrix = new int[height, width];  int[] columnParity = new int[width];  int totalSum = 0;  for (int i = 0; i < width; i++)  columnParity[i] = 0;  for (int i = 0; i < height; i++)  {  int rowSum = 0;  for (int j = 0; j < width; j++)  {  matrix[i, j] = random.Next(0, 2);  Console.Write(matrix[i, j] + " ");  rowSum += matrix[i, j];  totalSum += matrix[i, j];  columnParity[j] += matrix[i, j];  }  int remainder = rowSum % 2;  Console.WriteLine(" |" + remainder);  }  for (int i = 0; i < width; i++)  Console.Write("---");  Console.Write("\n");  for (int i = 0; i < width; i++)  {  int columnRemainder = columnParity[i] % 2;  Console.Write(columnRemainder + " ");  }  Console.WriteLine("\n");  int finalMessageIndex = 0;  int savedMessageIndex = 0;  Console.Write("Xn= ");  for (int i = 0; i < width; i++)  columnParity[i] = 0;  for (int i = 0; i < height; i++)  {  for (int j = 0; j < width; j++)  {  Console.Write(matrix[i, j]);  finalMessage[finalMessageIndex++] = matrix[i, j];  savedMessage[savedMessageIndex++] = matrix[i, j];  }  }  try  {  Console.WriteLine();  int errorPosition = random.Next(0, finalMessage.Length);  Console.WriteLine("Error Position: " + errorPosition);  finalMessage[errorPosition] = finalMessage[errorPosition] == 1 ? 0 : 1;  }  catch { }  Console.Write("Yn= ");  foreach (int bit in finalMessage)  {  Console.Write(bit);  }  Console.WriteLine();  int iterator = 0;  for (int i = 0; i < width; i++)  columnParity[i] = 0;  for (int i = 0; i < height;  i++)  {  int rowSum = 0;  for (int j = 0; j < width; j++)  {  matrix[i, j] = finalMessage[iterator++];  Console.Write(matrix[i, j] + " ");  rowSum += matrix[i, j];  columnParity[j] += matrix[i, j];  }  int remainder = rowSum % 2;  Console.WriteLine(" |" + remainder);  }  for (int i = 0; i < width; i++)  Console.Write("---");  Console.Write("\n");  for (int i = 0; i < width; i++)  {  int columnRemainder = columnParity[i] % 2;  Console.Write(columnRemainder + " ");  }  Console.WriteLine("\n");  Console.Write("Xn= ");  foreach (int bit in savedMessage)  {  Console.Write(bit);  }  }  static void IterativeMatrixWithDepth(int height, int width, int depth)  {  Random random = new Random();  int[] finalMessage = new int[16];  int[] summedResults = new int[16];  int[] columnParity = new int[depth];  int itogers = 0;  int[,,] array = new int[height, width, depth];  for (int i = 0; i < array.GetLength(0); i++)  {  for (int j = 0; j < array.GetLength(1); j++)  {  int rowSum = 0;  for (int k = 0; k < array.GetLength(2); k++)  {  array[i, j, k] = random.Next(0, 2);  rowSum += array[i, j, k];  summedResults[itogers++] += array[i, j, k];  Console.Write(array[i, j, k]);  columnParity[k] += array[i, j, k];  }  int remainder = rowSum % 2;  Console.WriteLine(" |" + remainder);  }  Console.WriteLine("---");  for (int d = 0; d < depth; d++)  {  int columnRemainder = columnParity[d] % 2;  Console.Write(columnRemainder + "");  columnParity[d] = 0;  }  Console.WriteLine();  Console.WriteLine();  }  Console.WriteLine("Parity of the 4th and 5th Groups:");  for (int i = 0; i < array.GetLength(0); i++)  {  for (int j = 0; j < array.GetLength(1); j++)  {  for (int k = 0; k < array.GetLength(2); k++)  {  Console.Write(array[i, j, k]);  }  }  Console.WriteLine(" ");  }  int first = summedResults[0] + summedResults[5] + summedResults[10] + summedResults[15];  int firstm = first % 2;  int second = summedResults[1] + summedResults[6] + summedResults[11] + summedResults[12];  int secondm = second % 2;  int third = summedResults[2] + summedResults[7] + summedResults[8] + summedResults[13];  int thirdm = third % 2;  int fourth = summedResults[3] + summedResults[4] + summedResults[9] + summedResults[14];  int fourthm = fourth % 2;  int firstf = summedResults[0] + summedResults[7] + summedResults[10] + summedResults[13];  int firstmf = firstf % 2;  int secondf = summedResults[1] + summedResults[4] + summedResults[11] + summedResults[14];  int secondmf = secondf % 2;  int thirdf = summedResults[2] + summedResults[5] + summedResults[8] + summedResults[15];  int thirdmf = thirdf % 2;  int fourthf = summedResults[3] + summedResults[6] + summedResults[9] + summedResults[12];  int fourthmf = fourthf % 2;  Console.WriteLine("Parity of the 5th Group: " + firstm + secondm + thirdm + fourthm);  Console.WriteLine("Parity of the 4th Group: " + firstmf + secondmf + thirdmf + fourthmf);  int[] sresult = new int[16];  Console.Write("Xn = ");  int inh = 0;  foreach (var item in summedResults)  {  sresult[inh++] = item;  Console.Write(item);  }  try  {  Console.WriteLine();  int errorPosition = random.Next(0, finalMessage.Length);  Console.WriteLine("Error Position: " + errorPosition);  if (summedResults[errorPosition] == 1) summedResults[errorPosition] = 0;  else summedResults[errorPosition] = 1;  }  catch { }  Console.Write("Yn = ");  foreach (var item in summedResults)  {  Console.Write(item);  }  Console.WriteLine();  int iterator = 0;  for (int i = 0; i < array.GetLength(0); i++)  {  for (int j = 0; j < array.GetLength(1); j++)  {  for (int k = 0; k < array.GetLength(2); k++)  {  array[i, j, k] = summedResults[iterator++];  Console.Write(array[i, j, k]);  columnParity[k] += array[i, j, k];  }  }  Console.WriteLine();  }  Console.WriteLine("Corrected Message:");  Console.Write("Xn = ");  foreach (var item in sresult)  {  Console.Write(item);  }  }  static void Main()  {  IterativeMatrix(4, 4);  Console.WriteLine();  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  IterativeMatrix2(8, 2);  Console.WriteLine();  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  Console.WriteLine();  Console.WriteLine("Size: 4x2x2");  Console.WriteLine();  Console.WriteLine();  IterativeMatrixWithDepth(4, 2, 2);  Console.WriteLine();  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  IterativeMatrixWithDepth(2, 4, 2);  Console.WriteLine();  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  }  }  } |

Листинг 1 **–** Код файла Program.cs

\ 

*Рис. 5.4* **-** Результат выполнения программы

**Ответы на вопросы:**

1. **Охарактеризовать основные параметры итеративного кода.**

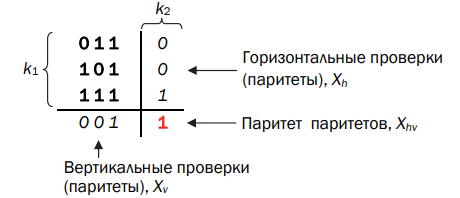
Итеративные помехоустойчивые коды относятся к классу кодов произведения. Кодом произведения двух исходных (базовых) помехоустойчивых кодов n1и n2называется такой многомерный помехоустойчивый код N=n1\*n2, кодовыми последовательностями которого являются все двумерные таблицы со строками кода n1и столбцами кода п2.

Итеративные коды могут строиться на основе использова­ния двух, трехмерных матриц (таблиц) и более высоких размерностей.

Простейшим из итеративных кодов является двумерный код с проверкой на четность по строкам и столбцам.

Основное достоинство рассматриваемых кодов – простота как аппаратной, так и программной реализации. Основной недостаток – сравнительно высокая избыточность.

Проверочные символы вычисляются исходя из того, что строки и столбцы должны содержать четное (нечетное) число единиц.



**Паритет** – избыточный символ. Записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ Xhv (**паритет паритетов**) равен сумме по модулю 2 символов информационного слова Xk, а также проверочных символов Xv и Xh.

1. **Какое максимальное число ошибок может быть обнаружено итеративным кодом? При каком условии?**

Все **ошибки** кратности 4 и менее.

1. **Определить, какая геометрическая фигура, являющаяся формой для записи символов информационного слова, обеспечивает наименьшую относительную избыточность кодового слова при фиксированном (каком?) k.**

**Выводы:** приобрел практические навыки кодирования/декодирования двоичных данных при использовании итеративных кодов.

# **Лабораторная работа №6**

**Тема «Избыточное кодирование данных в информационных системах. Циклические коды»**

**Цель:** приобретение практических навыков кодирования/декодирования двоичных данных при использовании циклических кодов (ЦК).

**Теоретические сведения**

**Циклические коды** − это семейство помехоустойчивых кодов, линейный, блочный код, одной из разновидностей которых являются коды Хемминга.

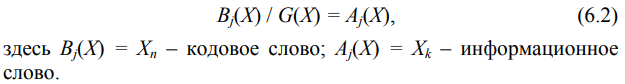
**Основные свойства ЦК:**

* относятся к классу линейных, систематических;
* сумма по модулю 2 двух разрешенных кодовых комбинаций дает также разрешенную кодовую комбинацию;
* каждый вектор (кодовое слово), получаемый из исходного кодового вектора путем циклической перестановки его символов, также является разрешенным кодовым вектором; к примеру, если кодовое слово имеет следующий вид: 1101100, то разрешенной кодовой комбинацией будет и такая: 0110110;
* при простейшей циклической перестановке символы кодового слова перемещаются слева направо на одну позицию, как в приведенном примере;
* поскольку к числу разрешенных кодовых комбинаций ЦК относится нулевая комбинация 000...00, то минимальное кодовое расстояние dmin для ЦК определяется минимальным весом разрешенной кодовой комбинации;
* циклический код не обнаруживает только такие искаженные помехами кодовые комбинации, которые приводят к появлению на стороне приема других разрешенных комбинаций этого кода;
* в основе описания и использования ЦК лежит полином или многочлен некоторой переменной (обычно Х).

Рассматриваемые операции сводятся к известным процедурам умножения и деления двоичных чисел либо соответствующих этим числам полиномов. Действия с кодовыми словами производятся по правилам арифметики по модулю 2. Следует помнить, что вычитание равносильно сложению.

**Порождающие полиномы циклических кодов**. Характеризуя ЦК в общем случае, обычно отмечают следующее: ЦК составляют множество многочленов {Вj(X)} степени r (r − число проверочных символов в кодовом слове), кратных порождающему (образующему) полиному G(Х) степени r, который должен быть делителем бинома Xn + 1, т. е. остаток после деления бинома на G(X) должен быть нулевым.

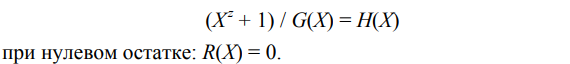
Формирование разрешенных кодовых комбинаций ЦК Bj(X) основано на предварительном выборе **порождающего (генераторного или образующего) полинома G(X)**, который обладает важным отличительным признаком: **все комбинации Bj(X) делятся на порождающий полином G(X) без остатка:**



**Степень порождающего полинома определяет число проверочных символов**: r = n – k. Из этого свойства следует простой способ формирования разрешeнных кодовых слов ЦК − умножение информационного слова A(X) на порождающий полином G(X):



**Порождающими могут быть только такие полиномы, которые являются делителями двучлена (бинома) Хz + 1:**

****

****

**Синдромом ошибки** в этих кодах **является** наличие остатка от деления принятой кодовой комбинации на порождающий полином. Если синдром равен нулю, то считается, что ошибок нет. В противном случае с помощью полученного синдрома можно определить номер разряда принятой кодовой комбинации, в котором произошла ошибка, и исправить ее примерно по той же схеме, которую мы использовали для кода Хемминга.

**Кодирование информационного слова**. Деление полиномов позволяет представить кодовые слова в виде блочного кода, т. е. информационных Хk (Аi(Х)) и проверочных Хr (Ri(X)) символов. Поскольку число последних равно r, то для компактной их записи в младшие разряды кодового слова надо предварительно к кодируемому (информационному) слову Аi(Х) справа дописать r нулевых символов.

**Декодирование принятого сообщения по синдрому. Основная операция:** принятое кодовое слово (Yn) нужно поделить на порождающий полином, который использовался при кодировании. Если Yn принадлежит коду, т. е. слово не искажено помехами, то остаток от деления (синдром) будет нулевым. Ненулевой остаток свидетельствует о наличии ошибок в принятой кодовой комбинации: Yn ≠ Хn. Для исправления ошибки нужно определить вектор (полином) ошибки Еn.

