Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Электронная тетрадь**

по Защите Информации

Студентка: Кохнюк А. С.

ФИТ 3 курс 2 группа

Преподаватель: Урбанович П. П.

Ассистент: Ржеутская Н. В.

Минск 2022

# **Лабораторная работа №1**

**Тема «Разработка и внедрение политики безопасности организации учреждения»**

**Вариант 5**

**Цель:** приобретение практических навыков разработки и внедрения эффективной политики информационной безопасности организации или учреждения.

**Задачи:**

1. Научиться выделять и классифицировать особенности информационной или информационно-вычислительной системы (ИВС) конкретной организации или учреждения как объекта защиты.
2. Овладеть навыками принятия обоснованных решений по организационному и правовому регулированию проблем, относящихся к состоянию безопасности ИВС, обеспечению необходимого уровня защиты информации в ИВС.
3. Овладеть основными приемами анализа угроз информационной безопасности ИВС.
4. Научиться выявлять все возможные угрозы и их источники информационной безопасности в организации или учреждении, анализировать и оценивать собранные данные.
5. Разработать концепцию, основные элементы политики безопасности для организации или учреждения по указанному преподавателем варианту задания.
6. Разработать мероприятия по внедрению предложенной Вами политики безопасности.
7. Результаты выполнения лабораторной работы оформить в виде описания разработанной политики безопасности, а также плана мероприятий по ее реализации

**Теоретические сведения**

**Политика информационной безопасности (ПИБ)** организации или учреждения – совокупность правил, процедур, практических методов, руководящих принципов, документированных управленческих решений, направленных на защиту информации и связанных с ней ресурсов и используемых всеми сотрудниками организации или учреждения в своей деятельности.

**Информационная (информационно-вычислительная) система** – организационно упорядоченная совокупность документов, технических средств и информационных технологий, реализующая информационные (информационно-вычислительные) процессы.

**Информационные процессы** – процессы сбора, накопления, хранения, обработки (переработки), передачи и использования информации.

**Информационные ресурсы** – отдельные документы или массивы документов в информационных системах.

**Объект** – пассивный компонент системы, хранящий, перерабатывающий, передающий или принимающий информацию; примеры объектов: страницы, файлы, папки, директории, компьютерные программы, устройства (мониторы, диски, принтеры и т. д.).

**Субъект** – активный компонент системы, который может инициировать поток информации; примеры субъектов: пользователь, процесс либо устройство.

**Доступ** – специальный тип взаимодействия между объектом и субъектом, в результате которого создается поток информации от одного к другому.

**Атака** – попытка несанкционированного преодоления защиты системы.

**Несанкционированный доступ (НСД**) – доступ к информации, устройствам ее хранения и обработки, а также к каналам передачи, реализуемый без ведома (санкции) владельца и нарушающий тем самым установленные правила доступа.

**Защита информации** – организационные, правовые, программно-технические и иные меры по предотвращению угроз информационной безопасности и устранению их последствий.

**Безопасность информации** – защищенность информации от нежелательного (для соответствующих субъектов информационных отношений) ее разглашения (нарушения конфиденциальности), искажения (нарушения целостности), утраты или снижения степени доступности информации, а также незаконного ее тиражирования.

**Безопасность любого ресурса информационной системы** складывается из обеспечения трех его характеристик: конфиденциальности, целостности и доступности, также могут быть включены другие, такие как аутентичность, подотчетность, надежность; **информационная безопасность** – все аспекты, связанные с определением, достижением и поддержанием конфиденциальности, целостности, доступности информации или средств ее обработки:

* **конфиденциальность** (англ. confidentiality) компонента системы заключается в том, что он доступен только тем субъектам доступа (пользователям, программам, процессам), которым предоставлены на то соответствующие полномочия;
* **целостность** (англ. integrity) компонента предполагает, что он может быть модифицирован только субъектом, имеющим для этого соответствующие права; целостность является гарантией корректности (неизменности, работоспособности) компонента в любой момент времени;
* **доступность** (англ. availability) компонента означает, что имеющий соответствующие полномочия субъект может в любое время без особых проблем получить доступ к необходимому компоненту системы (ресурсу).

Для создания эффективной системы информационной безопасности организации или учреждения целесообразно разработать:

* концепцию информационной безопасности, которая определяет в целом цели политики и основные ее принципы в увязке со статусом, целями и задачами организации или учреждения;
* стандарты (менеджмента качества) – правила и принципы защиты информации по каждому конкретному направлению деятельности;
* процедуры – описание конкретных действий по защите информации при работе с ней: персональных данных, порядка доступа к информационным носителям, системам и ресурсам;
* инструкции, содержащие подробное описание (алгоритмы) действий по организации информационной защиты и обеспечению разработанных стандартов и процедур;
* план мероприятий по обучению персонала и тестированию знаний сотрудников, имеющих доступ к информационным ресурсам.

**Практическое задание**

Разработать политику информационной безопасности организации согласно варианту, а также план мероприятий по ее реализации.

1. **Обоснование актуальности, цели и задачи разработки ПИБ в организации.**

Основной целью, на достижение которой направлена ПИБ, является минимизация ущерба от событий, таящих угрозу безопасности информации, посредством их предотвращения или сведения их последствий к минимуму.

Для достижения цели необходимо обеспечивать решение следующих **задач:**

* Своевременное выявление, оценка и прогнозирование источников угроз ИБ;
* Создание механизма оперативного реагирования на угрозы ИБ;  
  Предотвращение и/или снижение ущерба от реализации угроз ИБ;
* Защита от вмешательств в процесс функционирования Информационной Системы (ИС)  
  посторонних лиц;
* Соответствие требованиям законодательства по информационной безопасности Республики Беларусь, нормативно-методических документов и договорным обязательствам в части ИБ;
* Обеспечение непрерывности критических бизнес-процессов;
* Достижение адекватности мер по защите от угроз ИБ;
* Изучение партнёров, клиентов, конкурентов и кандидатов на работу;
* Недопущение проникновения структур организованной преступности и отдельных лиц с противоправными намерениями;
* Выявление, предупреждение и пресечение возможной противоправной и иной негативной деятельности сотрудников;
* Повышение деловой репутации и корпоративной культуры;

1. **Объекты защиты.**

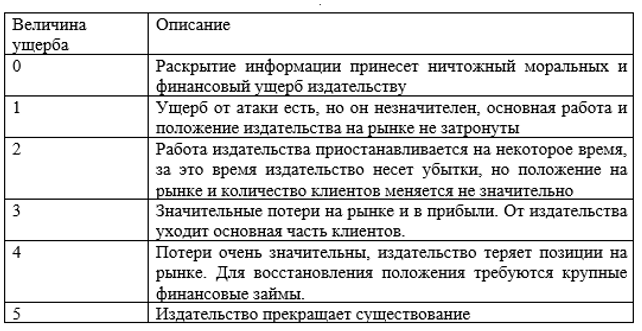
**Объекты защиты:**

* коммерческая тайна издательства, данные о ее договорах, финансовых  
  взаимоотношениях, бухгалтерская информация;
* коммерческая тайна клиентов и партнеров организации, данные об их активах, имуществе, платежах, произошедших страховых событиях;
* персональные данные сотрудников компании и сотрудников клиентов, эта информация иногда включает номера автомобилей, водительских удостоверений, кредитных карт;

1. **Основные угрозы и их источники.**

* Действия внутреннего или внешнего злоумышленника (несанкционированный, в том числе удаленный доступ с целью нарушения работоспособности ИВС, кражи, удаления или модификации информации, несанкционированного распространение материальных носителей за пределами организации);
* наблюдение за источниками информации;
* подслушивание конфиденциальных разговоров и акустических сигналов работающих механизмов;
* перехват электрических, магнитных и электромагнитных полей, электрических сигналов и радиоактивных излучений;
* разглашение информации компетентными людьми;
* несанкционированное распространение информации через поля и электрические сигналы, случайно возникшие в аппаратуре;
* воздействие стихийных сил (наводнения, пожары и т. п.);  
  сбои и отказы в аппаратуре сбора, обработки и передачи информации;  
  отказы системы электроснабжения;
* воздействие мощных электромагнитных и электрических помех (промышленных и природных).

1. **Оценка угроз, рисков и уязвимостей.**

****

**Условная численная шкала для оценки ущерба издательства от НСД**

|  |  |
| --- | --- |
| **Вероятность события** | **Средняя частота события (НСД)** |
| 0 | Данный вид атаки отсутствует |
| 0,1 | Реже, чем раз в год |
| 0,2 | Около 1 раза в год |
| 0,3 | Около 1 раза в месяц |
| 0,4 | Около 1 раза в неделю |
| 0,5 | Практически ежедневно |

**Оценка рисков**

|  |  |  |  |
| --- | --- | --- | --- |
| **Описание атаки** | **Ущерб** | **Вероятность** | **Риск (Ущерб\*Вероятность)** |
| Спам (переполнение почтового ящика) | 1 | 0,4 | 0,4 |
| Копирование жесткого диска из центрального | 3 | 0,1 | 0,3 |
| Непреднамеренный рассказ конфиденциальной информации сотрудниками издательства | 3 | 0,4 | 1,2 |
| Раскрытие основных планов и стратегий развития издательство | 4 | 0,1 | 0,4 |
| Раскрытие личных данных клиентов издательств | 3 | 0,2 | 0,6 |
| **Итого** | **14** | **1,2** | **2,9** |

1. **Меры, методы и средства обеспечения требуемого уровня защищенности информационных ресурсов.**

Информационная безопасность издательства должна обеспечиваться целым комплексом мер, среди которых:

* административно-правовые;
* организационные;
* программно-технические  
   Данные меры следует применять совместно. Опираться система защиты должна на управление персоналом компании и контроль над ним. Меры технического характера не менее важны, но не могут существовать в отрыве от организационных мер.
* VPN
* Шифрование
* Резервное копирование данных
* Тщательно отбирать сотрудников
* Гидро-защита техники

**Вывод:**

Комплексное применение современных технических средств в работе службы безопасности издательства может обеспечить высокий уровень защиты информации от утечек и несанкционированного доступа.

Следует учитывать, что все предпринимаемые действия должны в полной мере соответствовать требованиям законодательства. В частности, нарушение  
конфиденциальности данных юридической фирмы и их клиентов зачастую приводит к вымогательству и шантажу, инсайдерской торговле и недобросовестной конкуренции. Это не только нанесет урон репутации – юридическая фирма понесет ответственность – начиная от финансовой и закачивая уголовной

**Ответы на вопросы:**

1. **Охарактеризовать актуальность и основные причины проблемы информационной безопасности организации, страны.**

Обеспечение ИБ на предприятиях и в учреждениях, как правило, является неотъемлемой частью общей системы управления, необходимой для достижения уставных целей и задач. Значимость систематической целенаправленной деятельности по обеспечению ИБ становится тем более высокой, чем выше степень автоматизации бизнес-процессов. Значимость обеспечения ИБ в некоторых случаях может определяться наличием в общей системе информационных потоков предприятия сведений, составляющих не только коммерческую, но и государственную тайну, а также другие виды конфиденциальной информации: сведения, составляющие банковскую тайну, различные виды персональных данных, в том числе врачебная тайна, интеллектуальная собственность компаний-партнеров и т. п.

1. **Сформулировать цели и задачи политики информационной безопасности.**

* конфиденциальность данных – доступ есть только у лиц, имеющих на это полномочия;
* доступность информационных систем с находящимися в них данными конкретным пользователям, у которых есть право доступа к таким сведениям;
* целостность данных предполагает блокировку несанкционированного изменения информации;
* подлинность – полнота и общая точность информации;
* неотказуемость – возможность определить источник или авторство информации.

1. **Как правильно проводить оценку рисков?**

Оценка рисков включает в себя следующие **действия и мероприятия**:

* идентификация значимых угроз и уязвимостей для идентифицированных ресурсов;
* оценка вероятности возникновения угроз и уязвимостей;
* вычисление рисков; оценивание рисков по заранее определенной шкале риска.

При оценивании рисков учитываются:

* ценность ресурсов;
* оценка значимости угроз;
* эффективность существующих и планируемых средств защиты.

1. **Что должна включать в себя программа внедрения политики информационной безопасности?**

Политика информационной безопасности должна рассматриваться как система, как комплекс инструментов по защите информации.

Основу таких мероприятий составляют инструкции, содержащие подробное описание (алгоритмы) действий по организации информационной защиты и обеспечению разработанных стандартов и процедур, и план мероприятий по обучению персонала и тестированию знаний сотрудников, имеющих доступ к информационным ресурсам.

Можно выделить следующие общие **направления мероприятий**:

* *управление персоналом* (для каждой должности должны существовать квалификационные требования по ИБ, в должностные инструкции должны входить разделы, касающиеся информационной безопасности, каждого работника нужно научить мерам безопасности теоретически и на практике);
* *физическая защита инфраструктуры ИВС* (защита от утечки информации по техническим каналам, инженерные способы защиты и т. д.);
* *поддержание работоспособности ИВС* (создание инфраструктуры, включающий в себя как технические, так и процедурные регуляторы и способной обеспечить любой наперед заданный уровень работоспособности на всем протяжении жизненного цикла информационной системы.);
* *реагирование на нарушения режима безопасности ИВС* (может быть регламентировано в рамках отдельно взятой организации. В настоящее время осуществляется только мониторинг компьютерных преступлений в национальном масштабе и на мировом уровне.);
* *планирование восстановительных работ* (слаженность действий персонала во время и после аварии, наличие заранее подготовленных резервных производственных площадок, официально утвержденную схему переноса на резервные площадки основных информационных ресурсов, схему возвращения к нормальному режиму работы).

**Вывод:** приобрела практические навыки разработки и внедрения эффективной политики информационной безопасности организации или учреждения.

# **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;  using System;  namespace lab2  {  class Program  {  static void Main(string[] args)  {    //First task  Console.WriteLine("Энтропия алфавитов");  Console.WriteLine("Кириллица: " + Task.EntropyOfAlphabet(Task.Alphabets.Cyrillic));  Console.WriteLine("Латинский: " + Task.EntropyOfAlphabet(Task.Alphabets.Latin));  //Second task  Console.WriteLine("Бинарный: " + Task.EntropyOfAlphabet(Task.Alphabets.Binary));  Console.WriteLine();  //Third task  Console.WriteLine("Количество информации в моем полном имени (латиницей): " + (Task.EntropyOfAlphabet(Task.Alphabets.Latin) \* "КохнюкАлександраСергеевна".Length));  Console.WriteLine("Количество информации в моем полном имени (ASCII): " +  (Task.EntropyOfAlphabet(Task.Alphabets.Binary) \*  System.Text.Encoding.Unicode.GetBytes("КохнюкАлександраСергеевна").Length));  //Fourth task  Console.WriteLine("Количество информации в моем полном имени (ASCII, вероятность ошибки 0.1): " +  (Task.EntropyOfAlphabet(Task.Alphabets.Binary, 0.1f) \*  System.Text.Encoding.Unicode.GetBytes("Kohnyuk Aleksandra Sergeevnah").Length));  Console.WriteLine("Количество информации в моем полном имени (ASCII, вероятность ошибки 0.5)6 " +  (Task.EntropyOfAlphabet(Task.Alphabets.Binary, 0.5f) \*  System.Text.Encoding.Unicode.GetBytes("Kohnyuk Aleksandra Sergeevna").Length));  Console.WriteLine("Количество информации в моем полном имени (ASCII, вероятность ошибки 1): " +  (Double.IsNaN(Task.EntropyOfAlphabet(Task.Alphabets.Binary, 1f)) ? 0 : Task.EntropyOfAlphabet(Task.Alphabets.Binary, 1f) \*  System.Text.Encoding.Unicode.GetBytes("Kohnyuk Aleksandra Sergeevna").Length));  }  }  } |

Листинг *1* – Код файла Program.cs

|  |
| --- |
| using System.IO;  using System.Collections.Generic;  using System;  using System.Linq;  namespace lab2  {  public static class Task  {  public enum Alphabets  {  Latin,  Cyrillic,  Binary  }  public static double EntropyOfAlphabet(Alphabets Alphabet, float errorProbability = 0)  {  string alphabet = "";  string path = "";  if (Alphabet == Alphabets.Latin)  {  alphabet = "qwertyuiopasdfghjklzxcvbnm";  path = "latin.txt";  }  else if (Alphabet == Alphabets.Cyrillic)  {  path = "cyrillic.txt";  alphabet = "йцукенгшщзхъфывапролджэячсмитьбю";  }  else if (Alphabet == Alphabets.Binary)  {  path = "binary.bin";  alphabet = "01";  }  Dictionary<char, int> numberOfOccurrences = new Dictionary<char, int>();//встреча  foreach (var ch in alphabet)  numberOfOccurrences.Add(ch, 0);  //подсчитываем кол-во встреч  using (StreamReader sr = new StreamReader(path))  {  string text = sr.ReadToEnd();  text = text.ToLower();  foreach (var ch in text.Select((value, i) => new { i, value }))  {  if (alphabet.Contains(ch.value))  numberOfOccurrences[ch.value]++;  else  text.Remove(ch.i);  }  double answer = 0;  foreach (var ch in alphabet)  {  if (numberOfOccurrences[ch] != 0)  {  double P = (double)numberOfOccurrences[ch] / (double)text.Length \* (1 - errorProbability);  answer += P \* Math.Log2(P);  }  }  return -answer;  }  }  }  } |

Листинг *2* – Код файла Task.cs

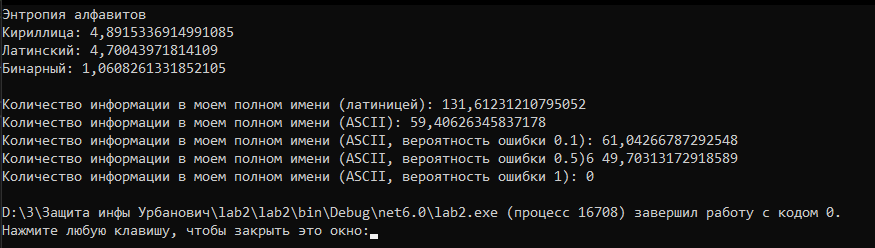
****

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

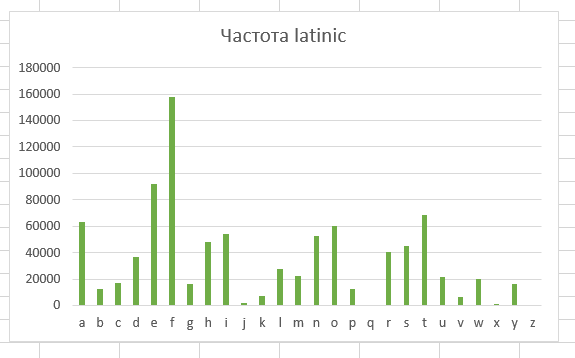
****

Рис. *1*.*2* – Частота латинского алфавита

****

Рис. *1*.*3* – Частота русского 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.

