

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

**КУРСОВАЯ РАБОТА**

Тема: «РЕАЛИЗАЦИЯ МОДЕЛИ КАНАЛА СВЯЗИ НА ОСНОВЕ ЦИКЛИЧЕСКИХ КОДОВ»

Дисциплина: «Теория информации»

Специальность: 10.05.01 Компьютерная безопасность

Специализация: Математические методы защиты информации

Обозначение курсовой работы ТИ.730000.000 Группа ВКБ32

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Р. Приходько

подпись, дата

Курсовая работа защищена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ст. преподаватель, И.А. Алферова

подпись, дата

Ростов-на-Дону

2023



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**ЗАДАНИЕ**

на выполнение курсовой работы

Тема «Реализация модели канала связи на основе циклических кодов»

Дисциплина: Теория информации

Обучающийся: Приходько Алексей Романович

Обозначение курсовой работы ТИ.730000.000 Группа: ВКБ32

Срок представления работы к защите «\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 г.

Исходные данные для курсовой работы:

1. Колесник В.Д., Мирончиков Е.Т. Декодирование циклических кодов.
2. Блейхут Р.Э., Теория и практика кодов, контролирующих ошибки.

|  |  |  |  |
| --- | --- | --- | --- |
| **Содержание пояснительной записки**  Введение:  Помехоустойчивое кодирование необходимо для устранения ошибок, которые возникают в процессе передачи или хранения информации. Помехоустойчивое кодирование является важным инструментом для обеспечения надежной передачи данных в различных областях, таких как телекоммуникации, компьютерные сети, хранение данных и другие. | | | |
| Разделы основной части: | | | |
| 1. Помехоустойчивое кодирование. Общая информация по кодированию.   Циклические коды.   1. Разработка программного средства. Обоснование выбора языка и среды разработки.   Архитектура программного средства. Блок-схемы кода и алгоритма циклических кодов. Основные методы и классы. Демонстрация программного средства   1. Тестирование программного средства.   Заключение:  Помехоустойчивое кодирование является необходимым для защиты информации от помех и ошибок, которые могут возникнуть в процессе передачи и хранения данных. В представленной работе был проведен анализ помехоустойчивого кодирования, а также программно реализована модель канала связи на основе циклических коды. | | | |
| Руководитель работы | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | И.А. Алферова |
|  |  |  |
| Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | А.Р. Приходько |

Содержание

[Введение 5](#_Toc137650437)

[1 Помехоустойчивое кодирование 6](#_Toc137650438)

[1.1 Общая информация по кодированию 6](#_Toc137650439)

[1.2 Циклические коды 9](#_Toc137650440)

[2 Разработка программного средства 14](#_Toc137650441)

[2.1 Обоснование выбора языка и среды разработки 14](#_Toc137650442)

[2.2 Архитектура программного средства 15](#_Toc137650443)

[2.3 Блок-схемы кода и алгоритма циклических кодов 16](#_Toc137650444)

[2.4 Основные методы и классы 19](#_Toc137650445)

[2.5 Демонстрация программного средства 20](#_Toc137650446)

[3 Тестирование программного средства 24](#_Toc137650447)

[Заключение 29](#_Toc137650448)

[Перечень использованных информационных ресурсов 30](#_Toc137650449)

[Приложение А 31](#_Toc137650450)

## Введение

Помехоустойчивое кодирование необходимо для устранения ошибок, которые возникают в процессе передачи или хранения информации. При передаче информации по каналу связи возникают помехи, ошибки и небольшая часть информации теряется. Кодирование происходит путем добавления дополнительной информации в исходное сообщение, которая позволяет обнаруживать и исправлять ошибки при их возникновении.

Первые коды с обнаружением ошибок были предложены Ричардом Хэммингом в 1950-х годах, затем им были разработаны более сложные коды, такие как циклические коды, которые являются наиболее ценным достоянием теории кодирования.

Помехоустойчивое кодирование является важным инструментом для обеспечения надежной передачи данных в различных областях, таких как телекоммуникации, компьютерные сети, хранение данных и другие.

Объект исследования: циклические коды.

Предмет исследования: программная реализация модели канала связи на основе циклических кодов.

Целью работы является исследование возможностей циклических кодов и программная реализация их на практике.

Задачи исследования:

1. Изучение принципов построения помехоустойчивых кодов с заданными свойствами, методов кодирования и декодирования помехоустойчивых кодов, в частности циклических.
2. Изучение принципов построения циклических кодов и их свойств.
3. Программная реализация модели канала связи на основе циклических кодов.

# 1 Помехоустойчивое кодирование

## 1.1 Общая информация по кодированию

История помехоустойчивого кодирования начинается с изобретения в 1950 году американским математиком Ричардом Хеммингом двоичного кода, способного обнаруживать и исправлять ошибки в передаваемых сообщениях. В дальнейшем Хемминг разработал целый ряд кодов, которые стали широко применяться в телекоммуникационных системах и компьютерных сетях.

