Учреждение образования

«Белорусский государственный технологический университет»

**Отчёт за практические занятия по предмету:**

Защита информации и надежность информационных систем

Выполнила:

Студентка 3 курса 2 группы ФИТ

Шастовская Марина Сергеевна

# **2022г.Лабораторная работа №1**

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

**Вариант 11**

**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.**

* коммерческая тайна учебного заведения (университета), данные о ее договорах, финансовых  
  взаимоотношениях, бухгалтерская информация;
* персональные данные преподавателей и студентов.

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;
* 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.**

|  |  |
| --- | --- |
| Величина ущерба | Описание |
| **0** | Раскрытие информации принесет ничтожный моральный и финансовый ущерб университету |
| **1** | Ущерб от атаки есть, но он незначителен, основная работа не нарушена. |
| **2** | Работа  Финансовые операции не ведутся в течение некоторого времени, за это время университет терпит убытки, но его престиж и количество преподователей изменяется минимально |
| **3** | Значительные потери финансов. От университета уходит ощутимая часть преподавателей |
| **4** | Потери очень значительны, университет на период до года теряет данные о студентах и преподавателях, финасы. |
| **5** | Университет прекращает существование |

**Условная численная шкала для оценки ущерба учебного заведения (университета) 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 университета | 3 | 0,4 | 1,2 |
| Раскрытие основных планов и стратегий развития учебное заведение (университет) | 4 | 0,1 | 0,4 |
| Раскрытие личных данных студентов и преподователей | 3 | 0,2 | 0,6 |
| **Итого** | **14** | **1,2** | **2,9** |

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

Для образовательной среды проблема стоит шире: в ограждении студента от информации, которая может негативно повлиять на его формирование и развитие, то есть о пропаганде различной направленности. Кроме того, все еще слабо осознана та часть проблемы, которая связана с общением в социальных сетях, которые сегодня все чаще подменяют собой живое общение. В виртуальном пространстве действуют совершенно иные правила, где психически неокрепшая личность не может эффективно противостоять угрозам, запугиванию, откровенным попыткам растления. И сегодня именно этот фактор начинает выходить на первые роли в обеспечении информационной безопасности в ее широком понимании: не только технической, но и когнитивной сферы во всей ее полноте.

**Ситуации:**

**-Несанкционированный доступ к данным.** Эта группа угроз включает в себя подмену данных в электронных журналах, архивах, хищение информации экзаменационных билетов, личных данных студентов и их родственников и т.п. В большинстве рекомендаций по организации схем обеспечения информационной безопасности специалисты ограничиваются только этой, технической сферой.

**- Фильтрация нежелательной информации.** Эта группа угроз напрямую связана с противодействием экстремистской идеологии, но не ограничивается только ей. При рассмотрении угроз доступа к нежелательной информации следует также учитывать вопросы распространения порнографии, провокационных материалов, пропаганды наркотиков и алкоголя и т.п.

- **Проблемы регулирования использования социальных сетей.** Именно в этой зоне осуществляется активное давление на студентов, запугивание, а также сравнительно новый феномен киберхулиганства.

**-** **Кибертерроризм.** Несмотря на то, что эта группа угроз находится в ведении соответствующих силовых ведомств, частично она может решаться и на уровне учебных заведений. Создание безопасной информационно-технологической среды серьезно осложняет возможные кибератаки на объекты образования, которые могут привести к нарушению функционирования управляющих автоматических систем и последующему повреждению инфраструктуры. Следует, впрочем, отметить, что эта группа угроз остается пока во многом гипотетической, так как учебные заведения в силу низкой их насыщенности автоматизированными управляющими системами не рассматриваются в качестве приоритетных целей для кибератак.

Информационная безопасность учебной организации (университет) должна обеспечиваться целым комплексом **мер**, среди которых:

* административно-правовые;
* организационные;
* программно-технические  
   Данные меры следует применять совместно. Опираться система защиты должна на управление сотрудниками и студентами университета и контроль над ним. Меры технического характера не менее важны, но не могут существовать в отрыве от организационных мер.
* Шифрование
* Резервное копирование данных
* Тщательно отбирать сотрудников
* Использование легальных программ

**Вывод:**

К решению всех описанных проблем можно подходить на различных уровнях. Оптимальным вариантом могла бы стать разработка единой платформы для всех образовательных учреждений и ее централизованная удаленная поддержка высококвалифицированными специалистами, которые обладают необходимыми знаниями в области обеспечения безопасности информационных систем.

**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).

**Энтропи́я (информационная)** — мера хаотичности информации, неопределённость появления какого-либо символа первичного алфавита. При отсутствии информационных потерь численно равна количеству информации на символ передаваемого сообщения.

**Алфавит** – это общее число знаков или символов (N), используемых для генерации или передачи сообщений.

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

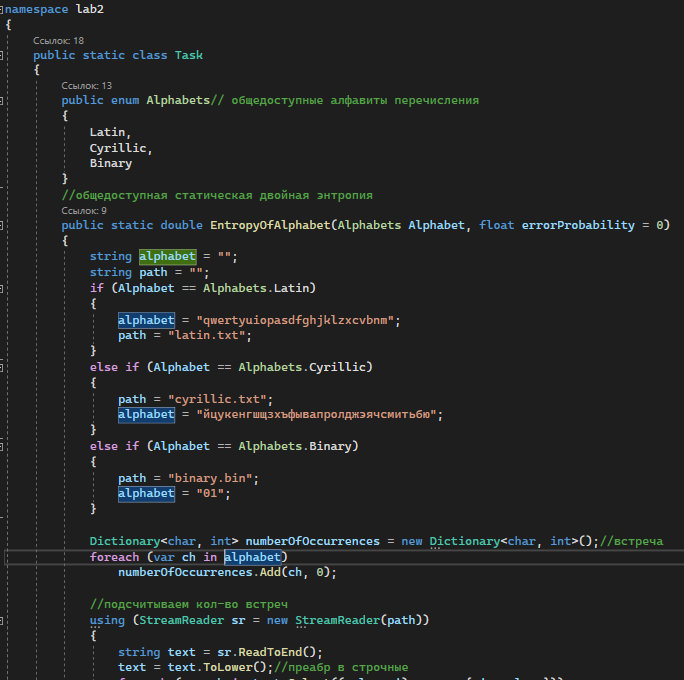
**ASCII**  - стандартом [кодирования символов](https://en.wikipedia.org/wiki/Character_encoding) для электронной связи.

**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)).



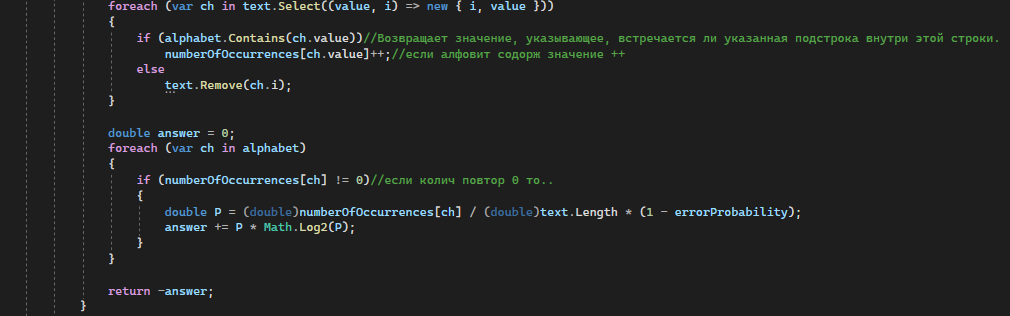


Рис. *1* – Код файла Task.cs

Выполнить задание пункта (в) при условии, что вероятность ошибочной передачи единичного бита сообщения составляет: 0,1; 0,5; 1,0.

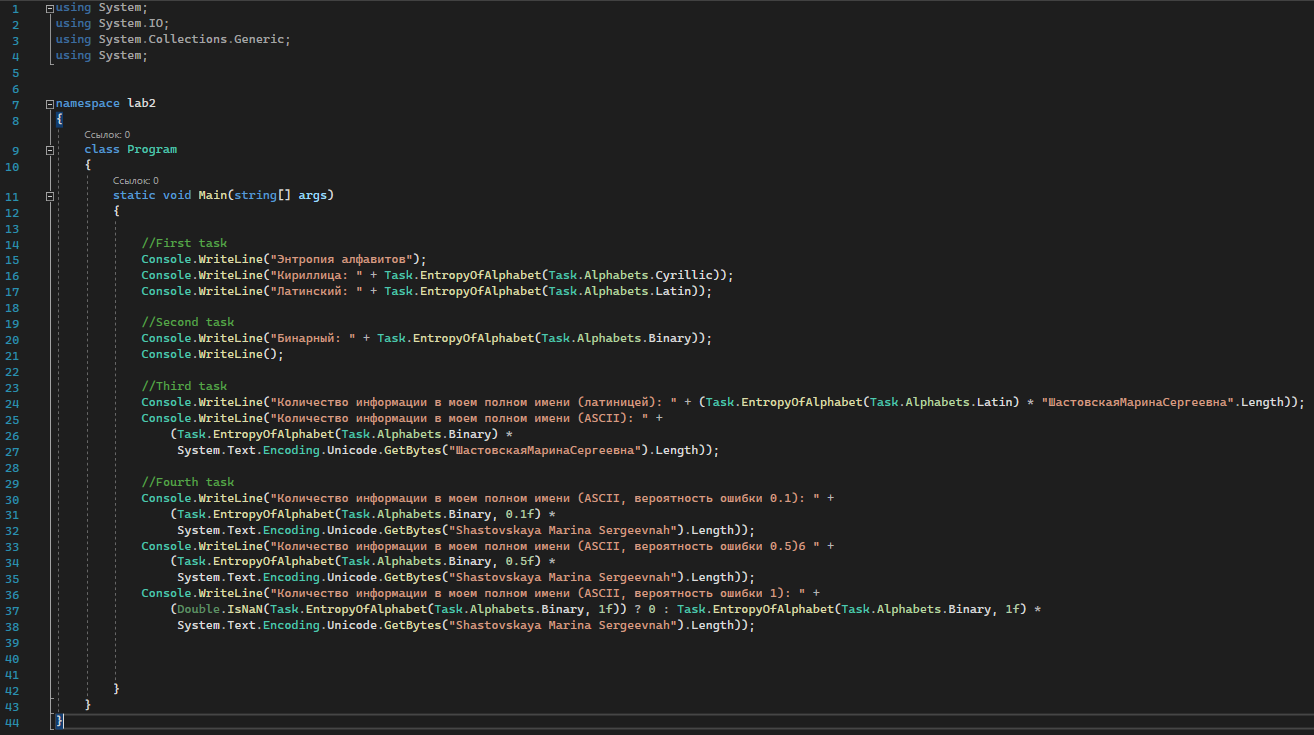


Рис. 2 – Код файла Program.cs

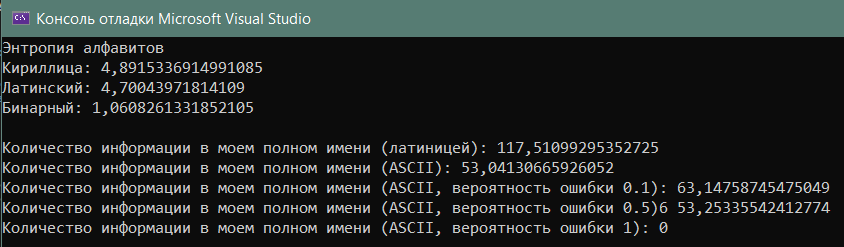


Рис. 3 – Результат выполнения программы

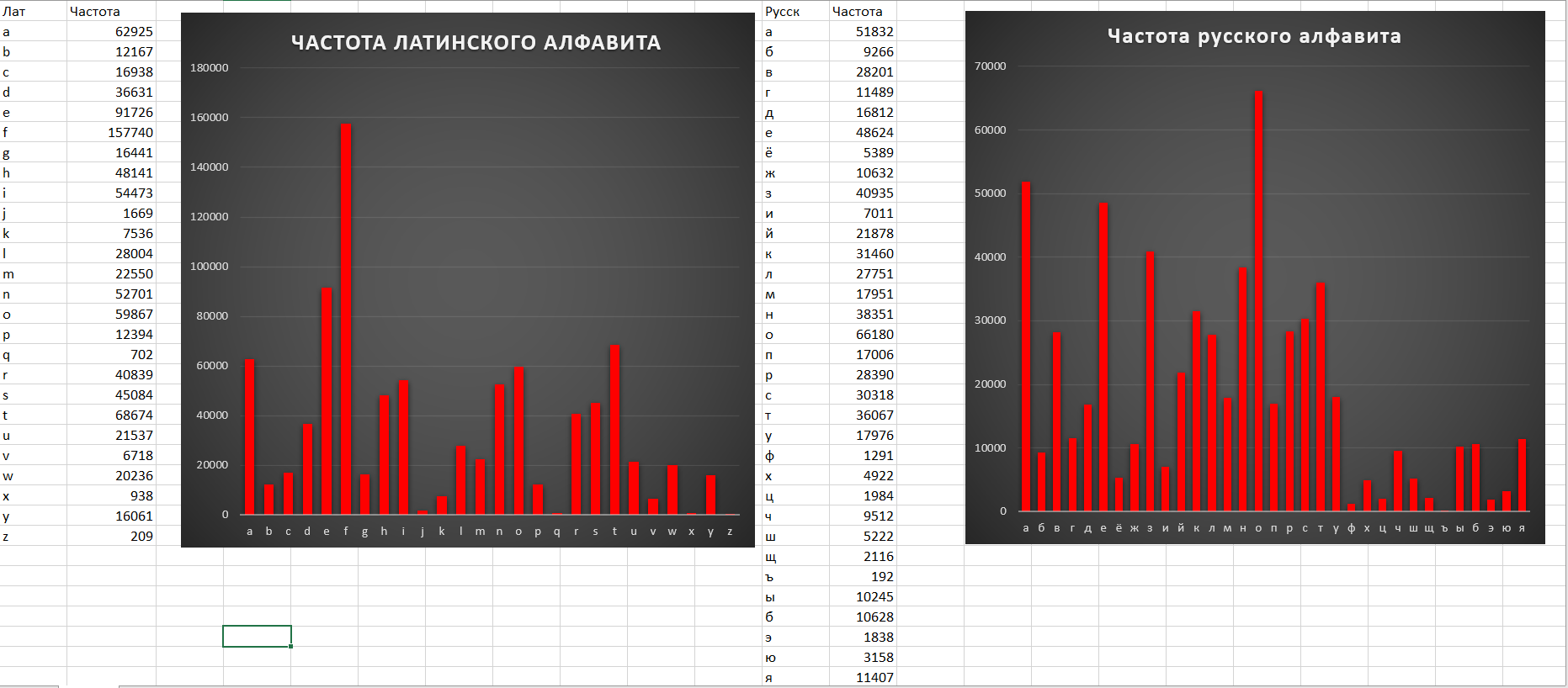


Рис. *4* – Частота алфавитов

**Вопросы для контроля и самоконтроля**

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

**Алфавит** – это общее число знаков или символов (N), используемых для генерации или передачи сообщений.

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.

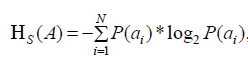
1. **От чего зависит энтропия алфавита?**

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).**

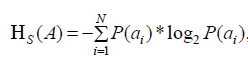


1. **Что нужно знать для вычисления энтропии алфавита?**

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.

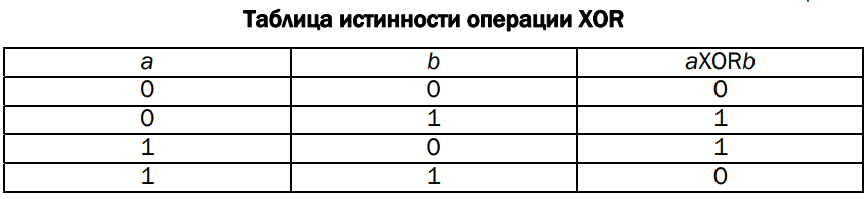
Утверждение восьмибитных кодировок (ASCII) как стандарта принесло некоторые проблемы. К этому моменту уже существовала определенная инфраструктура, использующая семибитные кодировки. Известны проблемы с «обрезанием восьмого бита» в системе электронной почты. Утверждение восьмибитного символа дало 256 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).