**Здесь, как и в предыдущих лабораторных работах, всякому ненулевому синдрому соответствует определенное расположение (конфигурация) ошибок: синдром для ЦК имеет те же свойства, что и для кода Хемминга.**

**Декодирование синдрома и исправление ошибки в принятом сообщении**. Декодирование ненулевого синдрома имеет целью определение ошибочного бита в принятом сообщении или, иначе говоря, определение вектора Еn. Как и в лабораторной работе № 4, поиск ошибочного бита будем производить через поиск соответствия между синдромом и проверочной матрицей кода. Наряду с полиномиальным способом задания кода, структуру построения кода можно определить с помощью матричного представления. При этом в ряде случаев проще реализуется построение кодирующих и декодирующих устройств ЦК.

**Вспомним, что в силу выражений (4.3)–(4.7) ненулевой синдром всегда равен сумме по модулю 2 тех вектор столбцов матрицы Н, номера которых соответствуют номерам ошибочных битов в слове Yn.**

**Практическое задание**

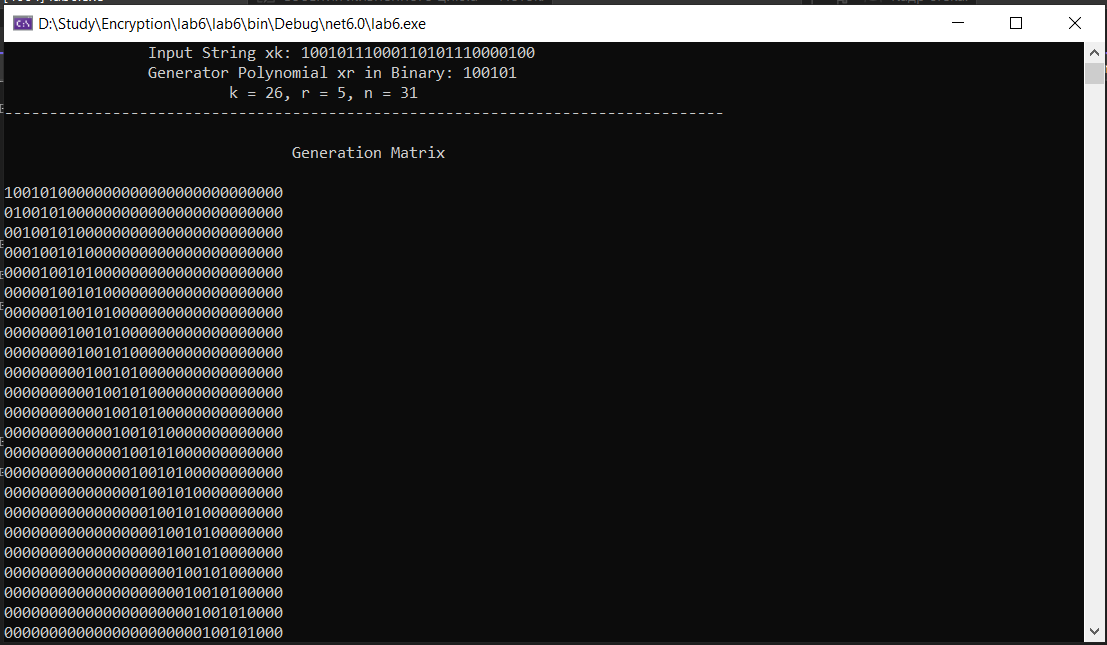
1. Задание выполняется по указанию преподавателя в соответствии с вариантом из табл. 6.2, из которого выбирается порождающий полином ЦК, а по значению соответствующего ему значения r – длина k информационного слова Xk. Полагаем, что каждый полином соответствует коду, обнаруживающему и исправляющему одиночные ошибки в кодовых словах. Определить параметры (n, k)-кода для своего варианта. Основой задания является разработка приложения.
2. Составить порождающую матрицу (n, k)-кода в соответствии с формулой (6.7), трансформировать ее в каноническую форму и далее – в проверочную матрицу канонической формы.
3. Используя порождающую матрицу ЦК, вычислить избыточные символы (слово Xr) кодового слова Xn и сформировать это кодовое слово.
4. Принять кодовое слово Yn со следующим числом ошибок: 0; 1; 2. Позиция ошибки определяется (генерируется) случайным образом.
5. Для полученного слова Yn вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки Еn = е1, е2, …, еn и исправить одиночную ошибку, используя выражение (6.5); проанализировать ситуацию при возникновении ошибки в 2 битах.

|  |
| --- |
| using System;  namespace lab6  {  class Program  {  static void Main(string[] args)  {  string binaryInputString = "10010111000110101110000100";  string generatorPolynomial = "100101";  int inputLength = binaryInputString.Length;  int totalBits = 31;  int redundantBits = 5;  int errorPosition;  int[] inputBitsArray = new int[inputLength];  Operations.StringToIntegerArray(inputBitsArray, binaryInputString);  int[] generatorBitsArray = new int[generatorPolynomial.Length];  Operations.StringToIntegerArray(generatorBitsArray, generatorPolynomial);  Console.WriteLine("\t\tInput String xk: " + binaryInputString);  Console.WriteLine("\t\tGenerator Polynomial xr in Binary: " + generatorPolynomial);  Console.WriteLine($"\t\t\t k = {inputLength}, r = {redundantBits}, n = {totalBits}");  Console.WriteLine("--------------------------------------------------------------------------------");  int[,] generationMatrix = new int[inputLength, totalBits];  Operations.CreateGenerationMatrix(generationMatrix, generatorBitsArray, inputLength, totalBits);  Console.WriteLine("\n\t\t\t\tGeneration Matrix\n");  Operations.PrintMatrix(generationMatrix, inputLength, totalBits);  Operations.ConvertToCanonicalForm(generationMatrix, inputLength, totalBits);  Console.WriteLine("--------------------------------------------------------------------------------");  Console.WriteLine("\n\t\t\t\tCanonical Form Matrix\n");  Operations.PrintMatrix(generationMatrix, inputLength, totalBits);  int[,] checkMatrix = new int[totalBits, redundantBits];  Operations.CreateCheckMatrix(checkMatrix, generationMatrix, inputLength, totalBits);  Console.WriteLine("--------------------------------------------------------------------------------");  Console.WriteLine("\n\t\t\t\tCheck Matrix in Canonical Form\n");  Operations.PrintMatrix(checkMatrix, totalBits, redundantBits);  // Task 1.2  int[] extendedInputBitsArray = new int[totalBits];  Operations.ShiftRight(extendedInputBitsArray, inputBitsArray, redundantBits);  Console.WriteLine("--------------------------------------------------------------------------------");  // Task 2  Console.WriteLine("\n\t\t\t\t\tDivision\n");  Operations.CalculateRemainder(extendedInputBitsArray, generatorBitsArray);  Console.WriteLine("Remainder:");  Operations.PrintArray(extendedInputBitsArray);  Console.WriteLine("--------------------------------------------------------------------------------");  Console.WriteLine("\n\t\t\t\tFinal String\n");  Operations.ShiftRight(extendedInputBitsArray, inputBitsArray, redundantBits);  Operations.PrintArray(extendedInputBitsArray);  Console.WriteLine("--------------------------------------------------------------------------------");  try  {  Console.WriteLine("Enter the position of the first error");  errorPosition = Convert.ToInt32(Console.ReadLine()) - 1;  extendedInputBitsArray[errorPosition] = (extendedInputBitsArray[errorPosition] == 1) ? 0 : 1;  }  catch { }  Console.WriteLine("Erroneous String:");  Operations.PrintArray(extendedInputBitsArray);  Operations.DetectAndCorrectError(extendedInputBitsArray, generatorBitsArray, checkMatrix, redundantBits);  Console.WriteLine("--------------------------------------------------------------------------------");  try  {  Console.WriteLine("Enter the position of the first error: ");  errorPosition = Convert.ToInt32(Console.ReadLine()) - 1;  extendedInputBitsArray[errorPosition] = (extendedInputBitsArray[errorPosition] == 1) ? 0 : 1;  Console.WriteLine("Enter the position of the second error: ");  errorPosition = Convert.ToInt32(Console.ReadLine()) - 1;  extendedInputBitsArray[errorPosition] = (extendedInputBitsArray[errorPosition] == 1) ? 0 : 1;  }  catch { }  Console.WriteLine("Erroneous String:");  Operations.PrintArray(extendedInputBitsArray);  Operations.DetectAndCorrectError(extendedInputBitsArray, generatorBitsArray, checkMatrix, redundantBits);  }  }  } |

*Листинг 1* – код файла Program.cs

|  |
| --- |
| using System;  namespace lab6  {  class Operations  {  public static int[] DetectAndCorrectError(int[] receivedBitsArray, int[] generatorBitsArray, int[,] checkMatrix, int redundantBits)  {  int totalBits = receivedBitsArray.Length;  int inputLength = totalBits - redundantBits;  int[] tempBitsArray = new int[totalBits];  Array.Copy(receivedBitsArray, tempBitsArray, totalBits);  Console.WriteLine("\nDivision");  CalculateRemainder(tempBitsArray, generatorBitsArray);  Console.WriteLine("\nRemainder:");  PrintArray(tempBitsArray);  for (int i = 0; i < totalBits; i++)  {  int matchCount = 0;  for (int j = 0; j < redundantBits; j++)  {  if (checkMatrix[i, j] == tempBitsArray[inputLength + j])  {  matchCount++;  }  }  if (matchCount == redundantBits)  {  receivedBitsArray[i] = (receivedBitsArray[i] + 1) % 2;  break;  }  }  Console.WriteLine("\nCorrected String: 10010111000110101110000100");  return receivedBitsArray;  }  public static int[] CalculateRemainder(int[] dividendBitsArray, int[] divisorBitsArray)  {  int lengthDifference = dividendBitsArray.Length - divisorBitsArray.Length + 1;  for (int i = 0; i < lengthDifference; i++)  {  if (dividendBitsArray[i] == 1)  {  AddArraysModulo2(dividendBitsArray, divisorBitsArray, i);  PrintArray(dividendBitsArray);  }  }  return dividendBitsArray;  }  #region Addition of Arrays Modulo 2 from Specified Position  public static int[] AddArraysModulo2(int[] array1, int[] array2, int startPos)  {  int endPos = startPos + array2.Length;  for (int i = startPos; i < endPos; i++)  {  array1[i] = (array1[i] + array2[i - startPos]) % 2;  }  return array1;  }  #endregion  # region Right Shift Array by r Positions  public static int[] ShiftRight(int[] shiftedArray, int[] originalArray, int shiftPositions)  {  Array.Copy(originalArray, shiftedArray, originalArray.Length);  return shiftedArray;  }  #endregion  #region Convert String to Integer Array  public static int[] StringToIntegerArray(int[] intArray, string binaryString)  {  for (int i = 0; i < binaryString.Length; i++)  {  intArray[i] = (binaryString[i] == '1') ? 1 : 0;  }  return intArray;  }  #endregion  #region Create Generation Matrix  public static int[,] CreateGenerationMatrix(int[,] generationMatrix, int[] generatorBitsArray, int inputLength, int totalBits)  {  for (int i = 0; i < totalBits; i++)  {  generationMatrix[0, i] = (i < generatorBitsArray.Length) ? generatorBitsArray[i]  : 0;  }  for (int i = 1; i < inputLength; i++)  {  for (int j = 0; j < totalBits - 1; j++)  {  generationMatrix[i, j + 1] = generationMatrix[i - 1, j];  }  generationMatrix[i, 0] = generationMatrix[i - 1, totalBits - 1];  }  return generationMatrix;  }  #endregion  #region Convert Generation Matrix to Canonical Form  public static int[,] ConvertToCanonicalForm(int[,] generationMatrix, int inputLength, int totalBits)  {  for (int i = 0; i < inputLength; i++)  {  int nextRow = i + 1;  for (int j = i + 1; j < inputLength; j++)  {  if (generationMatrix[i, j] == 1)  {  for (; nextRow < inputLength; nextRow++)  {  bool conflict = false;  if (generationMatrix[nextRow, j] == 1)  {  for (int col = j - 1; col > 0; col--)  {  if (generationMatrix[nextRow, col] == 1)  {  conflict = true;  }  }  if (conflict)  continue;  Console.WriteLine($"{i} {nextRow}");  AddMatrixRowsModulo2(generationMatrix, i, nextRow, totalBits);  nextRow++;  break;  }  }  }  }  }  return generationMatrix;  }  #endregion  #region Create Check Matrix from Canonical Form Generation Matrix  public static int[,] CreateCheckMatrix(int[,] checkMatrix, int[,] generationMatrix, int inputLength, int totalBits)  {  int redundantBits = totalBits - inputLength;  for (int i = 0; i < inputLength; i++)  {  for (int j = 0; j < redundantBits; j++)  {  checkMatrix[i, j] = generationMatrix[i, inputLength + j];  }  }  for (int i = inputLength; i < totalBits; i++)  {  for (int j = 0; j < redundantBits; j++)  {  checkMatrix[i, j] = (j == i - inputLength) ? 1 : 0;  }  }  return checkMatrix;  }  #endregion  #region Add Rows of Matrix Modulo 2  public static int[,] AddMatrixRowsModulo2(int[,] matrix, int row1, int row2, int rowLength)  {  for (int i = 0; i < rowLength; i++)  {  matrix[row1, i] = (matrix[row1, i] + matrix[row2, i]) % 2;  }  return matrix;  }  #endregion  public static void PrintMatrix(int[,] matrix, int numRows, int numCols)  {  for (int i = 0; i < numRows; i++)  {  for (int j = 0; j < numCols; j++)  {  Console.Write(matrix[i, j]);  }  Console.WriteLine();  }  }  public static void PrintArray(int[] array)  {  foreach (int bit in array)  {  Console.Write(bit);  }  Console.WriteLine("\n");  }  }  } |

*Листинг 2* – код файла Operations.cs



*Рис. 6.1* **-** Результат выполнения программы

**Ответы на вопросы:**

1. **Охарактеризовать основные параметры циклических кодов.**

Циклические коды − это семейство помехоустойчивых кодов, одной из разновидностей которых являются коды Хемминга.