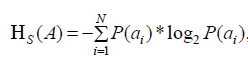
1. **От чего зависит энтропия алфавита?**

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).**

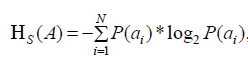


1. **Что нужно знать для вычисления энтропии алфавита?**

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:** приобрела 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.

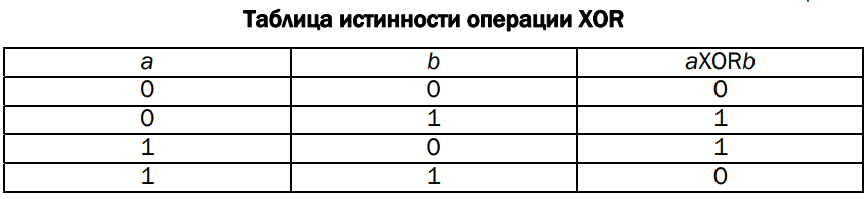
Утверждение восьмибитных кодировок (ASCII) как стандарта принесло некоторые проблемы. К этому моменту уже существовала определенная инфраструктура, использующая семибитные кодировки. Известны проблемы с «обрезанием восьмого бита» в системе электронной почты. Утверждение восьмибитного символа дало 256 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;  using System.Threading.Tasks;  namespace lab3  {  public static class LW3  {  public static string toBase64(string path)  {  string text;  using (StreamReader sr = new StreamReader(path))  {  text = sr.ReadToEnd();  }  return Convert.ToBase64String(System.Text.Encoding.Unicode.GetBytes(text));  }  public static void createBase64Doc()  {  using (FileStream fs = new FileStream("base64.txt", FileMode.OpenOrCreate))  {  fs.Write(System.Text.Encoding.Unicode.GetBytes(toBase64("latin.txt")));  }  }  public static string createReport()  {  double base64ES = LW2.EntropyOfAlphabet(LW2.Alphabets.Base64),  latinES = LW2.EntropyOfAlphabet(LW2.Alphabets.Latin),  base64EC = Math.Log2("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/".Length),  latinEC = Math.Log2("qwertyuiopasdfghjklzxcvbnm".Length);  string report = "";  report += "Base64 энтропия (Шеннон): " + base64ES + "\n";  report += "Latin энтропия (Шеннон): " + latinES + "\n";  report += "Base64 энтропия (Хартли): " + base64EC + "\n";  report += "Latin энтропия (Хартли): " + latinEC + "\n";  report += "Избыточность base64: " + (base64EC - base64ES) / base64EC \* 100 + "%\n";  report += "Избыточность latin: " + (latinEC - latinES) / latinEC \* 100 + "%\n";  return report;  }  public static byte[] XOR(byte[] buf1, byte[] buf2)  {  byte[] ans = new byte[buf1.Length];  if (buf1.Length == buf2.Length)  {  for (int i = 0; i < buf1.Length; i++)  {  ans[i] = (byte)(buf1[i] ^ buf2[i]);  }  }  return ans;  }  public static byte[] myXOR(byte[] buf1, byte[] buf2)  {  byte[] ans = new byte[buf1.Length];  if (buf1.Length == buf2.Length)  {  for (int i = 0; i < buf1.Length; i++)  {  string buf1In2 = Convert.ToString(buf1[i], 2);  string buf2In2 = Convert.ToString(buf2[i], 2);  string bitAns = "";  for (int j = 0; j < buf2In2.Length; j++)  {  if (buf1In2[j] == '0' && buf2In2[j] == '0'  || buf1In2[j] == '1' && buf2In2[j] == '1')  bitAns += '0';  else  bitAns += '1';  }  ans[i] = (byte)Convert.ToInt32(bitAns, 2);  }  }  return ans;  }  }  } |

Листинг *1* – Код файла LW3.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab3  {  public static class LW2  {  public enum Alphabets  {  Latin,  Cyrillic,  Binary,  Base64  }  public static double EntropyOfAlphabet(Alphabets Alphabet, float errorProbability = 0)  {  string alphabet = "";  string path = "";  if (Alphabet == Alphabets.Latin)  {  alphabet = "qwertyuiopasdfghjklzxcvbnm";  path = "latin.txt";  }  else if (Alphabet == Alphabets.Cyrillic)  {  path = "cyrillic.txt";  alphabet = "абвгдежзийклмнопрстуфхцчшщъьюя";  }  else if (Alphabet == Alphabets.Binary)  {  path = "binary.bin";  alphabet = "01";  }  else if (Alphabet == Alphabets.Base64)  {  path = "base64.txt";  alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";  }  Dictionary<char, int> numberOfOccurrences = new Dictionary<char, int>();  foreach (var ch in alphabet)  numberOfOccurrences.Add(ch, 0);  using (StreamReader sr = new StreamReader(path))  {  string text = sr.ReadToEnd();  text = text.ToLower();  foreach (var ch in text.Select((value, i) => new { i, value }))  {  if (alphabet.Contains(ch.value))  numberOfOccurrences[ch.value]++;  else if (Alphabet != Alphabets.Base64)  text.Remove(ch.i);  }  double answer = 0;  foreach (var ch in alphabet)  {  if (numberOfOccurrences[ch] != 0)  {  double P = (double)numberOfOccurrences[ch] / (double)text.Length \* (1 - errorProbability);  answer += P \* Math.Log2(P);  }  }  return -answer;  }  }  }  } |

Листинг *2* – Код файла LW2.cs

|  |
| --- |
| using System;  using System.Text;  namespace lab3  {  class Program  {  static void Main(string[] args)  {  LW3.createBase64Doc();  Console.WriteLine(LW3.createReport());  Console.Write("ASCII XOR: ");  string surname = "Kohnyuk",  name = "Sasha",  surnameASCII = "",  nameASCII = "";  foreach (var ch in surname)  surnameASCII += Convert.ToInt32(ch);  foreach (var ch in name)  nameASCII += Convert.ToInt32(ch);  while (surnameASCII.Length != nameASCII.Length)  nameASCII += '0';  foreach (var ch in LW3.XOR(Encoding.Unicode.GetBytes(surnameASCII), Encoding.Unicode.GetBytes(nameASCII)))  Console.Write(ch);  Console.WriteLine();  Console.Write("Base64 XOR: ");  string name64 = Convert.ToBase64String(Encoding.Unicode.GetBytes(name)),  surname64 = Convert.ToBase64String(Encoding.Unicode.GetBytes(surname));  while (surname64.Length != name64.Length)  name64 += '0';  foreach (var ch in LW3.XOR(Encoding.Unicode.GetBytes(surname64), Encoding.Unicode.GetBytes(name64)))  Console.Write(ch);  Console.WriteLine();  Console.WriteLine();  Console.Write("aXORbXORb: \t");  byte[] aXORbXORb = LW3.XOR(Encoding.Unicode.GetBytes(surname64), LW3.XOR(Encoding.Unicode.GetBytes(name64), Encoding.Unicode.GetBytes(surname64)));  foreach (var ch in aXORbXORb)  Console.Write(ch);  Console.WriteLine();    }  }  } |

Листинг *2* – Код файла Program.cs

**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:** приобрела 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



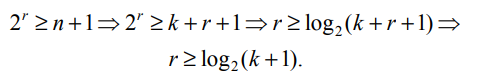
В последнем выражении символ «T» означает транспонирование, а X n = 1x , 2 x , …, xn. Матрица Н, называемая проверочной, равна



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.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace Laba4  {  class Program  {  static void Main(string[] args)  {  string str;  string binaryString;  do  {  Console.Clear();  Console.Write("Введите строку: ");  str = Console.ReadLine();  binaryString = ToBinary(ConvertToByteArray(str, Encoding.ASCII));  Console.WriteLine("\nВаша строка в кодах ASCII: " + binaryString);  }  while (str.Length < 2);  int k = binaryString.Length;  int r = LenghtHemminga(k);  int n = k + r;  int[] mas = new int[binaryString.Length + r];  int[,] checkMatrix = new int[n, r];  int error;  //Преобразование строки в массив  mas = StrInMas(binaryString, k);  Console.Write("\nВходная строка ");  OutMass(mas, k);  //Получение проверочной матрицы  checkMatrix = CheckMatrix(k);  Console.WriteLine("\nПроверочная матрица");  OutMass(checkMatrix, k);  //Добавление проверочных битов  Sindrom(checkMatrix, mas, k);  Console.WriteLine("\nПолная строка");  OutMass(mas, k);  try  {  Console.WriteLine("Введите номер бита первой ошибки");  error = Convert.ToInt32(Console.ReadLine()) - 1;  if (mas[error] == 1) mas[error] = 0;  else mas[error] = 1;  Console.WriteLine("Введите номер бита второй ошибки");  error = Convert.ToInt32(Console.ReadLine()) - 1;  if (mas[error] == 1) mas[error] = 0;  else mas[error] = 1;  }  catch { }  Console.WriteLine("\nСтрока с ошибкой");  OutMass(mas, k);  Console.WriteLine("\nСтрока с вектором ошибки");  mas = SearchError(mas, checkMatrix, k);  Console.WriteLine("\nСтрока без ошибки");  OutMass(mas, k);  }  //Считаем r (кол-во пров. симв.)  public static int LenghtHemminga(int k)  {  int r = (int)(Math.Log(k, 2) + 1.99f);  return r;  }  //Создание пров. матрицы  public static int[,] CheckMatrix(int k)  {  int r = LenghtHemminga(k);  int n = r + k;  double rDouble = r - 1;  int rPow = (int)(Math.Pow(2, rDouble));  int[,] mas = new int[n, r];  int[,] combinations = new int[rPow, r];  for (int i = 0; i < rPow; i++)  for (int j = 0; j < r; j++)  combinations[i, j] = 0;  //генератор бит.мн.  for (int segmentLenght = 0; segmentLenght < r - 2; segmentLenght++)  {  if (segmentLenght \* r > k) break;  for (int i = 0; i < segmentLenght + 2; i++)  {  combinations[segmentLenght \* r, i] = 1;  }  for (int segmentPositin = 1; segmentPositin < r; segmentPositin++)  {  for (int i = 0; i < r - 1; i++)  {  combinations[segmentLenght \* r + segmentPositin, i + 1] = combinations[segmentLenght \* r + segmentPositin - 1, i];  }  combinations[segmentLenght \* r + segmentPositin, 0] = combinations[segmentLenght \* r + segmentPositin - 1, r - 1];  }  if (segmentLenght == r - 3)  {  for (int i = 0; i < r; i++)  {  combinations[rPow - 1, i] = 1;  }  }  }  for (int i = 0; i < k; i++)  for (int j = 0; j < r; j++)  mas[i, j] = combinations[i, j];  for (int i = 0; i < r; i++)  mas[i + k, i] = 1;  return mas;  }  //Поиск синдрома  public static int[] Sindrom(int[,] CheckMatrix, int[] mas, int k)  {  int r = LenghtHemminga(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;  }  //Нахождение ошибок  public static int[] SearchError(int[] mas, int[,] checkMatrix, int k)  {  int r = LenghtHemminga(k);  int n = r + k;  int[] beforeSindrom = new int[r];  //запоминаем проверочные биты  for (int i = k; i < n; i++)  {  beforeSindrom[i - k] = mas[i];  }  mas = Sindrom(checkMatrix, mas, k);  //Складываем синдром по модулю два  for (int i = k, j = 0; i < n; i++)  {  if (beforeSindrom[i - k].Equals(mas[i]))  {  mas[i] = 0;  j++;  //если сумма по модулю два все пров. бит равна нулю  if (j == r)  {  for (int l = k; l < n; l++)  {  mas[l] = beforeSindrom[l - k];  }  return mas;  }  }  else  {  mas[i] = 1;  }  }  for (int i = 0; i < n; i++)  {  int l = 0;  for (int j = 0; j < r; j++)  {  if (checkMatrix[i, j].Equals(mas[j + k])) l++;  }  if (l == r)  {  mas[i] = (mas[i] + 1) % 2;  }  }  OutMass(mas, k);  mas = Sindrom(checkMatrix, mas, k);  return mas;  }  //Преобразование строки в массив  public static int[] StrInMas(string binaryString, int k)  {  int r = LenghtHemminga(k);  int[] mas = new int[binaryString.Length + r];  for (int i = 0; i < binaryString.Length; i++)  {  if (binaryString[i] == 49)  mas[i] = 1;  else mas[i] = 0;  }  return mas;  }  //вывод матрицы  public static void OutMass(int[,] mas, int k)  {  int r = LenghtHemminga(k);  int n = r + k;  for (int i = 0; i < r; i++)  {  for (int j = 0; j < n; j++)  {  Console.Write(mas[j, i]);  if (j + 1 == k) Console.Write("|");  }  Console.WriteLine();  }  }  //вывод одномерного массива  public static void OutMass(int[] mas, int k)  {  int n = LenghtHemminga(k) + k;  for (int i = 0; i < n; i++)  {  if (i == k) Console.Write("|");  Console.Write(mas[i]);  }  Console.WriteLine("\n");  }  public static byte[] ConvertToByteArray(string binaryString, Encoding encoding)  {  return encoding.GetBytes(binaryString);  }  public static String ToBinary(Byte[] data)  {  return string.Join("", data.Select(byt => Convert.ToString(byt, 2).PadLeft(8, '0')));  }  }  } |

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

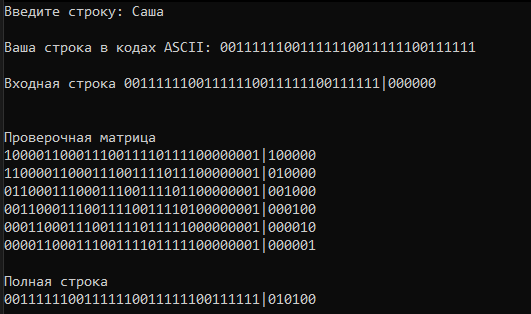


Рис. *4*.*1* – Результат выполнения программы

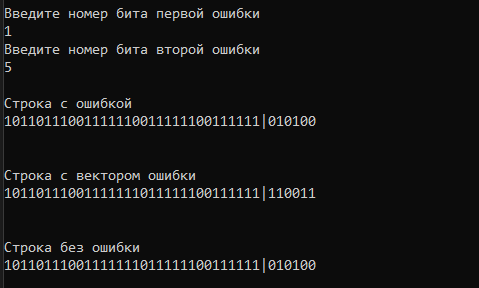


Рис. *4.2* – Результат выполнения программы

**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:** приобрела 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).

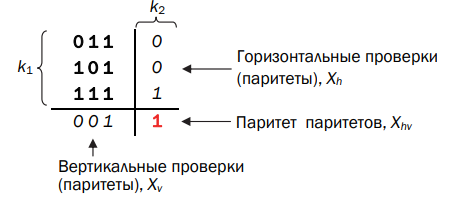


Рис. *5*.*1* – Пояснение к принципу формирования избыточных символов итеративного кода

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:

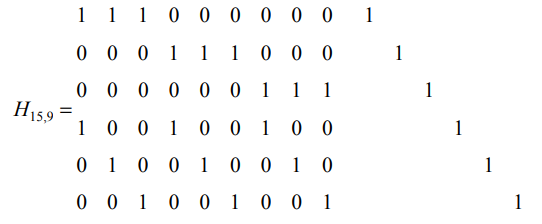


Рис. *5*.*2* – Проверочная матрица

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).