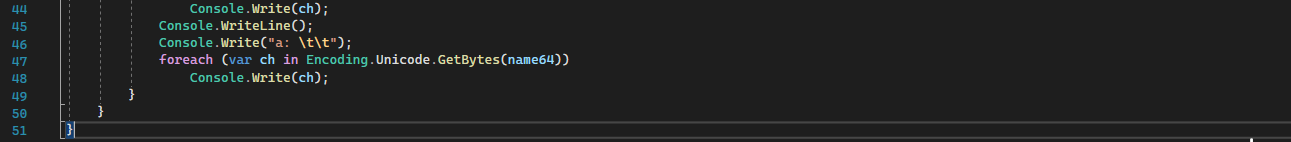
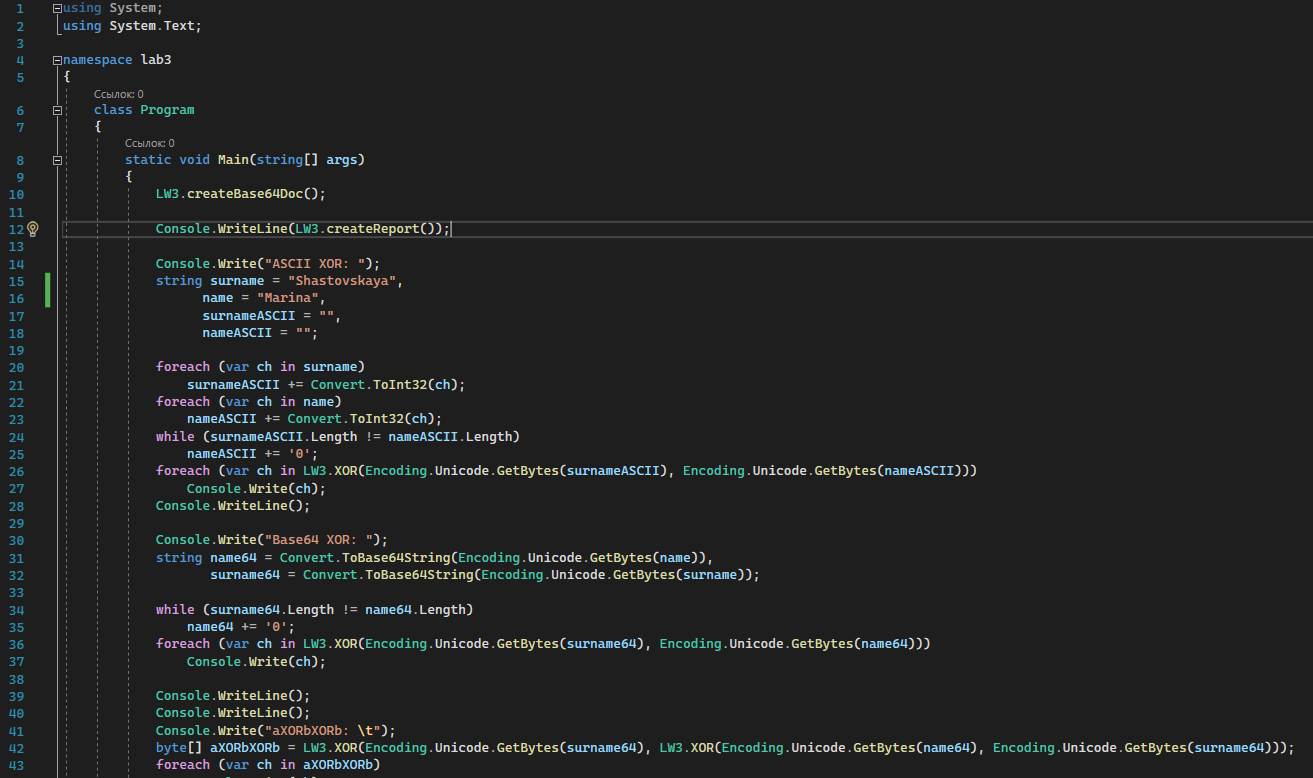
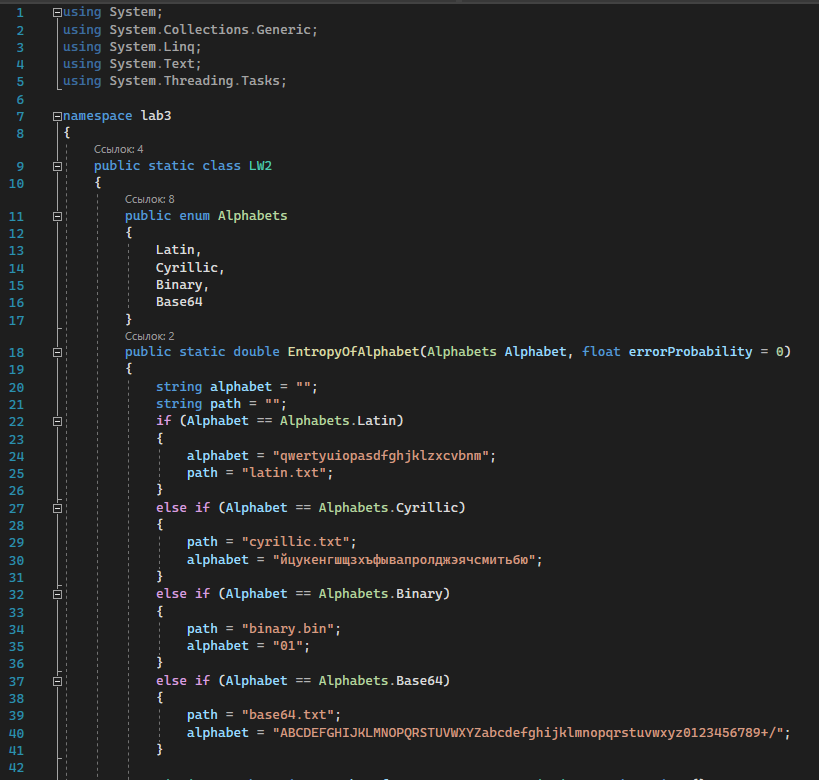


Рис. *1* – Код файла Program.cs



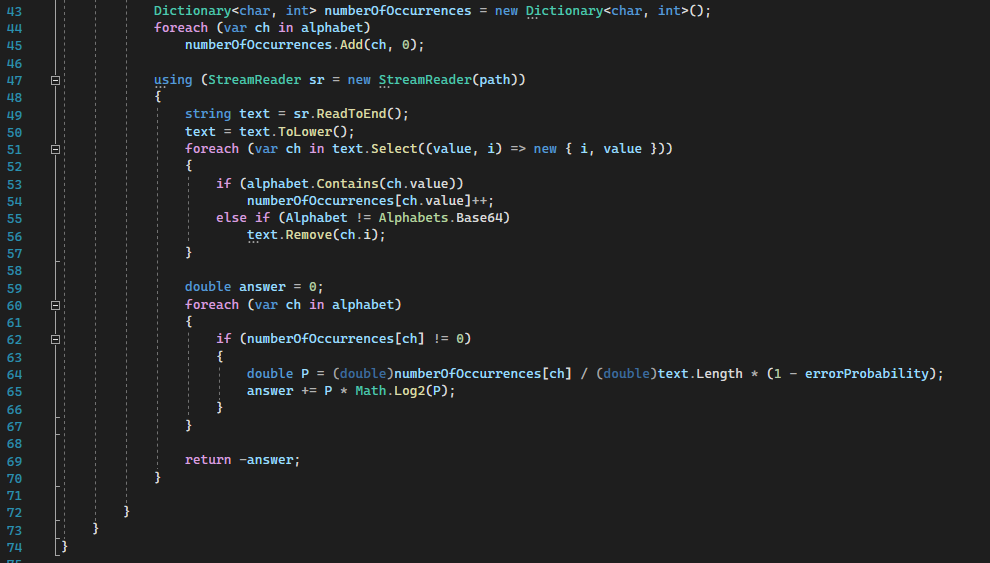


Рис. *2* – Код файла LW2.cs

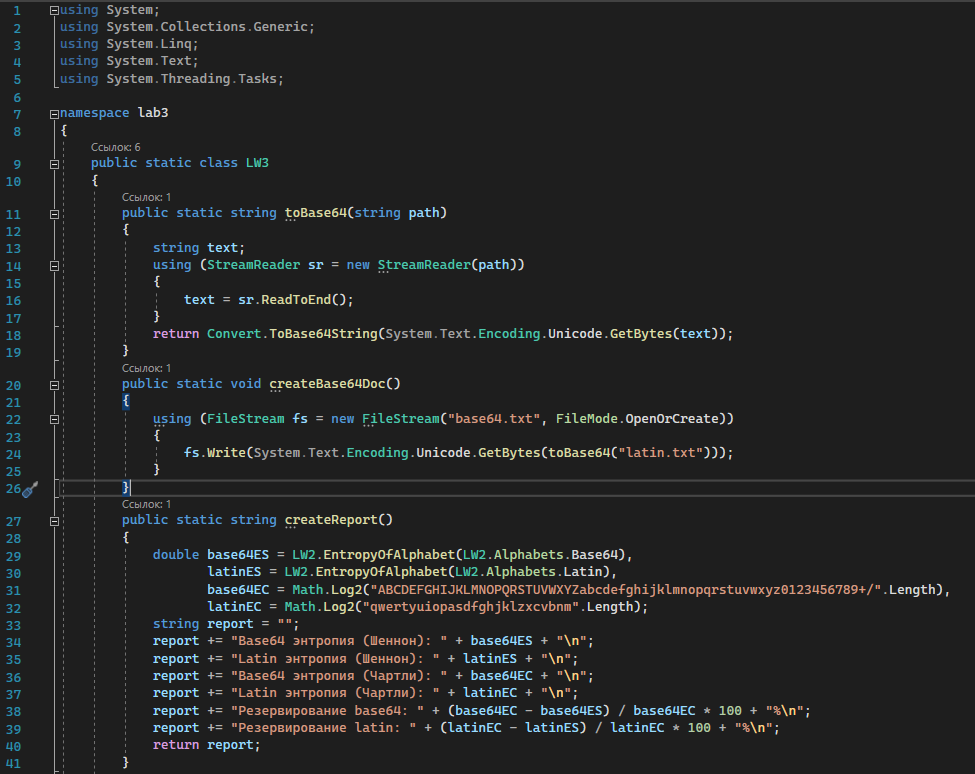


Рис. *3* – Код файла LW3.cs

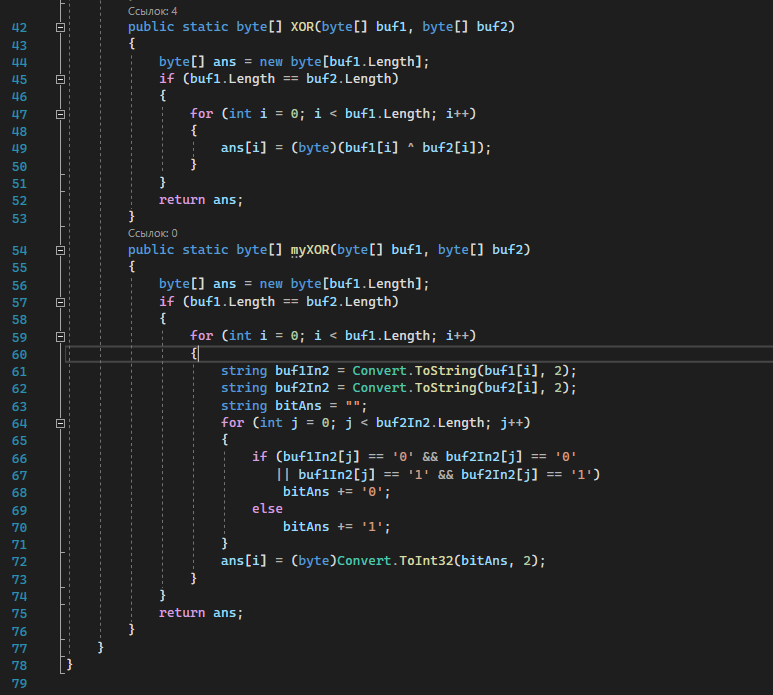
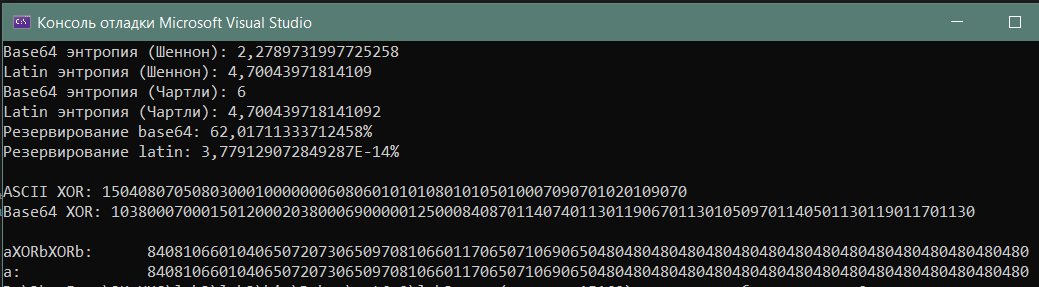


Рис. *4* – Код файла LW3.cs



**Ответы на вопросы:**

1. **Что такое base64?**

**Base64** — стандарт кодирования двоичных данных при помощи только 64 символов [ASCII](https://ru.wikipedia.org/wiki/ASCII).

**Base64** - это группа схожих 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?**
2. **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?**
2. **Результатом операции аXORb (а – каждый байт строки, b – некоторая неизменная величина) будет строка:**

**1f180d1e1f04051c404c0f19**

**1f180308050d024c030a4c18**

**04094c1f18030009024c1c00**

**Найти значение b.**

**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.

**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 знам (результате) вследствие неправильно запрограммированного алгоритма или неправильно составленной программы.

Как следует из вышеприведенного определения, надежность есть внутреннее свойство объекта, заложенное в него при изготовлении и проявляющееся во время эксплуатации. Вторая особенность надежности состоит в том, что она проявляется во времени. И третья особенность надежности выражается по-разному при различных условиях эксплуатации и различных режимах применения объекта (информационной системы в целом, отдельного ее блока, канала передачи сообщения, оперативной или внешней памяти компьютера).

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

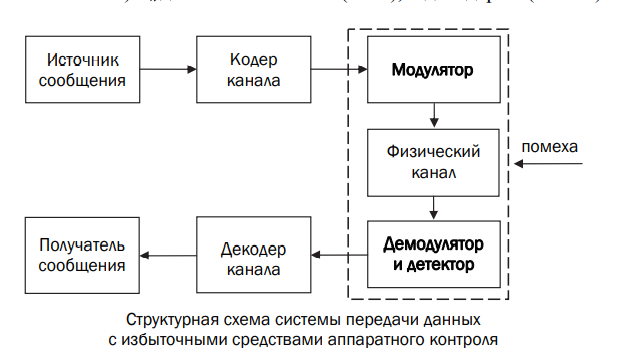
**Безотказность** – это свойство технического объекта непрерывно сохранять работоспособное состояние в течение некоторого времени (или наработки). Наработка, как правило, измеряется в единицах времени.

**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). Большинство современных цифровых систем и устройств (в том числе компьютеры и компьютерные системы, отдельные блоки и модули компьютеров – полупроводниковая, магнитная или оптическая память) содержат специальные средства, призванные автоматически восстанавливать работоспособность этих объектов при нарушении нормального функционирования.

Такие специальные средства контроля называются избыточными. На рисунке приведена упрощенная структурная схема системы передачи данных с избыточными средствами аппаратного контроля.

Не вдаваясь в детали функционирования блоков, названия которых на схеме выделены полужирным, рассмотрим основные принципы работы представленной схемы в соответствии с задачами данной лабораторной работы. Как видим, сначала осуществляется формирование данных в виде двоичных символов. Затем кодер канала вносит в принятую информационную последовательность некоторую избыточность (данный процесс называется кодированием или помехоустойчивым кодированием), которую декодер может использовать для исправления возникающих при передаче данных по каналу связи ошибок.

Таким образом, простейшая структурная схема дополнена двумя интересующими нас блоками: кодером (канала), осуществляющим преобразование исходного сообщения (информационного слова) Хk (длина сообщения – k символов) в избыточное сообщение (кодовое слово) Xn длиной n символов (n > k), и декодером (канала).



Изначальной причиной нарушения нормальной работы цифрового устройства являются технические дефекты (неисправности), возникающие внутри узлов или блоков устройства либо в каналах связи между ними.

Дефекты или неисправности могут приводить либо к кратковременному нарушению достоверности работы устройства (сбой), либо к полной и окончательной потере достоверности (отказ).

В каждом из этих случаев следствием неисправности являются ошибки в информации (информационные ошибки). Чаще всего причиной ошибок бывают внешние помехи, как это показано на рисунке. Количество таких ошибок (количество ошибочных двоичных символов) принято называть кратностью ошибки.

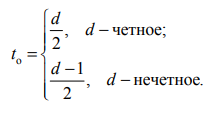
Пример. Исходная (правильная) информационная последовательность Хk = 1001. Длина этой последовательности равна 4 битам (k = 4). Некоторая из перечисленных причин привела к тому, что в этой последовательности появились две ошибки (кратность ошибки равна двум): Yk = 1111. Здесь Yk – сообщение на выходе канала (без учета его кодирования/декодирования).

Обнаружение и/или исправление подобных ошибок как раз и призваны обеспечить кодер и декодер. Дополнительную информацию по рассматриваемым аспектам можно найти в пособиях [5, 9].

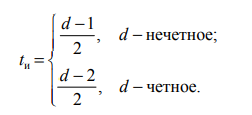
При использовании избыточных кодов исходные данные делятся на блоки из k битов (**называются информационными битами).** В процессе кодирования каждый k-битный блок данных преобразуется, как было отмечено выше, в блок из n битов (**кодовое слово**). Число k часто называется размерностью кода. Таким образом, к каждому блоку данных в процессе кодирования присоединяются **r = n – k** битов, которые называют **избыточными битами** (redundant bits), битами четности (parity bits) или контрольными битами (check bits); новой информации они не несут.

Для обозначения описанного кода обычно пользуются записью (n, k) и говорят, что данный код использует n символов для передачи (хранения) k символов сообщения. Отношение числа битов данных к общему числу битов k/n именуется степенью кодирования (code rate) – доля кода, которая приходится на полезную информацию. Еще одним важным параметром кода является расстояние Хемминга (d), которое показывает, что два кодовых слова различаются по крайней мере в d позициях. В общем случае код позволяет обнаруживать tо ошибок:

В общем случае код позволяет обнаруживать tо ошибок:



Количество исправляемых кодом ошибок иt определяется следующим образом:



Во всех этих простых математических выражениях d соответствует минимальному кодовому расстоянию Хемминга анализируемого кода.

**Практическое задание**

1. На основе информационного сообщения, представленного символами английского алфавитов, служебными символами и цифрами, содержащегося в некотором текстовом файле,

сформировать информационное сообщение в двоичном виде; длина сообщения в бинарном виде должна быть не менее 16 символов.