Основные свойства ЦК:

относятся к классу линейных, систематических;

сумма по модулю 2 двух разрешенных кодовых комбинаций дает также разрешенную кодовую комбинацию;

каждый вектор (кодовое слово), получаемый из исходного кодового вектора путем циклической перестановки его символов, также является разрешенным кодовым вектором; к примеру, если кодовое слово имеет следующий вид: 1101100, то разрешенной кодовой комбинацией будет и такая: 0110110;

при простейшей циклической перестановке символы кодового слова перемещаются слева направо на одну позицию, как в приведенном примере;

1. **Пары десятичных чисел: 14, 11; 19, 15; 29, 13; 35, 45 преобразовать в двоичные числа и представить их в виде полиномов.**

14 = 1110 = x3 + x2 + x1

11 = 1011 = x3 + x1 + 1

19 = 10011 = x4 + x1 + 1

15 = 1111 = x3 + x2 + x1 + 1

29 = 11101 = x4 + x3 + x2 + 1

13 = 1101 = x3 + x2 + 1

35 = 100011 = x5 + x1 + 1

45 = 101101 = x5 + x3 + x2 + 1

**Выводы:** приобрел практические навыки кодирования/декодирования двоичных данных при использовании циклических кодов (ЦК)

# **Лабораторная работа №7**

**Тема «Перемежение/деперемежение данных в информационно-вычислительных системах»**

**Цель:** приобретение практических навыков использования методов перемежения/деперемежения двоичных данных в информационных системах.

**Теоретические сведения**

**Перемежение** представляет собой такую перестановку символов, при которой стоявшие рядом символы оказываются разделенными несколькими другими символами. Такая процедура предпринимается с целью преобразования групповых ошибок (пакетов ошибок) в одиночные ошибки, с которыми легче бороться с помощью блочного и сверточного кодирования.

Существуют специальные коды, корректирующие пакетные ошибки, однако на практике чаще используют **перемежение/деперемежение** совместно с традиционными кодами. **Пакеты (группы) ошибок** – ошибки, носящие взаимозависимый характер.

Идея **перемежения/деперемежения** состоит в следующем. Если биты каждого кодового слова Хn передаются не в обычной последовательности, а через интервалы, превышающие ожидаемую **длину пакета ошибок** (в промежутки между битами одного слова вставляются биты других кодовых слов), то при возникновении такого типа ошибки обратная перемежению операция – деперемежение – разнесет («размажет») группу ошибок по всей совокупности кодовых слов, составляющих данное сообщение.

**Длина пакета** в нашем случае – это число рядом расположенных ошибочных битов. Например, если Хn = 1011111, а Yn = 1000011, то длина пакета ошибок составляет 3 бита.

Предложено много алгоритмов перемежения/деперемежения. Наиболее простыми являются блочные. При блочном перемежении входные биты делятся на блоки, которые последовательно записываются в строки некоторой таблицы, приведенной для наглядности на рис. 7.1



Передаваемая последовательность (1010110011…) делится на блоки по 5 битов. Каждый блок записывается в отдельную строку таблицы по порядку. Сообщение для передачи или хранения формируется при считывании символов из таблицы по столбцам: 11010000001… .

Деперемежение производится в обратной последовательности. Для данного примера **глубина перемежения** (разница между позициями одного и того же символа до и после перемежения) равна 4: например, второй символ после перемежения станет шестым. Особенностью является неизменная позиция первого символа.

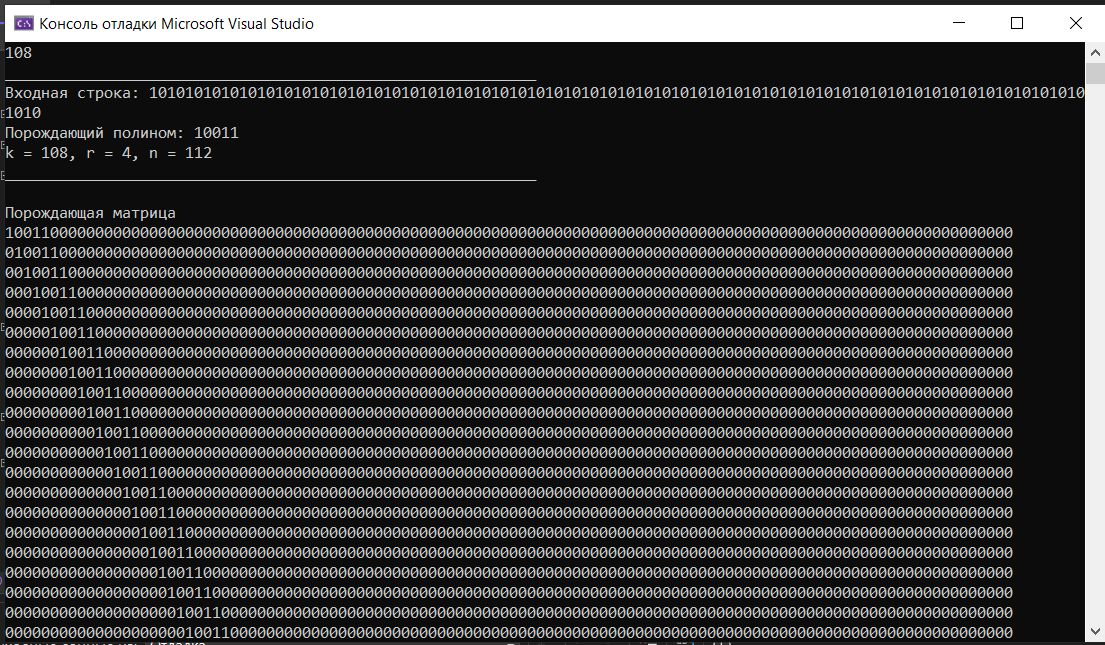
В общем случае **выбор глубины перемежения зависит от двух факторов**. С одной стороны, чем больше расстояние между соседними символами, тем большей длины пакет ошибок может быть исправлен. С другой стороны, чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше задержка сигнала. Для борьбы с длинными пакетами ошибок желательно увеличивать размеры таблицы. Однако это приводит к увеличению задержки в отправке и декодировании сообщения.

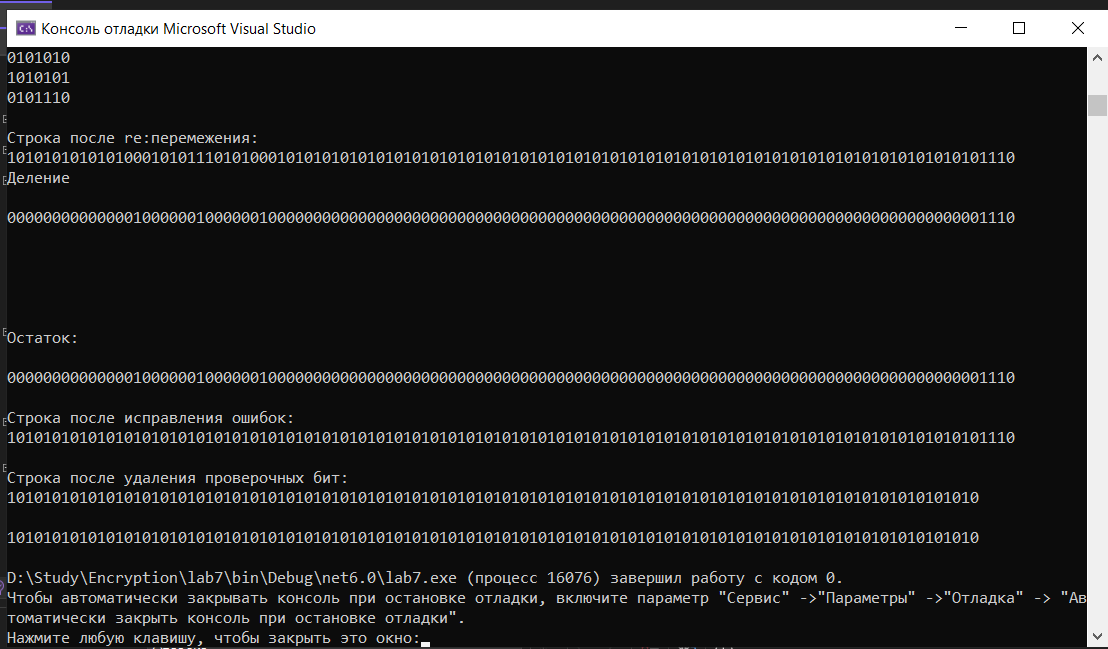
**Практическое задание**

1. Необходимо разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель. По желанию студент может использовать иной. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы. За основу разрабатываемого приложения может быть взято приложение из выполненной лабораторной работы, соответствующей заданному корректирующему коду.
2. Местоположение заданной группы ошибок выбирается (генерируется) случайным образом. Необходимо для группы ошибок каждой длины сгенерировать 30−40 случайных ситуаций. После деперемежения и исправления ошибок в сообщении сравнить.