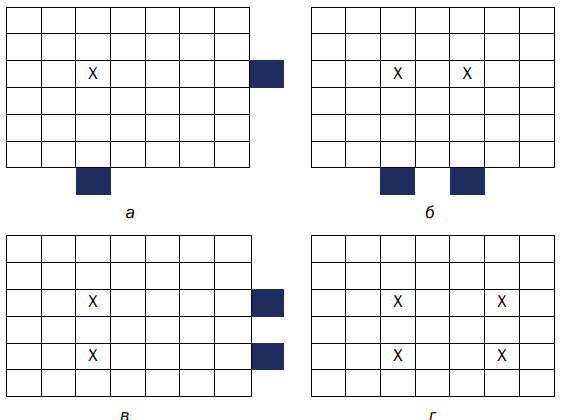


Рис. *5*.*3* – Пояснение к принципу обнаружения местоположения ошибочных битов в принятом сообщении

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;

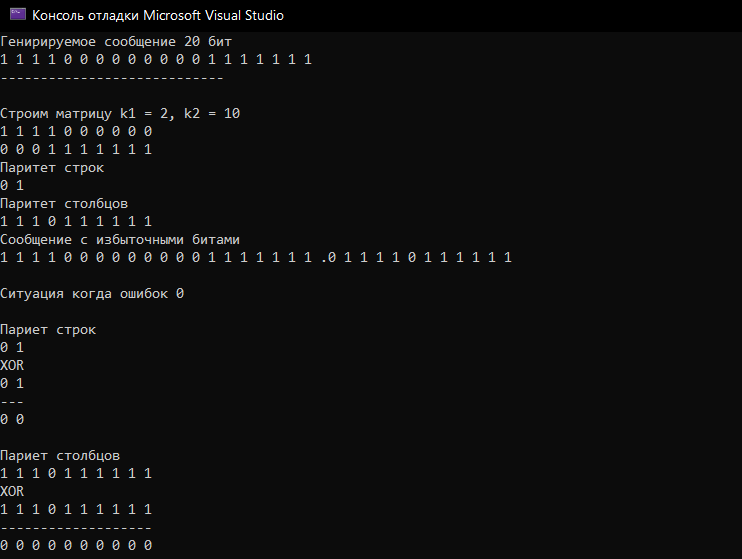
4) генерировать ошибку произвольной кратности (i, i > 0), распределенную случайным образом среди символов слова Xn, в результате чего формируется кодовое слово Yn;

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’);

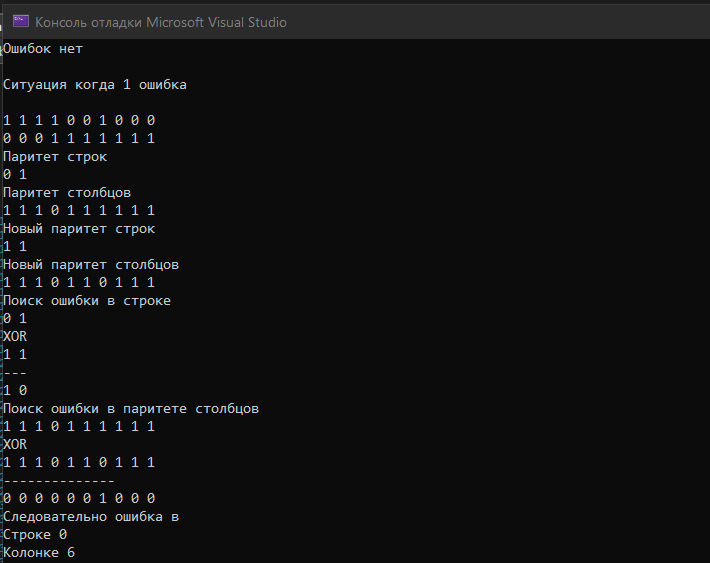
6) выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов Xn и Yn’; результат анализа может быть представлен в виде отношения общего числа сгенерированных кодовых слов с ошибками определенной одинаковой кратности (с одной ошибкой, с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

|  |
| --- |
| using System;  namespace Lab  {  class Program  {  static void Main(string[] args)  {    Console.WriteLine("Генирируемое сообщение 20 бит");  int[] array\_message = new int[20];  Random rand = new Random();  for (int i = 0; i < array\_message.Length; i++) {  array\_message[i] = rand.Next(0,2);  Console.Write(array\_message[i] + " ");  }  Console.WriteLine('\n' + "----------------------------");  Console.WriteLine('\n' + "Строим матрицу k1 = 2, k2 = 10");  for (int i = 0; i < array\_message.Length; i++)  {  if(i == 10)  {  Console.WriteLine("");  }  Console.Write(array\_message[i] + " ");  }  Console.WriteLine("");  #region Расчет паритета строк  Console.WriteLine("Паритет строк");  int[] array\_paritet\_rows = new int[2];  int flag = 0;  for (int i = 0; i < array\_message.Length; i++)  {  if (array\_message[i] == 1)  {  flag += 1;  }  if (i == 9)  {  if (flag % 2 == 0) { array\_paritet\_rows[0] = 0; }  else { array\_paritet\_rows[0] = 1; }  flag = 0;  }  if (i == 19)  {  if (flag % 2 == 0) { array\_paritet\_rows[1] = 0; }  else { array\_paritet\_rows[1] = 1; }  flag = 0;  }  }  #endregion  for (int i = 0; i < array\_paritet\_rows.Length; i++)  {  Console.Write(array\_paritet\_rows[i] + " ");  }  Console.WriteLine();  #region Расчет паритета столбцов  Console.WriteLine("Паритет столбцов");    int[] array\_paritet\_colums = new int[10];    if ((array\_message[0] + array\_message[10]) % 2 == 0) { array\_paritet\_colums[0] = 0; }  else { array\_paritet\_colums[0] = 1; }  if ((array\_message[1] + array\_message[11]) % 2 == 0) { array\_paritet\_colums[1] = 0; }  else { array\_paritet\_colums[1] = 1; }  if ((array\_message[2] + array\_message[12]) % 2 == 0) { array\_paritet\_colums[2] = 0; }  else { array\_paritet\_colums[2] = 1; }  if ((array\_message[3] + array\_message[13]) % 2 == 0) { array\_paritet\_colums[3] = 0; }  else { array\_paritet\_colums[3] = 1; }  if ((array\_message[4] + array\_message[14]) % 2 == 0) { array\_paritet\_colums[4] = 0; }  else { array\_paritet\_colums[4] = 1; }  if ((array\_message[5] + array\_message[15]) % 2 == 0) { array\_paritet\_colums[5] = 0; }  else { array\_paritet\_colums[5] = 1; }  if ((array\_message[6] + array\_message[16]) % 2 == 0) { array\_paritet\_colums[6] = 0; }  else { array\_paritet\_colums[6] = 1; }  if ((array\_message[7] + array\_message[17]) % 2 == 0) { array\_paritet\_colums[7] = 0; }  else { array\_paritet\_colums[7] = 1; }  if ((array\_message[8] + array\_message[18]) % 2 == 0) { array\_paritet\_colums[8] = 0; }  else { array\_paritet\_colums[8] = 1; }  if ((array\_message[9] + array\_message[19]) % 2 == 0) { array\_paritet\_colums[9] = 0; }  else { array\_paritet\_colums[9] = 1; }  for (int i = 0; i < array\_paritet\_colums.Length; i++)  {  Console.Write(array\_paritet\_colums[i] + " ");  }  #endregion  Console.WriteLine('\n'+"Сообщение с избыточными битами");  for (int i = 0; i < array\_message.Length; i++)  {  Console.Write(array\_message[i] + " ");  }  Console.Write(".");  for (int i = 0; i < array\_paritet\_rows.Length; i++)  {  Console.Write(array\_paritet\_rows[i] + " ");  }  for (int i = 0; i < array\_paritet\_colums.Length; i++)  {  Console.Write(array\_paritet\_colums[i] + " ");  }  Console.WriteLine("");  Console.WriteLine("");  #region Ситуация когда ошибок 0  Console.WriteLine("Ситуация когда ошибок 0");  Console.WriteLine('\n' + "Париет строк");  for (int i = 0; i < array\_paritet\_rows.Length; i++)  {  Console.Write(array\_paritet\_rows[i] + " ");  }  Console.WriteLine('\n' + "XOR");  for (int i = 0; i < array\_paritet\_rows.Length; i++)  {  Console.Write(array\_paritet\_rows[i] + " ");  }  Console.WriteLine('\n' + "---" + '\n' + "0 0");  Console.WriteLine('\n' + "Париет столбцов");  for (int i = 0; i < array\_paritet\_colums.Length; i++)  {  Console.Write(array\_paritet\_colums[i] + " ");  }  Console.WriteLine('\n' + "XOR");  for (int i = 0; i < array\_paritet\_colums.Length; i++)  {  Console.Write(array\_paritet\_colums[i] + " ");  }  Console.WriteLine('\n' + "-------------------" + '\n' + "0 0 0 0 0 0 0 0 0 0");  Console.WriteLine('\n' + "Ошибок нет");  #endregion  Console.WriteLine("");  #region Ситуация когда ошибка 1  Console.WriteLine("Ситуация когда 1 ошибка");  int error\_in\_rows = 0;  int error\_in\_colums = 0;  int[] array\_message\_with\_one\_error = new int[20];  array\_message\_with\_one\_error = array\_message;  int rand\_index = rand.Next(0, 20);  if (array\_message\_with\_one\_error[rand\_index] == 1) { array\_message\_with\_one\_error[rand\_index] = 0; }  else  {  array\_message\_with\_one\_error[rand\_index] = 1;  }  Console.WriteLine("");  for (int i = 0; i < array\_message\_with\_one\_error.Length; i++)  {  if (i == 10)  {  Console.WriteLine("");  }  Console.Write(array\_message\_with\_one\_error[i] + " ");  }  Console.WriteLine("");  Console.WriteLine("Паритет строк");  for (int i = 0; i < array\_paritet\_rows.Length; i++)  {  Console.Write(array\_paritet\_rows[i] + " ");  }  Console.WriteLine("");  Console.WriteLine("Паритет столбцов");  for (int i = 0; i < array\_paritet\_colums.Length; i++)  {  Console.Write(array\_paritet\_colums[i] + " ");  }  Console.WriteLine("");  #region Новый паритет строк  Console.WriteLine("Новый паритет строк");  int[] array\_paritet\_rows\_with\_one\_error = new int[2];  for (int i = 0; i < array\_message\_with\_one\_error.Length; i++)  {  if (array\_message\_with\_one\_error[i] == 1)  {  flag += 1;  }  if (i == 9)  {  if (flag % 2 == 0) { array\_paritet\_rows\_with\_one\_error[0] = 0; }  else { array\_paritet\_rows\_with\_one\_error[0] = 1; }  flag = 0;  }  if (i == 19)  {  if (flag % 2 == 0) { array\_paritet\_rows\_with\_one\_error[1] = 0; }  else { array\_paritet\_rows\_with\_one\_error[1] = 1; }  flag = 0;  }  }  for (int i = 0; i < array\_paritet\_rows\_with\_one\_error.Length; i++)  {  Console.Write(array\_paritet\_rows\_with\_one\_error[i] + " ");  }  Console.WriteLine("");  #endregion  #region Новый паритет столбцов  Console.WriteLine("Новый паритет столбцов");  int[] array\_paritet\_colums\_with\_one\_error = new int[10];  if ((array\_message\_with\_one\_error[0] + array\_message\_with\_one\_error[10]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[0] = 0; }  else { array\_paritet\_colums\_with\_one\_error[0] = 1; }  if ((array\_message\_with\_one\_error[1] + array\_message\_with\_one\_error[11]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[1] = 0; }  else { array\_paritet\_colums\_with\_one\_error[1] = 1; }  if ((array\_message\_with\_one\_error[2] + array\_message\_with\_one\_error[12]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[2] = 0; }  else { array\_paritet\_colums\_with\_one\_error[2] = 1; }  if ((array\_message\_with\_one\_error[3] + array\_message\_with\_one\_error[13]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[3] = 0; }  else { array\_paritet\_colums\_with\_one\_error[3] = 1; }  if ((array\_message\_with\_one\_error[4] + array\_message\_with\_one\_error[14]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[4] = 0; }  else { array\_paritet\_colums\_with\_one\_error[4] = 1; }  if ((array\_message\_with\_one\_error[5] + array\_message\_with\_one\_error[15]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[5] = 0; }  else { array\_paritet\_colums\_with\_one\_error[5] = 1; }  if ((array\_message\_with\_one\_error[6] + array\_message\_with\_one\_error[16]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[6] = 0; }  else { array\_paritet\_colums\_with\_one\_error[6] = 1; }  if ((array\_message\_with\_one\_error[7] + array\_message\_with\_one\_error[17]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[7] = 0; }  else { array\_paritet\_colums\_with\_one\_error[7] = 1; }  if ((array\_message\_with\_one\_error[8] + array\_message\_with\_one\_error[18]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[8] = 0; }  else { array\_paritet\_colums\_with\_one\_error[8] = 1; }  if ((array\_message\_with\_one\_error[9] + array\_message\_with\_one\_error[19]) % 2 == 0) { array\_paritet\_colums\_with\_one\_error[9] = 0; }  else { array\_paritet\_colums\_with\_one\_error[9] = 1; }  for (int i = 0; i < array\_paritet\_colums\_with\_one\_error.Length; i++)  {  Console.Write(array\_paritet\_colums\_with\_one\_error[i] + " ");  }  #endregion  #region Ошибка в строке  Console.WriteLine('\n'+"Поиск ошибки в строке");  for (int i = 0; i < array\_paritet\_rows.Length; i++)  {  Console.Write(array\_paritet\_rows[i] + " ");  }  Console.WriteLine('\n' + "XOR");  for (int i = 0; i < array\_paritet\_rows\_with\_one\_error.Length; i++)  {  Console.Write(array\_paritet\_rows\_with\_one\_error[i] + " ");  }  Console.WriteLine('\n' + "---");  int[] rows\_paritet\_with\_errors = new int[2];  if ((array\_paritet\_rows[0] == 0 && array\_paritet\_rows\_with\_one\_error[0] == 0) || (array\_paritet\_rows[0] == 1 && array\_paritet\_rows\_with\_one\_error[0] == 1))  {  rows\_paritet\_with\_errors[0] = 0;  }  else  {  rows\_paritet\_with\_errors[0] = 1;  }  if ((array\_paritet\_rows[1] == 0 && array\_paritet\_rows\_with\_one\_error[1] == 0) || (array\_paritet\_rows[1] == 1 && array\_paritet\_rows\_with\_one\_error[1] == 1))  {  rows\_paritet\_with\_errors[1] = 0;  }  else  {  rows\_paritet\_with\_errors[1] = 1;  }  for (int i = 0; i < rows\_paritet\_with\_errors.Length; i++)  {  Console.Write(rows\_paritet\_with\_errors[i] + " ");  if(rows\_paritet\_with\_errors[i] == 1)  {  error\_in\_rows = i;  }  }  #endregion  #region Ошибка в колонке  Console.WriteLine('\n' + "Поиск ошибки в паритете столбцов");  for (int i = 0; i < array\_paritet\_colums.Length; i++)  {  Console.Write(array\_paritet\_colums[i] + " ");  }  Console.WriteLine('\n' + "XOR");  for (int i = 0; i < array\_paritet\_colums\_with\_one\_error.Length; i++)  {  Console.Write(array\_paritet\_colums\_with\_one\_error[i] + " ");  }  Console.WriteLine('\n' + "--------------");  int[] colums\_paritet\_with\_errors = new int[10];  if ((array\_paritet\_colums[0] == 0 && array\_paritet\_colums\_with\_one\_error[0] == 0) || (array\_paritet\_colums[0] == 1 && array\_paritet\_colums\_with\_one\_error[0] == 1))  {  colums\_paritet\_with\_errors[0] = 0;  }  else { colums\_paritet\_with\_errors[0] = 1; }  if ((array\_paritet\_colums[1] == 0 && array\_paritet\_colums\_with\_one\_error[1] == 0) || (array\_paritet\_colums[1] == 1 && array\_paritet\_colums\_with\_one\_error[1] == 1))  {  colums\_paritet\_with\_errors[1] = 0;  }  else { colums\_paritet\_with\_errors[1] = 1; }  if ((array\_paritet\_colums[2] == 0 && array\_paritet\_colums\_with\_one\_error[2] == 0) || (array\_paritet\_colums[2] == 1 && array\_paritet\_colums\_with\_one\_error[2] == 1))  {  colums\_paritet\_with\_errors[2] = 0;  }  else { colums\_paritet\_with\_errors[2] = 1; }  if ((array\_paritet\_colums[3] == 0 && array\_paritet\_colums\_with\_one\_error[3] == 0) || (array\_paritet\_colums[3] == 1 && array\_paritet\_colums\_with\_one\_error[3] == 1))  {  colums\_paritet\_with\_errors[3] = 0;  }  else { colums\_paritet\_with\_errors[3] = 1; }  if ((array\_paritet\_colums[4] == 0 && array\_paritet\_colums\_with\_one\_error[4] == 0) || (array\_paritet\_colums[4] == 1 && array\_paritet\_colums\_with\_one\_error[4] == 1))  {  colums\_paritet\_with\_errors[4] = 0;  }  else { colums\_paritet\_with\_errors[4] = 1; }  if ((array\_paritet\_colums[5] == 0 && array\_paritet\_colums\_with\_one\_error[5] == 0) || (array\_paritet\_colums[5] == 1 && array\_paritet\_colums\_with\_one\_error[5] == 1))  {  colums\_paritet\_with\_errors[5] = 0;  }  else { colums\_paritet\_with\_errors[5] = 1; }  if ((array\_paritet\_colums[6] == 0 && array\_paritet\_colums\_with\_one\_error[6] == 0) || (array\_paritet\_colums[6] == 1 && array\_paritet\_colums\_with\_one\_error[6] == 1))  {  colums\_paritet\_with\_errors[6] = 0;  }  else { colums\_paritet\_with\_errors[6] = 1; }  if ((array\_paritet\_colums[7] == 0 && array\_paritet\_colums\_with\_one\_error[7] == 0) || (array\_paritet\_colums[7] == 1 && array\_paritet\_colums\_with\_one\_error[7] == 1))  {  colums\_paritet\_with\_errors[7] = 0;  }  else { colums\_paritet\_with\_errors[7] = 1; }  if ((array\_paritet\_colums[8] == 0 && array\_paritet\_colums\_with\_one\_error[8] == 0) || (array\_paritet\_colums[8] == 1 && array\_paritet\_colums\_with\_one\_error[8] == 1))  {  colums\_paritet\_with\_errors[8] = 0;  }  else { colums\_paritet\_with\_errors[8] = 1; }  if ((array\_paritet\_colums[9] == 0 && array\_paritet\_colums\_with\_one\_error[9] == 0) || (array\_paritet\_colums[9] == 1 && array\_paritet\_colums\_with\_one\_error[9] == 1))  {  colums\_paritet\_with\_errors[9] = 0;  }  else { colums\_paritet\_with\_errors[9] = 1; }  for (int i = 0; i < colums\_paritet\_with\_errors.Length; i++)  {  Console.Write(colums\_paritet\_with\_errors[i] + " ");  if(colums\_paritet\_with\_errors[i] == 1)  {  error\_in\_colums = i;  }  }  #endregion  Console.WriteLine('\n' + "Следовательно ошибка в ");  Console.WriteLine("Строке " + error\_in\_rows);  Console.WriteLine("Колонке " + error\_in\_colums);  Console.WriteLine("Исправленная матрица");  for (int i = 0; i < array\_message\_with\_one\_error.Length; i++)  {  if (i == 10)  {  Console.WriteLine("");  }  if(i == (error\_in\_colums + error\_in\_rows))  {  if (error\_in\_rows == 1)  {  if (array\_message\_with\_one\_error[i + 9] == 1) { array\_message\_with\_one\_error[i + 9] = 0; }  else { array\_message\_with\_one\_error[i + 9] = 1; }  }  else  {  if (array\_message\_with\_one\_error[i] == 1) { array\_message\_with\_one\_error[i] = 0; }  else { array\_message\_with\_one\_error[i] = 1; }  }    }  Console.Write(array\_message\_with\_one\_error[i] + " ");  }  #endregion  }  }  } |