В 1960-х годах были разработаны свёрточные коды, которые позволяют обеспечить высокую корректирующую способность при передаче информации по каналам с шумом и помехами. В 1970-х годах были созданы блочные коды, которые позволяют обеспечить высокую корректирующую способность при передаче информации по каналам с ошибками.

Сегодня исследования в области помехоустойчивого кодирования продолжаются, и появляются новые методы и технологии, которые позволяют обеспечить более эффективную защиту информации от ошибок и помех при передаче и хранении.

Цель помехоустойчивого кодирования – добавить избыточность в передаваемое сообщение, чтобы обнаружить и исправить ошибки, возникающие при передаче по каналу связи. Для этого используются кодеры и декодеры, которые добавляют и удаляют избыточность в сообщении.

Основные задачи помехоустойчивого кодирования состоят в построении кодов с высокой корректирующей способностью, обеспечивающих максимальный ЭВК (эффективный выигрыш в канале) при требуемой скорости передачи информации [4]. Для этого используются различные методы кодирования и декодирования, которые позволяют обнаруживать и исправлять ошибки в передаваемой информации.

Помехоустойчивые коды можно разделить на коды с обнаружением ошибок и коды с исправлением ошибок. Коды с обнаружением ошибок могут определить наличие ошибок в передаваемом сообщении, но не могут их исправить. Коды с исправлением ошибок могут обнаружить и исправить ошибки в передаваемом сообщении [5].

Все помехоустойчивые коды можно разделить на два основных класса: блочные и непрерывные (рекуррентные или цепные).

В блочных кодах каждому сообщению (или элементу сообщения) сопоставляется кодовая комбинация (блок) из определенного количества разрядов. Блоки кодируются и декодируются отдельно друг от друга.

Блочные коды могут быть равномерными, когда длина кодовых комбинаций постоянна, или неравномерными, когда непостоянно.

В непрерывных кодах введение избыточности в последовательность входных символов осуществляется без разбивки ее на отдельные блоки. Процессы кодирования и декодирования в непрерывных кодах имеют также непрерывный характер.

Как блочные, так и непрерывные коды в зависимости от методов внесения избыточности подразделяются на разделимые и неразделимые. В разделимых кодах четко разграничена роль отдельных символов. Одни символы являются информационными, другие являются проверочными и служат для обнаружения и исправления ошибок. Разделимые блочные коды называются обычно -кодами, где – длина кодовых комбинаций, – число информационных символов в комбинациях [6].

Неразделимые коды не имеют четкого разделения кодовой комбинации на информационные и проверочные символы.

Разделимые блочные коды делятся, в свою очередь, на несистематические (нелинейные) и систематические (линейные). Несистематические разделимые коды строятся таким образом, что проверочные символы определяются как сумма подблоков длины , на которые разделяется блок информационных символов.

У разделимых систематических кодов значение проверочных символов определяется в результате проведения линейных операций над определенными информационными символами. Для случая двоичных кодов каждый проверочный символ выбирается таким, чтобы его сумма по модулю два с определенными информационными символами стала равной нулю (т.е. сумма единиц была четной). Декодирование сводится к проверке на четность определенных групп символов. В результате таких проверок дается информация о наличии ошибок, а в случае необходимости – о позиции символов, где имеются ошибки [1].

Существует несколько распространенных методов исправления ошибок в кодах, используемых в современных компьютерных сетях и системах хранения данных. Некоторые из наиболее распространенных методов исправления ошибок в кодах включают:

* код Хэмминга – это тип блочного кода, который может обнаруживать и исправлять однобитовые ошибки при передаче данных;
* свёрточные коды – это коды, которые относятся к непрерывным блочным кодам. Такие коды используются для исправления ошибок при передаче данных. Это особенно полезно для исправления ошибок при передаче данных в режиме реального времени, например, при потоковой передаче видео и аудио;
* циклический код – это линейный, блочный код, который обладает свойством цикличности, то есть каждая циклическая перестановка кодового слова также является кодовым словом. Такие коды особенно хорошо применяются для оценки достоверности информации, размещенных в заголовках пакетов данных.

Большинство из наиболее распространенных методов исправления ошибок являются линейными, блочными, главная характеристика таких кодов состоит в том, что это – канальный код фиксированной длины. Блочные, линейные коды позволяют представить информационные и кодовые слова в виде двоичных векторов, что позволяет описать процессы кодирования и декодирования с помощью аппарата линейной алгебры, с учетом того, что компонентами вводимых векторов и матриц являются символы «0» и «1». Операции над двоичными компонентами производятся при этом по правилам арифметики по модулю 2.

Минимальное расстояние Хемминга является важной характеристикой линейного блочного кода.

Расстоянием Хемминга (метрикой Хемминга) между двумя кодовыми словами и называется количество отличных бит на соответствующих позициях, то есть число «единиц» в векторе .

Минимальное расстояние линейного кода является минимальным из всех расстояний Хемминга всех пар кодовых слов.

Вес вектора — расстояние Хемминга между этим вектором и нулевым вектором, иными словами — число ненулевых компонент вектора.