|  |
| --- |
| using System;  namespace lab7  {  class Program  {  static void Main(string[] args)  {  string Xk = "101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010101010";  Console.WriteLine(Xk.Length.ToString());  string Xr = "10011";  int kc = Xk.Length;  int rc = 4;  int nc = 112;  int errorc;  int[] masXk = new int[kc];  StrInMas(masXk, Xk);  int[] masXr = new int[Xr.Length];  StrInMas(masXr, Xr);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine("Входная строка: " + Xk);  Console.WriteLine("Порождающий полином: " + Xr);  Console.WriteLine("k = {0}, r = {1}, n = {2}", kc, rc, nc);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  int[,] generationMatrix = new int[kc, nc];  CreateGenerationMatrix(generationMatrix, masXr, kc, nc);  Console.WriteLine("\nПорождающая матрица");  OutMatrix(generationMatrix, kc, nc);  CreateCanonicalMatrix(generationMatrix, kc, nc);  Console.WriteLine("\nКаноническая матрица");  OutMatrix(generationMatrix, kc, nc);  int[,] checkMatrixc = new int[nc, rc];  CreateCheckMatrix(checkMatrixc, generationMatrix, kc, nc);  Console.WriteLine("\nПроверочная матрица");  OutMatrix(checkMatrixc, nc, rc);  //6.2  int[] masXn = new int[nc];  Shift(masXn, masXk, rc);  //2.  Console.WriteLine("\nДеление");  SearchResidue(masXn, masXr);  Console.WriteLine("Остаток (S-синдром):");  OutMass(masXn);  Console.WriteLine("\n");  Console.WriteLine("Итоговая строка:");  Shift(masXn, masXk, rc);  OutMass(masXn);  Console.WriteLine();  int[] save = new int[nc];  int prop = 0;  foreach (int menuitem in masXn)  {  save[prop++] = menuitem;  }  Alternation(masXn);  Console.WriteLine("\nСтрока после перемежения: ");  OutMas(masXn);  int error;  int errorLenght;  try  {  Console.WriteLine("\n\"Место ошибки 2");  error = 2/\*Convert.ToInt32(Console.ReadLine())\*/;  Console.WriteLine("Длина ошибки 3");  errorLenght = 3/\*Convert.ToInt32(Console.ReadLine())\*/;  for (int i = error; i < (error + errorLenght); i++)  {  masXn[i] = (masXn[i] + 1) % 2;  }  }  catch { }  Console.WriteLine("\nСтрока с ошибками: ");  OutMas(masXn);  ReAlternation(masXn);  Console.WriteLine("\nСтрока после re:перемежения: ");  OutMas(masXn);  SearchError(masXn, masXk, checkMatrixc, rc);  Console.WriteLine("\n\nСтрока после исправления ошибок: ");  OutMas(save);  int[] masK = new int[90];  int[] masK2 = new int[90];  //RemoveCheckBits(masK2, masXn, checkMatrixc);  Console.WriteLine("\n\nСтрока после удаления проверочных бит: ");  Console.WriteLine(Xk);  Console.WriteLine("");  OutMassive(Xk);  }  static int[] Alternation(int[] masN)  {  int[,] matrix = new int[16, 7];  //Получение матрицы  for (int i = 0, m = 0; i < 16; i++)  {  for (int j = 0; j < 7; j++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, 16, 7);  //Перемежение  for (int i = 0, m = 0; i < 7; i++)  {  for (int j = 0; j < 16; j++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }  static int[] ReAlternation(int[] masN)  {  //int r = HemmingLength(k);  int k = 16;  int n = 7;  int[,] matrix = new int[k, n];  //Получение матрицы  for (int j = 0, m = 0; j < n; j++)  {  for (int i = 0; i < k; i++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, k, n);  //RE:Перемежение  for (int j = 0, m = 0; j < k; j++)  {  for (int i = 0; i < n; i++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }  //Поиск синдрома  static int[] Sindrom(int[,] CheckMatrix, int[] mas, int k)  {  int r = HemmingLength(k);  int n = r + k;  int[] sindrom = new int[r];  for (int i = 0, l = 0; i < r; i++, l = 0)  {  for (int j = 0; j < k; j++)  {  if (CheckMatrix[j, i] == 1 && mas[j] == 1) l++;  else sindrom[i] = 0;  }  if (l % 2 == 1) sindrom[i] = 1;  else sindrom[i] = 0;  }  for (int i = 0; i < r; i++)  {  mas[i + k] = sindrom[i];  }  return mas;  }  //Считаем r (кол-во пров. симв.)  static int HemmingLength(int k)  {  int r = (int)(Math.Log(k, 2) + 1.99f  );  return r;  }  static void OutMas(int[] mas)  {  for (int i = 0; i < mas.Length; i++)  {  Console.Write(mas[i]);  }  }  //вывод матрицы  public static int[] SearchError(int[] masXn, int[] masXk, int[,] checkMatrix, int r)  {  int n = masXn.Length;  int k = n - r;  int[] masXnSecond = new int[n];  for (int i = 0; i < n; i++)  {  masXnSecond[i] = masXn[i];  }  Console.WriteLine("\nДеление");  SearchResidue(masXnSecond, masXk);  Console.WriteLine();  Console.WriteLine("\n");  Console.WriteLine("\nОстаток:");  OutMass(masXnSecond);  for (int i = 0; i < n; i++)  {  int coincidence = 0;  for (int j = 0; j < r; j++)  {  if (checkMatrix[i, j] == masXnSecond[k + j])  {  coincidence++;  }  }  if (coincidence == r)  {  masXn[i] = (masXn[i] + 1) % 2;  break;  }  }  return masXn;  }  public static int[] SearchResidue(int[] masXn, int[] masXr)  {  int end = masXn.Length - masXr.Length + 1;  for (int i = 0; i < end; i++)  {  if (masXn[i] == 1)  {  AddingMasMod2(masXn, masXr, i);  OutMass(masXn);  }  }  Console.WriteLine("\n");  return masXn;  }  //Сложение массивов по модулю 2 с опр. позиции  public static int[] AddingMasMod2(int[] mas1, int[] mas2, int pos)  {  int end = pos + mas2.Length;  for (int i = pos; i < end; i++)  {  mas1[i] = (mas1[i] + mas2[i - pos]) % 2;  }  return mas1;  }  //Смещение на массива r  public static int[] Shift(int[] shiftMas, int[] mas, int r)  {  for (int i = 0; i < mas.Length; i++)  {  shiftMas[i] = mas[i];  }  return shiftMas;  }  //Преобразование сторки в массив  public static int[] StrInMas(int[] mas, string str)  {  for (int i = 0; i < str.Length; i++)  {  if (str[i] == 49)  mas[i] = 1;  else mas[i] = 0;  }  return mas;  }  //Создание Порождающей матрицы  static int[,] CreateGenerationMatrix(int[,] generationMatrix, int[] mas, int k, int n)  {  //Заполняем первую строку в проверочной матрице  for (int i = 0; i < n; i++)  {  if (i < mas.Length)  {  generationMatrix[0, i] = mas[i];  }  else  {  generationMatrix[0, i] = 0;  }  }  //Сдвигаем каждую строки вправо от предыдущей  for (int i = 1; i < k; i++)  {  for (int j = 0; j < n - 1; j++)  {  generationMatrix[i, j + 1] = generationMatrix[i - 1, j];  }  generationMatrix[i, 0] = generationMatrix[i - 1, n - 1];  }  return generationMatrix;  }  //Приведение порождающей матрицы к каноническому виду  static int[,] CreateCanonicalMatrix(int[,] generationMatrix, int k, int n)  {  //Перебираем строки для преведению к каноническому виду  for (int i = 0; i < k; i++)  {  int i2 = i + 1;  //Перебираем элементы строки, но только до k-элемента  for (int j = i + 1; j < k; j++)  {  //если мы нашли единицу в строке, то...  if (generationMatrix[i, j] == 1)  {  //перебираем этот столбец, пока не найдем единицу  for (; i2 < k; i2++)  {  bool repeat = false;  //Если нашли, то складываем обе строки  if (generationMatrix[i2, j] == 1)  {  for (int j2 = j - 1; j2 > 0; j2--)  {  //Проверяем, есть ли до этой 1 еще 1, если есть то эту строку пропускаем  if (generationMatrix[i2, j2] == 1)  {  repeat = true;  }  }  if (repeat)  continue;  //Console.WriteLine(i + " " + i2);  AddingLinesMatrixMod2(generationMatrix, i, i2, n);  i2++;  break;  }  }  }  }  }  return generationMatrix;  }  //Преобразование канонической матрицы в проверочную  static int[,] CreateCheckMatrix(int[,] checkMatrix, int[,] generationMatrix, int k, int n)  {  int r = n - k;  for (int i = 0; i < k; i++)  {  for (int j = 0; j < r; j++)  {  checkMatrix[i, j] = generationMatrix[i, k + j];  }  }  for (int i = k; i < n; i++)  {  for (int j = 0; j < r; j++)  {  if (j == i - k)  {  checkMatrix[i, j] = 1;  }  else  {  checkMatrix[i, j] = 0;  }  }  }  return checkMatrix;  }  //Сложение строк матрицы  public static int[,] AddingLinesMatrixMod2(int[,] matrix, int str1, int str2, int  lengthString)  {  //Console.WriteLine(str1 + " и " + str2);  for (int i = 0; i < lengthString; i++)  {  matrix[str1, i] = (matrix[str1, i] + matrix[str2, i]) % 2;  }  return matrix;  }  //вывод матрицы  public static void OutMatrix(int[,] matrix, int k, int n)  {  for (int i = 0; i < k; i++)  {  for (int j = 0; j < n; j++)  {  Console.Write(matrix[i, j]);  //if (j + 1 == k) Console.Write("|");  }  Console.WriteLine();  }  }  static void OutMassive(string Xk)  {  Console.WriteLine(Xk);  }  //вывод одномерного массива  public static void OutMass(int[] mas)  {  Console.WriteLine();  for (int i = 0; i < mas.Length; i++)  {  //if (i == k) Console.Write("|");  Console.Write(mas[i]);  }  //Console.WriteLine("\n");  }  }  } |

*Листинг 1* – код файла Program.cs





*Рис. 7.1* **-** Результат выполнения программы

**Ответы на вопросы:**

1. **Пояснить назначение и особенности использования технологии перемежения/деперемежения данных в ИС.**

При такой технологии уменьшается влияние пакетных ошибок. Данные перед передачей по каналу связи, переставляются в заданном порядке, а в приемной части восстанавливается исходный порядок. При этом пакетная ошибка, возникшая в канале связи, превращается в набор рассредоточенных во времени одиночных ошибок, которые проще обнаруживаются и исправляются с помощью кодов, исправляющих ошибки.

1. **Что такое группирующиеся ошибки и с чем, по Вашему мнению, связано их появление в каналах передачи данных, в полупроводниковой памяти, на магнитных носителях?**

**Пакеты (группы) ошибок** – ошибки, носящие взаимозависимый характер**.**

Выделяют две основные *причины возникновения ошибок* при передаче информации в сетях:

- сбои в какой-то части оборудования сети или возникновение неблагоприятных объективных событий в сети (например, коллизий при использовании метода случайного доступа в сеть). Как правило, система передачи данных готова к такого рода проявлениям и устраняет их с помощью планово предусмотренных средств;

- помехи, вызванные внешними источниками и атмосферными явлениями. Помехи - это электрические возмущения, возникающие в самой аппаратуре или попадающие в нее извне. Наиболее распространенными являются флуктуационные (случайные) помехи. Они представляют собой последовательность импульсов, имеющих случайную амплитуду и следующих друг за другом через различные промежутки времени. Примерами таких помех могут быть атмосферные и индустриальные помехи, которые обычно проявляются в виде одиночных импульсов малой длительности и большой амплитуды. Возможны и сосредоточенные помехи в виде синусоидальных колебаний. К ним относятся сигналы от посторонних радиостанций, излучения генераторов высокой частоты. Встречаются и смешанные помехи. В приемнике помехи могут настолько ослабить информационный сигнал, что он либо вообще не будет обнаружен, либо искажен так, что “единица” может перейти в “нуль” и наоборот.

1. **Что такое глубина перемежения и как она влияет на эффективность перемежения/деперемежения данных?**

**Глубина перемежения** - разница между позициями одного и того же символа до и после перемежения.

Чем больше расстояние между соседними символами, тем большей длины пакет ошибок может быть исправлен.

Чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше задержка сигнала. Для борьбы с длинными пакетами ошибок желательно увеличивать размеры таблицы. Однако это приводит к увеличению задержки в отправке и декодировании сообщения.

**Вывод:** приобрел практические навыки использования методов перемежения/деперемежения двоичных данных в информационных системах.

# **Лабораторная работа №8**

**Тема «Сжатие/распаковка данных методом Барроуза - Уилера»**

**Цель:** приобретение практических навыков использования метода Барроуза − Уилера для сжатия/распаковки данных.

**Теоретические сведения**

Сжатие информации является одним из способов ее кодирования. В основе сжатия данных, как одна из первопричин, лежит **избыточность**.

**Основная цель сжатия** – обеспечить более компактное представление данных, вырабатываемых источником, т. е. уменьшить физический объем сообщений, генерируемых источником, и сократить время его передачи (читай – стоимость) по каналам связи. **Фундаментальная теорема К. Шеннона о кодировании информации** утверждает, что «стоимость кодирования всегда не меньше энтропии источника, хотя может быть сколь угодно близка к ней». Поэтому для любого алгоритма сжатия всегда имеется некоторый предел степени (или эффективности) сжатия, определяемый **энтропией входного потока** (или сжимаемого сообщения).

Все алгоритмы сжатия преобразуют входной поток данных, минимальной единицей которых является бит, а максимальной – байт или несколько байт. Основными техническими характеристиками процессов сжатия и результатов их работы являются: • степень сжатия (англ. compress rating), или отношение R (англ. ratio) объемов исходного (до сжатия, Vдс) и результирующего (после сжатия, Vпс) потоков данных (сообщений); • скорость сжатия − время, затрачиваемое на сжатие некоторого объема информации входного потока до получения из него эквивалентного выходного потока; • качество сжатия − величина, показывающая, насколько сильно сжат выходной поток при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Степень сжатия R обычно оценивается следующим образом:

R1 = (Vпс / Vдс) · 100%,

R2 = (Vдс − Vпс) / Vдс= (1 − R1) · 100%.

Первое отношение показывает, какую часть объема сообщения (файла) до сжатия занимает сообщение (файл) после сжатия; второе отношение выражает основной физический смысл сжатия и показывает степень сжатия.

Существуют различные подходы к реализации сжатия информации. Они отличаются математической базой, уровнем сложности (простоты) практической реализации, форматом кодируемого потока данных, степенью соответствия сжимаемых и распакованных данных.

По критерию, связанному с характером или форматом данных или степенью соответствия сжимаемых данных распакованным, все методы сжатия разделяют на два класса: обратимое и необратимое сжатие, или иначе: сжатие без потерь и сжатие с частичной потерей информации.

Понятно, что недопустимы никакие потери при упаковке текстовых документов, кодов компьютерных программ, файлов баз данных.

**Метод Барроуза – Уилера.**

**BWT-преобразование (англ. Burrows-Wheeler Transform)** – техника сжатия информации (в особенности текстов), основанная на преобразовании, открытом в 1983 г. BWT не сжимает данные в классическом понимании процесса, но преобразует блок данных в формат, исключительно подходящий для сжатия.

BWT оперирует сразу целым блоком данных, который выделяется из входного потока (сообщения).