Листинг 1 **–** Код файла Program.cs



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



*Рис. 5.5* **-** Результат выполнения программы



*Рис. 5.6* **-** Результат выполнения программы

**Ответы на вопросы:**

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

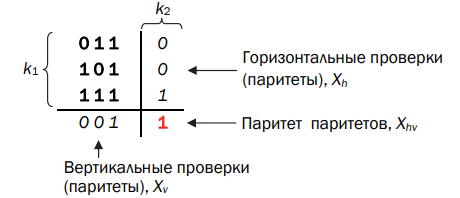
Итеративные помехоустойчивые коды относятся к классу кодов произведения. Кодом произведения двух исходных (базовых) помехоустойчивых кодов n1и n2называется такой многомерный помехоустойчивый код N=n1\*n2, кодовыми последовательностями которого являются все двумерные таблицы со строками кода n1и столбцами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

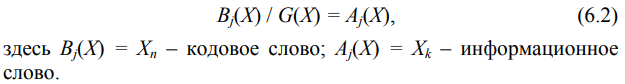
**Основные свойства ЦК:**

* 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.

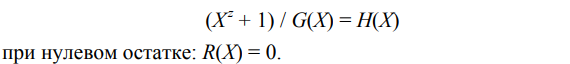
Формирование разрешенных кодовых комбинаций ЦК Bj(X) основано на предварительном выборе **порождающего (генераторного или образующего) полинома G(X)**, который обладает важным отличительным признаком: **все комбинации Bj(X) делятся на порождающий полином G(X) без остатка:**



**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**

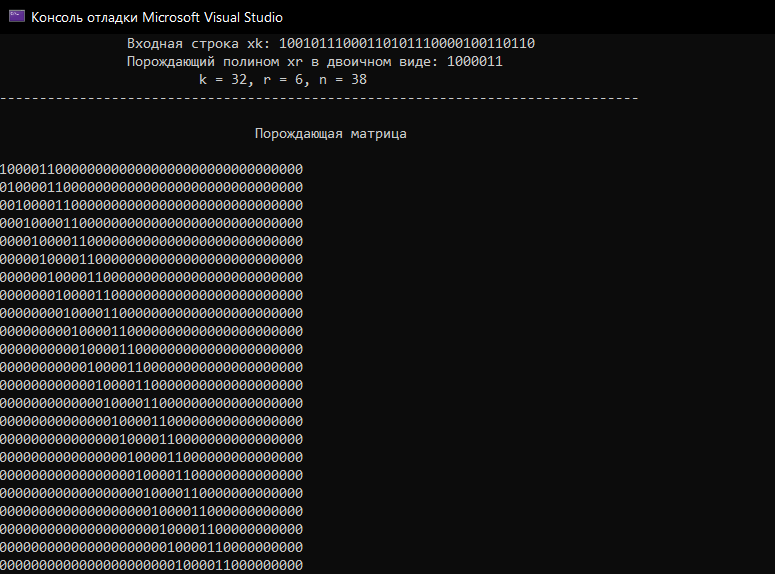
1. Задание выполняется по указанию преподавателя в соответствии с вариантом из табл. 6.2, из которого выбирается порождающий полином ЦК, а по значению соответствующего ему значения r – длина k информационного слова Xk. Полагаем, что каждый полином соответствует коду, обнаруживающему и исправляющему одиночные 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.

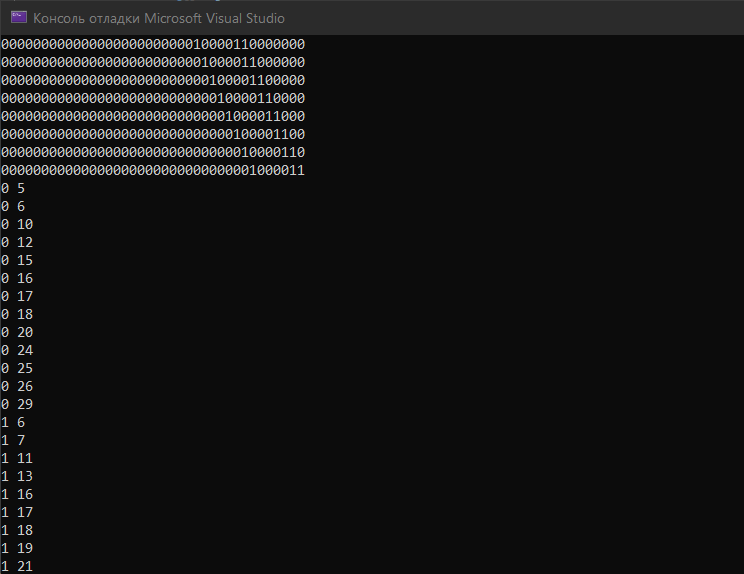
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab6  {  class Program  {  static void Main(string[] args)  {  string Xk = "10010111000110101110000100110110";  string Xr = "1000011";  int k = Xk.Length;  int n = 38;  int r = 6;  int error;  int[] masXk = new int[k];  Operations.StringInArray(masXk, Xk);  int[] masXr = new int[Xr.Length];  Operations.StringInArray(masXr, Xr);  Console.WriteLine("\t\tВходная строка xk: " + Xk);  Console.WriteLine("\t\tПорождающий полином xr в двоичном виде: " + Xr);  Console.WriteLine("\t\t\t k = {0}, r = {1}, n = {2}", k, r, n);  Console.WriteLine("--------------------------------------------------------------------------------");  int[,] generationMatrix = new int[k, n];  Operations.CreateGenerationMatrix(generationMatrix, masXr, k, n);  Console.WriteLine("\n\t\t\t\tПорождающая матрица\n");  Operations.PrintMatrix(generationMatrix, k, n);  Operations.CreateMatrixCanon(generationMatrix, k, n);  Console.WriteLine("--------------------------------------------------------------------------------");  Console.WriteLine("\n\t\t\t\tКаноническая матрица\n");  Operations.PrintMatrix(generationMatrix, k, n);  int[,] checkMatrix = new int[n, r];  Operations.CreateMatrixForCheck(checkMatrix, generationMatrix, k, n);  Console.WriteLine("--------------------------------------------------------------------------------");  Console.WriteLine("\n\t\t\t\tПроверочная матрица канонической формы\n");  Operations.PrintMatrix(checkMatrix, n, r);  // Task 1.2  int[] masXn = new int[n];  Operations.ShiftR(masXn, masXk, r);  Console.WriteLine("--------------------------------------------------------------------------------");  // Task 2  Console.WriteLine("\n\t\t\t\t\tДеление\n");  Operations.SearchRes(masXn, masXr);  Console.WriteLine("Остаток:");  Operations.PrintArray(masXn);  Console.WriteLine("--------------------------------------------------------------------------------");  Console.WriteLine("\n\t\t\t\tИтоговая строка\n");  Operations.ShiftR(masXn, masXk, r);  Operations.PrintArray(masXn);  Console.WriteLine("--------------------------------------------------------------------------------");  try  {  Console.WriteLine("Введите место первой ошибки");  error = Convert.ToInt32(Console.ReadLine()) - 1;  if (masXn[error] == 1) masXn[error] = 0;  else masXn[error] = 1;  }  catch { }  Console.WriteLine("Ошибочная строка:");  Operations.PrintArray(masXn);  Operations.SearchingMistake(masXn, masXr, checkMatrix, r);  Console.WriteLine("--------------------------------------------------------------------------------");  try  {  Console.WriteLine("Место первой ошибки: ");  error = Convert.ToInt32(Console.ReadLine()) - 1;  if (masXn[error] == 1)  masXn[error] = 0;  else masXn[error] = 1;  Console.WriteLine("Место второй ошибки: ");  error = Convert.ToInt32(Console.ReadLine()) - 1;  if (masXn[error] == 1)  masXn[error] = 0;  else masXn[error] = 1;  }  catch { }  Console.WriteLine("Ошибочная строка:");  Operations.PrintArray(masXn);  Operations.SearchingMistake(masXn, masXr, checkMatrix, r);  }  }  } |

*Листинг 1* – код файла Program.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  namespace lab6  {  class Operations  {  public static int[] SearchingMistake(int[] masXn, int[] masXr, 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");  SearchRes(masXnSecond, masXr);  Console.WriteLine("\nОстаток:");  PrintArray(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;  }  }  Console.WriteLine("\nИсправленная строка:");  PrintArray(masXn);  return masXn;  }  public static int[] SearchRes(int[] masXn, int[] masXr)  {  int end = masXn.Length - masXr.Length + 1;  for (int i = 0; i < end; i++)  {  if (masXn[i] == 1)  {  AdditionWithArraysMod2(masXn, masXr, i);  PrintArray(masXn);  }  }  return masXn;  }  #region Сложение массивов по модулю 2 с опр. позиции  public static int[] AdditionWithArraysMod2(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;  }  #endregion  # region Смещение на массива r  public static int[] ShiftR(int[] shiftMas, int[] mas, int r)  {  for (int i = 0; i < mas.Length; i++)  {  shiftMas[i] = mas[i];  }  return shiftMas;  }  #endregion  #region Преобразование сторки в массив  public static int[] StringInArray(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;  }  #endregion  #region Создание порождающей матрицы  public static int[,] CreateGenerationMatrix(int[,] generationMatrix, int[] mas, int k, int n)  {  for (int i = 0; i < n; i++)  {  if (i < mas.Length)  {  generationMatrix[0, i] = mas[i];  }  else  {  generationMatrix[0, i] = 0;  }  }  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;  }  #endregion  #region Приведение порождающей матрицы к каноническому виду  public static int[,] CreateMatrixCanon(int[,] generationMatrix, int k, int n)  {  for (int i = 0; i < k; i++)  {  int i2 = i + 1;  for (int j = i + 1; j < k; j++)  {  if (generationMatrix[i, j] == 1)  {  for (; i2 < k; i2++)  {  bool repeat = false;  if (generationMatrix[i2, j] == 1)  {  for (int j2 = j - 1; j2 > 0; j2--)  {  if (generationMatrix[i2, j2] == 1)  {  repeat = true;  }  }  if (repeat)  continue;  Console.WriteLine(i + " " + i2);  AddingLinesMatrixMod2(generationMatrix, i, i2, n);  i2++;  break;  }  }  }  }  }  return generationMatrix;  }  #endregion  #region Преобразование канонической матрицы в проверочную  public static int[,] CreateMatrixForCheck(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;  }  #endregion  #region Сложение строк матрицы  public static int[,] AddingLinesMatrixMod2(int[,] matrix, int str1, int str2, int lengthString)  {  for (int i = 0; i < lengthString; i++)  {  matrix[str1, i] = (matrix[str1, i] + matrix[str2, i]) % 2;  }  return matrix;  }  #endregion  public static void PrintMatrix(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]);  }  Console.WriteLine();  }  }  public static void PrintArray(int[] mas)  {  for (int i = 0; i < mas.Length; i++)  {  Console.Write(mas[i]);  }  Console.WriteLine("\n");  }  }  } |