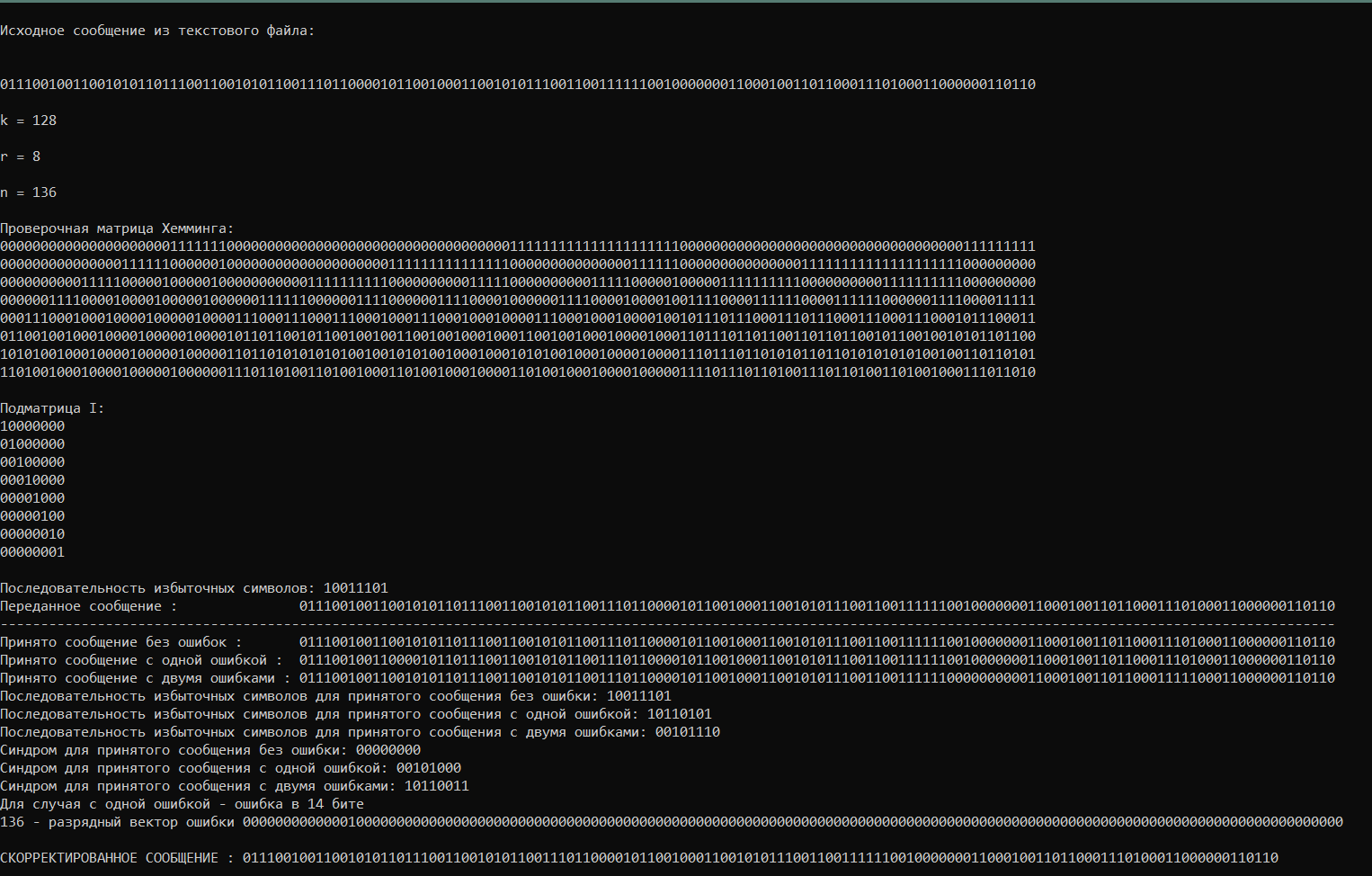
2. Для полученного информационного слова построить проверочную матрицу Хемминга (значение минимального кодового расстояния согласовать с преподавателем - положим, dmin = 3).

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’).

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; проанализировать ситуацию при возникновении ошибки в 2 битах.

**ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ**

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

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

**Кодирование Хэмминга** предусматривает как возможность обнаружения ошибки, так и возможность её исправления.

Смысл алгоритма кодирования и декодирования по Хеммингу представляется так:  
Исходный текст, который необходимо передать по ненадежному каналу, рубится на куски определенной одинаковой длины (пакеты) и в конец каждого такого пакета помещается избыточная информация (контрольные биты).  
Принятую кодограмму, алгоритм декодирования, также анализирует по пакетам Проверяется четность пакета в целом и четность каждой из групп  
В этом случае, группой называется комбинация информационных бит (различной длины), причем, обязательно дополненная одним контрольным битом, который как раз и должен обеспечивать четность всей группы.

Вот и вся премудрость По результатам проверки четности, декодирующий алгоритм однозначно определит:

* Пакеты без ошибок
* Пакеты с одинарной ошибкой (которую автоматически исправит)
* Пакеты с двойной ошибкой (не исправит, но сообщит, что пакет с двойной ошибкой)

## **Код Хэмминга -** это алгоритм самоконтролирующегося и самокорректирующегося кода, 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.

## **Простой алгоритм Хемминга:**

## который может исправлять лишь одну ошибку (существуют более совершенные модификации данного алгоритма, которые позволяют обнаруживать (и если возможно исправлять) большее количество ошибок).

## **Код Хэмминга состоит из двух частей**.

## Первая часть кодирует **исходное сообщение**, вставляя в него в определённых местах контрольные биты (вычисленные особым образом).

## Вторая часть **получает входящее сообщение и заново вычисляет контрольные биты** (по тому же алгоритму, что и первая часть). Если все вновь вычисленные контрольные биты совпадают с полученными, то сообщение получено без ошибок. В противном случае, выводится сообщение об ошибке и при возможности ошибка исправляется.

# **Лабораторная работа №5**

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

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

**Теоретические сведения**

Итеративные коды относятся к классу кодов произведения.

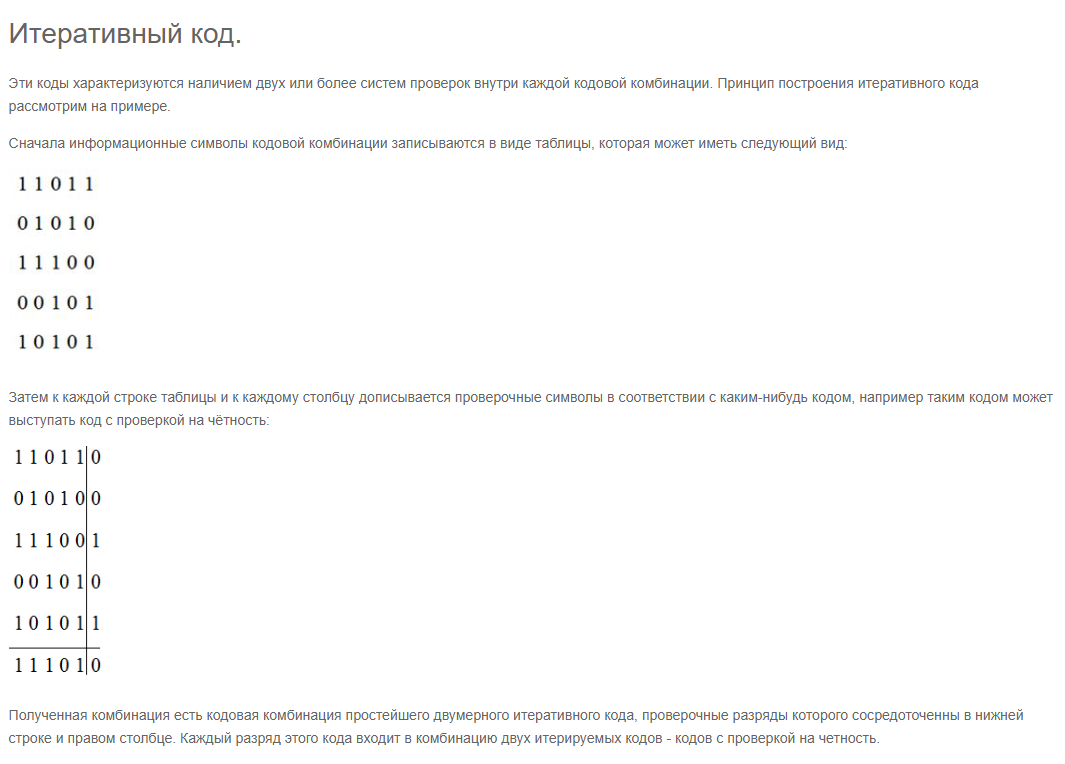
Кодом произведения двух исходных (базовых) помехоустойчивых кодов называется такой многомерный помехоустойчивый код, кодовыми последовательностями которого являются все двумерные таблицы со строками кода (k1) и столбцами кода (k2).

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 Избыточное кодирование данных в информационных системах. Итеративные коды 49 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).



Итеративные коды характеризуются наличием двух и более систем проверок внутри каждой кодовой комбинации.

Итеративный код матричный код, строки и столбцы которого представляют собой комбинации группового кода.

**Ход работы**

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) по

а) двум, б) трем, в) четырем направлениям (группам паритетов);

1. Формировать кодовое слово Xn, присоединением избыточных символов к информационному слову;
2. Генерировать ошибку произвольной кратности (i, i>0), распределенную случайным образом среди символов слова Xn, в результате чего формируется кодовое слово Yn;
3. Определять местоположение ошибочных символов итеративным кодом в слове Yn в соответствии с используемыми группами паритетов по п. 1.2 и исправлять ошибочные символы (результат исправления – слово Yn’;
4. 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.

|  |
| --- |
| static void IterativeMatrix(int height , int width){    Random rand = new Random();  int[] itog = new int[16];  int[] save = new int[16];  int[,] generateMessage = new int[height, width];  int[] sc = new int[width];  int eachRow, i, j;  int[] gorParity = new int[width];  int[] verParity = new int[height];  int result = 0;  for ( i = 0; i < width; i++) sc[i] = 0;  for (i = 0; i < height; i++)  {  eachRow = 0;  for (j = 0; j < width; j++)  {  generateMessage[i,j] = rand.Next(0,2);  Console.Write(generateMessage[i,j] + " ");  eachRow += generateMessage[i,j];  result += generateMessage[i, j];  gorParity[i] = eachRow;  sc[j] += generateMessage[i,j];  }  int l = eachRow % 2;  Console.WriteLine(" |" + l);    }  for (i = 0; i < width; i++)  Console.Write("---");  Console.Write("\n");  for (i = 0; i < width; i++)  {  int z = sc[i] % 2;  verParity[i] = z;  Console.Write( z + " ");  }  Console.WriteLine("\n");    int itogreg = 0;  int itogregd = 0;  Console.Write("Xn= ");  for (i = 0; i < width; i++) sc[i] = 0;  for (i = 0; i < height; i++)  {    for (j = 0; j < width; j++)  {  Console.Write(generateMessage[i, j]);  itog[itogreg++] = generateMessage[i, j];  save[itogregd++] = generateMessage[i, j];  }    }  // вертикальный паритет  int o = 0;  int[] savegor = new int[4];  Console.Write(" ");  foreach (int item in gorParity)  {  int l = item % 2;  savegor[o++] = l;  Console.Write(l);  }  int f = 0;  int[] savever = new int[4];  // горизонтальный паритет  Console.Write(" ");  Array.Reverse(verParity);  foreach (int item in verParity)  {  savever[f++] = item;  Console.Write(item);  }  int res = result % 2;  Console.WriteLine(" " +res);  foreach (int item in itog)  {  Console.Write(item);  }    try  {  Console.WriteLine();  var random = new Random();  int error = random.Next(0, itog.Length);  Console.WriteLine("Позиция ошибки: " + error);  if (itog[error] == 1) itog[error] = 0;  else itog[error] = 1;  }  catch { }  Console.Write("Yn= ");  foreach (int item in itog)  {  Console.Write(item);  }  Console.WriteLine();  int iter = 0;  for (i = 0; i < width; i++) sc[i] = 0;  for (i = 0; i < height; i++)  {  eachRow = 0;  for (j = 0; j < width; j++)  {  generateMessage[i, j] = itog[iter++];  Console.Write(generateMessage[i, j] + " ");  eachRow += generateMessage[i, j];  sc[j] += generateMessage[i, j];  gorParity[i] = eachRow;  }  int l = eachRow % 2;  Console.WriteLine(" |" + l );  }  for (i = 0; i < width; i++)  Console.Write("---");  Console.Write("\n");  for (i = 0; i < width; i++)  {  int z = sc[i] % 2;  verParity[i] = z;  Console.Write(z + " ");  }  Console.WriteLine("\n");  Console.WriteLine("Сравниваем паритеты:");  Console.Write(" ");  foreach (int item in gorParity)  {  int l = item % 2;  Console.Write(l);  }  // горизонтальный паритет  Console.Write(" ");  Array.Reverse(verParity);  foreach (int item in verParity)  {  Console.Write(item);  }  Console.WriteLine();  Console.Write(" ");  foreach (int it in savegor) {  Console.Write(it);  }  Console.Write(" ");  // Array.Reverse(savever);  foreach (int it in savever)  {  Console.Write(it);  }  Console.WriteLine();  Console.WriteLine("Исправляем, строка без ошибки:");  Console.Write("Xn= ");  foreach (int item in save) {  Console.Write(item);  }    } |

Листинг 1 – Функция итеративного кода

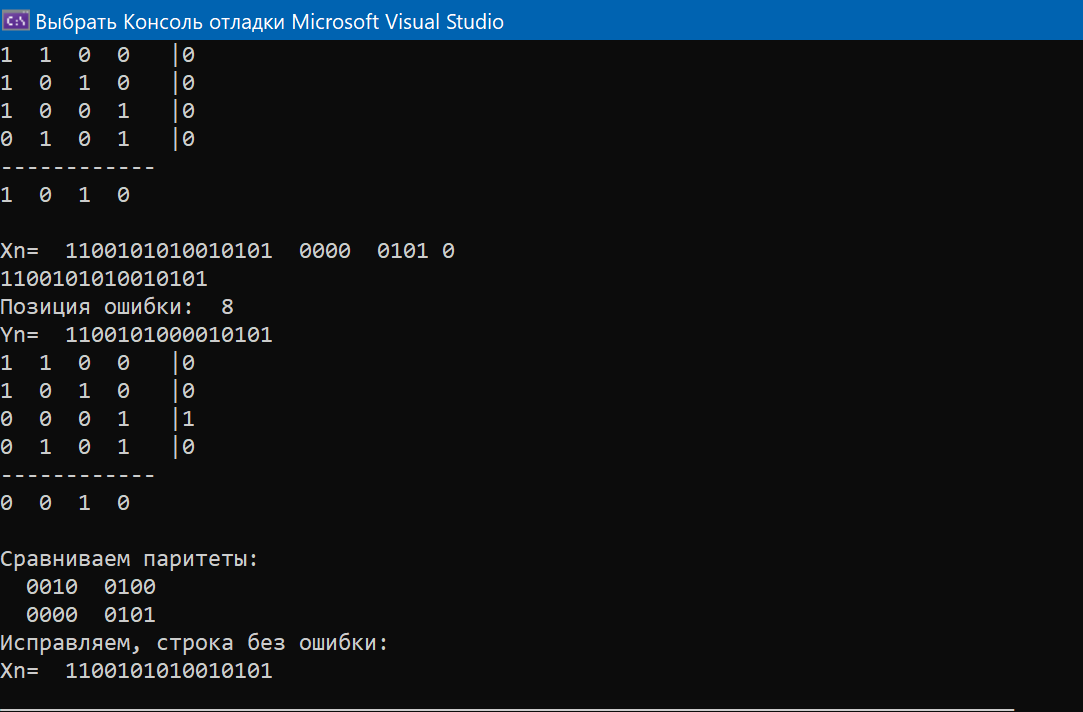


Рисунок 1 – Результат выполнения функции

|  |
| --- |
| static void IterativeMatrixWithDepth(int height, int width, int depth)  {    Random rand = new Random();  int[] itog = new int[16];    int[] sc = new int[width];    int[] result = new int[16];  int itogers = 0;  for (int i = 0; i < depth; i++) sc[i] = 0;    int[,,] array = new int[height, width, depth];  for (int i = 0; i < array.GetLength(0); i++)  {    for (int j = 0; j < array.GetLength(1); j++)  {    int slow = 0;  for (int k = 0; k < array.GetLength(2); k++)  {    array[i, j, k] = rand.Next(0, 2);  slow += array[i, j, k];  result[itogers++] += array[i, j, k];  Console.Write(array[i, j, k]);  sc[k] += array[i, j, k ];  }  int l = slow % 2;  Console.WriteLine(" |" + l);    }  Console.WriteLine("---");  for (int d = 0; d < depth; d++)  {  int z = sc[d] % 2;    Console.Write(z + "");  sc[d] = 0;  }  Console.WriteLine();  Console.WriteLine();  }  Console.WriteLine("Паритет 4 - 5 группы:");  for (int i = 0; i < array.GetLength(0); i++)  {  for (int j = 0; j < array.GetLength(1); j++)  {    for (int k = 0; k < array.GetLength(2); k++)  {  Console.Write(array[i, j, k]);  }  }  Console.WriteLine(" ");  }  int first = result[0] + result[5] + result[10] + result[15];  int firstm = first % 2;  int second = result[1] + result[6] + result[11] + result[12];  int secondm = second % 2;  int third = result[2] + result[7] + result[8] + result[13];  int thirdm = third % 2;  int fourth = result[3] + result[4] + result[9] + result[14];  int fourthm = fourth % 2;  int firstf = result[0] + result[7] + result[10] + result[13];  int firstmf = firstf % 2;  int secondf = result[1] + result[4] + result[11] + result[14];  int secondmf = secondf % 2;  int thirdf = result[2] + result[5] + result[8] + result[15];  int thirdmf = thirdf % 2;  int fourthf = result[3] + result[6] + result[9] + result[12];  int fourthmf= fourthf % 2;  Console.WriteLine("Паритет 5 группы: " + firstm + secondm + thirdm + fourthm);  Console.WriteLine("Паритет 4 группы: " + firstmf + secondmf + thirdmf + fourthmf);  int[] sresult = new int[16];  Console.Write("Xn = ");  int inh = 0;  foreach (var item in result)  {  sresult[inh++] = item;  Console.Write(item);  }    try  {  Console.WriteLine();  var random = new Random();  int error = random.Next(0, itog.Length);  Console.WriteLine("Позиция ошибки: " + error);  if (result[error] == 1) result[error] = 0;  else result[error] = 1;  }  catch { }  Console.Write("Yn = ");  foreach (var item in result)  {  Console.Write(item);  }  Console.WriteLine();  int itogers2 = 0;  int[] result2 = new int[16];  int iter = 0;  for (int i = 0; i < array.GetLength(0); i++)  {  for (int j = 0; j < array.GetLength(1); j++)  {  int slow = 0;  for (int k = 0; k < array.GetLength(2); k++)  {  array[i, j, k] = result[iter++];  slow += array[i, j, k];  result2[itogers2++] += array[i, j, k];  Console.Write(array[i, j, k]);  sc[k] += array[i, j, k];  }  int l = slow % 2;  Console.WriteLine(" |" + l);  }  Console.WriteLine("---");  for (int d = 0; d < depth; d++)  {  int z = sc[d] % 2;  Console.Write(z + "");  sc[d] = 0;  }  Console.WriteLine();  Console.WriteLine();  }  first = result2[0] + result2[5] + result2[10] + result2[15];  firstm = first % 2;  second = result2[1] + result2[6] + result2[11] + result2[12];  secondm = second % 2;  third = result2[2] + result2[7] + result2[8] + result2[13];  thirdm = third % 2;  fourth = result2[3] + result2[4] + result2[9] + result2[14];  fourthm = fourth % 2;  firstf = result2[0] + result2[7] + result2[10] + result2[13];  firstmf = firstf % 2;  secondf = result2[1] + result2[4] + result2[11] + result2[14];  secondmf = secondf % 2;  thirdf = result2[2] + result2[5] + result2[8] + result2[15];  thirdmf = thirdf % 2;  fourthf = result2[3] + result2[6] + result2[9] + result2[12];  fourthmf = fourthf % 2;  Console.WriteLine("Паритет 5 группы: " + firstm + secondm + thirdm + fourthm);  Console.WriteLine("Паритет 4 группы: " + firstmf + secondmf + thirdmf + fourthmf);  Console.WriteLine("Исправленное сообщение:");  Console.Write("Xn = ");  foreach (var item in sresult)  {  Console.Write(item);  }  } |