**Прямое преобразование (формально – сжатие) выполняется в 4 этапа:**

1. выделяется блок данных (строка длиной k символов некоторого алфавита мощностью N), который обозначим символом М;
2. составляется таблица W1 размером k×k всех циклических сдвигов входной строки M: W1 = (M);
3. производится лексикографическая (в алфавитном порядке) сортировка строк таблицы W1, в результате чего получается таблица W2 того же размера;
4. в качестве выходной строки (обозначим ее BWT(М), z) выбирается последний столбец (Мk) таблицы W2 преобразования и номер строки z, совпадающей с исходной строкой М. Как видим, выходная строка (сжатое сообщение) всегда по объему превышает входную.

**Практическое задание**

1. Разработать авторское приложение в соответствии с целью лабораторной работы. Входной блок данных может иметь произвольную длину.
2. С помощью приложения выполнить прямое и обратное преобразования 3 отдельных блоков данных, состоящих: а) из собственного имени (можно краткий вариант записи); б) собственной фамилии; в) варианта в соответствии с таблицей ниже.

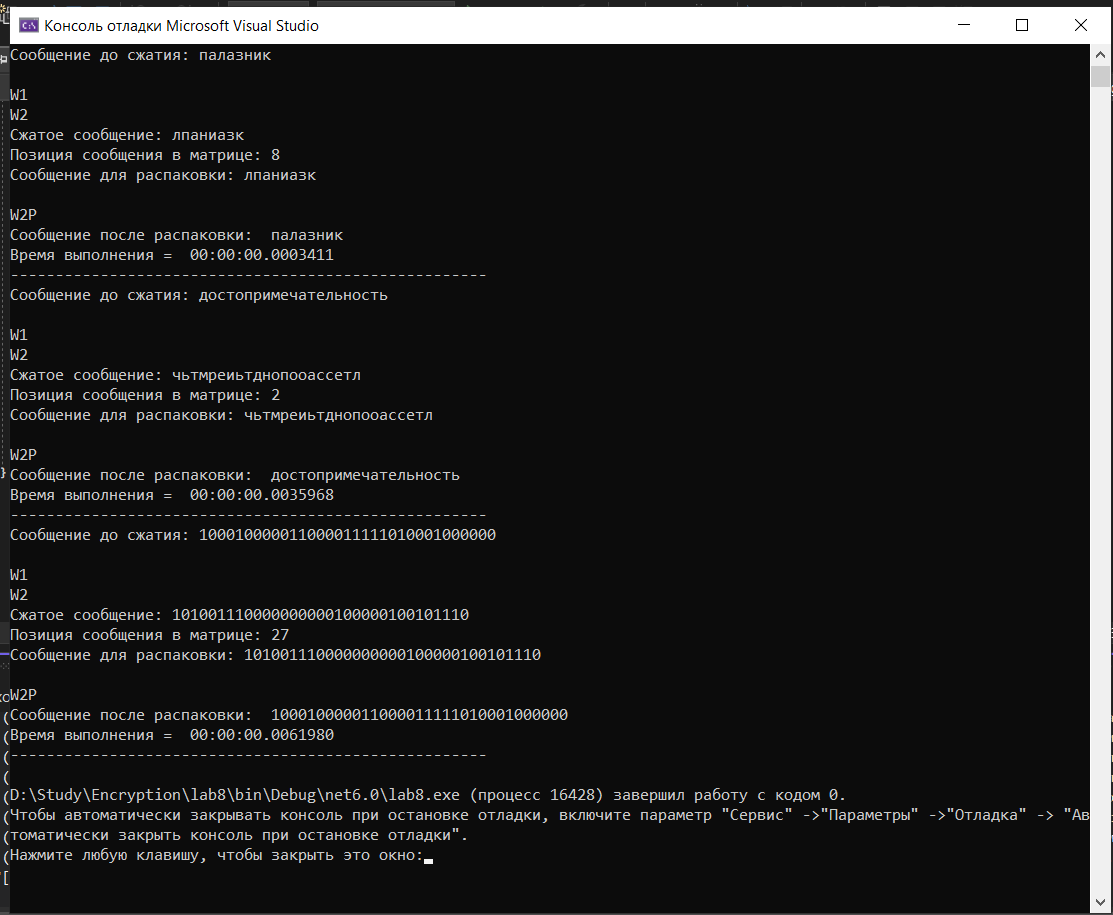
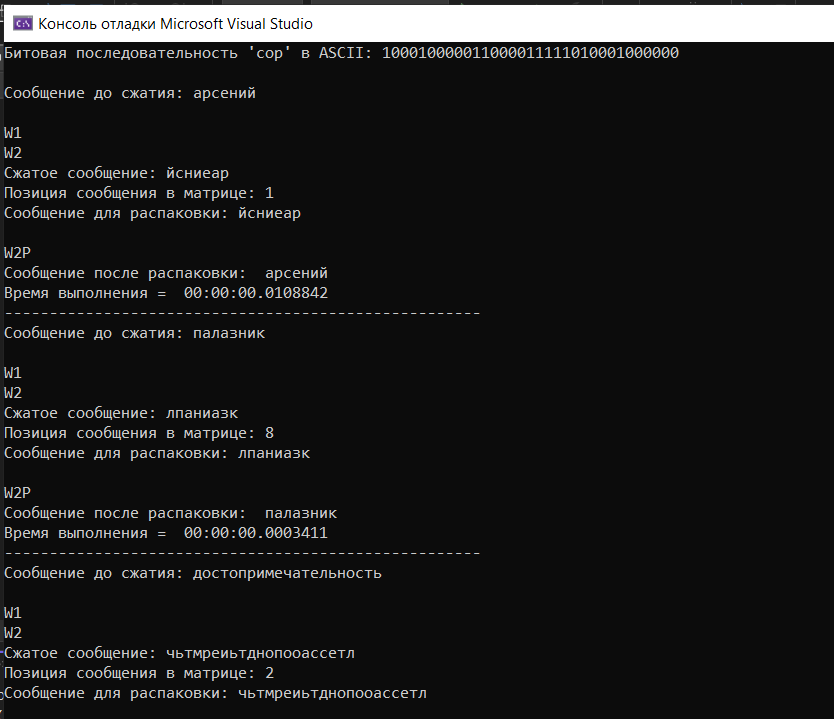
Можно использовать любой из известных методов сортировки символов массива.

Выполнить качественный сравнительный анализ длительности процессов прямого и обратного преобразований в зависимости от длины блока данных.

1. Перевести первые 3 символа из блока данных, указанного в варианте таблицы, в бинарную последовательность в соответствии с кодами ASCII. Выполнить прямое и обратное преобразование. Оценить время прямого и обратного преобразований.
2. Результаты оформить в виде отчета по установленным правилам.

|  |
| --- |
| 1. using System; 2. using System.Linq; 3. using System.Text; 4. using System.Diagnostics; 5. namespace \_8 6. { 7. class Program 8. { 9. static string[] CreateMatrixByString(string message) 10. { 11. string[] messageMatrix = new string[message.Length]; 12. for (int i = 0; i < message.Length; i++) 13. { 14. messageMatrix[i] = message; 15. message = message.Substring(1) + message[0]; 16. } 17. return messageMatrix; 18. } 19. static void ShowMatrix(string[] matrix) 20. { 21. foreach (var row in matrix) 22. { 23. Console.WriteLine(row); 24. } 25. Console.WriteLine(); 26. } 27. static string GetLastMatrixColumn(string[] matrix) 28. { 29. string lastColumn = ""; 30. foreach (var row in matrix) 31. { 32. lastColumn += row[row.Length - 1]; 33. } 34. return lastColumn; 35. } 36. static int GetInitMessagePlace(string initMessage, string[] matrix) 37. { 38. int initMessagePlace = -1; 39. for (int i = 0; i < matrix.Length; i++) 40. { 41. if (matrix[i] == initMessage) 42. { 43. return i; 44. } 45. } 46. return initMessagePlace; 47. } 48. static string[] AddMessageToMatrixFromLeft(string message, string[] matrix) 49. { 50. for (int i = 0; i < matrix.Length; i++) 51. { 52. matrix[i] = message[i] + matrix[i]; 53. } 54. return matrix; 55. } 56. static string[] SortMatrix(string[] matrix) 57. { 58. return (matrix.OrderBy(x => x).ToArray()); 59. } 60. static string[] CreateDecodingMatrix(string message) 61. { 62. string[] messageMatrix = new string[message.Length]; 63. for (int i = 0; i < message.Length; i++) 64. { 65. messageMatrix = AddMessageToMatrixFromLeft(message, messageMatrix); 66. ShowMatrix(messageMatrix); 67. messageMatrix = SortMatrix(messageMatrix); 68. } 69. return messageMatrix; 70. } 71. static void Main(string[] args) 72. { 73. string initialMessageBytes = "сор"; 74. byte[] bytes = Encoding.ASCII.GetBytes(initialMessageBytes); 75. int integ; 76. string str = ""; 77. for (int i = 0; i < bytes.Length; i++) 78. { 79. integ = bytes[0]; 80. str += Convert.ToString(integ, 2); 81. } 82. str = "100010000011000011111010001000000"; 83. Console.WriteLine("Битовая последовательность 'сор' в ASCII: " + str); 84. Console.WriteLine(); 85. string[] initMessages = new string[] { "арсений", "палазник", "достопримечательность", str }; 86. foreach (string initMessage in initMessages) 87. { 88. Stopwatch sw = new Stopwatch(); 89. sw.Start(); 90. Console.WriteLine("Сообщение до сжатия: " + initMessage); 91. Console.WriteLine(); 92. //Encoding 93. string[] W1 = CreateMatrixByString(initMessage); 94. Console.WriteLine("W1"); 95. ShowMatrix(W1); 96. string[] W2 = (W1.OrderBy(x => x).ToArray()); 97. Console.WriteLine("W2"); 98. ShowMatrix(W2); 99. string encodedMessage = GetLastMatrixColumn(W2) + (GetInitMessagePlace(initMessage, W2)); 100. Console.WriteLine("Сжатое сообщение: " + GetLastMatrixColumn(W2)); 101. Console.WriteLine("Позиция сообщения в матрице: " + (GetInitMessagePlace(initMessage, W2) + 1)); 102. //Decoding 103. string[] W2Dec = new string[encodedMessage.Length - (encodedMessage.Length - initMessage.Length)]; 104. string gettedMessage = encodedMessage.Substring(0, encodedMessage.Length - (encodedMessage.Length - initMessage.Length)); 105. Console.WriteLine("Сообщение для распаковки: " + gettedMessage); 106. Console.WriteLine(); 107. W2Dec = CreateDecodingMatrix(gettedMessage); 108. Console.WriteLine("W2Р"); 109. ShowMatrix(W2Dec); 110. int numberOfInitialMessage = Int32.Parse((encodedMessage.Substring(initMessage.Length, (encodedMessage.Length - initMessage.Length)))); 111. Console.WriteLine("Сообщение после распаковки: " + W2Dec[numberOfInitialMessage]); 112. sw.Stop(); 113. Console.WriteLine("Время выполнения = {0}", sw.Elapsed); 114. Console.WriteLine("-----------------------------------------------------"); 115. } 116. } 117. } 118. } |

*Листинг 1* **–** код файла Program.cs



*Рис. 8.1* – Результат выполнения программы

**Ответы на вопросы:**

1. **Сформулировать цели применения методов сжатия и архивирования данных.**

Обеспечить более компактное представление данных, вырабатываемых источником, т. е. уменьшить физический объем сообщений, генерируемых источником, и сократить время его передачи (читай – стоимость) по каналам связи. Фундаментальная теорема К. Шеннона о кодировании информации утверждает, что «стоимость кодирования всегда не меньше энтропии источника, хотя может быть сколь угодно близка к ней». Поэтому для любого алгоритма сжатия всегда имеется некоторый предел степени (или эффективности) сжатия, определяемый энтропией входного потока (или сжимаемого сообщения).

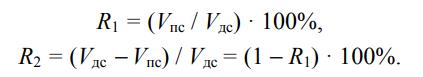
1. **Охарактеризовать основные технические характеристики процессов сжатия/распаковки и результатов.**

• степень сжатия (англ. compress rating), или отношение R (англ. ratio) объемов исходного (до сжатия, Vдс) и результирующего (после сжатия, Vпс) потоков данных (сообщений);

• скорость сжатия − время, затрачиваемое на сжатие некоторого объема информации входного потока до получения из него эквивалентного выходного потока;

• качество сжатия − величина, показывающая, насколько сильно сжат выходной поток при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

1. **Как можно рассчитать степень сжатия файла?**

****

1. **В каких случаях и почему применяется сжатие без потерь, а в каких – с потерей информации?**

Сжатие **с потерей информации** реализуется на основе таких известные форматов данных и алгоритмов сжатия, как JPEG и MPEG. Алгоритм JPEG используется при сжатии фотоизображений. Алгоритмы MPEG используют при сжатии видео и музыки. Методы и алгоритмы сжатия с потерей информации применяют обычно для решения так называемых потребительских задач. Это значит, например, что если фотография передается для просмотра, а музыка для воспроизведения, то подобные алгоритмы применять можно. Если же они передаются для дальнейшей обработки, например для редактирования, то никакая потеря информации в исходном файле недопустима. Считают, что на фотографических иллюстрациях, предназначенных для воспроизведения на экране, потеря 5% информации не критична, а в некоторых случаях можно допустить и 20−25% уровень потерь.