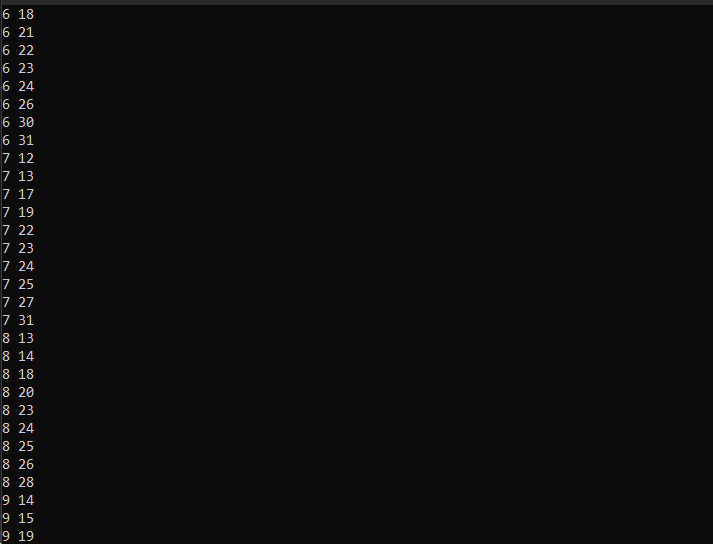
*Листинг 2* – код файла Operations.cs







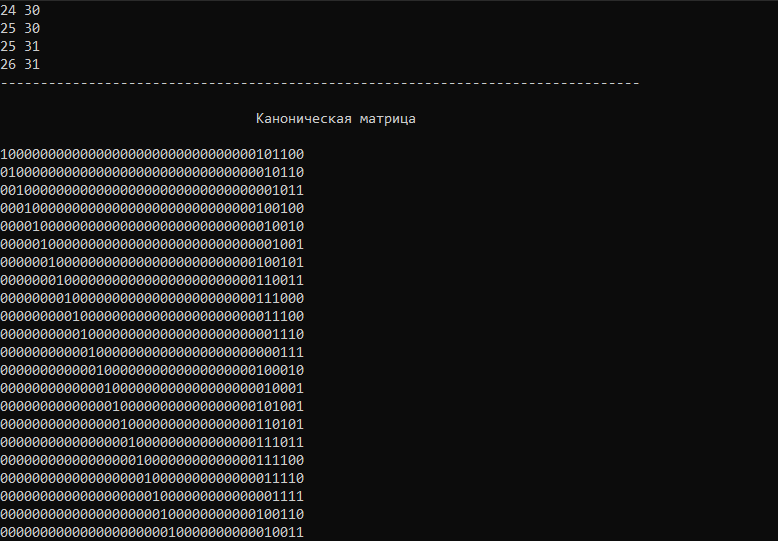


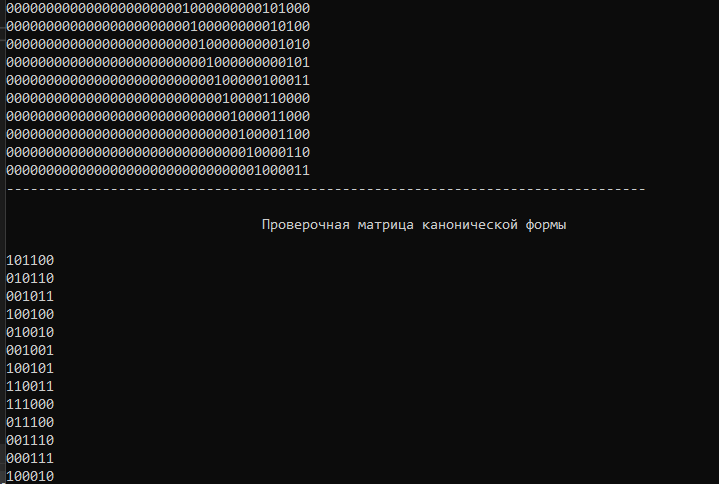
****

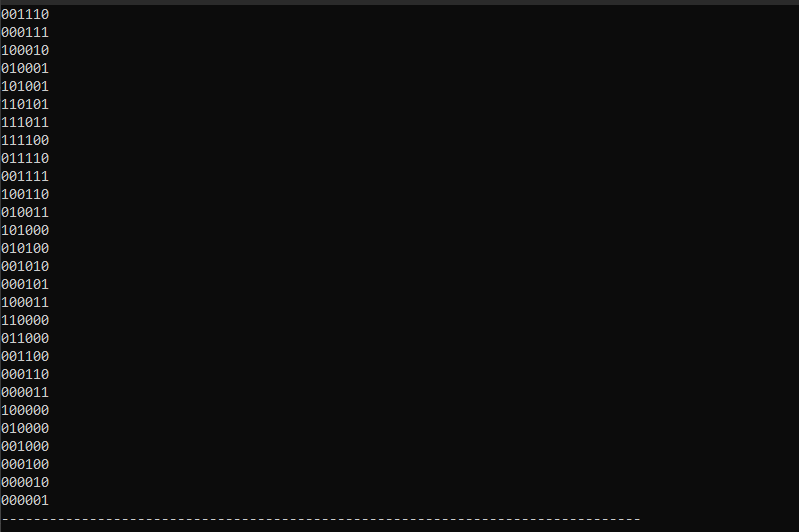
****

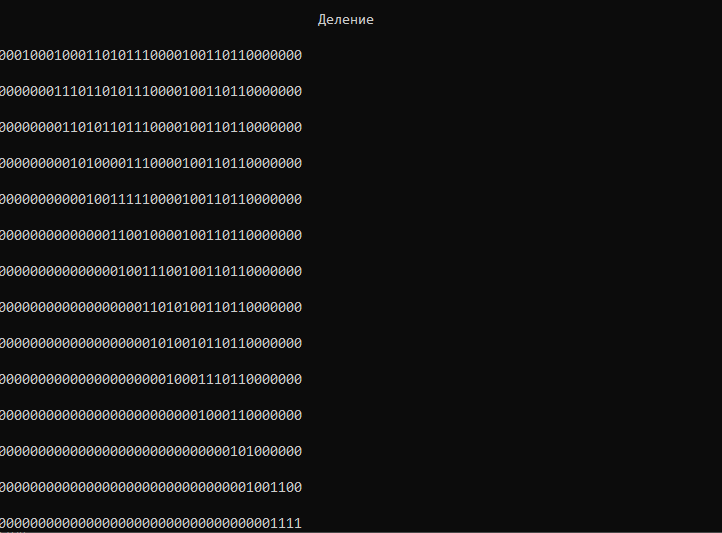
****

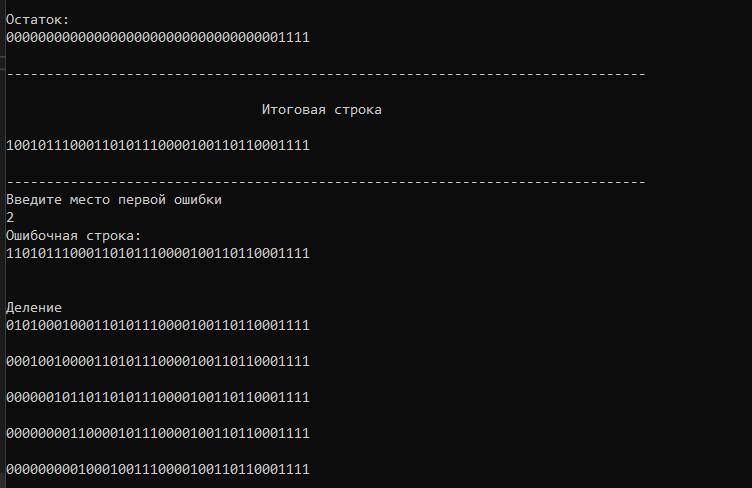
****

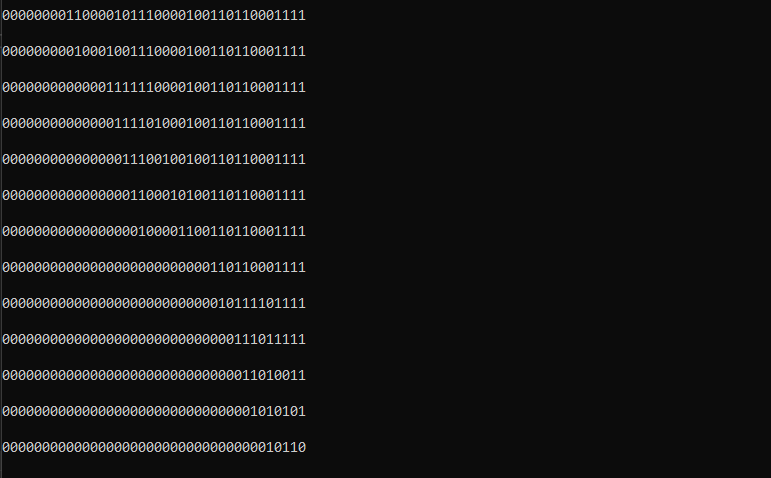
****

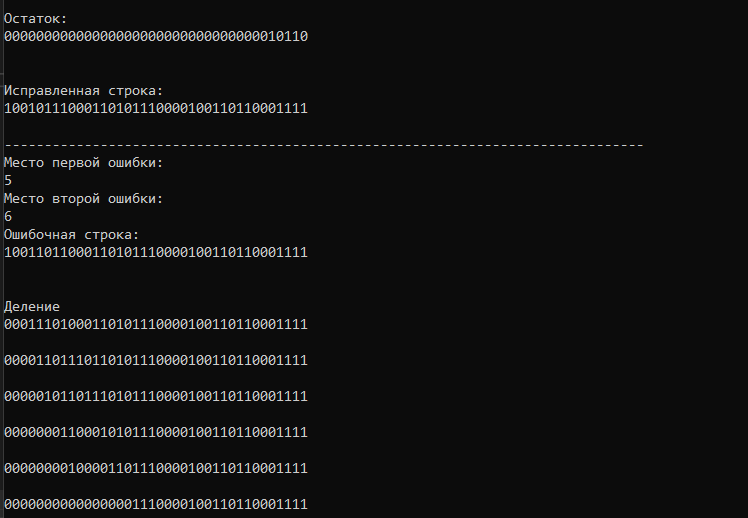
****

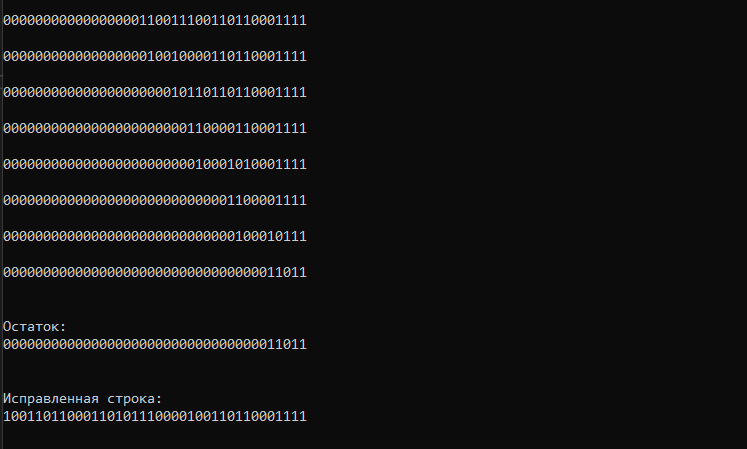
****

****

****

****

****

****

*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:** приобрела 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.

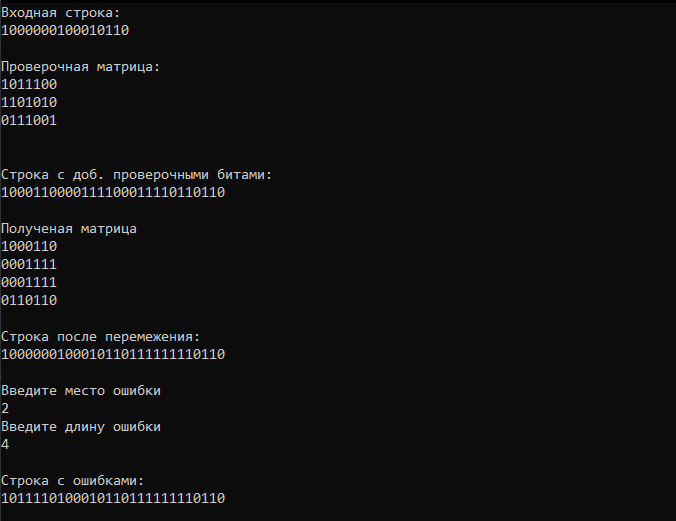
В общем случае **выбор глубины перемежения зависит от двух факторов**. С одной стороны, чем больше расстояние между соседними символами, тем большей длины пакет ошибок может быть исправлен. С другой стороны, чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше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

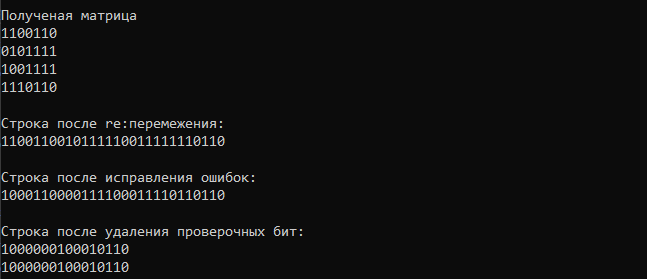
**Практическое задание**

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)  {  int lenghtK = 16; //Должна быть равна 2^n  int k = (int)(Math.Sqrt(lenghtK));  int r = HemmingLength(k);  int n = k + r;  int lenghtN = lenghtK + (r \* k);  int[] masK = new int[lenghtK];  int[] masK2 = new int[lenghtK]; //сюда запишиться итоговая строка после всех манипуляций  int[] masN = new int[lenghtK + (r \* k)];  int[,] checkMatrix = new int[n, r];  int error;  int errorLenght;  GenerationRandMasMod2(masK);  Console.WriteLine("Входная строка: ");  OutMas(masK);  Console.WriteLine("\n\nПроверочная матрица: ");  checkMatrix = CheckMatrix(k);  OutMatrixInv(checkMatrix, n, r);  AddCheckBits(masK, masN, checkMatrix);  Console.WriteLine("\n\nСтрока с доб. проверочными битами: ");  OutMas(masN);  Alternation(masN, k);  Console.WriteLine("\nСтрока после перемежения: ");  OutMas(masN);  try  {  Console.WriteLine("\n\nВведите место ошибки");  error = Convert.ToInt32(Console.ReadLine());  Console.WriteLine("Введите длину ошибки");  errorLenght = Convert.ToInt32(Console.ReadLine());  for (int i = error; i < (error + errorLenght); i++)  {  masN[i] = (masN[i] + 1) % 2;  }  }  catch { }  Console.WriteLine("\nСтрока с ошибками: ");  OutMas(masN);  ReAlternation(masN, k);  Console.WriteLine("\nСтрока после re:перемежения: ");  OutMas(masN);  SearchErrorLong(masN, checkMatrix, k);  Console.WriteLine("\n\nСтрока после исправления ошибок: ");  OutMas(masN);  RemoveCheckBits(masK2, masN, checkMatrix);  Console.WriteLine("\n\nСтрока после удаления проверочных бит: ");  OutMas(masK2);  Console.WriteLine("");  OutMas(masK);    }  static int[] SearchErrorLong(int[] masN, int[,] checkMatrix, int k)  {  int r = HemmingLength(k);  int n = r + k;  for (int i = 0; i < k; i++)  {  int[] temp = new int[n];  for (int j = 0; j < n; j++)  {  temp[j] = masN[(n \* i) + j];  }  //Получение проверочных битов каждой строки  //Console.WriteLine("\nTemp");  SearchError(temp, checkMatrix, k);  //OutMas(temp);  //Запись строки в массив, для получения одной большой строки  for (int j = 0; j < n; j++)  {  masN[i \* n + j] = temp[j];  }  }  return masN;  }  static int[] RemoveCheckBits(int[] masK, int[] masN, int[,] checkMatrix)  {  int lenghtK = masK.Length; //Должна быть равна 2^n  int lenghtN = masN.Length;  int k = (int)(Math.Sqrt(lenghtK));  int r = HemmingLength(k);  int n = k + r;  int[,] matrix = new int[k, n];  //Разбиение массива на отдельные строки  for (int i = 0; i < k; i++)  {  int[] temp = new int[n];  for (int j = 0; j < n; j++)  {  temp[j] = masN[(n \* i) + j];  }  //Запись строки в массив, для получения одной большой строки  for (int j = 0; j < k; j++)  {  masK[i \* k + j] = temp[j];  }  }  return masK;  }  static int[] AddCheckBits(int[] masK, int[] masN, int[,] checkMatrix)  {  int lenghtK = masK.Length; //Должна быть равна 2^n  int lenghtN = masN.Length;  int k = (int)(Math.Sqrt(lenghtK));  int r = HemmingLength(k);  int n = k + r;  int[,] matrix = new int[k, n];  //Разбиение массива на отдельные строки  for (int i = 0; i < k; i++)  {  int[] temp = new int[n];  for (int j = 0; j < k; j++)  {  temp[j] = masK[(k \* i) + j];  }  //Получение проверочных битов каждой строки  Sindrom(checkMatrix, temp, k);  // Console.WriteLine("");  //OutMas(temp);  //Запись строки в массив, для получения одной большой строки  for (int j = 0; j < n; j++)  {  masN[i \* n + j] = temp[j];  }  }  return masN;  }  static int[] Alternation(int[] masN, int k)  {  int r = HemmingLength(k);  int n = k + r;  int[,] matrix = new int[k, n];  //Получение матрицы  for (int i = 0, m = 0; i < k; i++)  {  for (int j = 0; j < n; j++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, k, n);  //Перемежение  for (int i = 0, m = 0; i < n; i++)  {  for (int j = 0; j < k; j++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }  static int[] ReAlternation(int[] masN, int k)  {  int r = HemmingLength(k);  int n = k + r;  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;  }  static int[] GenerationRandMasMod2(int[] mas)  {  Random rnd = new Random();  for (int i = 0; i < mas.Length; i++)  {  mas[i] = rnd.Next(2);  }  return mas;  }  //Создание пров. матрицы  static int[,] CheckMatrix(int k)  {  int r = HemmingLength(k);  int n = r + k;  double rDouble = r - 1;  int rPow = (int)(Math.Pow(2, rDouble));  int[,] mas = new int[n, r];  int[,] combinations = new int[rPow, r];  for (int i = 0; i < rPow; i++)  for (int j = 0; j < r; j++)  combinations[i, j] = 0;  //генератор бит.мн.  for (int segmentLenght = 0; segmentLenght < r - 2; segmentLenght++)  {  if (segmentLenght \* r > k) break;  for (int i = 0; i < segmentLenght + 2; i++)  {  combinations[segmentLenght \* r, i] = 1;  }  for (int segmentPositin = 1; segmentPositin < r; segmentPositin++)  {  for (int i = 0; i < r - 1; i++)  {  combinations[segmentLenght \* r + segmentPositin, i + 1] = combinations[segmentLenght \* r + segmentPositin - 1, i];  }  combinations[segmentLenght \* r + segmentPositin, 0] = combinations[segmentLenght \* r + segmentPositin - 1, r - 1];  }  if (segmentLenght == r - 3)  {  for (int i = 0; i < r; i++)  {  combinations[rPow - 1, i] = 1;  }  }  }  for (int i = 0; i < k; i++)  for (int j = 0; j < r; j++)  mas[i, j] = combinations[i, j];  for (int i = 0; i < r; i++)  mas[i + k, i] = 1;  return mas;  }  //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;  }  //Нахождение ошибок  static int[] SearchError(int[] mas, int[,] checkMatrix, int k)  {  int r = HemmingLength(k);  int n = r + k;  int[] beforeSindrom = new int[r];  //запоминаем проверочные биты  for (int i = k; i < n; i++)  {  beforeSindrom[i - k] = mas[i];  }  mas = Sindrom(checkMatrix, mas, k);  //Складываем синдром по модулю два  for (int i = k, j = 0; i < n; i++)  {  if (beforeSindrom[i - k].Equals(mas[i]))  {  mas[i] = 0;  j++;  //если сумма по модулю два все пров. бит равна нулю  if (j == r)  {  for (int l = k; l < n; l++)  {  mas[l] = beforeSindrom[l - k];  }  return mas;  }  }  else  {  mas[i] = 1;  }  }  for (int i = 0; i < n; i++)  {  int l = 0;  for (int j = 0; j < r; j++)  {  if (checkMatrix[i, j].Equals(mas[j + k])) l++;  }  if (l == r)  {  mas[i] = (mas[i] + 1) % 2;  }  }  //OutMas(mas);  mas = Sindrom(checkMatrix, mas, k);  return mas;  }  static void OutMas(int[] mas)  {  for (int i = 0; i < mas.Length; i++)  {  Console.Write(mas[i]);  }  }  //вывод матрицы  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 OutMatrixInv(int[,] matrix, int k, int n)  {  for (int j = 0; j < n; j++)  {  for (int i = 0; i < k; i++)  {  Console.Write(matrix[i, j]);  }  Console.WriteLine();  }  }  }  } |

*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:** приобрела 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.

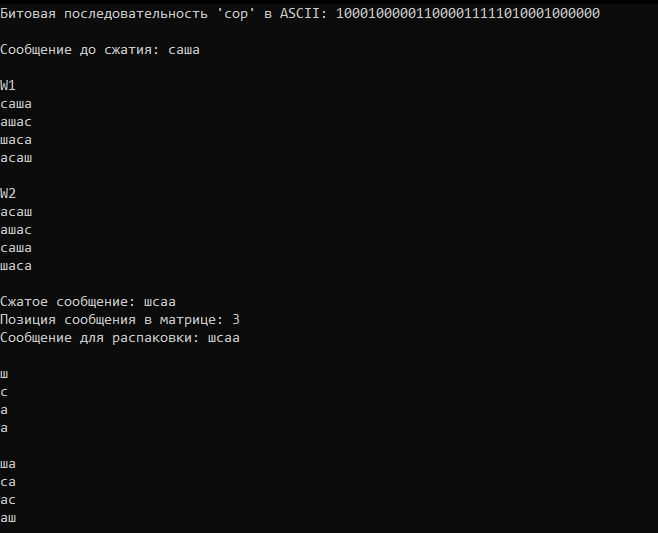
Можно использовать любой из известных методов сортировки символов массива.