Листинг 2 – Функция итеративного кода с нахождением паритетов 4 и 5 группы

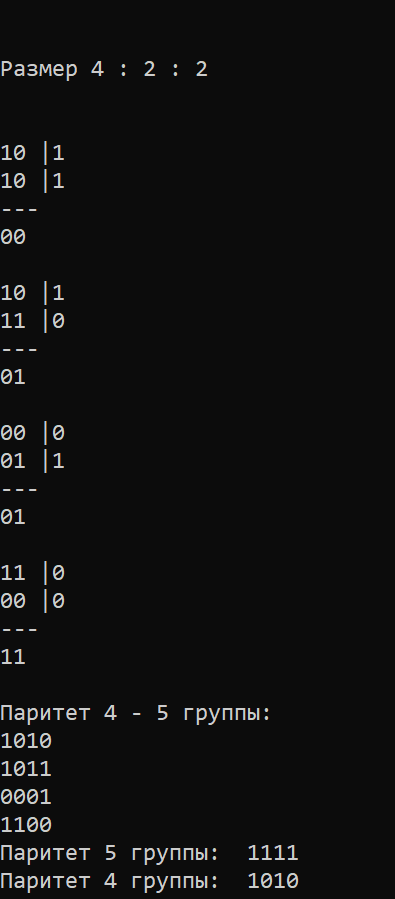


Рисунок 2 – Результат выполнения функции с нахождением паритетов 4 и 5 группы

Анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов Xn и Yn:

В ходе выполнения лабораторной работы, мы можем заметить, что в ситуации, где сообщение Xn отличается на 1 бит от Yn, сообщение подвержено исправлению. Однако в ситуации с двумя ошибками, обнаружить место ошибки невозможно, поскольку при появлении ошибки на одном уровне, сумма по модулю два не будет исправлять сообщение. Следуя от частного к общему, можем прийти к выводу, что итеративный код может исправлять с большей точностью нечетное количество ошибок. Так же при увеличении групп паритетов, шанс обнаружения увеличивается. Наглядный пример:

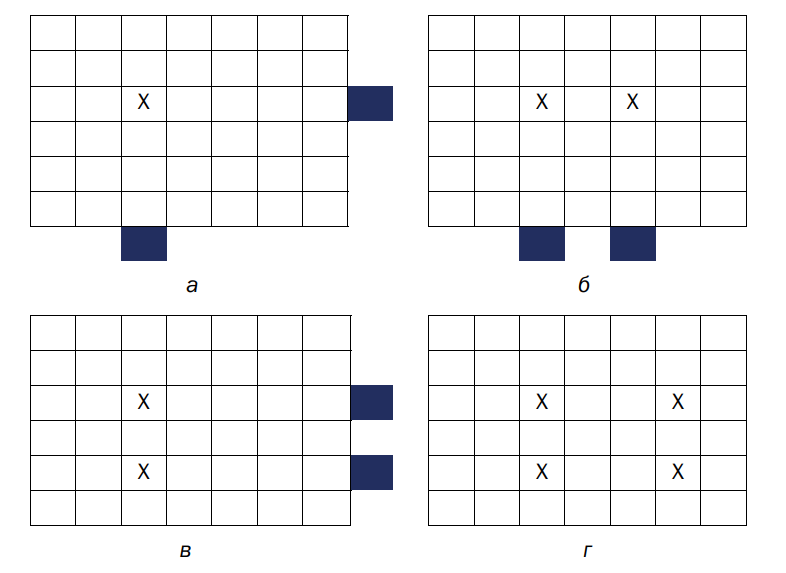


Рисунок 3 – Наглядный пример проблемы обнаружения двух ошибок

**Ответы на вопросы:**

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

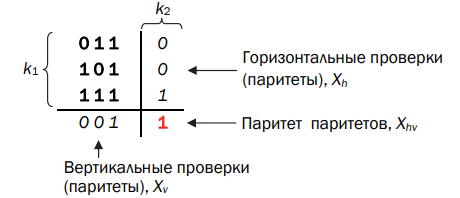
Итеративные помехоустойчивые коды относятся к классу кодов произведения. Кодом произведения двух исходных (базовых) помехоустойчивых кодов n1и n2называется такой многомерный помехоустойчивый код N=n1\*n2, кодовыми последовательностями которого являются все двумерные таблицы со строками кода n1и столбцами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

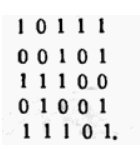


Паритет – избыточный символ. Записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ Xhv (паритет паритетов) равен сумме по модулю 2 символов информационного слова Xk, а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

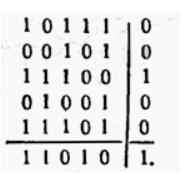
1. Охарактеризовать основные параметры итеративного кода.

Идея построения итеративных кодов принадлежит Элайесу [144, 145]. Эти коды характеризуются наличием двух или более систем проверок внутри каждой кодовой комбинации. Принцип построения итеративного кода проще всего показать на примере.

Сначала информационные символы кодовой комбинации записываются в виде таблицы, которая, например, может иметь такой вид:



Затем к каждой строке таблицы и к каждому столбцу дописываются проверочные символы в соответствии с каким-нибудь кодом Применим в качестве кода код с проверкой на четность:



Полученная комбинация есть кодовая комбинация простейшего двумерного итеративного кода, проверочные разряды которого сосредоточены в нижней строке и правом столбце. Каждый информационный разряд этого кода входит в комбинацию двух итерируемых кодов — кодов с проверкой на четность. Передача одного сообщения комбинацией итеративного кода обычно происходит по строкам последовательно, от первой строки к последней. Приведенный код является простейшим итеративным кодом.

**2. Сравнить основные параметры кодов Хемминга и итеративных кодов**

# Коды Хэмминга

Кодом Хэмминга называется (*n, k*) – код, который задается матрицей проверок *H(n,k)*, имеющей строк и столбцов, причем столбцами *H(n,k)* являются все различные ненулевые двоичные последовательности длины *m* (*m* – разрядные двоичные числа от 1 до ).

Длина кодовой комбинации кода Хэмминга равна .

Число информационных элементов определяется как .

Итак, код Хэмминга полностью задается числом *m* – количеством проверочных элементов в кодовой комбинации.

Зная вид матрицы *H(n,k)*, можно определить корректирующие свойства (*n, k*) – кода Хэмминга. Так как все столбцы матрицы проверок различны, то никакие два столбца *H(n,k)* не являются линейно зависимыми. Наряду с этим, для любого числа *m* всегда можно указать три столбца матрицы *H(n,k)*, которые линейно зависимы, например, столбцы, соответствующие числам 1, 2, 3. Следовательно, для любого (*n, k*) – кода Хэмминга *dmin*=3.

**Итеративные коды.**

На основе (*n, n-*1) – кодов с *dmin*=2 или кодов Хэмминга с *dmin*=3 и *dmin*=4 можно построить коды с более высокими корректирующими свойствами. Для этой цели, наряду с защитой каждой передаваемой комбинации описанным выше способом, осуществляют помехоустойчивое кодирование одноименных разрядов групп передаваемых комбинаций. Процесс кодирования можно пояснить при помощи *рис. 5.6*.

Комбинации простого кода, подлежащие передаче по системе связи, записываются в виде таблицы – каждая комбинация составляет отдельную строку этой таблицы (информационные символы). Затем осуществляется кодирование по строкам и столбцам. В общем случае для кодирования строк и кодирования столбцов можно использовать различные коды. Избыточные элементы дописываются к каждой строке (проверка по строкам) и к каждому столбцу (проверка по столбцам). Проверка проверок осуществляется кодированием столбцов, составленных из избыточных элементов строк или кодированием строк, составленных из проверок столбцов. Последующее введение избыточности осуществляется для защиты блоков информации, представленных на рис. 5.6. Процесс кодирования поясняется на *рис. 5.7*. Из блоков информации, защищенных двумя проверками, составляется параллелепипед. Избыточные разряды третьей проверки образуют параллелепипед, выделенный утолщенной линией.

**3. Составить проверочные матрицы кодов Хемминга для кодирования 9-битных сообщений по схеме, представленной на рис. 5.1 (с учетом и без учета символа Xhv).**

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

Все ошибки кратности 4 и менее.

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

k.

**Вывод**: в данной работе был рассмотрен вариант итеративного кодирования информации, рассмотрены его плюсы и минусы. Было установлено, что итеративные коды могут корректировать несколько ошибок одинаковой четности.

# **Лабораторная работа №6**

**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, 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). Для более глубокого изучения параметров и свойств ЦК, равно как и других корректирующих кодов, полезно ознакомиться с классическими книгами, например [16−18].

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.

Основная операция: принятое кодовое слово (Yn) нужно поделить на порождающий полином, который использовался при кодировании.

**Ход работы**

1. 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.

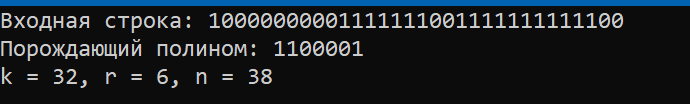
****

Рисунок 1 – Параметры кода и полином

1. **Составить** порождающую матрицу (n, k)-кода в соответствии с формулой (6.7**), трансформировать** ее в каноническую форму и далее – в проверочную матрицу канонической формы.

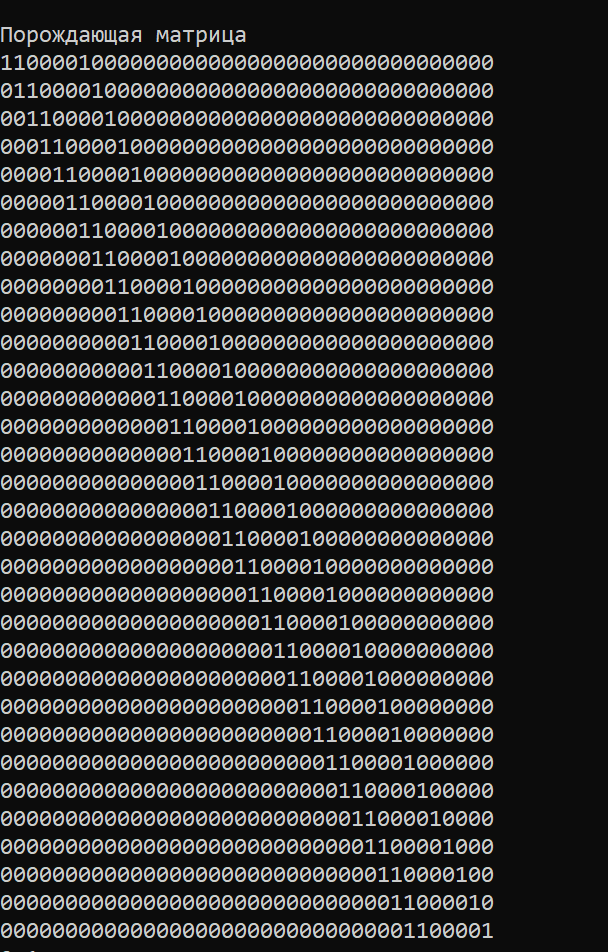


Рисунок 2 – Порождающая матрица

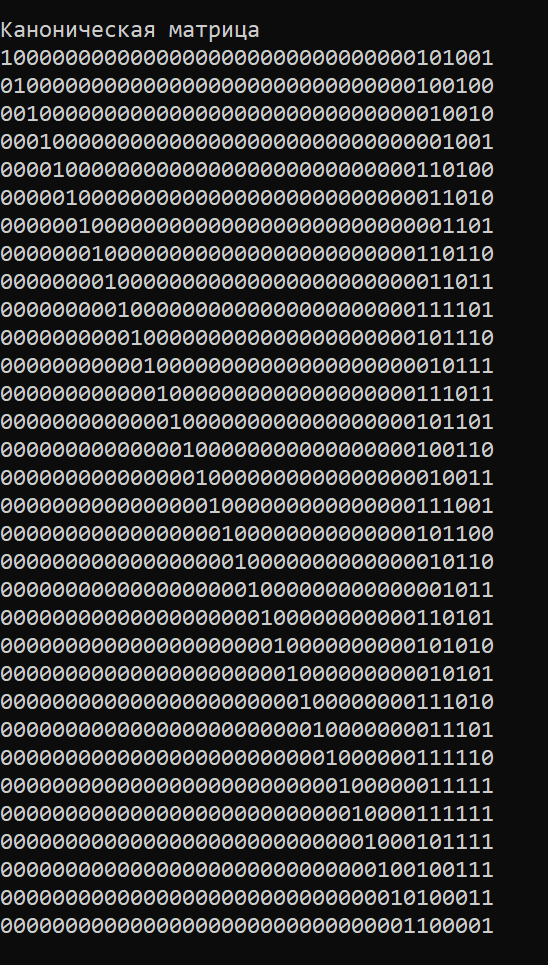


Рисунок 3 – Каноническая матрица

1. Используя порождающую матрицу ЦК, **вычислить избыточные** символы (слово Xr) кодового слова Xn и сформировать это  
   кодовое слово.

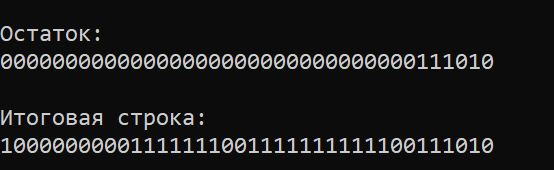


Рисунок 3 – Сформированные избыточные символы с кодовым словом

1. **Принять кодовое слово** Yn со следующим числом ошибок:  
   0; 1; 2. Позиция ошибки определяется (генерируется) случайным  
   образом.