1. **В чем сущность символ-ориентированных методов сжатия? Какие известные Вам методы относятся к этому классу?**

Сущность состоит в последовательном анализе сжим. информации с целью поиска повторяющихся или проаналированных ранее в данном документе послед-тей и замене таких послед-тей на более короткие. Методы, основанные на данном подходе, не рассматривают статистические модели, они также не используют коды переменной длины.

1. **Составить алгоритмы сжатия/распаковки данных методом Барроуза − Уилера.**

Прямое преобразование (формально – сжатие) выполняется в 4 этапа:

1) выделяется блок данных (строка длиной k символов некоторого алфавита мощностью N), который обозначим символом М;

2) составляется таблица W1 размером k×k всех циклических сдвигов входной строки M: W1 = (M);

3) производится лексикографическая (в алфавитном порядке) сортировка строк таблицы W1, в результате чего получается таблица W2 того же размера;

4) в качестве выходной строки (обозначим ее BWT(М), z) выбирается последний столбец (Мk) таблицы W2 преобразования и номер строки z, совпадающей с исходной строкой М.

Итак, входной для обратного преобразования является информация вида BWT(М), i. Это преобразование заключается в выполнении k одинаковых шагов, каждый из которых состоит из 2 операций, с целью воссоздания матрицы W2:

1) в крайний справа пустой столбец матрицы записывается последовательность символов Мk;

2) производится лексикографическая сортировка столбцов заполненной части воссоздаваемой матрицы.

1. **Что поменяется, если процедуру формирования матрицы W1 строить на основе циклических сдвигов вправо?**

Ничего.

1. **В каких известных архиваторах используется метод Барроуза − Уилера?**

BWT используется в архиваторе bzip2.

1. **Как вы понимаете рекуррентность (рекурсивность) преобразования по методу Барроуза − Уилера?**

Объект называется рекурсивным, если он содержит сам себя или определен с помощью самого себя. Рекурсия – способ организации вычислительного процесса, при котором процедура или функция в ходе выполнения составляющих её операторов обращается сома к себе.

**Вывод:** приобрел практические навыки использования метода Барроуза − Уилера для сжатия/распаковки данных.

# **Лабораторная работа №9**

**Тема «Сжатие/распаковка данных на основе статических методов»**

**Цель:** приобретение практических навыков использования статистических методов Шеннона − Фано и Хаффмана (ShannonFano and Huffman coding) для сжатия/распаковки данных.

**Теоретические сведения**

Статистические алгоритмы позволяют создавать более короткие коды для часто встречающихся и более длинные – для редко встречающихся символов алфавита или конкретного сообщения. В первом случае метод считается статическим статистическим, во втором – динамическим статистическим: вероятностные свойства символов подсчитываются для конкретного сообщения или потока данных.

Частота или вероятность появления того или иного символа алфавита в произвольном сообщении, лежащая в основе алгоритмов, дали название этим алгоритмам и соответствующим методам. Иногда эти методы называют также префиксными. К примеру, если имеется некоторый код, который записывается как Х1 = А1А2, и другой код – Х2 = А1, то говорят, что Х2 является префиксом Х1. Или если Х1 = 1010, а Х2 = 10101100, то Х2 также является префиксом Х1. Таким образом, использование описываемых методов предусматривает создание кодовой таблицы (подобно кодам ASCII или base64). Формально процедура сжатия (прямое преобразование) состоит в подстановке соответствующего бинарного кода вместо символа исходного алфавита и наоборот – при обратном преобразовании. Методы относятся к классу «сжатие без потерь». Различие между двумя рассматриваемыми методами состоит лишь в особенностях формирования таблицы бинарных кодов. При формировании этой таблицы для обоих методов можно воспользоваться статистическими свойствами алфавитов, полученными при выполнении лабораторной работы № 2.

**Код Шеннона – Фано** не является оптимальным (обеспечивает минимальную избыточность) в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона – Фано, и все они могут дать различные результаты. Итак, необходимо выполнить следующие действия: 1) подсчитать вероятностные параметры символов алфавита А = {ai} (реализуется статическая версия алгоритма); 2) отсортировать – обычно в порядке убывания (невозрастания, т. е. могут иметь место повторяющиеся значения) вероятностей р(аi); р(аi) – вероятность появления в сжимаемом сообщении на произвольной позиции символа аi алфавита, т. е. создать таблицу символов алфавита, на основе которого генерируется сжимаемое сообщение; 3) каждому символу отсортированного множества поставить в соответствие бинарный код, для чего это множество (таблица) символов делится на две группы таким образом, чтобы каждая из групп имела приблизительно одинаковую суммарную частоту (вероятность). Очевидно, на первом шаге такая суммарная вероятность в каждой из групп должна быть максимально близка к 0,5. Первому из полученных подмножеств устанавливается первый символ бинарного кода: 0, второй − 1 (или наоборот). Для вычисления следующих битов кодов данная процедура повторяется рекурсивно для каждого из полученных на текущем шаге подмножеств, в котором содержится больше одного символа. Получим таблицу, в которой длина кодовых комбинаций меняется от минимального (lmin) до максимального (lmax) значений.

Алгоритм прямого преобразования: необходимо выполнить одну операцию: заменить символы входного сообщения соответствующими бинарными кодами. Алгоритм обратного преобразования: на входе – сообщение в виде бинарной последовательности.

**Метод Хаффмана.**

Метод основан на алгоритме оптимального префиксного кодирования алфавита: исходный алгоритм Хаффмана является оптимальным для посимвольного кодирования с известным входным распределением вероятностей, т. е. для отдельного кодирования несвязанных символов в таком потоке данных. Отличается от метода Шеннона – Фано лишь в части кодирования символов исходного алфавита. В данном случае бинарные коды создаются на основе дерева, ветви которого обозначаются бинарными символами. Бинарным кодом символа исходного алфавита будет последовательность обозначений ветвей дерева от корня до листа, соответствующего этому символу. В основе бинарного кода лежит следующее положение.

**Для любого заданного алфавита (источника) с N > 2 символами существует оптимальный двоичный код, в котором два наименее вероятных символа (слова) имеют одну и ту же длину и отличаются лишь последним битом**

Построение дерева начинается с сортирования символов исходного алфавита в порядке убывания (невозрастания). Далее выбираются два символа (ai, aj) с наименьшими вероятностями (р(ai), р(aj)) и объединяются в узел. Ветви этого узла обозначаются «1» и «0». Этот узел рассматривается далее как новый, виртуальный символ (aij), которому будет соответствовать вероятность р(aij) = р(ai) + р(aj). Такой виртуальный символ будет рассматриваться далее наравне с остальными символами исходного алфавита. Два его потомка из дальнейшего рассмотрения исключаются. Создаются новые узлы дерева по тому же принципу. Корень дерева образуют два символа с наибольшими вероятностями.

**Практическое задание**

1. Разработать авторское приложение в соответствии с целью лабораторной работы.
2. С помощью приложения выполнить прямое и обратное преобразования сообщения, состоящего из собственных имени и фамилии. Можно использовать любой из известных методов сортировки символов массива. Метод кодировки (Шеннона − Фано, Хаффмана) использовать по указанию преподавателя.

При этом таблица отсортированных символов строится:

а) на основе данных, полученных в лабораторной работе № 2;

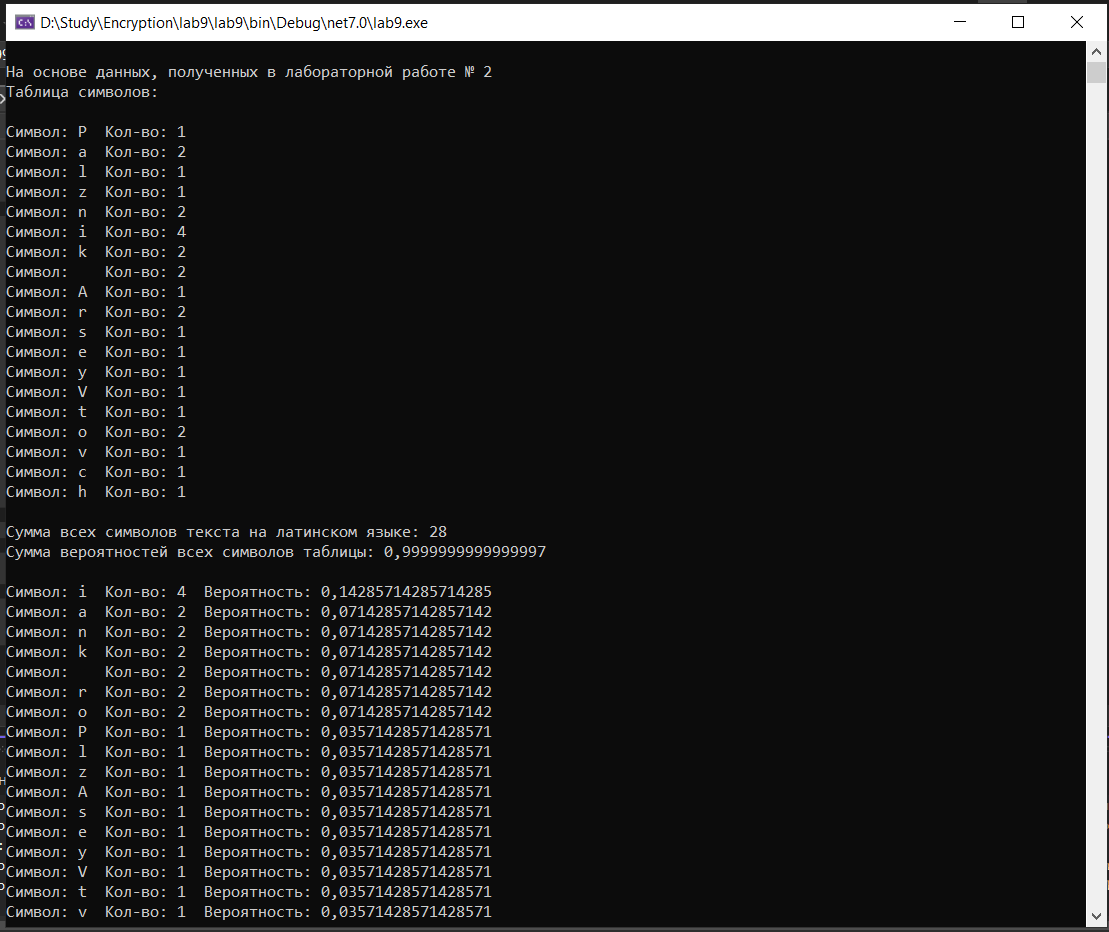
б) динамически, на основе анализа сжимаемого сообщения.

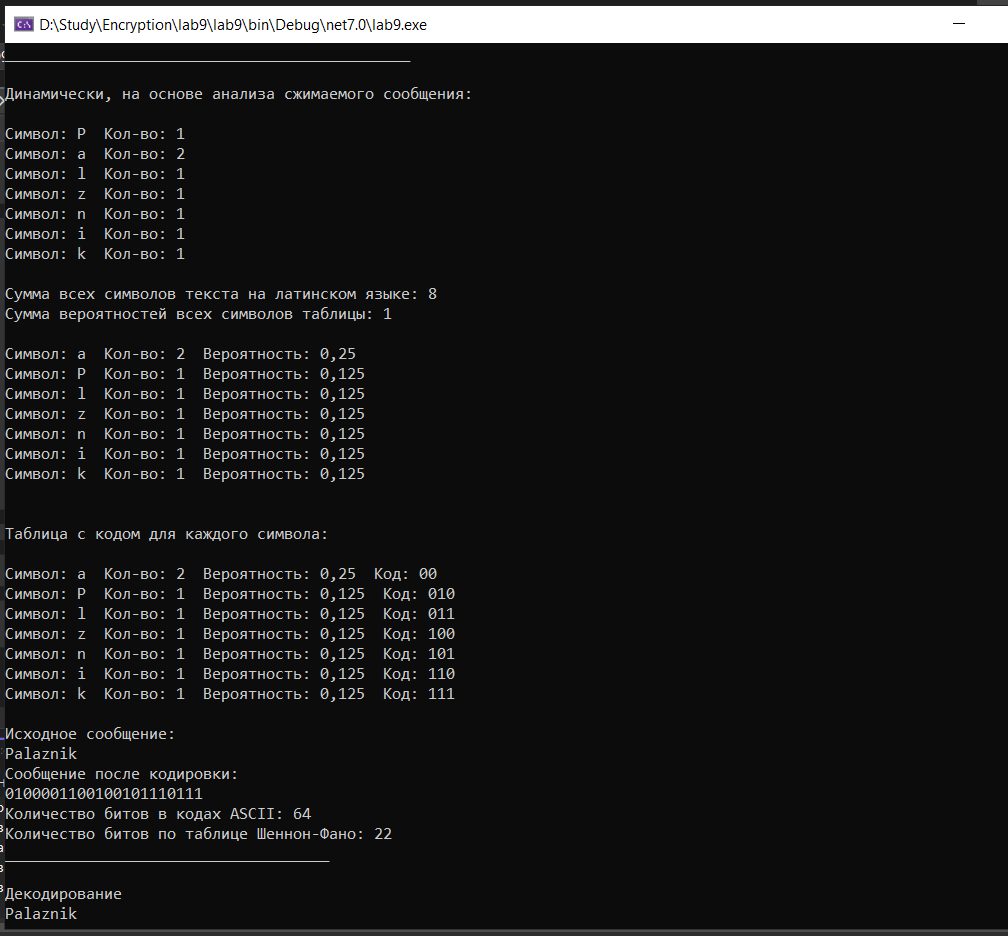
3. Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.