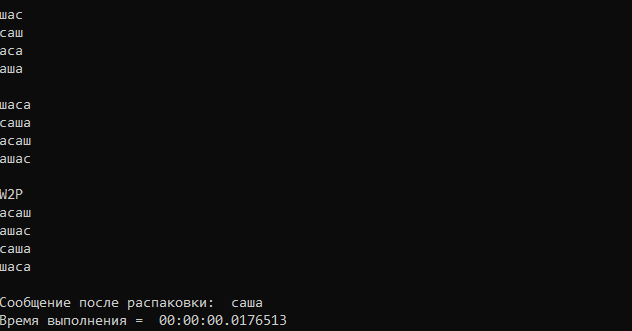
Выполнить качественный сравнительный анализ длительности процессов прямого и обратного преобразований в зависимости от длины блока данных.

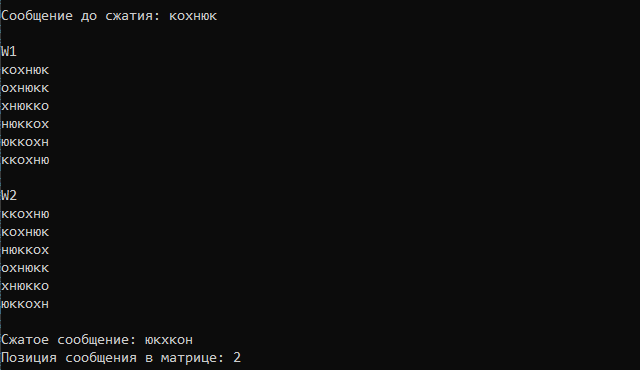
1. Перевести первые 3 символа из блока данных, указанного в варианте таблицы, в бинарную последовательность в соответствии с кодами ASCII. Выполнить прямое и обратное преобразование. Оценить время прямого и обратного преобразований.
2. Результаты оформить в виде отчета по установленным правилам.

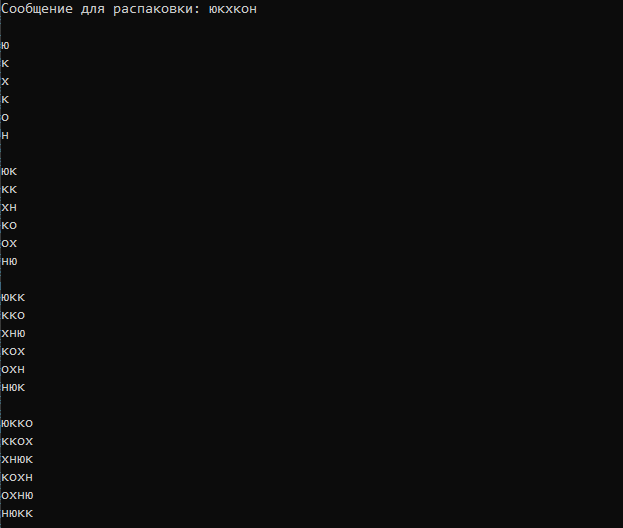
|  |
| --- |
| using System;  using System.Linq;  using System.Text;  using System.Diagnostics;  namespace lab8  {  class Program  {  static string[] CreateMatrixByString(string message)  {  string[] messageMatrix = new string[message.Count()];  for (int i = 0; i < message.Count(); i++)  {  messageMatrix[i] = message;  message = message.Substring(1) + message[0];  }  return messageMatrix;  }  static void ShowMatrix(string[] matrix)  {  foreach (var row in matrix)  {  Console.WriteLine(row);  }  Console.WriteLine();  }  static string GetLastMatrixColumn(string[] matrix)  {  string lastColumn = "";  foreach (var row in matrix)  {  lastColumn += row[row.Length - 1];  }  return lastColumn;  }  static int GetInitMessagePlace(string initMessage, string[] matrix)  {  int initMessagePlace = -1;  for (int i = 0; i < matrix.Count(); i++)  {  if (matrix[i] == initMessage)  {  return i;  }  }  return initMessagePlace;  }  static string[] AddMessageToMatrixFromLeft(string message, string[] matrix)  {  for (int i = 0; i < matrix.Length; i++)  {  matrix[i] = message[i] + matrix[i];  }  return matrix;  }  static string[] SortMatrix(string[] matrix)  {  return (matrix.OrderBy(x => x).ToArray());  }  static string[] CreateDecodingMatrix(string message)  {  string[] messageMatrix = new string[message.Length];  for (int i = 0; i < message.Length; i++)  {  messageMatrix = AddMessageToMatrixFromLeft(message, messageMatrix);  ShowMatrix(messageMatrix);  messageMatrix = SortMatrix(messageMatrix);  }  return messageMatrix;  }  static void Main(string[] args)  {  string initialMessageBytes = "сор";  byte[] bytes = Encoding.ASCII.GetBytes(initialMessageBytes);  int integ;  string str = "";  for (int i = 0; i < bytes.Length; i++)  {  integ = bytes[0];  str += Convert.ToString(integ, 2);  }  str = "100010000011000011111010001000000";  Console.WriteLine("Битовая последовательность 'пес' в ASCII: " + str);  Console.WriteLine();  string[] initMessages = new string[] { "саша", "кохнюк", "песнетворчество", str };  foreach (string initMessage in initMessages)  {  Stopwatch sw = new Stopwatch();  sw.Start();  Console.WriteLine("Сообщение до сжатия: " + initMessage);  Console.WriteLine();  //Encoding  string[] W1 = CreateMatrixByString(initMessage);  Console.WriteLine("W1");  ShowMatrix(W1);  string[] W2 = (W1.OrderBy(x => x).ToArray());  Console.WriteLine("W2");  ShowMatrix(W2);  string encodedMessage = GetLastMatrixColumn(W2) + (GetInitMessagePlace(initMessage, W2));  Console.WriteLine("Сжатое сообщение: " + GetLastMatrixColumn(W2));  Console.WriteLine("Позиция сообщения в матрице: " + (GetInitMessagePlace(initMessage, W2) + 1));  //Decoding  string[] W2Dec = new string[encodedMessage.Length - (encodedMessage.Length - initMessage.Length)];  string gettedMessage = encodedMessage.Substring(0, encodedMessage.Length - (encodedMessage.Length - initMessage.Length));  Console.WriteLine("Сообщение для распаковки: " + gettedMessage);  Console.WriteLine();  W2Dec = CreateDecodingMatrix(gettedMessage);  Console.WriteLine("W2Р");  ShowMatrix(W2Dec);  int numberOfInitialMessage = Int32.Parse((encodedMessage.Substring(initMessage.Length, (encodedMessage.Length - initMessage.Length))));  Console.WriteLine("Сообщение после распаковки: " + W2Dec[numberOfInitialMessage]);  sw.Stop();  Console.WriteLine("Время выполнения = {0}", sw.Elapsed);  Console.WriteLine("-----------------------------------------------------");  }  }  }  } |

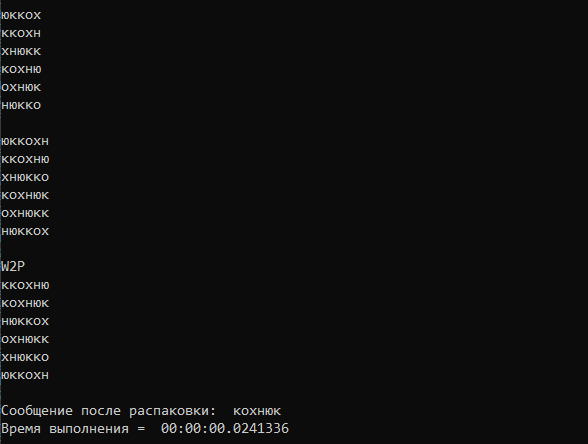
*Листинг 1* **–** код файла Program.cs

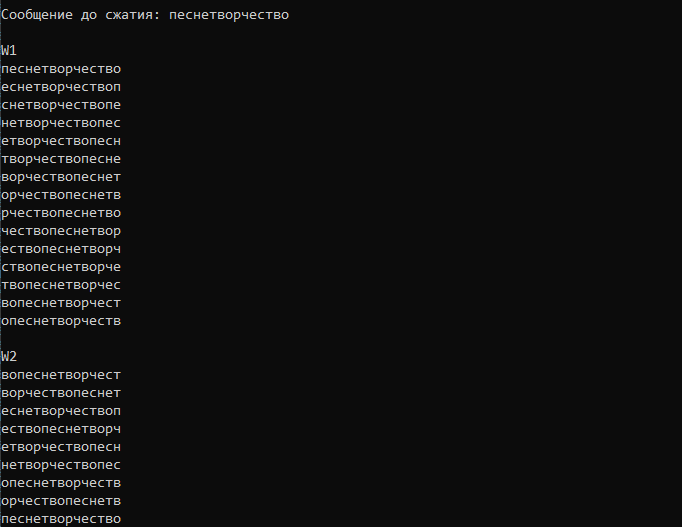
****

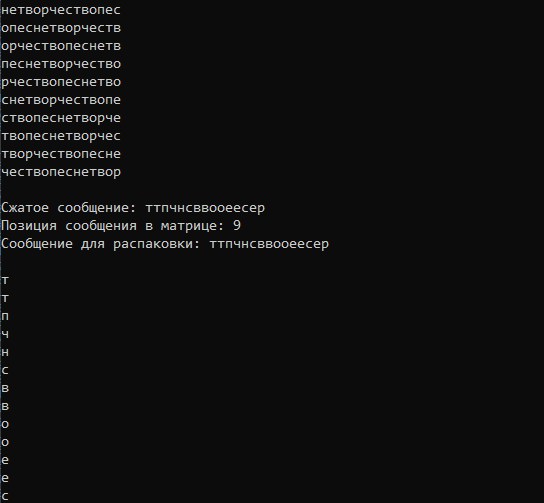
****

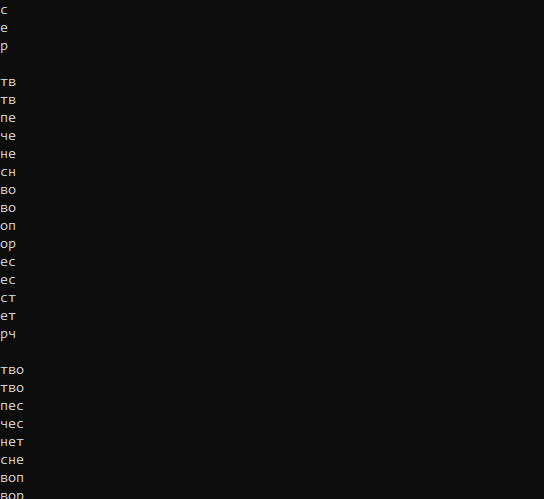
****

****

****

****

****

****

****

****

****

*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).

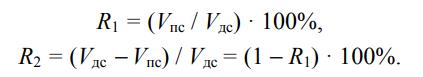
1. **Охарактеризовать основные технические характеристики процессов сжатия/распаковки и результатов.**

• 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:** приобрела 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;

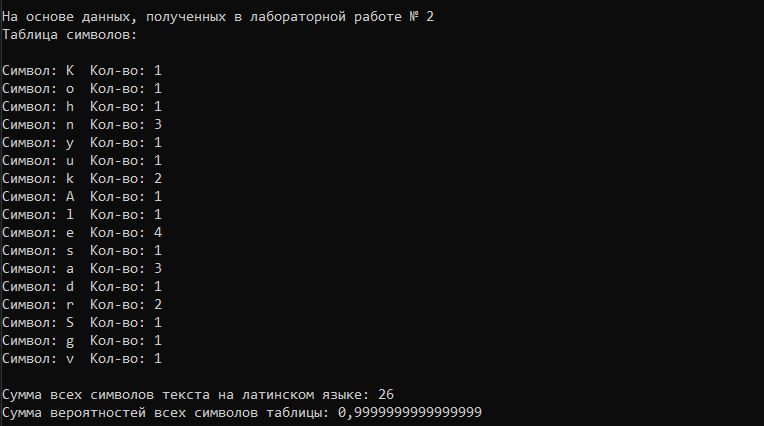
б) динамически, на основе анализа сжимаемого сообщения.

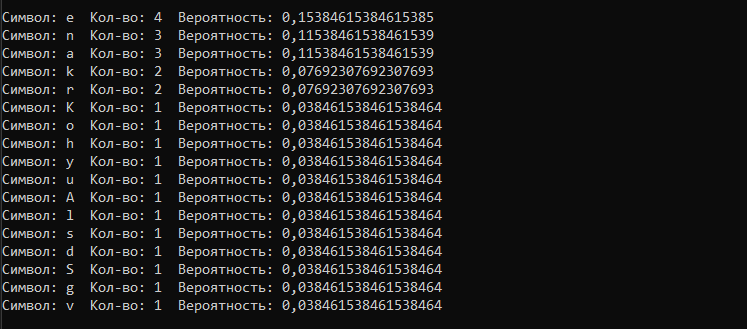
3. Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.

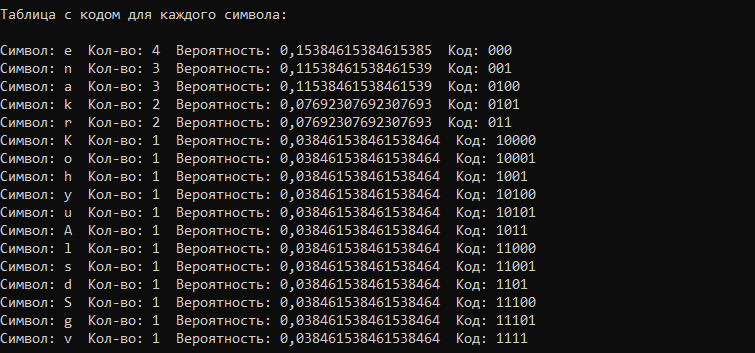
4. Результаты оформить в виде отчета по установленным правилам

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.IO;  namespace lab9  {  public class ShannonFanoSymbol  {  public char symbol;  public int count;  public double viorite;  public string code;  public ShannonFanoSymbol(char sim, int count, double vior, string code)  {  this.viorite = vior;  this.symbol = sim;  this.count = count;  this.code = code;  }  public static List<ShannonFanoSymbol> AddSymbols(List<ShannonFanoSymbol> symbols, string line)  {  foreach (var character in line)  {  if (symbols.Find(x => x.symbol == character) == null)  {  symbols.Add(new ShannonFanoSymbol(character, 1, 0.0, ""));  }  else  {  symbols.Where(x => x.symbol == character).ToList().ForEach(x => x.count++);  }  }  return symbols;  }  public static void Show(List<ShannonFanoSymbol> symbols)  {  foreach (var symbol in symbols)  {  Console.Write("Символ: {0} Кол-во: {1} ", symbol.symbol, symbol.count);  if (symbol.viorite != 0)  {  Console.Write("Вероятность: {0}", symbol.viorite);  }  if (symbol.code != "")  {  Console.Write(" Код: {0}", symbol.code);  }  Console.WriteLine();  }  Console.WriteLine();  }  }  class Program  {  static void Main(string[] args)  {  List<ShannonFanoSymbol> symbols = new List<ShannonFanoSymbol>();  using (StreamReader stream = new StreamReader(@"D:\3\Защита инфы Урбанович\lab 9\latin.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);  for (int i = 0; i < symbols.Count; i++)  {  symbols[i].viorite = symbols[i].count / symbolssum;  }  Console.WriteLine("Сумма вероятностей всех символов таблицы: " + (symbols.Sum(x => x.viorite)));  Console.WriteLine();  symbols = symbols.OrderByDescending(x => x.viorite).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 = "Kohnyuk Sasha Sergeevna";  string decodingOfFIO = "";  foreach (var charFIO in blockofFIO)  {  decodingOfFIO += (symbols.Where(x => x.symbol == charFIO).FirstOrDefault()).code;  }  Console.WriteLine("Исходное сообщение: ");  Console.WriteLine(blockofFIO);  Console.WriteLine("Сообщение после кодировки: ");  Console.WriteLine(decodingOfFIO);  Console.WriteLine("Количество битов в кодах ASCII: " + blockofFIO.Count() \* 8);  Console.WriteLine("Количество битов по таблице Шеннон-Фано: " + decodingOfFIO.Count());  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  Console.WriteLine("Декодирование: ");  Console.WriteLine();  string Encoded = "";  string FIOdecoded = "";  for (int i = 0; i < decodingOfFIO.Count(); 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 = "Kohnyuk";  symbols = ShannonFanoSymbol.AddSymbols(symbols, message);  ShannonFanoSymbol.Show(symbols);  symbolssum = symbols.Sum(x => x.count);  Console.WriteLine("Сумма всех символов текста на латинском языке: " + symbolssum);  for (int i = 0; i < symbols.Count; i++)  {  symbols[i].viorite = symbols[i].count / symbolssum;  }  Console.WriteLine("Сумма вероятностей всех символов таблицы: " + (symbols.Sum(x => x.viorite)));  Console.WriteLine();  symbols = symbols.OrderByDescending(x => x.viorite).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 = "Kohnyuk";  decodingOfFIO = "";  foreach (var charFIO in blockofFIO)  {  decodingOfFIO += (symbols.Where(x => x.symbol == charFIO).First()).code;  }  Console.WriteLine("Исходное сообщение: ");  Console.WriteLine(blockofFIO);  Console.WriteLine("Сообщение после кодировки: ");  Console.WriteLine(decodingOfFIO);  Console.WriteLine("Количество битов в кодах ASCII: " + blockofFIO.Count() \* 8);  Console.WriteLine("Количество битов по таблице Шеннон-Фано: " + decodingOfFIO.Count());  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine();  Console.WriteLine("Декодирование");  Encoded = "";  FIOdecoded = "";  for (int i = 0; i < decodingOfFIO.Count(); 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.viorite) / 2))  {  probability += symbols[counter].viorite;  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**: приобрела 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.