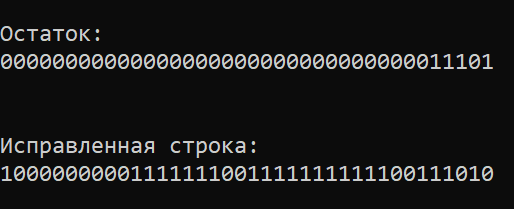


Рисунок 4 – Остаток и исправленная строка при одной ошибке

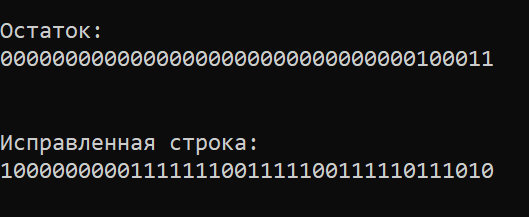


Рисунок 5 – Остаток и исправленная строка при двух ошибках

|  |
| --- |
| from numpy.polynomial import polynomial as P  import math  import numpy as np  # вычисляем длину строки  def str\_len(X\_k):  k = len(X\_k)  print('Длина сообщения = "',k,'"')  return k  # вычисляем r  R\_Culc = {  'test':3,  '4':5  }  # добавочный полином  def Edition\_R\_Polinom(r):  r\_poly = []  for i in range(r):  r\_poly.append(0)  r\_poly.append(1)  print('Полином для умножения = ',r\_poly)  return r\_poly  # получаем начальную строку в виде массива  def str\_to\_arr(string):  X\_k\_list = list(string)  X\_k\_list = list(map(int,X\_k\_list))  return X\_k\_list  # полином  Poly = {  'test':[1,1,0,1],  '4':[1,0,0,1,0,1]  }  # умножение полиномов  def multiply\_result(poly\_1,poly\_2):  return P.polymul(poly\_1,poly\_2)  # деление полиномов  def division\_result(poly\_1,poly\_2):  res = P.polydiv(poly\_1,poly\_2)  res = [abs(elem) for elem in res]  return res  # добавление остатка к исходному сообщению  def FinalMessage(poly\_1,poly\_2):  res = []  res.extend(poly\_2)  res.extend(poly\_1)  return res  # проверка ошибки  def Mistake\_Check(Message, EncodePoly):  res = division\_result(Message,EncodePoly)  print(res)  for i in res[1]:  if int(i)==1 or int(i)==-1:  return 0  return 1  # генерация матрицы  def GenerateMatrix(K\_R,K,poly):    Matrix = [ [0]\*K\_R for i in range(K)]  for i in range(K):  Matrix[i]=poly  poly=[poly[-1]] + poly[:-1]  for i in range(K-1):  for j in range(i+1,K):  if Matrix[i][j]==1:  Matrix[i] = list(np.array(Matrix[i])+np.array(Matrix[j]))  for m in range(K):  for l in range(K\_R):  if Matrix[m][l]%2==0:  Matrix[m][l]=0    return Matrix  # подматрица H  def SubMatrixH(K,R,Matrix):  H = []  for i in range(K):  for j in range(K,K+R):  H.append(Matrix[i][j])    H = np.reshape(H,(K,R))  H = np.concatenate((H,np.eye(R,dtype=int).reshape((R,R))))  return H  # исправление ошибок  def ClearMistake(K\_R,K,Fin\_message,curr\_poly):  poly\_for\_matrix = curr\_poly.copy()  poly\_for\_matrix.reverse()  for i in range(K\_R-K):  poly\_for\_matrix.append(0)  matr = GenerateMatrix(K\_R,K,poly\_for\_matrix)  #print(matr)  H = SubMatrixH(K,K\_R-K,matr)  print(H)  divres = division\_result(Fin\_message,curr\_poly)  divres = list(map(int,divres[1]))  if len(divres)!=K:  for i in range(K-len(divres)):  divres.append(0)  print(divres)  E = []  index = 99999  for i in range(K\_R):  if divres==list(H[i]):  index = i;  print(index)  for i in range(K\_R):  if i==index:  E.append(1)  else:  E.append(0)  E.reverse()  print(E)  Corrected\_Message = list(np.array(Fin\_message)+np.array(E))  for i in range(len(Corrected\_Message)):  if Corrected\_Message[i]%2 == 0:  Corrected\_Message[i]=0  print(Corrected\_Message)  # сбор всего  def Cycle\_Code(X\_k):  X\_k\_list = str\_to\_arr(X\_k)  K = str\_len(X\_k)  curr\_r = R\_Culc['4']  r\_poly = Edition\_R\_Polinom(curr\_r)  curr\_poly = Poly['4']  X\_k\_mul\_X\_Pow\_R = list(map(int,multiply\_result(X\_k\_list,r\_poly)))  print('Строка с избыточными битами = ',X\_k\_mul\_X\_Pow\_R)  div\_res = division\_result(X\_k\_mul\_X\_Pow\_R,curr\_poly)  print(div\_res)  Fin\_message = FinalMessage(X\_k\_list,list(map(int,div\_res[1])))  print('Итоговая строка = ',Fin\_message)    Fin\_message = [1, 0, 1, 1, 1, 1, 0, 1, 0, 1]  if Mistake\_Check(Fin\_message,curr\_poly)==1:  print('Ошибок нет')  else:  print('Есть ошибка')  ClearMistake(K+curr\_r,K,Fin\_message,curr\_poly)    def main():  X\_k = '10101'  print('Сообщение = "',X\_k,'"')  Cycle\_Code(X\_k)  main() |

Рисунок 6 – Листинг кода

**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;

**2.Сравнить основные параметры кодов Хемминга, итеративных и циклических кодов.**

Кодом Хэмминга называется (*n, k*) – код, который задается матрицей проверок *H(n,k)*, имеющей строк и столбцов, причем столбцами *H(n,k)* являются все различные ненулевые двоичные последовательности длины *m* (*m* – разрядные двоичные числа от 1 до ).

Длина кодовой комбинации кода Хэмминга равна .

Число информационных элементов определяется как .

Итак, код Хэмминга полностью задается числом *m* – количеством проверочных элементов в кодовой комбинации.

Зная вид матрицы *H(n,k)*, можно определить корректирующие свойства (*n, k*) – кода Хэмминга. Так как все столбцы матрицы проверок различны, то никакие два столбца *H(n,k)* не являются линейно зависимыми. Наряду с этим, для любого числа *m* всегда можно указать три столбца матрицы *H(n,k)*, которые линейно зависимы, например, столбцы, соответствующие числам 1, 2, 3. Следовательно, для любого (*n, k*) – кода Хэмминга *dmin*=3.

**Циклический код** — линейный, блочный код, обладающий свойством цикличности, то есть каждая циклическая перестановка кодового слова также является кодовым словом. Используется для **преобразования информации для защиты её от ошибок** (см. Обнаружение и исправление ошибок).

***3.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

**Вывод**: в данной работе был рассмотрен циклический вид кодирования информации. Данный вид кодирования позволяет представлять информацию в другом формате, на основе деления двоичной информации на некоторый порождающий полином. Было установлено, что при декодировании у нас есть возможность исправления одной ошибки, как в предыдущих видах кодирования.

# **Лабораторная работа №7**

**Перемежение/деперемежение данных в информационно-вычислительных системах**

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

**Теоретические сведения**

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

Проанализированные и исследованные нами коды, как и большинство других кодов, которые были разработаны для увеличения надежности каналов передачи и хранения информации, наиболее эффективны, когда возникающие ошибки статистически независимы, т. е. вероятность передачи (хранения в памяти) любого символа одинакова. Однако довольно часто распределение ошибок носит взаимозависимый характер. В таких случаях говорят о группах (или пакетах) ошибок. Такие ошибки характерны и для каналов передачи, и для устройств памяти (главным образом магнитной и полупроводниковой; cм., например, [9−10, 19−22]).

Существуют специальные коды, корректирующие пакетные ошибки, однако на практике чаще используют перемежение/деперемежение совместно с традиционными кодами

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.

Рассмотренный метод блочного перемежения применяется в SM. К числу других используемых на практике относятся следующие методы перемежения/деперемежения: псевдослучайный, S-типа (применяется в турбо-кодировании: CDMA (Codе Division Multiple Access) – стандарт беспроводной связи множественного доступа с кодовым разделением каналов и др.); циклически-сдвиговый; сверточный; случайный; диагональный; многошаговый [5, 23, 24].

Перемежение (перестановка) символов также является основой некоторых классов криптографических методов [1], которые мы будем анализировать с практической точки зрения в другой части курса.

**Ход работы**

Разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель.

Используется Циклический код, длинна пакета ошибок 3,5,7, число столбцов в матрице 7, длинна сообщений 14 байт, длинна информационного слова 6.

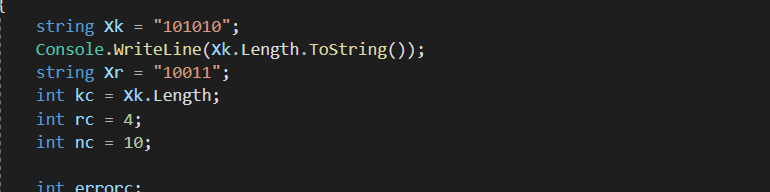
****

Рисунок 1 – Параметры кода

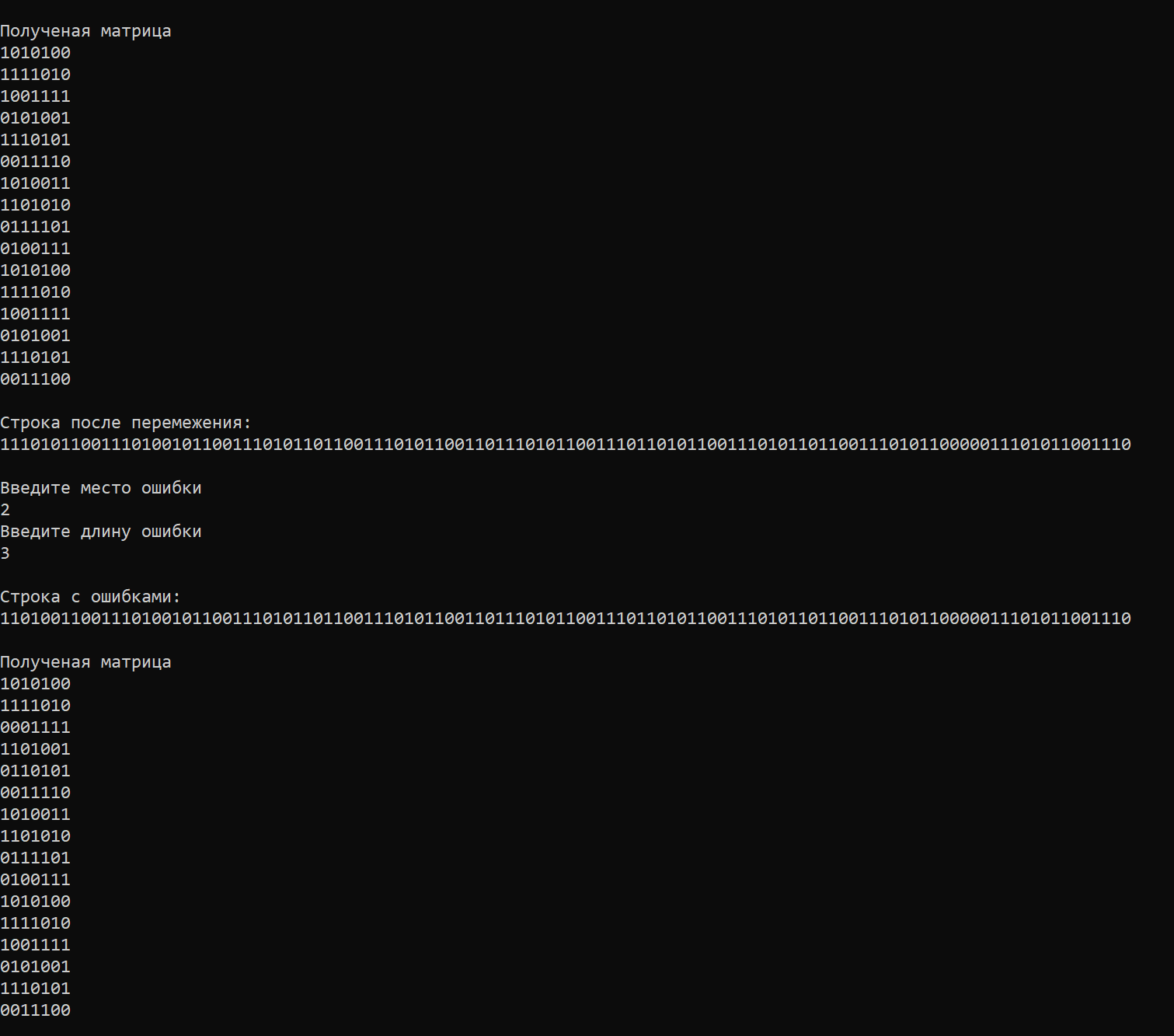


Рисунок 2 – Результат перемежения/деперемежения

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  namespace SevenLab  {  public static class Extensions  {  public static IEnumerable<IEnumerable<T>> Split<T>(this T[] arr, int size)  {  return arr.Select((s, i) => arr.Skip(i \* size).Take(size)).Where(a => a.Any());  }  }  class Program  {  static void Main(string[] args)  {  string Xk = "101010";  Console.WriteLine(Xk.Length.ToString());  string Xr = "10011";  int kc = Xk.Length;  int rc = 4;  int nc = 10;  int errorc;  int[] masXk = new int[kc];  StrInMas(masXk, Xk);  int[] masXr = new int[Xr.Length];  StrInMas(masXr, Xr);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine("Входная строка: " + Xk);  Console.WriteLine("Порождающий полином: " + Xr);  Console.WriteLine("k = {0}, r = {1}, n = {2}", kc, rc, nc);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  int[,] generationMatrix = new int[kc, nc];  CreateGenerationMatrix(generationMatrix, masXr, kc, nc);  Console.WriteLine("\nПорождающая матрица");  OutMatrix(generationMatrix, kc, nc);  CreateCanonicalMatrix(generationMatrix, kc, nc);  Console.WriteLine("\nКаноническая матрица");  OutMatrix(generationMatrix, kc, nc);  int[,] checkMatrixc = new int[nc, rc];  CreateCheckMatrix(checkMatrixc, generationMatrix, kc, nc);  Console.WriteLine("\nПроверочная матрица");  OutMatrix(checkMatrixc, nc, rc);  //6.2  int[] masXn = new int[nc];  Shift(masXn, masXk, rc);  //2.  Console.WriteLine("\nДеление");  SearchResidue(masXn, masXr);  Console.WriteLine("Остаток (S-синдром):");  OutMass(masXn);  Console.WriteLine("\n");  Console.WriteLine("Итоговая строка:");  Shift(masXn, masXk, rc);  OutMass(masXn);  Console.WriteLine();  int[] save = new int[nc];  int prop = 0;  foreach (int menuitem in masXn)  {  save[prop++] = menuitem;  }  int altero = 0;  int[] alter = new int[110];  for (int i = 0; i < 11; i++)  {  foreach (var item in masXn)  {  alter[altero++] = item;  }  }  List<int> list = alter.ToList<int>();  list.Add(0);  list.Add(0);  // You can convert it back to an array if you would like to  masXn = list.ToArray();  Alternation(masXn);  Console.WriteLine("\nСтрока после перемежения: ");  OutMas(masXn);  int error;  int errorLenght;  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++)  {  masXn[i] = (masXn[i] + 1) % 2;  }  }  catch { }  Console.WriteLine("\nСтрока с ошибками: ");  OutMas(masXn);    ReAlternation(masXn);  Console.WriteLine("\nСтрока после re:перемежения: ");  List<int> listt = new List<int>(masXn);  listt.RemoveAt(masXn.Length - 1);  masXn = listt.ToArray();  List<int> listt2 = new List<int>(masXn);  listt2.RemoveAt(masXn.Length - 1);  masXn = listt2.ToArray();  OutMas(masXn);  int jojo = 0;  int[] errorw = new int[110];  foreach (var item in masXn)  {  errorw[jojo++] = item;  }  //var nChunks = 11;  //var totalLength = masXn.Count();  //var chunkLength = (int)Math.Ceiling(totalLength / (double)nChunks);  //var parts = Enumerable.Range(0, 10)  // .Select(i => masXn.Skip(i \* chunkLength).Take(chunkLength));  int size = 10;  var arrays = masXn.Split(size);  Console.WriteLine();    foreach (var array in arrays)  {  Console.Write(String.Join("", array));  }    int[] itog = new int[110];  int ite = 0;  #region  int[] m1 = new int[10];  int[] m2 = new int[10];  int[] m3 = new int[10];  int[] m4 = new int[10];  int[] m5 = new int[10];  int[] m6 = new int[10];  int[] m7 = new int[10];  int[] m8 = new int[10];  int[] m9 = new int[10];  int[] m10 = new int[10];  int[] m11 = new int[10];  m1 = arrays.ElementAt(0).ToArray();  m2 = arrays.ElementAt(1).ToArray();  m3 = arrays.ElementAt(2).ToArray();  m4 = arrays.ElementAt(3).ToArray();  m5 = arrays.ElementAt(4).ToArray();  m6 = arrays.ElementAt(5).ToArray();  m7 = arrays.ElementAt(6).ToArray();  m8 = arrays.ElementAt(7).ToArray();  m9 = arrays.ElementAt(8).ToArray();  m10 = arrays.ElementAt(9).ToArray();  m11 = arrays.ElementAt(10).ToArray();  Console.WriteLine("Попааааааааа");  foreach (int item in m1)  {  Console.Write(item);  }  foreach (int item in m2)  {  Console.Write(item);  }  foreach (int item in m3)  {  Console.Write(item);  }  foreach (int item in m4)  {  Console.Write(item);  }  foreach (int item in m5)  {  Console.Write(item);  }  foreach (int item in m6)  {  Console.Write(item);  }  foreach (int item in m7)  {  Console.Write(item);  }  foreach (int item in m8)  {  Console.Write(item);  }  foreach (int item in m9)  {  Console.Write(item);  }  foreach (int item in m10)  {  Console.Write(item);  }  foreach (int item in m11)  {  Console.Write(item);  }  #endregion  Console.WriteLine("Попааааааааа");  int[] itogus = new int[10];  // SearchError(m1, masXk, checkMatrixc, rc);  //SearchError(m2, masXk, checkMatrixc, rc);  //SearchError(m3, masXk, checkMatrixc, rc);  //SearchError(m4, masXk, checkMatrixc, rc);  //SearchError(m5, masXk, checkMatrixc, rc);  //SearchError(m6, masXk, checkMatrixc, rc);  //SearchError(m7, masXk, checkMatrixc, rc);  //SearchError(m8, masXk, checkMatrixc, rc);  //SearchError(m9, masXk, checkMatrixc, rc);  //SearchError(m10, masXk, checkMatrixc, rc);  //SearchError(m11, masXk, checkMatrixc, rc);  itogus = SearchError(m1, masXk, checkMatrixc, rc)  .Concat(SearchError(m2, masXk, checkMatrixc, rc))  .Concat(SearchError(m3, masXk, checkMatrixc, rc))  .Concat(SearchError(m4, masXk, checkMatrixc, rc))  .Concat(SearchError(m5, masXk, checkMatrixc, rc))  .Concat(SearchError(m6, masXk, checkMatrixc, rc))  .Concat(SearchError(m7, masXk, checkMatrixc, rc))  .Concat(SearchError(m8, masXk, checkMatrixc, rc))  .Concat(SearchError(m9, masXk, checkMatrixc, rc))  .Concat(SearchError(m10, masXk, checkMatrixc, rc))  .Concat(SearchError(m11, masXk, checkMatrixc, rc))  .ToArray();  Console.WriteLine();  Console.WriteLine("Итооог");  foreach (var item in itogus)  {  Console.Write(item.ToString());  }  Console.WriteLine();  foreach (var item in errorw)  {  Console.Write(item);  }  Console.WriteLine("Итооог");  Console.WriteLine(); Console.WriteLine();  foreach (var item in itogus)  {  Console.Write(item.ToString());  }  //foreach (var item in parts.ElementAt(0))  //{  // first[ite++] = item;  // //itog[u++] = item;  //}  //Console.WriteLine("\nfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");  //foreach (var item in first)  //{  // Console.Write( item.ToString());  //}  ////SearchError(first, masXk, checkMatrixc, rc);  //int opp = 0;  //foreach (var item in parts.ElementAt(1))  //{  // second[opp++] = item;  // //itog[u++] = item;  //}  //Console.WriteLine("\nfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");  //foreach (var item in second)  //{  // Console.Write(item.ToString());  //}  // SearchError(secondso, masXk, checkMatrixc, rc);  // SearchError(second, masXk, checkMatrixc, rc);  //Array.Clear(first, 0, first.Length);  //ite = 0;  // SearchError(first, masXk, checkMatrixc, rc);  //SearchError(masXn, masXk, checkMatrixc, rc);  Console.WriteLine("\n\nСтрока после исправления ошибок: ");  OutMas(save);  //RemoveCheckBits(masK2, masXn, checkMatrixc);  Console.WriteLine("\n\nСтрока после удаления проверочных бит: ");  Console.WriteLine(Xk);  Console.WriteLine("");  OutMassive(Xk);    }    static int[] Alternation(int[] masN)  {  int[,] matrix = new int[16, 7];  //Получение матрицы  for (int i = 0, m = 0; i < 16; i++)  {  for (int j = 0; j < 7; j++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, 16, 7);  //Перемежение  for (int i = 0, m = 0; i < 7; i++)  {  for (int j = 0; j < 16; j++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }  static int[] ReAlternation(int[] masN)  {  //int r = HemmingLength(k);  int k = 16;  int n = 7;  int[,] matrix = new int[k, n];  //Получение матрицы  for (int j = 0, m = 0; j < n; j++)  {  for (int i = 0; i < k; i++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, k, n);  //RE:Перемежение  for (int j = 0, m = 0; j < k; j++)  {  for (int i = 0; i < n; i++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }      //Поиск синдрома  static int[] Sindrom(int[,] CheckMatrix, int[] mas, int k)  {  int r = HemmingLength(k);  int n = r + k;  int[] sindrom = new int[r];  for (int i = 0, l = 0; i < r; i++, l = 0)  {  for (int j = 0; j < k; j++)  {  if (CheckMatrix[j, i] == 1 && mas[j] == 1) l++;  else sindrom[i] = 0;  }  if (l % 2 == 1) sindrom[i] = 1;  else sindrom[i] = 0;  }  for (int i = 0; i < r; i++)  {  mas[i + k] = sindrom[i];  }  return mas;  }  //Считаем r (кол-во пров. симв.)  static int HemmingLength(int k)  {  int r = (int)(Math.Log(k, 2) + 1.99f);  return r;  }    static void OutMas(int[] mas)  {  for (int i = 0; i < mas.Length; i++)  {  Console.Write(mas[i]);  }  }  //вывод матрицы    public static int[] SearchError(int[] masXn, int[] 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");  SearchResidue(masXnSecond, masXr);  Console.WriteLine();  Console.WriteLine("\n");  Console.WriteLine("\nОстаток:");  OutMass(masXnSecond);  for (int i = 0; i < n; i++)  {  int coincidence = 0;  for (int j = 0; j < r; j++)  {  if (checkMatrix[i, j] == masXnSecond[k + j])  {  coincidence++;  }  }  if (coincidence == r)  {  masXn[i] = (masXn[i] + 1) % 2;  break;  }  }  Console.WriteLine("\nИсправленная строка:");  OutMass(masXn);  return masXn;  }  public static int[] SearchResidue(int[] masXn, int[] masXr)  {  int end = masXn.Length - masXr.Length + 1;  for (int i = 0; i < end; i++)  {  if (masXn[i] == 1)  {  AddingMasMod2(masXn, masXr, i);  OutMass(masXn);  }  }  Console.WriteLine("\n");  return masXn;  }  //Сложение массивов по модулю 2 с опр. позиции  public static int[] AddingMasMod2(int[] mas1, int[] mas2, int pos)  {  int end = pos + mas2.Length;  for (int i = pos; i < end; i++)  {  mas1[i] = (mas1[i] + mas2[i - pos]) % 2;  }  return mas1;  }  //Смещение на массива r  public static int[] Shift(int[] shiftMas, int[] mas, int r)  {  for (int i = 0; i < mas.Length; i++)  {  shiftMas[i] = mas[i];  }  return shiftMas;  }  //Преобразование сторки в массив  public static int[] StrInMas(int[] mas, string str)  {  for (int i = 0; i < str.Length; i++)  {  if (str[i] == 49)  mas[i] = 1;  else mas[i] = 0;  }  return mas;  }  //Создание Порождающей матрицы  static int[,] CreateGenerationMatrix(int[,] generationMatrix, int[] mas, int k, int n)  {  //Заполняем первую строку в проверочной матрице  for (int i = 0; i < n; i++)  {  if (i < mas.Length)  {  generationMatrix[0, i] = mas[i];  }  else  {  generationMatrix[0, i] = 0;  }  }  //Сдвигаем каждую строки вправо от предыдущей  for (int i = 1; i < k; i++)  {  for (int j = 0; j < n - 1; j++)  {  generationMatrix[i, j + 1] = generationMatrix[i - 1, j];  }  generationMatrix[i, 0] = generationMatrix[i - 1, n - 1];  }  return generationMatrix;  }  //Приведение порождающей матрицы к каноническому виду  static int[,] CreateCanonicalMatrix(int[,] generationMatrix, int k, int n)  {  //Перебираем строки для преведению к каноническому виду  for (int i = 0; i < k; i++)  {  int i2 = i + 1;  //Перебираем элементы строки, но только до k-элемента  for (int j = i + 1; j < k; j++)  {  //если мы нашли единицу в строке, то...  if (generationMatrix[i, j] == 1)  {  //перебираем этот столбец, пока не найдем единицу  for (; i2 < k; i2++)  {  bool repeat = false;  //Если нашли, то складываем обе строки  if (generationMatrix[i2, j] == 1)  {  for (int j2 = j - 1; j2 > 0; j2--)  {  //Проверяем, есть ли до этой 1 еще 1, если есть то эту строку пропускаем  if (generationMatrix[i2, j2] == 1)  {  repeat = true;  }  }  if (repeat)  continue;  Console.WriteLine(i + " " + i2);  AddingLinesMatrixMod2(generationMatrix, i, i2, n);  i2++;  break;  }  }  }  }  }  return generationMatrix;  }  //Преобразование канонической матрицы в проверочную  static int[,] CreateCheckMatrix(int[,] checkMatrix, int[,] generationMatrix, int k, int n)  {  int r = n - k;  for (int i = 0; i < k; i++)  {  for (int j = 0; j < r; j++)  {  checkMatrix[i, j] = generationMatrix[i, k + j];  }  }  for (int i = k; i < n; i++)  {  for (int j = 0; j < r; j++)  {  if (j == i - k)  {  checkMatrix[i, j] = 1;  }  else  {  checkMatrix[i, j] = 0;  }  }  }  return checkMatrix;  }  //Сложение строк матрицы  public static int[,] AddingLinesMatrixMod2(int[,] matrix, int str1, int str2, int lengthString)  {  //Console.WriteLine(str1 + " и " + str2);  for (int i = 0; i < lengthString; i++)  {  matrix[str1, i] = (matrix[str1, i] + matrix[str2, i]) % 2;  }  return matrix;  }  //вывод матрицы  public static void OutMatrix(int[,] matrix, int k, int n)  {  for (int i = 0; i < k; i++)  {  for (int j = 0; j < n; j++)  {  Console.Write(matrix[i, j]);  //if (j + 1 == k) Console.Write("|");  }  Console.WriteLine();  }  }  static void OutMassive(string Xk)  {  Console.WriteLine(Xk);  }  //вывод одномерного массива  public static void OutMass(int[] mas)  {  Console.WriteLine();  for (int i = 0; i < mas.Length; i++)  {  //if (i == k) Console.Write("|");  Console.Write(mas[i]);  }  //Console.WriteLine("\n");  }  }  } |