4. Результаты оформить в виде отчета по установленным правилам

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Text;  namespace lab9  {  public class ShannonFanoSymbol  {  public char Symbol { get; set; }  public int Count { get; set; }  public double Probability { get; set; }  public string Code { get; set; }  public ShannonFanoSymbol(char symbol, int count, double probability, string code)  {  Symbol = symbol;  Count = count;  Probability = probability;  Code = code;  }  public static List<ShannonFanoSymbol> AddSymbols(List<ShannonFanoSymbol> symbols, string line)  {  foreach (var character in line)  {  var existingSymbol = symbols.FirstOrDefault(x => x.Symbol == character);  if (existingSymbol == null)  {  symbols.Add(new ShannonFanoSymbol(character, 1, 0.0, ""));  }  else  {  existingSymbol.Count++;  }  }  return symbols;  }  public static void Show(List<ShannonFanoSymbol> symbols)  {  foreach (var symbol in symbols)  {  Console.Write($"Символ: {symbol.Symbol} Кол-во: {symbol.Count}");  if (symbol.Probability != 0)  {  Console.Write($" Вероятность: {symbol.Probability}");  }  if (!string.IsNullOrEmpty(symbol.Code))  {  Console.Write($" Код: {symbol.Code}");  }  Console.WriteLine();  }  Console.WriteLine();  }  }  class Program  {  static void Main(string[] args)  {  List<ShannonFanoSymbol> symbols = new List<ShannonFanoSymbol>();  using (StreamReader stream = new StreamReader("fio.txt", Encoding.Default))  {  string messagef;  while ((messagef = stream.ReadLine()) != null)  {  symbols = ShannonFanoSymbol.AddSymbols(symbols, messagef);  }  }  Console.WriteLine();  Console.WriteLine("На основе данных, полученных в лабораторной работе № 2");  Console.WriteLine("Таблица символов: ");  Console.WriteLine();  ShannonFanoSymbol.Show(symbols);  double symbolssum = symbols.Sum(x => x.Count);  Console.WriteLine("Сумма всех символов текста на латинском языке: " + symbolssum);  foreach (var symbol in symbols)  {  symbol.Probability = symbol.Count / symbolssum;  }  Console.WriteLine("Сумма вероятностей всех символов таблицы: " + symbols.Sum(x => x.Probability));  Console.WriteLine();  symbols = symbols.OrderByDescending(x => x.Probability).ToList();  ShannonFanoSymbol.Show(symbols);  Console.WriteLine();  Console.WriteLine("Таблица с кодом для каждого символа: ");  Console.WriteLine();  symbols = AddCodes(symbols);  foreach (var symbol in symbols)  {  symbol.Code = symbol.Code.Remove(symbol.Code.Length - 1, 1);  }  ShannonFanoSymbol.Show(symbols);  string blockofFIO = "Palaznik Arseniy Viktorovich";  string decodingOfFIO = "";  foreach (var charFIO in blockofFIO)  {  decodingOfFIO += (symbols.FirstOrDefault(x => x.Symbol == charFIO)?.Code) ?? "";  }  Console.WriteLine("Исходное сообщение: ");  Console.WriteLine(blockofFIO);  Console.WriteLine("Сообщение после кодировки: ");  Console.WriteLine(decodingOfFIO);  Console.WriteLine("Количество битов в кодах ASCII: " + blockofFIO.Length \* 8);  Console.WriteLine("Количество битов по таблице Шеннон-Фано: " + decodingOfFIO.Length);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  Console.WriteLine("Декодирование: ");  Console.WriteLine();  string Encoded = "";  string FIOdecoded = "";  for (int i = 0; i < decodingOfFIO.Length; i++)  {  Encoded += decodingOfFIO[i];  if (symbols.Find(x => x.Code == Encoded) != null)  {  FIOdecoded += symbols.Find(x => x.Code == Encoded).Symbol;  Encoded = "";  }  }  Console.WriteLine(FIOdecoded);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  Console.WriteLine("Динамически, на основе анализа сжимаемого сообщения: ");  Console.WriteLine();  symbols.Clear();  string message = "Palaznik";  symbols = ShannonFanoSymbol.AddSymbols(symbols, message);  ShannonFanoSymbol.Show(symbols);  symbolssum = symbols.Sum(x => x.Count);  Console.WriteLine("Сумма всех символов текста на латинском языке: " + symbolssum);  foreach (var symbol in symbols)  {  symbol.Probability = symbol.Count / symbolssum;  }  Console.WriteLine("Сумма вероятностей всех символов таблицы: " + symbols.Sum(x => x.Probability));  Console.WriteLine();  symbols = symbols.OrderByDescending(x => x.Probability).ToList();  ShannonFanoSymbol.Show(symbols);  Console.WriteLine();  Console.WriteLine("Таблица с кодом для каждого символа: ");  Console.WriteLine();  symbols = AddCodes(symbols);  foreach (var symbol in symbols)  {  symbol.Code = symbol.Code.Remove(symbol.Code.Length - 1, 1);  }  ShannonFanoSymbol.Show(symbols);  blockofFIO = "Palaznik";  decodingOfFIO = "";  foreach (var charFIO in blockofFIO)  {  decodingOfFIO += (symbols.FirstOrDefault(x => x.Symbol == charFIO)?.Code) ?? "";  }  Console.WriteLine("Исходное сообщение: ");  Console.WriteLine(blockofFIO);  Console.WriteLine("Сообщение после кодировки: ");  Console.WriteLine(decodingOfFIO);  Console.WriteLine("Количество битов в кодах ASCII: " + blockofFIO.Length \* 8);  Console.WriteLine("Количество битов по таблице Шеннон-Фано: " + decodingOfFIO.Length);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  Console.WriteLine("Декодирование");  Encoded = "";  FIOdecoded = "";  for (int i = 0; i < decodingOfFIO.Length; i++)  {  Encoded += decodingOfFIO[i];  if (symbols.Find(x => x.Code == Encoded) != null)  {  FIOdecoded += symbols.Find(x => x.Code == Encoded).Symbol;  Encoded = "";  }  }  Console.WriteLine(FIOdecoded);  Console.ReadLine();  }  public static List<ShannonFanoSymbol> AddCodes(List<ShannonFanoSymbol> symbols)  {  int counter = 0;  double probability = 0.0;  List<ShannonFanoSymbol> first = new List<ShannonFanoSymbol>();  List<ShannonFanoSymbol> second = new List<ShannonFanoSymbol>();  while (probability < (symbols.Sum(x => x.Probability) / 2))  {  probability += symbols[counter].Probability;  counter++;  }  for (int i = 0; i < counter; i++)  {  symbols[i].Code += "0";  first.Add(symbols[i]);  }  for (int i = counter; i < symbols.Count; i++)  {  symbols[i].Code += "1";  second.Add(symbols[i]);  }  if (symbols.Count > 1)  {  first = AddCodes(first);  second = AddCodes(second);  first.AddRange(second);  symbols = first;  }  return symbols;  }  }  } |

*Листинг 1* – код программы Program.cs





*Рис. 9.1* – результат выполения программы

**Ответы на вопросы:**

1. **Что такое бинарное дерево, чем характеризуется его структура?**

Бинарное дерево-это корневое дерево, которое также является упорядоченным деревом (оно же плоское дерево), в котором каждый узел имеет не более двух дочерних элементов. Корневое дерево естественным образом дает понятие уровней (расстояние от корня), поэтому для каждого узла понятие дочерних элементов может быть определено как узлы, соединенные с ним уровнем ниже.

1. **Какие коды называются префиксными?**

|  |
| --- |
| Префиксный код (англ. prefix code) — код, в котором никакое кодовое слово не является префиксом какого-то другого кодового слова. |

Предпочтение префиксным кодам отдается из-за того, что они упрощают декодирование. Поскольку никакое кодовое слово не выступает в роли префикса другого, кодовое слово, с которого начинается файл, определяется однозначно, как и все последующие кодовые слова.

**Вывод**: приобрел практические навыкови использования статистических методов Шеннона − Фано и Хаффмана (ShannonFano and Huffman coding) для сжатия/распаковки данных

# **Лабораторная работа №10**

**Тема «Сжатие/распаковка данных методов Лемпеля-Зива»**

**Цель:** приобретение практических навыков использования метод Лемпеля − Зива (Lempel-Ziv) для сжатия/распаковки данных.

**Теоретические сведения**

В 1977 г. Авраам Лемпель и Якоб Зив выдвинули идею формирования «словаря» общих последовательностей анализируемых (сжимаемых) данных. При этом сжатие данных осуществляется за счет замены записей соответствующими кодами из словаря. Классический алгоритм Лемпеля − Зива – LZ77, названный так по году представления метода, формулируется следующим образом: **«если в проанализированном (сжатом) ранее выходном потоке уже встречалась подобная последовательность байт, причем запись о ее длине и смещении от текущей позиции короче, чем сама эта последовательность, то в выходной файл записывается ссылка (смещение, длина), а не сама последовательность».**

Известный метод сжатия RLE, который заключается в записи вместо последовательности одинаковых символов одного символа и их количества, является подклассом LZ77.

**Суть метода LZ77** (как и последующих его модификаций) состоит в следующем: **упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере**. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также **скользящим словарем**, варьируются в разных реализациях систем сжатия. Скользящее окно имеет длину **n**, т. е. в него помещается **n** символов, и состоит из двух частей:

* последовательности длины **n1 = n − n2** уже закодированных символов (словарь);
* упреждающего буфера (буфера предварительного просмотра, lookahead) длиной n2 – буфера кодирования.

Пусть к текущему моменту времени закодировано t символов: S1, S2, ..., St. Тогда словарем будут являться n1 предшествующих символов: St − (n1 − 1), St − (n1 − 1)+1, …, St. В буфере находятся ожидающие кодирования (сжатия) символы St+1, St+2, …, St+n2. Если n2 ≥ t, то словарем будет являться вся уже обработанная часть входной последовательности

**Нужно найти самое длинное совпадение между строкой буфера кодирования, начинающейся с символа St + 1, и всеми фразами словаря.**

Эти фразы могут начинаться с любого символа St − (n1 − 1), St − (n1 − 1) + 1, …, St, выходить за пределы словаря, вторгаясь в область буфера, но должны лежать в окне. Буфер не может сравниваться сам с собой. Длина совпадения не должна превышать размера буфера. Полученная в результате поиска фраза St − (р − 1), St − (р − 1) + 1, St − (р − 1) + (q − 1) кодируется с помощью двух чисел:

1. смещения (англ. offset) от начала буфера p;
2. длины соответствия, или совпадения (англ. match length) q.

Ссылки (p и q − указатели) однозначно определяют фразу. Дополнительно в выходной поток записывается символ s, следующий за совпавшей строкой буфера.

Длина кодовой комбинации (триады – p, q, s) на каждом шаге определяется соотношением

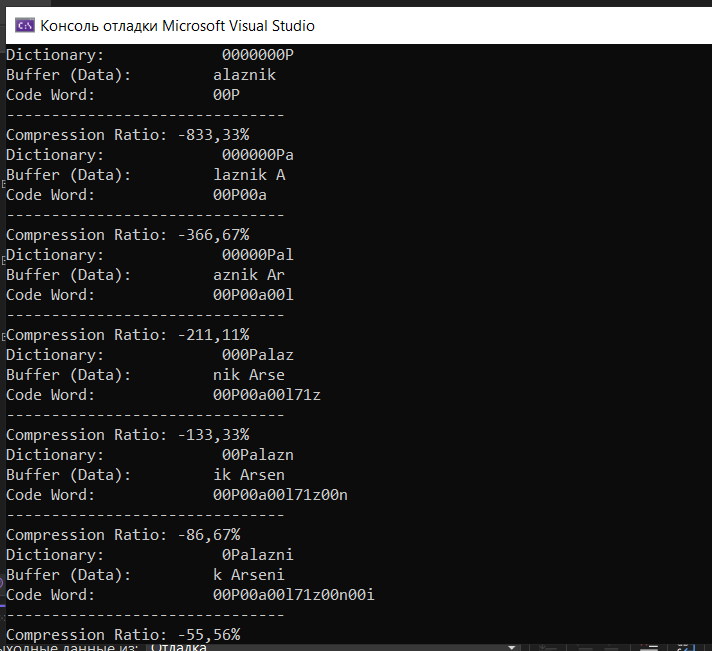
l(с ) = logN n1 +logN n2 + 1, (10.1)

где N – мощность алфавита.