**Ответить на вопросы:**

**Вывод:** приобрела 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],**

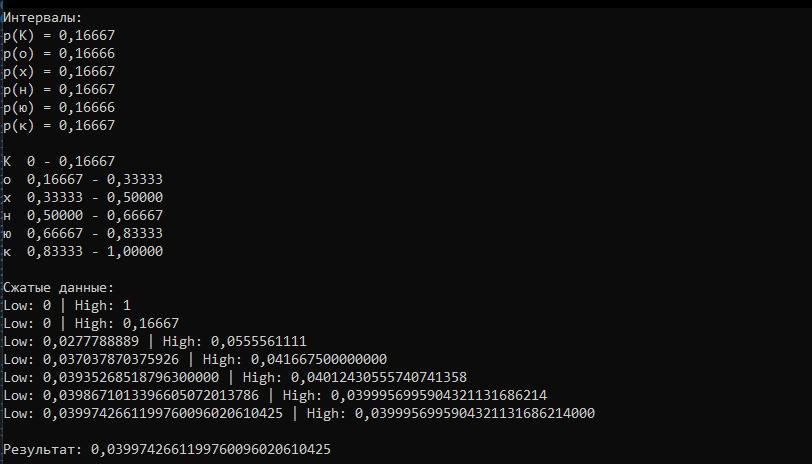
где код (i − 1) – число, анализ которого производился на предыдущем шаге – (i − 1)-м; H(αi − 1)0 и L(αi − 1)0 – соответственно верхняя и нижняя исходные границы символа сообщения, восстановленного на предыдущем шаге.

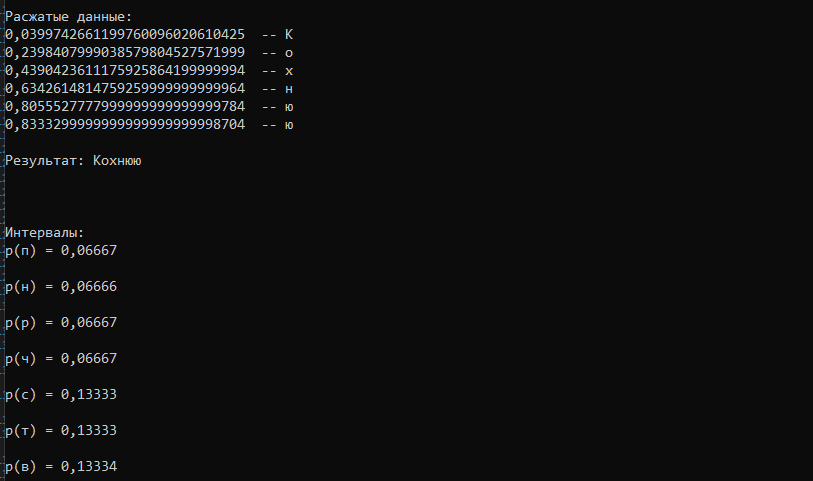
**Практическое задание**

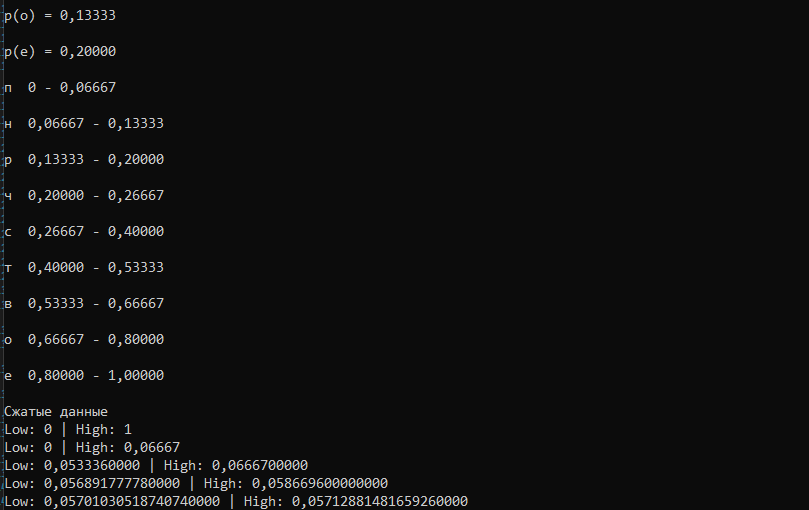
1. Разработать авторское приложение в соответствии с целью лабораторной работы.
2. С помощью приложения выполнить прямое и обратное преобразования сообщений в соответствии с таблицей. Каждый студент выполняет задание, состоящее из двух частей. Первая часть предусматривает кодирование/декодирование сообщения, 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.

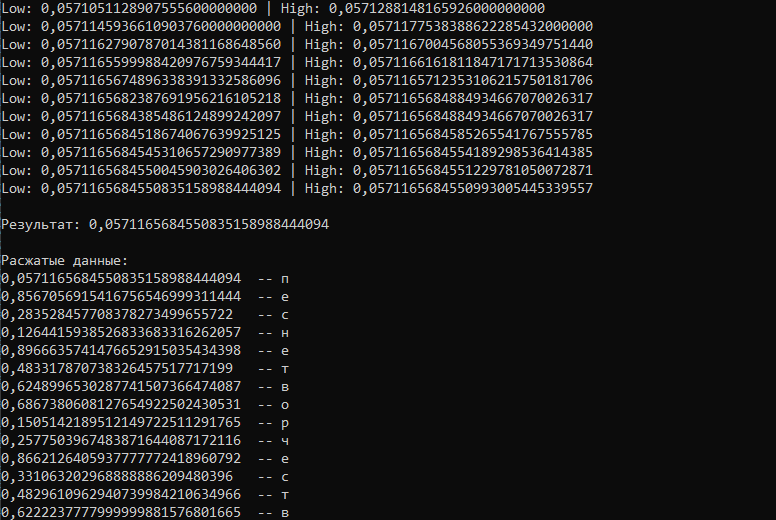
|  |
| --- |
| 1. using System.Text; 2. namespace lab11 3. { 4. public class Program 5. { 6. static void Main(string[] args) 7. { 8. { 9. string word = "Кохнюк"; 10. int wordLength = word.Length; 11. Compressor wordCompressor = new Compressor(); 12. wordCompressor.Build(word); 13. Console.WriteLine("Интервалы:"); 14. foreach (var i in wordCompressor.Nodes) 15. { 16. Console.WriteLine($"p({i.Symbol}) = {i.High - i.Low}"); 17. } 18. Console.WriteLine(); 19. foreach (var i in wordCompressor.Nodes) 20. { 21. Console.WriteLine($"{i.Symbol} {i.Low} - {i.High}"); 22. } 23. Console.WriteLine(); 24. var compressResult = wordCompressor.Compress(word); 25. Console.WriteLine("Сжатые данные:"); 26. Console.WriteLine(InfoString.Sb.ToString()); 27. Console.WriteLine($"Результат: {compressResult}\n"); 28. var decompressResult = wordCompressor.Decompress(compressResult, wordLength, wordLength / 2 + 1); 29. Console.WriteLine("Расжатые данные:"); 30. Console.WriteLine(InfoString.Sb.ToString()); 31. Console.WriteLine($"Результат: {decompressResult}"); 32. } 33. Console.WriteLine("\n\n"); 34. { 35. string word = "песнетворчество"; 36. int wordLength = word.Length; 37. Compressor wordCompressor = new Compressor(); 38. wordCompressor.Build(word); 39. Console.WriteLine("Интервалы:"); 40. foreach (var i in wordCompressor.Nodes) 41. { 42. Console.WriteLine($"p({i.Symbol}) = {i.High - i.Low}\n"); 43. } 44. foreach (var i in wordCompressor.Nodes) 45. { 46. Console.WriteLine($"{i.Symbol} {i.Low} - {i.High}\n"); 47. } 48. var compressResult = wordCompressor.Compress(word); 49. Console.WriteLine("Сжатые данные"); 50. Console.WriteLine(InfoString.Sb.ToString()); 51. Console.WriteLine($"Результат: {compressResult}\n"); 52. var decompressResult = wordCompressor.Decompress(compressResult, wordLength, wordLength / 2 + 1); 53. Console.WriteLine("Расжатые данные:"); 54. Console.WriteLine(InfoString.Sb.ToString()); 55. Console.WriteLine($"Результат: {decompressResult}"); 56. } 57. Console.ReadLine(); 58. } 59. } 60. public class Compressor 61. { 62. public List<Node> Nodes { get; set; } 63. public Dictionary<char, decimal> Frequencies { get; set; } 64. public Node ResultNode { get; set; } 65. public void Build(string source) 66. { 67. Nodes = new List<Node>(); 68. decimal inc = 1 / (decimal)source.Length; 69. Frequencies = new Dictionary<char, decimal>(); 70. for (int i = 0; i < source.Length; i++) 71. { 72. if (!Frequencies.ContainsKey(source[i])) 73. { 74. Frequencies.Add(source[i], 0); 75. } 76. Frequencies[source[i]] += inc; 77. } 78. Frequencies = Frequencies.OrderBy(x => x.Value).ToDictionary(x => x.Key, y => y.Value); 79. decimal low = 0; 80. foreach (var item in Frequencies) 81. { 82. Nodes.Add(new Node { Symbol = item.Key, Low = Math.Round(low, 5), High = Math.Round(low + item.Value, 5) }); 83. low += item.Value; 84. } 85. } 86. public decimal Compress(string source) 87. { 88. InfoString.Sb = new StringBuilder(); 89. ResultNode = new Node { Symbol = '\*', High = 1, Low = 0 }; 90. foreach (var item in source) 91. { 92. decimal oldHigh = ResultNode.High; 93. decimal oldLow = ResultNode.Low; 94. InfoString.Sb.Append(ResultNode.ToString()).Append(Environment.NewLine); 95. ResultNode.Symbol = '\*'; 96. ResultNode.High = oldLow + (oldHigh - oldLow) \* Nodes.Find(x => x.Symbol == item).High; 97. ResultNode.Low = oldLow + (oldHigh - oldLow) \* Nodes.Find(x => x.Symbol == item).Low; 98. } 99. InfoString.Sb.Append(ResultNode.ToString()).Append(Environment.NewLine); 100. return ResultNode.Low; 101. } 102. public string Decompress(decimal compress, int leng, int t) 103. { 104. StringBuilder sb = new StringBuilder(); 105. InfoString.Sb = new StringBuilder(); 106. for (int i = 0; i < leng; i++) 107. { 108. char symbol = Nodes.Find(x => Math.Round(compress, t) >= x.Low && Math.Round(compress, t) < x.High).Symbol; 109. InfoString.Sb.Append(compress.ToString() + $"\t-- {symbol}").Append(Environment.NewLine); 110. sb.Append(symbol); 111. Node tempNode = Nodes.Find(x => x.Symbol == symbol); 112. compress = (compress - tempNode.Low) / (tempNode.High - tempNode.Low); 113. } 114. return sb.ToString(); 115. } 116. } 117. public class Node 118. { 119. public char Symbol { get; set; } 120. public decimal High { get; set; } 121. public decimal Low { get; set; } 122. public override string ToString() 123. { 124. return string.Format("Low: {0} | High: {1}", Low.ToString(), High.ToString()); 125. } 126. } 127. public class InfoString 128. { 129. public static StringBuilder Sb { get; set; } 130. } 131. } |

**Листинг 1 –** код файла Program.cs

****

****

****

****

****

*Рис. 1* – результат выполнения программы

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

# **Лабораторная работа №12**

**Тема «Основы теории чисел и их использование в криптографии»**

**Цель:** приобретение практических навыков выполнения операций с числами для решения задач в области криптографии и разработка приложений для автоматизации этих операций.

**Теоретические сведения**

В основе современной криптографии лежит теория чисел.

Теория чисел, или высшая арифметика, – раздел математики, изучающий натуральные числа и иные похожие величины.

**Определение 1.** Множество всех целых чисел (обозначим буквой Z) есть набор всех действительных чисел без дробной части: {..., –3, –2, –1, 0, 1, 2, 3, ...}.

**Определение 2.** Натуральные числа являются подмножеством целых чисел и образуют множество N: {1, 2, 3, ...}.

**Определение 3**. Делимость – одно из основных понятий теории чисел. Если для некоторого целого числа a и натурального числа b существует целое число q, при котором bq = a, то говорят, что число a делится на b. В этом случае b называется делителем числа a, а a называется кратным числу b. При этом используются следующие обозначения:

a ⋮ b – a делится на b, или b | a – b делит a.

Из последнего определения следует, что:

• любое натуральное число является делителем нуля;

• единица является делителем любого целого числа;

• любое натуральное число является делителем самого себя.

**Определение 4.** Делитель a называется собственным делителем числа b, если 1 < |a| < |b|, и несобственным – в противном случае.

**Определение 5.** Всякое целое число а можно представить с помощью положительного целого числа b равенством вида а = bq + r, 0 ≤ r ≤ b. Число q называется неполным частным, а число r – остатком отделения а на b.

**Если число не имеет делителей, кроме самого себя и единицы, то оно называется простым, а если у числа есть еще делители, то составным.**

**Определение 6.** Натуральное число n называется простым, если n > 1 и не имеет положительных делителей, отличных от 1 и n.

Простое число не делится без остатка ни на одно другое число.

**Основная теорема арифметики**. Всякое натуральное число n, кроме 1, можно представить как произведение простых множителей: n = p1p2p3...pz, z > 1.

**решето Эратосфена –** алгоритм для поиска простых чисел.

**Простое число** – число, которое делится на цело только на себя и единицу.

Есть массив чисел, и мы рассматриваем первый элемент. Далее вычеркиваем все числа, которые нацело делятся на этот элемент.

**Алгоритм Евклида – используется для нахождения НОД двух целых положительных чисел.**

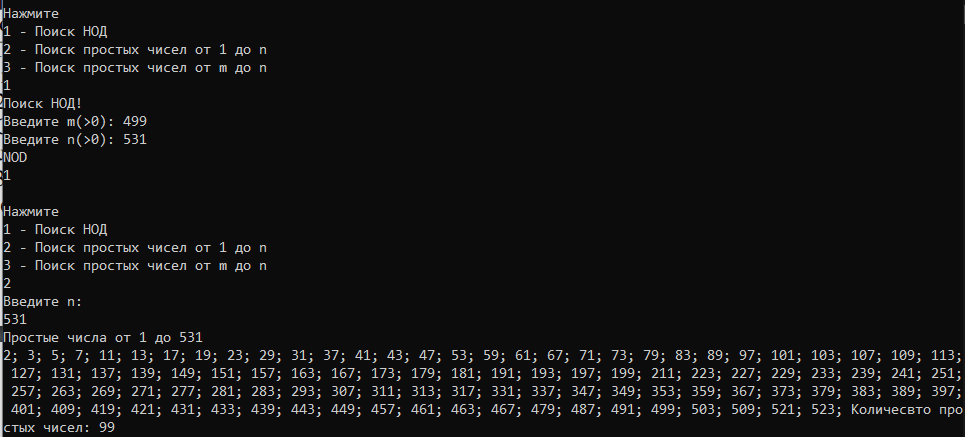
**Расширенный** **алгоритм** **Евклида** — **это** **расширение** **алгоритма** **Евклида**, которое вычисляет кроме наибольшего общего делителя (НОД) целых чисел a и b ещё и коэффициенты соотношения Безу, то есть целые x и y

**Практическое задание**

1. Используя L\_PROST, найти все простые числа в интервале [2, n]. Значение n соответствует варианту из табл. 1.2, указанному преподавателем. Подсчитать количество простых чисел в указанном интервале. Сравнить это число с n/ln(n) (см. выше пример 15).
2. Повторить п. 1 для интервала [m, n]. Сравнить полученные результаты с «ручными» вычислениями, используя «решето Эратосфена»
3. Записать числа m и n в виде произведения простых множителей (форма записи – каноническая).
4. Проверить, является ли число, состоящее из конкатенации цифр m ǀǀ n (табл. 1.2), простым.
5. Найти НОД (m, n).
6. Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции: • вычислять НОД двух либо трех чисел; • выполнять поиск простых чисел.
7. С помощью созданного приложения выполнить задания по условиям п. 1 и 2.