Листинг 1 – Код приложения

Сравнение полученной последовательности после исправления ошибок с передаваемой последовательностью:

Ситуация с группами с тремя ошибками:

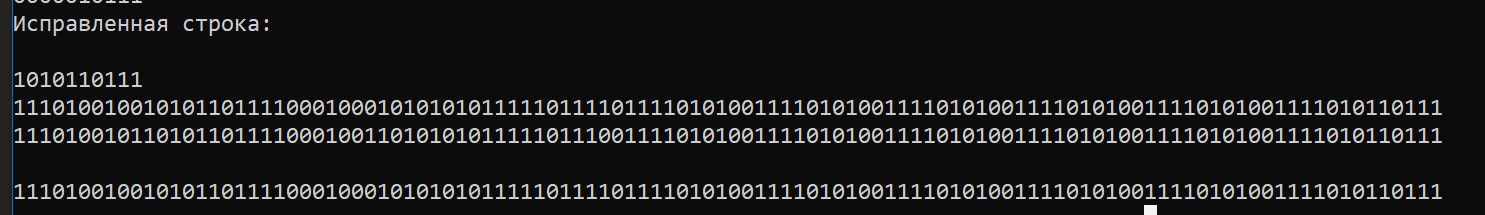


Рисунок 3 – Сравнение с 3 ошибками

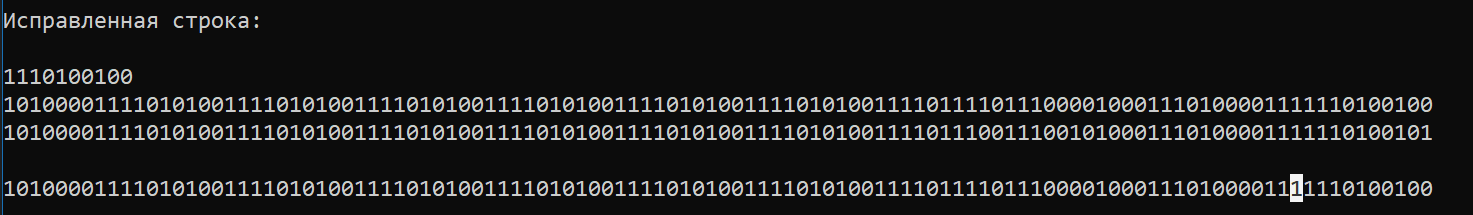


Рисунок 4 – Сравнение с 3 ошибками

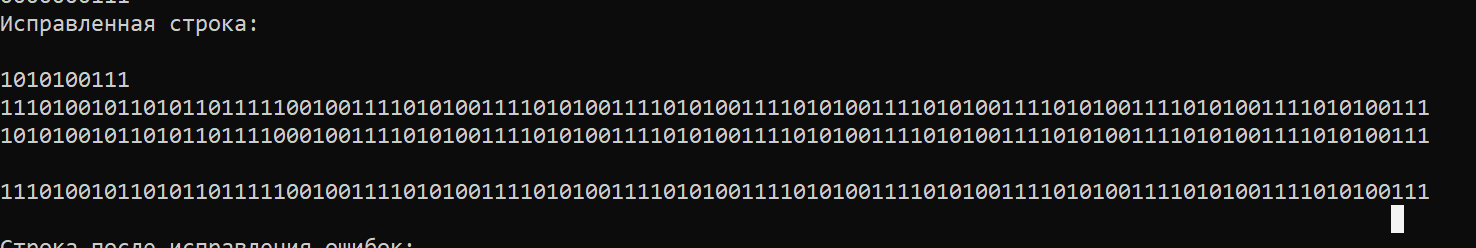


Рисунок 5 – Сравнение с 3 ошибками

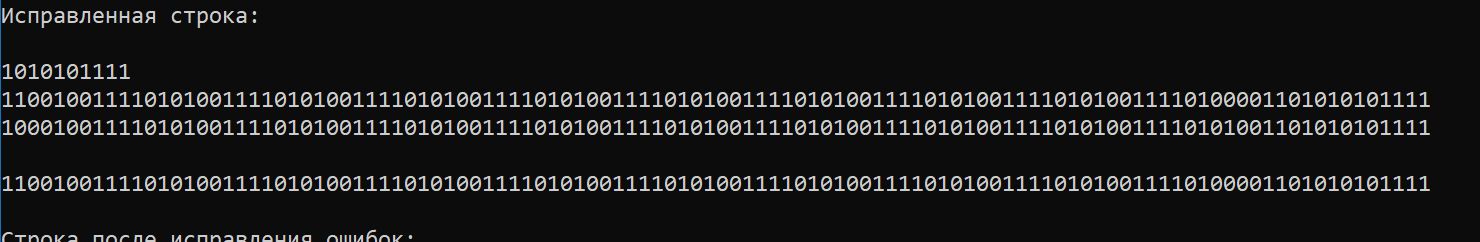
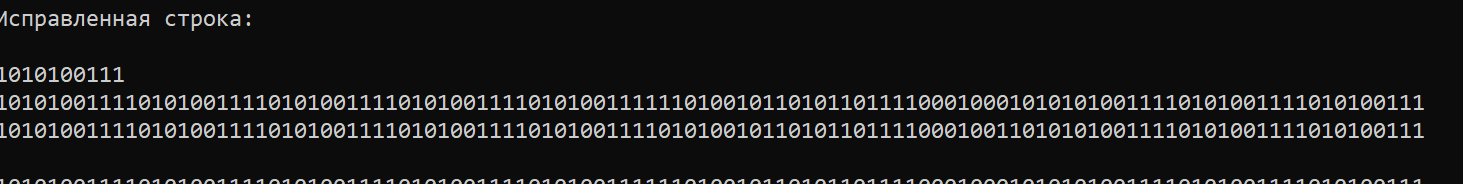


Рисунок 6 – Сравнение с 3 ошибками



Ситуация с группами с пятью ошибками:

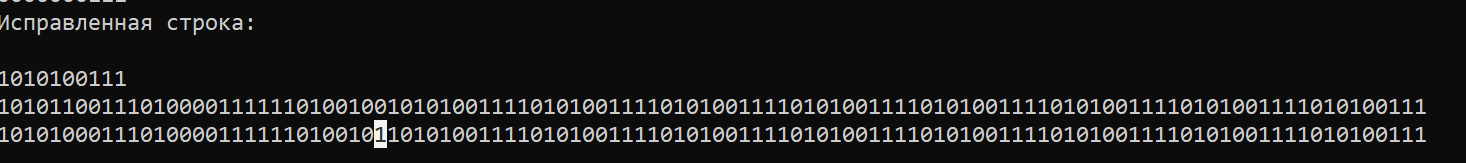


Рисунок 7 – Сравнение с 5 ошибками

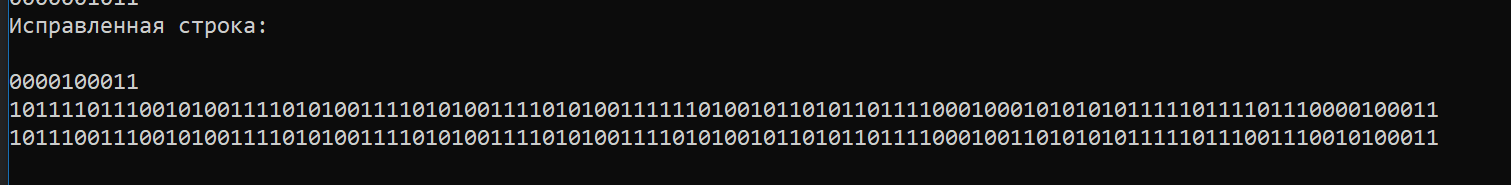


Рисунок 8 – Сравнение с 5 ошибками

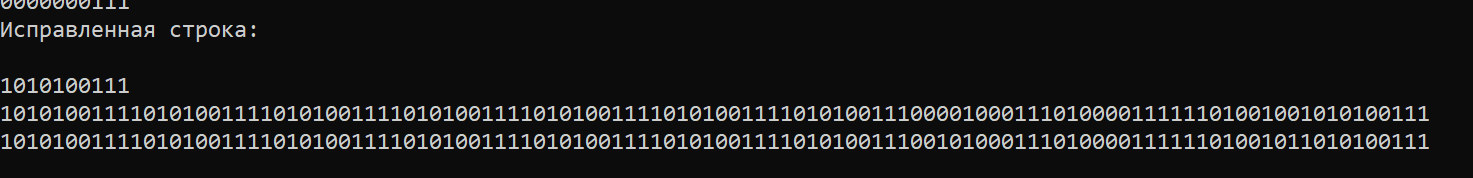


Рисунок 9 – Сравнение с 5 ошибками

Ситуация с группами с семью ошибками:

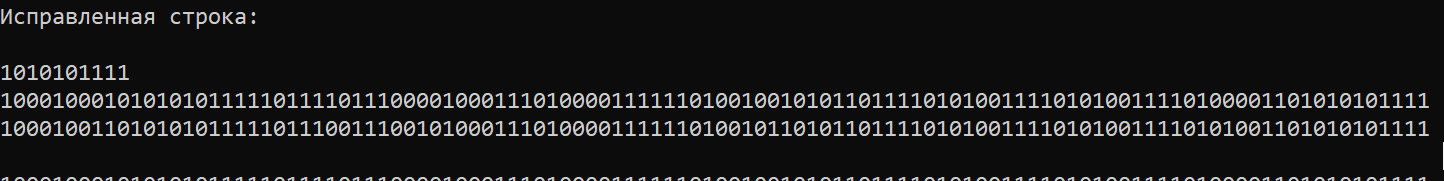


Рисунок 10 – Сравнение с 7 ошибками

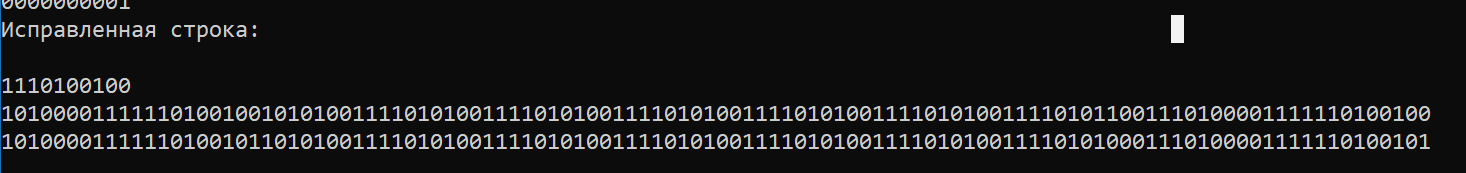


Рисунок 11 – Сравнение с 7 ошибками

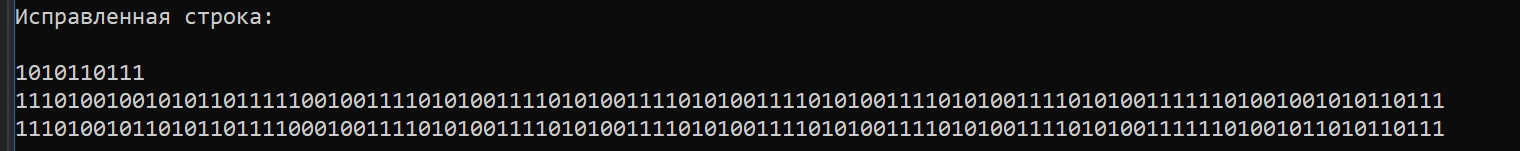


Рисунок 12 – Сравнение с 7 ошибками

**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.

**4.Пояснить принцип работы блокового метода перемежения**.

Метод перемежения подразумевает изменение порядка передачи сообщений.