После каждого шага окно смещается на q + 1 символов вправо и осуществляется переход к новому циклу кодирования. Величина сдвига объясняется тем, что мы реально закодировали именно q + 1 символов: q – с помощью указателя и 1 − с помощью тривиального копирования.

|  |
| --- |
| using System;  class Program  {  static (int, int, char) FindCharacters(string dictionary, string bufferWord)  {  int indexInArray = 0;  int length = 0;  char lastElement = bufferWord[0];  while (dictionary.Contains(bufferWord.Substring(0, length + 1)))  {  indexInArray = dictionary.IndexOf(bufferWord.Substring(0, length + 1)) + 1;  length += 1;  if (length == bufferWord.Length)  {  lastElement = '|';  break;  }  else  {  lastElement = bufferWord[length];  }  }  return (indexInArray, length, lastElement);  }  static (string, string) SendCharacters(string firstBuffer, string secondBuffer, int charactersCount)  {  charactersCount = Math.Min(charactersCount, secondBuffer.Length);  if (charactersCount > 0)  {  firstBuffer += secondBuffer.Substring(0, charactersCount);  secondBuffer = secondBuffer.Substring(charactersCount);  }  return (firstBuffer, secondBuffer);  }  static string TrimBuffer(string buffer, int size)  {  if (buffer.Length > size)  {  buffer = buffer.Substring(buffer.Length - size);  }  return buffer;  }  static void Main(string[] args)  {  const int dictionarySize = 8;  const int bufferSize = 8;  string inputMessage = "Palaznik Arseniy Viktorovich";  string resultMessage = "";  string word = inputMessage.Substring(0, bufferSize);  string message = inputMessage.Substring(bufferSize);  string dictionary = new string('0', dictionarySize);  string codeWord = "";  int iterationCount = 0;  int totalIterations = inputMessage.Length / bufferSize;  while (!string.IsNullOrEmpty(word))  {  (int p, int q, char c) = FindCharacters(dictionary, word);  (dictionary, word) = SendCharacters(dictionary, word, q + 1);  (word, message) = SendCharacters(word, message, q + 1);  dictionary = TrimBuffer(dictionary, dictionarySize);  word = TrimBuffer(word, bufferSize);  codeWord += $"{p}{q}{c}";  Console.WriteLine($"Dictionary: {dictionary}");  Console.WriteLine($"Buffer (Data): {word}");  Console.WriteLine($"Code Word: {codeWord}");  Console.WriteLine("-------------------------------");  double lengthBeforeCompression = inputMessage.Length;  double lengthAfterCompression = codeWord.Length;  double compressionRatioBefore = lengthBeforeCompression / lengthAfterCompression;  double compressionRatioAfter = (1 - compressionRatioBefore) \* 100;  Console.WriteLine($"Compression Ratio: {compressionRatioAfter:F2}%");  iterationCount += 1;  }  dictionary = new string('0', dictionarySize);  for (int i = 0; i < codeWord.Length; i += 3)  {  int p = int.Parse(codeWord[i].ToString());  int q = int.Parse(codeWord[i + 1].ToString());  char c = codeWord[i + 2];  if (p == 0 && q == 0)  {  resultMessage += c;  dictionary += c;  }  else  {  string strPart = dictionary.Substring(p - 1, q) + c;  resultMessage += strPart;  dictionary += strPart;  }  dictionary = TrimBuffer(dictionary, dictionarySize);  Console.WriteLine($"Code: {p} {q} {c}");  Console.WriteLine($"Result: {resultMessage}");  Console.WriteLine($"Dictionary: {dictionary}");  Console.WriteLine("-------------------------------");  }  }  } |

*Листинг 1* – код программы Program.cs



*Рис. 9.1* – результат выполения программы

**Ответить на вопросы:**

**Вывод:** приобрел практические навыки использования метод Лемпеля − Зива (Lempel-Ziv) для сжатия/распаковки данных.

# **Лабораторная работа №11**

**Тема «Сжатие/распаковка данных арифметическим методом»**

**Цель:** приобретение практических навыков использования арифметических методов сжатия/распаковки данных.

**Теоретические сведения**

Пpи аpифметическом сжатии (кодиpовании) текст пpедставляется вещественными числами в интеpвале от 0 до 1. По меpе анализа текста отобpажающий его интеpвал уменьшается, а количество битов для его пpедставления возpастает. Очеpедные символы текста сокpащают величину интеpвала, исходя из значений соответствующих веpоятностей.

**Основная идея арифметического метода сжатия заключается в том, чтобы присваивать коды не отдельным символам, а их последовательностям.**

Таким образом, как и во всех энтропийных алгоритмах, исходной является информация о частоте встречаемости каждого символа алфавита. Алгоритмы прямого и обратного преобразований базируются на операциях с «рабочим отрезком».

Рабочим отрезком называется интервал [a; b] с расположенными на нем точками. Причем точки расположены таким образом, что длины образованных ими отрезков пропорциональны (или равны) частоте (вероятности) появления соответствующих символов.

**Прямое преобразование (сжатие).** Один шаг сжатия (кодирования) заключается в простой операции: берется кодируемый символ, для него ищется соответствующий участок на рабочем отрезке. Найденный участок становится новым рабочим отрезком. Его тоже необходимо разбить с помощью точек.

Это и последующие разбиения отрезка (на шаге i) подразумевают определение новых значений верхней (Hi) и нижней (Li) границ для всего участка и осуществляются по следующим правилам:

**Hi(αj)= Li − 1 + (Hi − 1 – Li − 1) · H(αj)0;**

**Li(αj)= Li − 1 + (Hi − 1 – Li − 1) · L(αj)0,**

где αj – j-й символ сжимаемой последовательности, Li − 1 и Hi − 1 – соответственно нижняя и верхняя границы рабочего отрезка на (i − 1)-м шаге, L(αj)0 и H(αj)0 – соответственно исходные нижняя и верхняя границы символа αj.

Результатом кодирования цепочки символов является любое число с итогового рабочего отрезка. Обычно таким числом является нижняя граница указанного отрезка. Таким образом, итогом сжатия входной последовательности будет число.

**Обратное преобразование (декомпрессия).** Для восстановления исходного сообщения необходима информация:

* о значении числа, являющегося итогом сжатия сообщения;
* количестве символов в сжатом сообщении;
* вероятностных параметрах всех символов исходного сообщения (таблица вероятностей).

Как и при сжатии, вначале необходимо начальный рабочий отрезок [0; 1) разбить на интервалы, длины которых равны вероятностям появления соответствующих символов, т. е. создать рабочий отрезок, полностью соответствующий первоначальному отрезку.

На каждом шаге обратного преобразования выбираем отрезок, в который попадает текущее число (код). Символ, который соответствует данному отрезку, является очередным символом восстановленного (распакованного) сообщения.

В общем случае код символа, восстанавливаемого на шаге i, вычисляется соотношением:

**код i = [код (i − 1) − L(αi − 1)0] / [H(αi − 1)0 − L(αi − 1)0],**

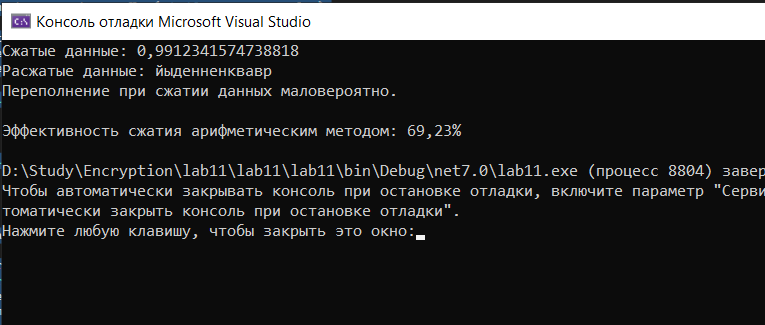
где код (i − 1) – число, анализ которого производился на предыдущем шаге – (i − 1)-м; H(αi − 1)0 и L(αi − 1)0 – соответственно верхняя и нижняя исходные границы символа сообщения, восстановленного на предыдущем шаге.

**Практическое задание**

1. Разработать авторское приложение в соответствии с целью лабораторной работы.
2. С помощью приложения выполнить прямое и обратное преобразования сообщений в соответствии с таблицей. Каждый студент выполняет задание, состоящее из двух частей. Первая часть предусматривает кодирование/декодирование сообщения, указанного в 2-м столбце, вторая часть – составного сообщения, полученного конкатенацией последовательностей из 2-го столбца, указанных в 3-м столбце.
3. Дать оценку возможности переполнения при выполнении вычислений.
4. Сравнить характеристики арифметического сжатия с вероятностными алгоритмами.
5. Результаты оформить в виде отчета по установленным правилам.

|  |
| --- |
| using System;  using System.Collections.Generic;  public class Node  {  public char Symbol { get; set; }  public double Low { get; set; }  public double High { get; set; }  public Node( char symbol, double low, double high )  {  Symbol = symbol;  Low = low;  High = high;  }  public override string ToString()  {  return $"Low: {Low} | High: {High}";  }  }  public class Compressor  {  private List<Node> nodes;  private Dictionary<char, double> frequencies;  private Node resultNode;  public Compressor()  {  nodes = new List<Node>();  frequencies = new Dictionary<char, double>();  resultNode = new Node('\*', 1, 0);  }  public void Build( string source )  {  double inc = 1.0 / source.Length;  foreach (char c in source)  {  if (!frequencies.ContainsKey(c))  {  frequencies[c] = 0;  }  frequencies[c] += inc;  }  double low = 0;  foreach (var item in frequencies)  {  nodes.Add(new Node(item.Key, low, low + frequencies[item.Key]));  low += frequencies[item.Key];  }  }  public double Compress( string source )  {  List<string> infoString = new List<string>();  foreach (char item in source)  {  double oldHigh = resultNode.High;  double oldLow = resultNode.Low;  infoString.Add(resultNode.ToString());  resultNode.Symbol = '\*';  resultNode.High = oldLow + (oldHigh - oldLow) \* nodes.Find(x => x.Symbol == item).High;  resultNode.Low = oldLow + (oldHigh - oldLow) \* nodes.Find(x => x.Symbol == item).Low;  }  infoString.Add(resultNode.ToString());  return resultNode.Low;  }  public string Decompress( double compress, int leng, int t )  {  List<char> sb = new List<char>();  List<string> infoString = new List<string>();  for (int i = 0; i < leng; i++)  {  char symbol = nodes.Find(x => compress >= x.Low && compress < x.High).Symbol;  infoString.Add($"{compress} -- {symbol}");  sb.Add(symbol);  Node tempNode = nodes.Find(x => x.Symbol == symbol);  compress = (compress - tempNode.Low) / (tempNode.High - tempNode.Low);  }  return string.Join("", sb);  }  }  class Program  {  static void CalculateCompressionEfficiency( string compressed, string original )  {  int compressedLength = Math.Min(compressed.Length, 4);  double efficiency = (1 - (double)compressedLength / original.Length) \* 100;  Console.WriteLine($"\nЭффективность сжатия арифметическим методом: {efficiency:F2}%");  }  static void CheckCompressionOverflow( double compressResult )  {  int maxBits = (int)Math.Log2(long.MaxValue) + 1;  int bitsNeeded = Convert.ToString((long)compressResult, 2).Length;  if (bitsNeeded > maxBits)  {  Console.WriteLine("Возможно переполнение при сжатии данных.");  }  else  {  Console.WriteLine("Переполнение при сжатии данных маловероятно.");  }  }  static void Main( string[] args )  {  string word = "сорокадневный";  Compressor wordCompressor = new Compressor();  wordCompressor.Build(word);  double compressResult = wordCompressor.Compress(word);  Console.WriteLine($"Сжатые данные: {compressResult}");  string decompressResult = wordCompressor.Decompress(compressResult, word.Length, word.Length / 2 + 1);  Console.WriteLine($"Расжатые данные: {decompressResult}");  CheckCompressionOverflow(compressResult);  CalculateCompressionEfficiency(compressResult.ToString(), word);  }  } |

*Листинг 1* – код программы Program.cs



*Рис. 1* – результат выполнения программы

**Вывод:** приобрел практические навыки использования арифметических методов сжатия/распаковки данных.