|  |
| --- |
| 1. using System; 2. using System.Collections.Generic; 3. namespace lab12 4. { 5. class Program 6. { 7. static void Main(string[] args) 8. { 9. int c = 5;  12. while (c == 5) 13. { 14. Console.WriteLine(); 15. Console.WriteLine("Нажмите"); 16. Console.WriteLine("1 - Поиск НОД"); 17. Console.WriteLine("2 - Поиск простых чисел от 1 до n "); 18. Console.WriteLine("3 - Поиск простых чисел от m до n "); 20. string selection = Console.ReadLine(); 21. switch (selection) 22. { 23. case "1": 24. Console.WriteLine("Поиск НОД!"); 25. Console.Write("Введите m(>0): "); 26. int a = Convert.ToInt32(Console.ReadLine()); 27. Console.Write("Введите n(>0): "); 28. int b = Convert.ToInt32(Console.ReadLine()); 29. int result = Nod(a, b); 30. Console.WriteLine("NOD"); 31. Console.WriteLine(result); 32. break; 33. case "2": 34. Console.WriteLine("Введите n: "); 35. int z = Convert.ToInt32(Console.ReadLine()); 36. Simple(z); 37. break; 38. case "3": 39. Console.WriteLine("Введите n: "); 40. int q = Convert.ToInt32(Console.ReadLine()); 41. Console.WriteLine("Введите m: "); 42. int x = Convert.ToInt32(Console.ReadLine()); 43. Simple2(q, x); 44. break; 45. default: 46. Console.WriteLine("Вы нажали неизвестную букву"); 47. break; 48. } 49. } 50. } 51. static int Nod(int a, int b) 52. { 53. if (a != 0 && b != 0) 54. { 55. if (a > b) 56. { 57. a = a % b; 58. } 59. else 60. { 61. b = b % a; 62. } 63. return Nod(a,b); 64. } 65. else 66. { 67. return a + b; 68. } 69. } 71. static void Simple (int z) 72. { 73. List<int> num = new List<int> { }; 74. for (int i = 2; i <= z; i++) 75. { 76. num.Add(i); 77. } 78. for (int i = 0; i < num.Count; i++) 79. { 80. for (int j = 2; j < z; j++) 81. num.Remove(num[i] \* j); 82. } 83. if (num.Count != 0) 84. { 85. Console.WriteLine("Простые числа от 1 до " + z); 86. foreach (int w in num) 87. { 88. Console.Write(w); 89. Console.Write("; "); 90. } 91. Console.WriteLine("Количесвто простых чисел: " + num.Count); 92. } 93. else 94. { 95. Console.WriteLine("Простых чисел в данном диапозоне нет!!!"); 96. } 97. } 99. static void Simple2(int q, int z) 100. { 101. List<int> num = new List<int> { }; 102. for (int i = 2; i <= z; i++) 103. { 104. num.Add(i); 105. } 106. for (int i = 0; i < num.Count; i++) 107. { 108. for (int j = 2; j < z; j++) 109. num.Remove(num[i] \* j); 110. } 111. if (num.Count != 0) 112. { 113. Console.WriteLine("Простые числа от "+ q + " до " + z); 114. foreach (int w in num) 115. { 116. if (w > q) 117. { 118. Console.Write(w); 119. Console.Write("; "); 120. } 121. } 123. } 124. else 125. { 126. Console.WriteLine("Простых чисел в данном диапозоне нет!!!"); 127. } 128. } 129. } 130. } |

*Листинг 1* – код файла Program.cs





*Рис. 12.1* – Результат выполнения программы

**Ответы на вопросы:**

1. **Дать определение понятий: целое число, натуральное число, делимость чисел, собственный делитель, НОД.**

**Целое** **число** (от латинского integer, означающего "**целое**")[a] в разговорной речи определяется как **число**, которое может быть записано без дробной составляющей.

Натуральные числа являются подмножеством целых чисел и образуют множество N: {1, 2, 3, ...}.

Делимость – одно из основных понятий теории чисел. Если для некоторого целого числа a и натурального числа b существует целое число q, при котором bq = a, то говорят, что число a делится на b. В этом случае b называется делителем числа a, а a называется кратным числу b.

Делитель a называется собственным делителем числа b, если 1 < |a| < |b|, и несобственным – в противном случае.

Наибольшее целое число, которое делит без остатка числа a и b, называется наибольшим общим делителем этих чисел – **НОД (a, b).**

1. **Сформулировать основную теорему арифметики. Представить примеры ее применения.**

Каждое натуральное число *n*, за исключением единицы, раскладывается в произведение простых сомножителей, причем единственным образом с точностью до порядка следования сомножителей:

https://konspekta.net/lektsiiorgimg/baza13/674373247956.files/image005.gif ,

где *p*1, …, *pk* – простые числа.

1. **Пояснить сущность проблемы факторизации и ее связь с прикладной криптографией.**

Сложность решения задачи разложения больших чисел на простые сомножители, известной как «проблема фактори-зации», определяет криптостойкость некоторых алгоритмов асимметричной криптографии, в частности алгоритма RSA. Свойство 4. Любое четное число, большее 2, представимо в виде суммы двух простых чисел, а любое нечетное, большее 5, представимо в виде суммы трех простых чисел.

**Выводы**: приобрела практическе навыки выполнения операций с числами для решения задач в области криптографии и разработки приложений для автоматизации этих операций.

# **Лабораторная работа №13**

**Тема «Исследование криптографических шифров на основе подстановки (замены) символов»**

**Цель:** изучение и приобретение практических навыков разработки и использования приложений для реализации подстановочных шифров.

**Теоретические сведения**

**Сущность подстановочного шифрования состоит в том, что исходный текст (из множества М) и зашифрованный текст (из множества С) основаны на использовании одного и того же или разных алфавитов, а тайной или ключевой информацией является алгоритм подстановки.**

Если исходить из того, что используемые алфавиты являются конечными множествами, то в общем случае каждой букве ax алфавита AM (ax ∈ AM) для создания сообщения Мi (Мi ∈ M) соответствует буква ay или множество букв {АхС} для создания шифртекста Сi (Сi ∈ С). Важно, чтобы во втором случае любые два множества (например, {АхС}b и {АхС}n, b ≠ n, 1 ≤ b, n, x, y ≤ N, N – мощность алфавита), используемые для замены разных букв открытого текста, не пересекались:

{АхС}b ∩ {АхС}n = 0.

Если в сообщении Мi содержится несколько букв ax, то каждая из них заменяется на символ ay либо на любой из символов {АхС}. За счет этого с помощью одного ключа можно сгенерировать различные Сi для одного и того же Мi. Так как множества {АхС}b и {АхС}n попарно не пересекаются, то по каждому символу Сi можно однозначно определить, какому множеству он принадлежит, и, следовательно, какую букву открытого сообщения Мi он заменяет. В силу этого открытое сообщение восстанавливается из зашифрованного однозначно.

Приведенные утверждения справедливы для следующих типов подстановочных шифров:

* моноалфавитных (шифры однозначной замены или простые подстановочные);
* полиграммных;
* омофонических (однозвучные шифры или шифры многозначной замены);
* полиалфавитных.

**Многоалфавиитные:**

В данных шифрах операция замены производится раздельно над каждым одиночным символом сообщения Мi. Для наглядной демонстрации шифра простой замены достаточно выписать под заданным алфавитом тот же алфавит, но в другом порядке или, например, со смещением. Записанный таким образом алфавит называют алфавитом замены.

Максимальное количество ключей для любого шифра этого вида не превышает N!, где N – количество символов в алфавите.

Для математического описания криптографического преобразования предполагаем, что зашифрованная буква ay (ay ∈ Сi), соответствующая символу aх (aх ∈ Мi), находится на позиции

y ≡ x + k mod N,

где x, y – индекс (порядковый номер, начиная с 0) символа в используемом алфавите; k – ключ. Для расшифрования сообщения Сi необходимо произвести расчеты, обратные выражению (2.1), т. е.

х ≡ у – k mod N.

Соотношениям (2.1) и (2.2) соответствует классический шифр подстановки – **шифр Цезаря**.

Существуют различные модификации шифра Цезаря, в частности, Атбаш и лозунговый шифр.

**Атбаш** - этот шифр состоит в замене каждой буквы другой буквой, которая находится в алфавите на таком же расстоянии от конца алфавита, как оригинальная буква – от начала. Например, в русском алфавите буква А заменяется на Я, буква Б – на Ю и т. д.

**Одним из существенных недостатков моноалфавитных шифров является их низкая криптостойкость. Зачастую метод криптоанализа базируется на частоте встречаемости букв исходного текста.**

Применяя одновременно операции сложения и умножения по модулю n над элементами множества (индексами букв алфавита), можно получить систему подстановок, которую называют **аффинной системой подстановок Цезаря.**

**Полиграммные:**

В таких шифрах одна подстановка соответствует сразу нескольким символам исходного текста.

Первым известным шифром этого типа является шифр **Порты**. Шифр представляется в виде таблицы. Наверху горизонтально и слева вертикально записывается стандартный алфавит. В ячейках таблицы записываются числа в определенном порядке. Одним из возможных вариантов такой таблицы для алфавита русского языка будет показанный ниже фрагмент таблицы на *рисунке 13.1.*

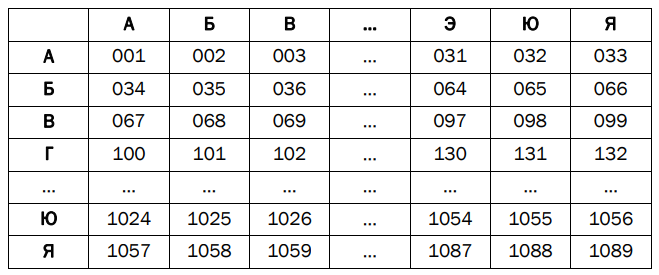
****

Рис. *13.1* – фрагмент шифра Порты для русского алфавита из 33 букв

Шифрование выполняется парами букв исходного сообщения. Первая буква пары указывает на строку, вторая – на столбец. В случае нечетного количества букв в сообщении Мi к нему добавляется вспомогательный символ, например «А».

**Пример**. Исходное сообщение Мi = «АВВА». Сообщение состоит из двух пар (биграмм): АВ и ВА – и будет зашифровано так: 003 067.

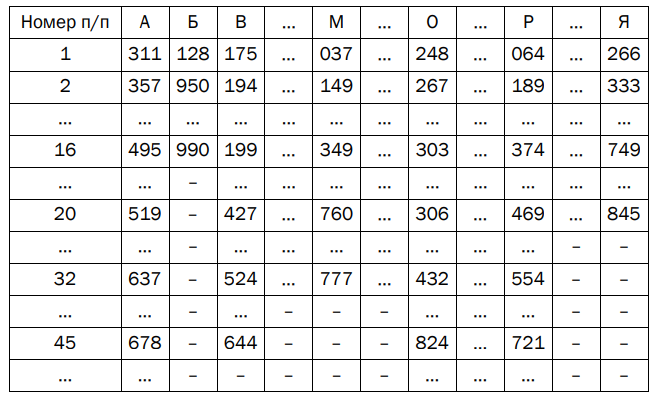
Другими известными полиграммными шифрами являются **шифр Плейфера и шифр Хилла**.

С точки зрения криптостойкости рассматриваемый тип шифров имеет преимущества перед моноалфавитными шифрами. Это связано с тем, что, во-первых, распределение частот групп букв значительно более равномерное, чем отдельных символов. Во-вторых, для эффективного частотного анализа требуется больший размер зашифрованного текста, так как число различных групп букв значительно больше, чем мощность алфавита.

**Омофонические шифры:**

**Омофонические шифры (омофоническая замена**), или однозвучные шифры подстановки, создавались с целью увеличить сложность частотного анализа шифртекстов путем маскировки реальных частот появления символов текста с помощью омофонии. Такие шифры позже **стали называться шифрами многозначной замены, или омофонами** (от греч. homos – одинаковый и phone – звук; слова, которые звучат одинаково, но пишутся по-разному и имеют разное значение.

Каждой букве ставится в соответствие несколько эквивалентов, число которых пропорционально частоте встречаемости буквы в открытом тексте Мi. В этих шифрах буквы исходного алфавита соответствуют более чем одному символу из алфавита замены. Обычно символам исходного текста с наивысшей частотой дают большее количество эквивалентов, чем более редким символам. Таким образом, распределение частоты становится более равномерным, сильно затрудняя частотный анализ. На рисунке 13.2 представлена таблица подстановок для системы омофонов.



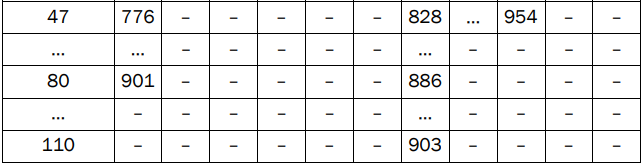


Рис. *13.2* – фрагмент подстановок для системы омофонов.

При шифровании символ исходного сообщения заменяется на любую подстановку из «своего» столбца. Если символ встречается повторно, то, как правило, используют разные подстановки. Например, исходное сообщение «АБРАМОВ» после зашифрования может выглядеть так: «357 990 374 678 037 828 175».

**Полиалфавитные:**

**Полиалфавитные (или многоалфавитные) шифры** состоят из нескольких шифров однозначной замены. Выбор варианта алфавита для зашифрования одного символа зависит от особенностей метода шифрования.

Альберти приводит первое точное описание многоалфавитного шифра на основе шифровального диска.

Он состоял из двух дисков – внешнего неподвижного и внутреннего подвижного, на которые были нанесены буквы алфавита. Процесс шифрования заключался в нахождении буквы открытого текста на внешнем диске и замене ее на букву с внутреннего диска, стоящую под ней. После этого внутренний диск сдвигался на одну позицию, и шифрование второй буквы производилось уже по-новому шифралфавиту. Ключом данного шифра являлся порядок расположения букв на дисках и начальное положение внутреннего диска относительно внешнего.

**Таблица Трисемуса**. Зашифрование осуществляется так: заготавливается таблица подстановки (так называемая «таблица Трисемуса» – таблица со стороной, равной N, где N – мощность алфавита), где первая строка – это алфавит, вторая – алфавит, сдвинутый на один символ, и т. д. При зашифровании первая буква открытого текста заменяется на букву, стоящую в первой строке, вторая – на букву, стоящую во второй строке, и т. д. После использования последней строки вновь возвращаются к первой. Пример 6. Рассмотрим процесс зашифрования сообщения Мi = «БГТУ», используя таблицу, фрагмент которой показан на рис. *13.3.* Стрелками на приведенном рисунке показан принцип зашифрования каждого символа открытого текста. Из этого следует, что шифртекст имеет вид: Сi = «БДФЦ».

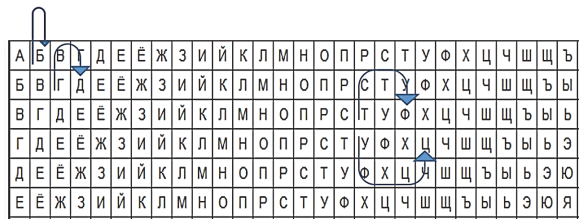


Рис. *13.3* – фрагмент таблицы Трисемуса для русского алфавита.

**Еще раз вспомним, что криптоанализ – это раздел криптологии, занимающийся методами взлома шифров или методами организации криптографических атак на шифры.**

**Атака с известным шифртекстом**. Предполагается, что противник знает алгоритм шифрования, у него имеется набор перехваченных шифрограмм, но он не знает секретный ключ.

**Атака с выбором шифртекста**. Криптоаналитик имеет возможность выбрать необходимое количество шифрограмм и получить соответствующие им открытые тексты. Он также может воспользоваться устройством расшифрования один или несколько раз для получения шифртекста в расшифрованном виде. Используя полученные данные, он может попытаться восстановить секретный ключ.

**Адаптивная атака с выбором шифртекста**. Криптоаналитик имеет возможность выбирать новые шифрограммы для расшифрования с учетом того, что ему известна некоторая информация из предыдущих сообщений.

**Атака с известным открытым текстом**. То же, что и предыдущая, но противник для некоторых шифрограмм получает в свое распоряжение соответствующие им открытые тексты.

**Атака с выбором открытого текста**. Криптоаналитик обладает некоторыми открытыми текстами и соответствующими шифртекстами. Кроме того, он имеет возможность зашифровать несколько предварительно выбранных открытых текстов (до начала атаки).

**Атака на основе связанных ключей**. Криптоаналитик знает не сами ключи, а некоторые различия (соотношения) между ними; реальные криптосистемы используют разные ключи, связанные известным соотношением, например, для каждого нового сообщения предыдущее значение ключа увеличивается на единицу или преобразуется на основе операции сдвига.

**Атака с выбором ключа**. Криптоаналитик задает часть ключа, а на оставшуюся часть ключа выполняет атаку на основе связанных ключей.

**Практическое задание**

1. Разработать авторское приложение в соответствии с целью лабораторной работы.

Приложение должно реализовывать следующие операции:

* выполнять зашифрование/расшифрование текстовых документов (объемом не менее 5 тысяч знаков), созданных на основе алфавита языка в соответствии с нижеследующей таблицей вариантов задания; при этом следует использовать шифры подстановки из третьего столбца данной таблицы.
* сформировать гистограммы частот появления символов для исходного и зашифрованного сообщений;
* оценить время выполнения операций зашифрования/расшифрования (напоминание: во многих языках программирования есть встроенные методы для замеров времени; при отсутствии такового в используемом языке можно воспользоваться разностью двух дат (например, в миллисекундах: время после выполнения программы – время до начала выполнения преобразования)).

|  |
| --- |
|  |

*Листинг 1* – код файла Program.cs

Рис. *13.1* – результат выполения программы.

**Ответы на вопросы:**

**Выводы:** изучила и приобрела практическе навыки разработки и использования приложений для реализации подстановочных шифров.