Для каналов с группированием ошибок часто применяют метод перемежения символов, или декорреляции ошибок. Он заключается в том, что символы, входящие в одну кодовую комбинацию, передаются не непосредственно друг за другом, а перемежаются символами других кодовых комбинаций

Блочное кодирование. Входная информация разбивается на блоки с определенным количеством символов, преобразуется дальше по определенному закону, обеспечивает обнаружение одиночных и групповых ошибок, и обладает возможностью их исправления. Существуют следующие виды блочного кодирования корректирующих кодов: коды повторения, линейные, коды Хемминга, циклические, Галлея, коды Риде-Соломона.

**5. В чем состоит основное отличие блочного перемежителя от других Вам известных?**

Метод перемежения подразумевает изменение порядка передачи сообщений. Входная информация разбивается на блоки с определенным количеством символов, преобразуется дальше по определенному закону, обеспечивает обнаружение одиночных и групповых ошибок, и обладает возможностью их исправления.

**Вывод**: в данной работе было рассмотрено перемежение и деперемежение как варианты кодирования информации. Было установлено, что такой способ должен использоваться в совокупности с другими корректирующими кодами, например Циклическим кодом. Перемежение/деперемежение позволяет разбить исходную строку, на строки нужной длины корректирующего кода, так чтобы можно было исправить как можно больше ошибок.

**Лабораторная работа № 8**

**СЖАТИЕ/РАСПАКОВКА ДАННЫХ МЕТОДОМ БАРРОУЗА – УИЛЕРА**

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

**Теоретические сведения**

Данная лабораторная работа открывает второй раздел практикума, относящийся к методам сжатия данных (англ. data compression), которые играют важную роль в решении проблемы защиты информации, ведь архивация данных необходима не только для экономии места на локальном дисковом носителе, но и для переноса информации, резервирования, резервного копирования и т. п.

Сжатие информации является одним из способов ее кодирования. В основе сжатия данных, 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

• 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

**Все методы сжатия разделяют на два класса: обратимое и необратимое** сжатие, или иначе: сжатие без потерь и сжатие с частичной потерей информации (англ. lossy compression).

**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).

Как видим, Сжатие/распаковка данных методом Барроуза – Уилера 79 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.

**11 вариант - сорокадневный**

**3**. 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.

**4.** Результаты оформить в виде отчета по установленным правилам.

using System;

using System.Linq;

using System.Text;

using System.Diagnostics;

namespace \_8

{

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 'сор' в ASCII: " + str);

Console.WriteLine();

string[] initMessages = new string[] { "марина", "шастовская", "сорокадневный", 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("-----------------------------------------------------");

}

}

}

}

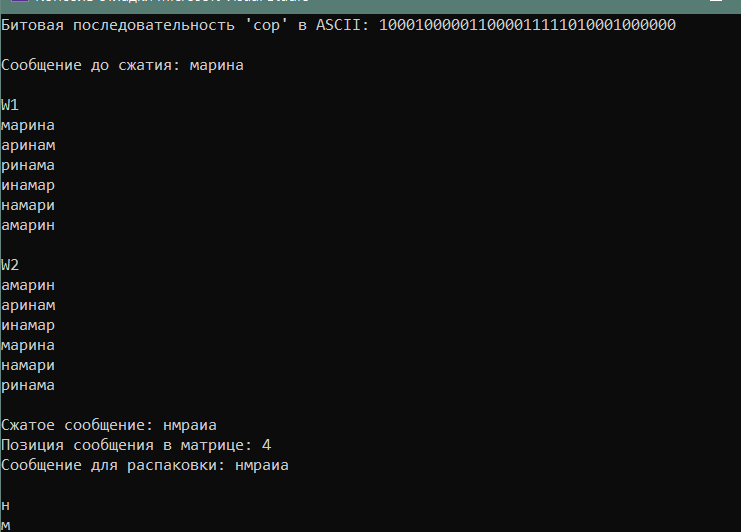


Рисунок 1 – Отладка программы

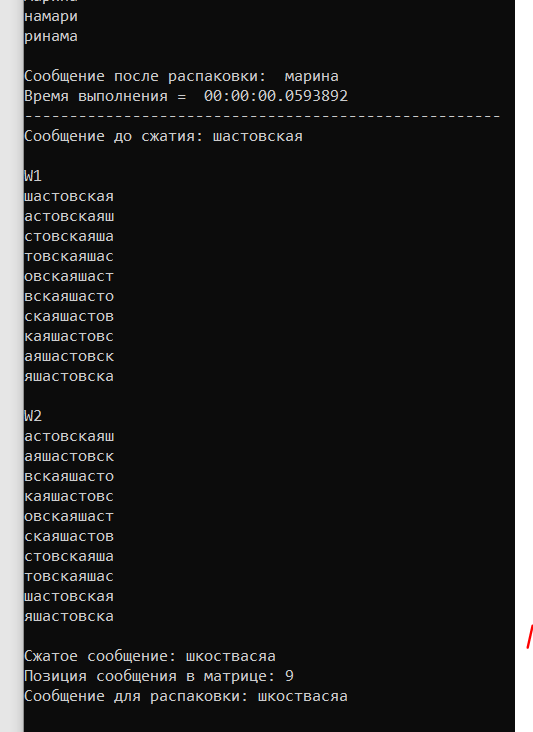


Рисунок 2 – Отладка программы

**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).

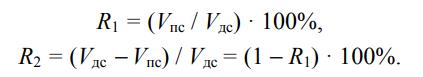
1. **Охарактеризовать основные технические характеристики процессов сжатия/распаковки и результатов.**

• 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.

**Лабораторная работа № 9**

**СЖАТИЕ/РАСПАКОВКА ДАННЫХ НА ОСНОВЕ СТАТИСТИЧЕСКИХ МЕТОДОВ**

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

**Теоретические сведения**

До появления уже упоминавшихся работ К. Шеннона кодирование символов алфавита при передаче сообщения по каналам связи осуществлялось одинаковым количеством битов, получаемым по формуле Хартли (см. формулу (2.2)). Позднее начали появляться способы, кодирующие символы разным числом битов в зависимости от вероятности появления их в тексте, подтверждение чему мы получили при выполнении лабораторной работы № 2. Таким образом, за счет использования для каждого значения байта кодов ASCII (символа алфавита) кода различной длины в соответствии с частостью (вероятностью появления этого символа в сообщении) можно значительно уменьшить общий размер данных.

Эта базовая идея лежит в основе алгоритмов статистических (вероятностных) методов сжатия: Шеннона − Фано и Хаффмана.

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.

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.

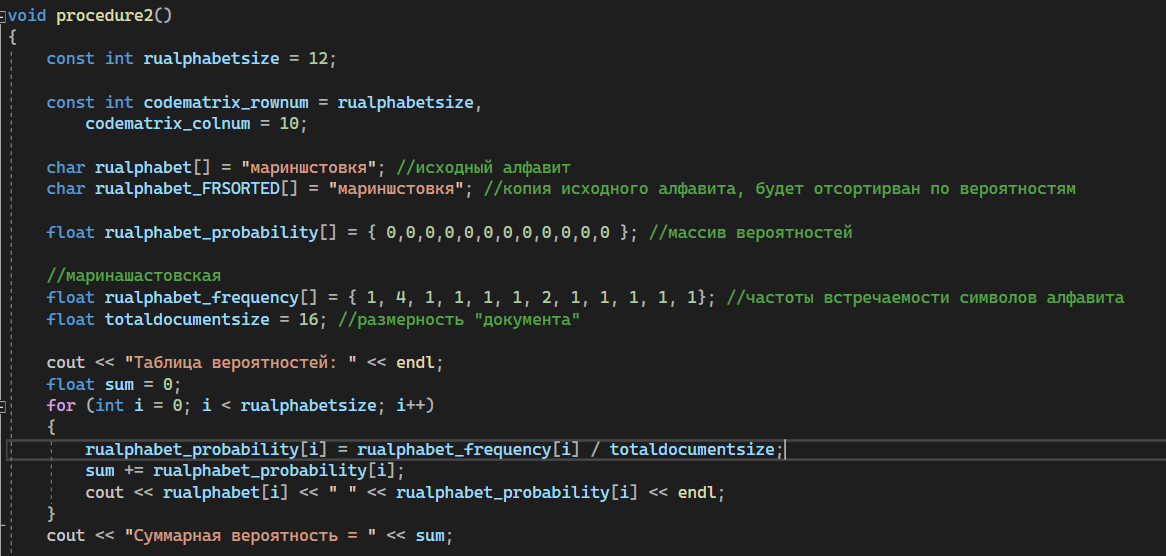
Методы относятся к классу «сжатие без потерь». Различие между двумя рассматриваемыми методами состоит лишь в особенностях формирования таблицы бинарных кодов. При формировании этой таблицы для обоих методов можно воспользоваться статистическими свойствами алфавитов, полученными при выполнении лабораторной работы № 2. преобразовании.

**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. Сжатие/распаковка данных на основе статистических методов 93 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.Выполнить прямое и обратное  
преобразования сообщения, состоящего из собственных имени и  
фамилии.

 Листинг 1 – Фрагмент программы

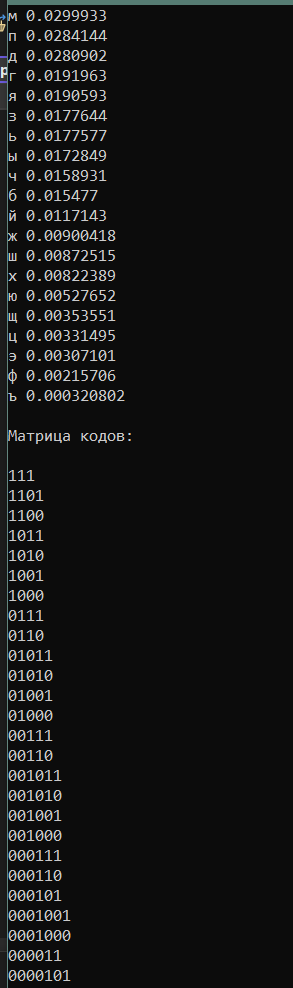
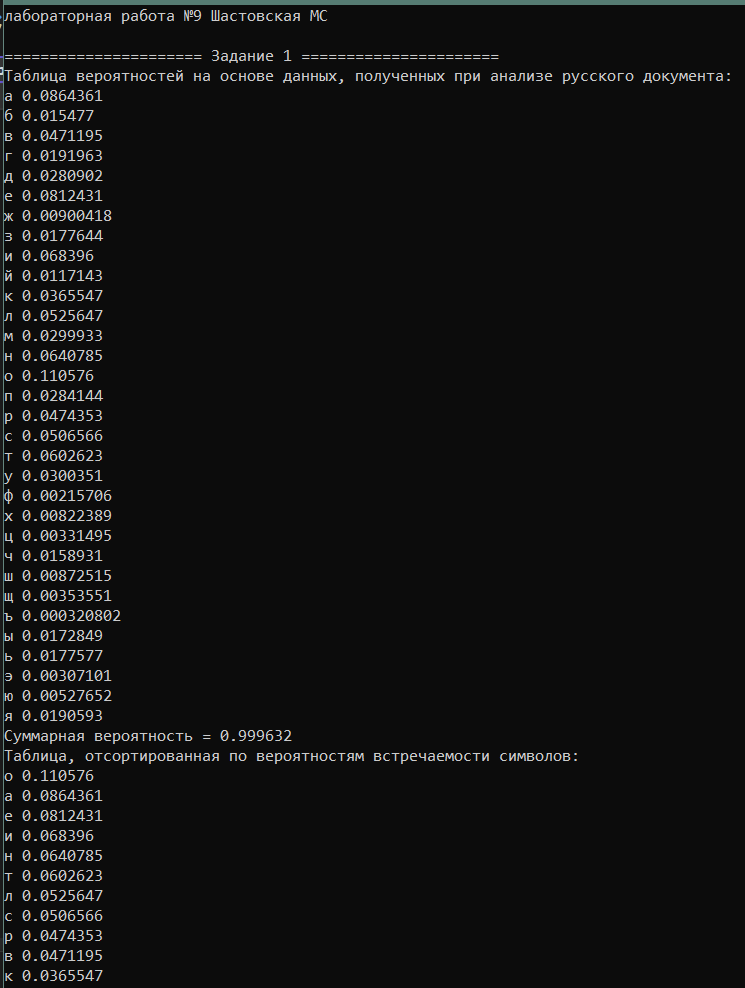


Рисунок 1 – Отладка программы

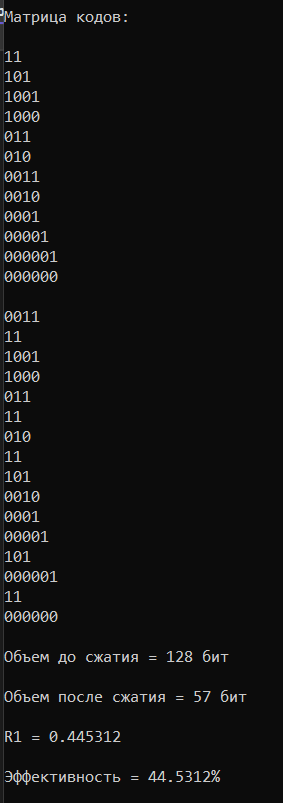
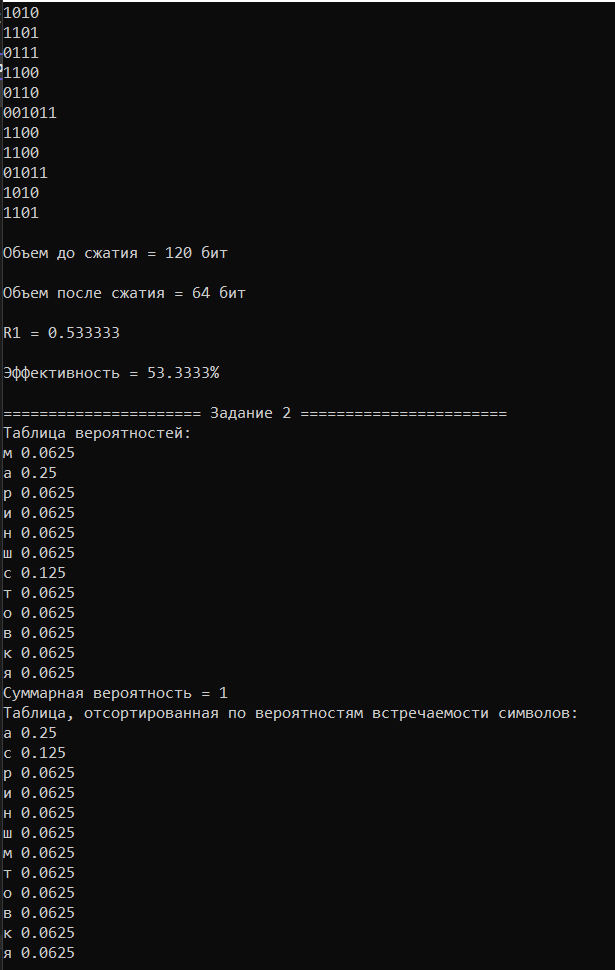


Рисунок 2 – Отладка программы

**Ответы на вопросы:**

**1. Что такое бинарное дерево, чем характеризуется его структура?**Бинарное дерево-это корневое дерево, которое также является упорядо-ченным 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.  
  
**2. 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 пре-фикса другого, кодовое слово, с которого начинается файл, определяется од-нозначно, как и все последующие кодовые слова.

**Вывод**: в данной работе был рассмотрен способ **сжатия данных на основе статистических данных – метод Шеннона – Фано**. Метод не является оптимальным так как при равном распределении вероятностей он может выдавать несколько различных вариантов результатов. Однако метод дает оптимальное распределение вероятностей и максимальное сжатие.

**Лабораторная работа № 10**

**СЖАТИЕ/РАСПАКОВКА ДАННЫХ**

**МЕТОДОМ ЛЕМПЕЛЯ − ЗИВА**

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

**Теоретические сведения**

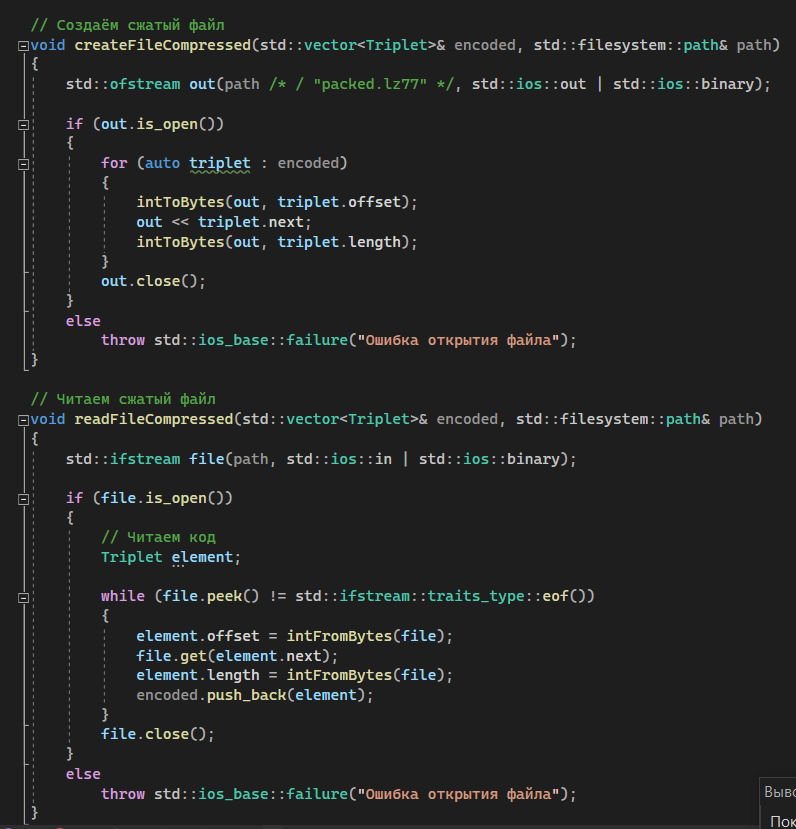
В 1977 г. Авраам Лемпель и Якоб Зив выдвинули идею формирования «словаря» общих последовательностей анализируемых (сжимаемых) данных. При этом сжатие данных осуществляется за счет замены записей соответствующими кодами из словаря.

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» (оригинальную статью см. по ссылке [33]). 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 (англ. sliding dictionary), 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. Передача одного символа в явном виде (s) позволяет разрешить проблему обработки еще ни разу не встречавшихся символов, но существенно увеличивает размер сжатого блока.

**Практическое задание**

1. Разработать авторское приложение в соответствии с целью лабораторной работы. При этом предусмотреть возможность оперативного изменения размеров окон (n1, n2).
2. С помощью приложения выполнить прямое и обратное преобразования произвольного текста длиной несколько килобайт. Формат представления параметров p и q выбрать по указанию преподавателя.
3. Изменяя размеры окон, оценить скорость и эффективность (используя соотношения на с. 76) выполнения операций сжатия/ распаковки.
4. Результаты оформить в виде отчета по установленным правилам.



Листинг 1 – Фрагмент программы

**Перед началом:**

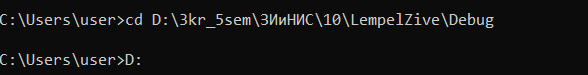


Рисунок 1 – Необходимое условие

**Для сжатия**:

Параметры: имя exe файла (исполняемый), затем имя файла для сжатия, затем флаг c (указывает, что сжимаем – compress)

**Для распаковки**:

Параметры: имя exe файла (исполняемый), затем имя сжатого файла, затем имя результирующего файла, затем флаг d (указывает, что разжимаем – decompress)

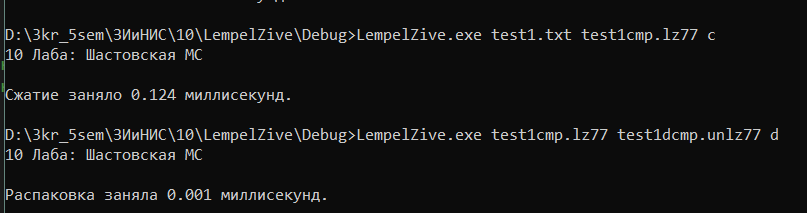


Рисунок 2 – Демонстрация сжатия и распаковки

В папке Debug лежат три папки. Первая – для сжатых файлов. Вторая – исходники (обычные файлы, их будем преобразовывать). Третья – распакованные.

**Ответы на вопросы**

**1. К какому классу методов сжатия относится метод LZ и почему?**  
LZ методы относятся к классу адаптивных словарных кодов. При ис-пользовании таких кодов составляется адаптивная таблица, или словарь, в ко-торую записываются последовательности символов источника и соответству-ющие им кодовые слова.  
**2. Какое значение имеет последний символ в сжатом текстовом сооб-щении методом LZ?**Показывает каким символом прервалась выбранная последовательность символов  
**3. Какие модификации метода LZ77 Вам известны? В чем заключается особенность модификаций?**алгоритмы типа LZ77 и алгоритмы типа LZ78  
Суть метода LZ77 (как и последующих его модификаций) состоит в сле-дующем: упаковщик постоянно хранит некоторое количество последних об-работанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые.

**Вывод**: в данной работе были приобретены практических навыков использования метод Лемпеля − Зива (Lempel-Ziv) для сжатия/распаковки данных.

**Лабораторная работа № 11**

**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.

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

**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. Например, для варианта 110 Лабораторная работа № 11 № 1 такой конкатенацией будет последовательность «летоисчислениевремяпрепровождение».

using System.Text;

namespace lab11

{

public class Program

{

static void Main(string[] args)

{

{

string word = "Шастовская";

int wordLength = word.Length;

Compressor wordCompressor = new Compressor();

wordCompressor.Build(word);

Console.WriteLine("Интервалы:");

foreach (var i in wordCompressor.Nodes)

{

Console.WriteLine($"p({i.Symbol}) = {i.High - i.Low}");

}

Console.WriteLine();

foreach (var i in wordCompressor.Nodes)

{

Console.WriteLine($"{i.Symbol} {i.Low} - {i.High}");

}

Console.WriteLine();

var compressResult = wordCompressor.Compress(word);

Console.WriteLine("Сжатые данные:");

Console.WriteLine(InfoString.Sb.ToString());

Console.WriteLine($"Результат: {compressResult}\n");

var decompressResult = wordCompressor.Decompress(compressResult, wordLength, wordLength / 2 + 1);

Console.WriteLine("Расжатые данные:");

Console.WriteLine(InfoString.Sb.ToString());

Console.WriteLine($"Результат: {decompressResult}");

}

Console.WriteLine("\n\n");

{

string word = "песнетворчество";

int wordLength = word.Length;

Compressor wordCompressor = new Compressor();

wordCompressor.Build(word);

Console.WriteLine("Интервалы:");

foreach (var i in wordCompressor.Nodes)

{

Console.WriteLine($"p({i.Symbol}) = {i.High - i.Low}\n");

}

foreach (var i in wordCompressor.Nodes)

{

Console.WriteLine($"{i.Symbol} {i.Low} - {i.High}\n");

}

var compressResult = wordCompressor.Compress(word);

Console.WriteLine("Сжатые данные");

Console.WriteLine(InfoString.Sb.ToString());

Console.WriteLine($"Результат: {compressResult}\n");

var decompressResult = wordCompressor.Decompress(compressResult, wordLength, wordLength / 2 + 1);

Console.WriteLine("Расжатые данные:");

Console.WriteLine(InfoString.Sb.ToString());

Console.WriteLine($"Результат: {decompressResult}");

}

Console.ReadLine();

}

}

public class Compressor

{

public List<Node> Nodes { get; set; }

public Dictionary<char, decimal> Frequencies { get; set; }

public Node ResultNode { get; set; }

public void Build(string source)

{

Nodes = new List<Node>();

decimal inc = 1 / (decimal)source.Length;

Frequencies = new Dictionary<char, decimal>();

for (int i = 0; i < source.Length; i++)

{

if (!Frequencies.ContainsKey(source[i]))

{

Frequencies.Add(source[i], 0);

}

Frequencies[source[i]] += inc;

}

Frequencies = Frequencies.OrderBy(x => x.Value).ToDictionary(x => x.Key, y => y.Value);

decimal low = 0;

foreach (var item in Frequencies)

{

Nodes.Add(new Node { Symbol = item.Key, Low = Math.Round(low, 5), High = Math.Round(low + item.Value, 5) });

low += item.Value;

}

}

public decimal Compress(string source)

{

InfoString.Sb = new StringBuilder();

ResultNode = new Node { Symbol = '\*', High = 1, Low = 0 };

foreach (var item in source)

{

decimal oldHigh = ResultNode.High;

decimal oldLow = ResultNode.Low;

InfoString.Sb.Append(ResultNode.ToString()).Append(Environment.NewLine);

ResultNode.Symbol = '\*';

ResultNode.High = oldLow + (oldHigh - oldLow) \* Nodes.Find(x => x.Symbol == item).High;

ResultNode.Low = oldLow + (oldHigh - oldLow) \* Nodes.Find(x => x.Symbol == item).Low;

}

InfoString.Sb.Append(ResultNode.ToString()).Append(Environment.NewLine);

return ResultNode.Low;

}

public string Decompress(decimal compress, int leng, int t)

{

StringBuilder sb = new StringBuilder();

InfoString.Sb = new StringBuilder();

for (int i = 0; i < leng; i++)

{

char symbol = Nodes.Find(x => Math.Round(compress, t) >= x.Low && Math.Round(compress, t) < x.High).Symbol;

InfoString.Sb.Append(compress.ToString() + $"\t-- {symbol}").Append(Environment.NewLine);

sb.Append(symbol);

Node tempNode = Nodes.Find(x => x.Symbol == symbol);

compress = (compress - tempNode.Low) / (tempNode.High - tempNode.Low);

}

return sb.ToString();

}

}

public class Node

{

public char Symbol { get; set; }

public decimal High { get; set; }

public decimal Low { get; set; }

public override string ToString()

{

return string.Format("Low: {0} | High: {1}", Low.ToString(), High.ToString());

}

}

public class InfoString

{

public static StringBuilder Sb { get; set; }

}

}

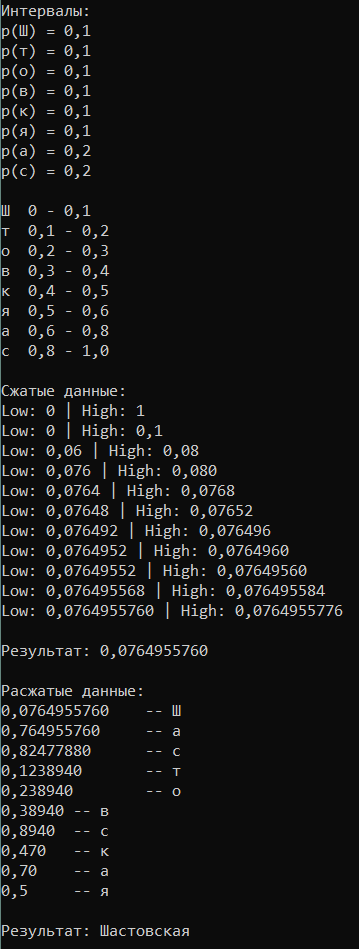
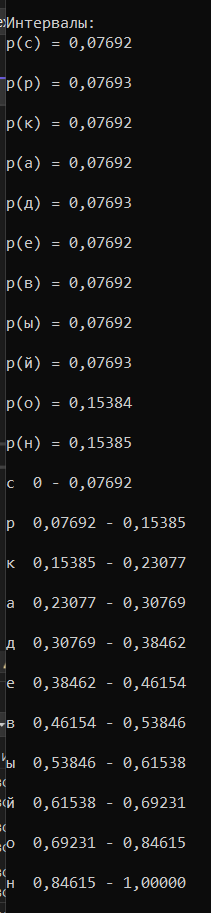
 

Рис.1 Отладка программы

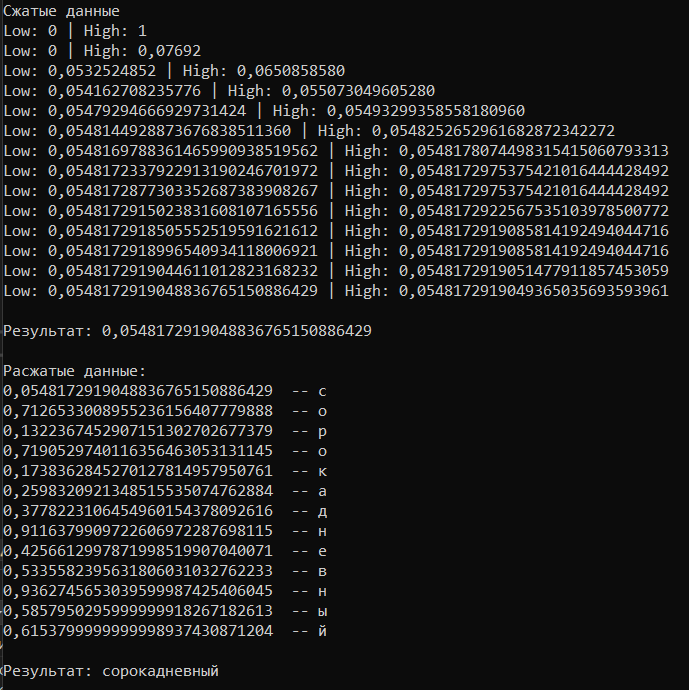


Рис.2 Отладка программы

**ОТВЕТЫ НА ВОПРОСЫ:**  
**1. В чем заключается суть арифметического метода сжатия?**Основная идея арифметического метода сжатия заключается в том, чтобы присваивать коды не отдельным символам, а их последовательностям  
**2. Пояснить принцип определения границ отрезков на каждом шаге (на каждом рабочем интервале) прямого и обратного преобразования.**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  
**3. Какой недостаток, свойственный префиксным методам, отсутствует в арифметическом кодировании?**  
Требуемая для представления интервала [Hi; Li] точность возрастает вместе с длиной анализируемого текста; постепенное выполнение алгоритма помогает преодолеть эту проблему.

**ЛАБОРАТОРНАЯ РАБОТА №12**

**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. В зависимости от используемых методов в теории чисел рассматривают несколько направлений.

Нас будут интересовать вопросы делимости целых чисел, вычисления наибольшего общего делителя (НОД), разложение числа на простые множители, малая теорема Ферма́, теорема Эйлера, элементы теории вычетов.

Простое число – если число не имеет делителей, кроме самого себя и единицы, то оно называется простым, а если у числа есть еще делители, то составным. Взаимно простые числа – целые числа, наибольший общий делитель которых равен 1.

Наибольший общий делитель двух и более натуральных чисел – это наибольшее из натуральных чисел, на которое делится каждое из данных чисел.

Каждое натуральное число, большее единицы, делится по крайней мере на два числа: на 1 и на само себя. Если число не имеет делителей, кроме самого себя и единицы, то оно называется простым, а если у числа есть еще делители, то составным.

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

Понятие делимости чисел (см. определение 3) является одним из важных в теории чисел.

**Расширенный алгоритм Евклида.** Этот алгоритм состоит из двух этапов: собственно алгоритма Евклида и вычислений на основе обратных подстановок или последовательного выражения остатков в каждом из шагов предыдущего этапа с соответствующим приведением подобных на каждом шаге.

**Ход работы**

Разработать авторское приложение в соответствии с целью лабораторной работы. Приложение должно реализовывать следующие операции:

* Вычислять НОД двух или трех чисел
* Выполнять поиск простых чисел

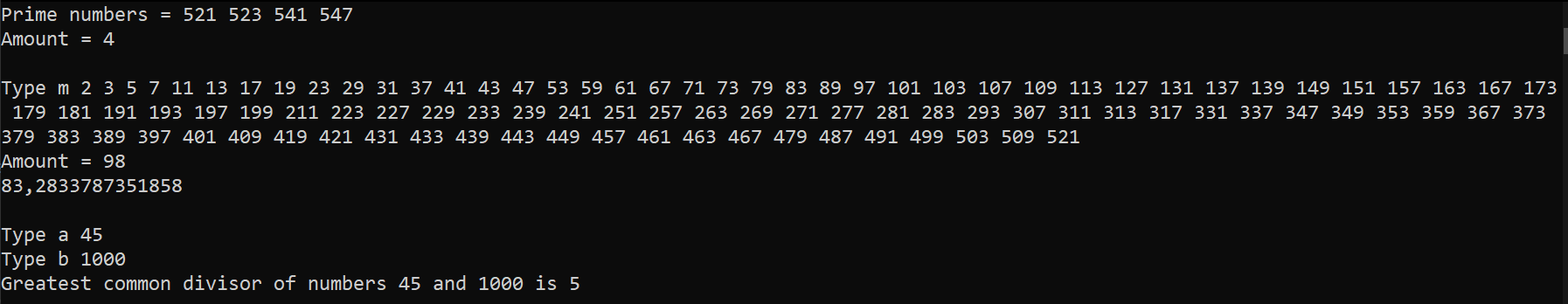
****

Рисунок 1 – Результат выполнения программы

|  |
| --- |
| using System;  namespace TwelvethLab  {  internal class Program  {  public static bool IsPrime(int number)  {  for (int i = 2; i < number; i++)  {  if (number % i == 0)  return false;  }  return true;  }  static int GCD(int a, int b)  {  while (b != 0)  {  var t = b;  b = a % b;  a = t;  }  return a;  }  static void Main(string[] args)  {  Console.Write("Prime numbers = ");  int count = 0;  for (int i = 521; i <= 553; i++)  {  if (IsPrime(i))  {  Console.Write($"{i} ");  count++;  }  }  Console.WriteLine("\nAmount = {0}",count);  Console.Write("\nType m ");  int count2 = 0;  for (int i = 2; i <= 521; i++)  {  if (IsPrime(i))  {  Console.Write($"{i} ");  count2++;  }  }  Console.WriteLine("\nAmount = {0}", count2);  double inter = 521/Math.Log(521);  Console.WriteLine(inter);  Console.Write("\nType a ");  int a = Convert.ToInt32(Console.ReadLine());  Console.Write("Type b ");  int b = Convert.ToInt32(Console.ReadLine());  Console.WriteLine("Greatest common divisor of numbers {0} and {1} is {2}",a,b,GCD(a,b));  Console.ReadLine();  }  }  } |

Листинг 1 – Код 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 разговор-ной 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).  
**2. Сформулировать основную теорему арифметики. Представить примеры ее применения.**Каждое натуральное число n, за исключением единицы, раскладывается в произведение простых сомножителей, причем единственным образом с точно-стью до порядка следования сомножителей:  
где p1, …, pk – простые числа.  
**3. Пояснить сущность проблемы факторизации и ее связь с прикладной криптографией.**Сложность решения задачи разложения больших чисел на простые сомножители, известной как «проблема факторизации», 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.

**Вывод**: в данной работе были приобретены практические навыки выполнения операций с числами для решения задач в области криптографии, закреплены знания по общей арифметике и алгоритмам нахождения простых чисел и НОД.