Минимальное расстояние Хемминга является важной характеристикой линейного блокового кода. Она определяет другую, не менее важную характеристику — корректирующую способность: , здесь угловые скобки обозначают округление «вниз». Корректирующая способность определяет, какое максимальное число ошибок в одном кодовом слове код может гарантированно исправить.

## 1.2 Циклические коды

Циклические коды используются для преобразования информации и защиты ее от ошибок, включая обнаружение и исправление ошибок. Они незаменимы при необходимости передавать информацию в каналах связи, в которых отсутствует возможность повторной передачи данных [9]. Данное название происходит от основного свойства этих кодов: если некоторая кодовая комбинация принадлежит циклическому коду, то комбинация, полученная циклической перестановкой исходной комбинации (циклическим сдвигом), также принадлежит данному коду [7].

Вторым свойством всех разрешенных комбинаций циклических кодов является их делимость без остатка на некоторый выбранный полином, называемый производящим или порождающим.

Циклические коды представляют кодовые комбинации в виде полинома. Так, -элементную кодовую комбинацию можно описать полиномом степени, в виде , где , причем соответствуют нулевым элементам комбинации, – ненулевым [3]. Так, например, комбинации битов 10101 соответствует полином: .

Порождающим полиномом циклического (n, k) кода C называется такой ненулевой полином , степень которого наименьшая, и коэффициент при старшей степени [10].

С порождающим полиномом связаны две важные теоремы и одно не менее важное следствие.

Первая теорема гласит, что если C – циклический код, и – его порождающий полином, то степень равна , и каждое кодовое слово может быть единственным образом представлено в виде , где степень меньше или равна .

Вторая теорема: — порождающий полином циклического кода — является делителем двучлена . У этой теоремы есть важное следствие, которое определяет число символов в информационном слове и количество проверочных символов : в качестве порождающего полинома можно выбирать любой полином делитель . Степень выбранного полинома будет определять количество проверочных символов , число информационных символов [2].

Для кодирования сообщения циклическим кодом используется следующий алгоритм:

* исходное сообщение представляется в виде последовательности битов, после чего преобразуется в полином;
* кодовое слово получается путём произведения информационного полинома на порождающий: ;
* полученное кодовое слово передается по каналу связи.

При декодировании сообщения циклическим кодом используются различные алгоритмы декодирования, наиболее известные из них: декодирование по таблице, декодирование циклических кодов Меггитта и декодирование с вылавливанием ошибок. Они позволяет определить ошибки в переданном сообщении и исправить их.

Алгоритм декодирования по таблице заключается в следующем: на вход поступает полином , если такой полином , что , то будет возвращён верный кодовый полином и исходя из него можно найти информационное слов , иначе выполняем следующие шаги:

1. Находим соответствующий синдром: .
2. Если присутствует в таблице, то определяем соответствующий ему полином ошибок , который показывает ошибочные биты, иначе возвращаем сообщение об ошибке декодирования.
3. Находим кодовое слово: .
4. Находим информационное слово: .

Главным недостатком данного алгоритма является то, что таблица подстановки может быть большой и требовать значительного объема памяти для хранения, особенно для более длинных кодов.

Ещё одним алгоритмом декодирования циклических кодов является алгоритм декодирования Меггитта. В данном случае на вход подаётся вектор , если и при этом , то на выходе будет возвращён вектор , иначе выполняем следующие шаги:

1. Вычисляем по принятому вектору .
2. Находим синдром .
3. Если полином присутствует в правой колонке сокращённой таблицы, то определяем соответствующий ему полином и переходим на шаг 7.
4. Для вычисляем следующее , если присутствует в правой колонке сокращённой таблицы, то определяем соответствующий ему полином , запоминаем число , а также переходим к шагу 6.
5. Возвращаем сообщение об ошибке декодирования.
6. Вычисляем: .
7. Находим кодовое слово: ; после чего находим информационное слово: .

Недостатком этого алгоритма является то, что данный алгоритм может быть не в состоянии исправить все ошибки в полученном кодовом слове, в зависимости от конкретного кода и шаблона ошибок. Алгоритм может быть дорогостоящим с точки зрения вычислений для больших кодов, особенно если таблицу поиска кодового слова по синдрому необходимо генерировать "на лету".

Одним из самых оптимальных алгоритмов является алгоритм декодирования с вылавливанием ошибок [8]:

1. Принятый многочлен делим с остатком на .
2. Пусть . Если , то полагается, что ошибок не происходило. Возвращаем в этом случае .
3. Если , то . Возвращаем кодовое слово без ошибки: .
4. Цикл до : вычисляем , если , то и возвращаем кодовое слово без ошибки: .

Именно этот алгоритм был выбран для программной реализации декодера циклических кодов.

Преимущества циклических кодов с исправлением ошибок включают:

* простые в реализации устройства кодирования и декодирования;
* циклическое свойство кодов обеспечивает эффективное кодирование и декодирование данных, которые передаются циклическим или периодическим образом;
* циклические коды являются частным случаем линейных кодов, что означает, что они могут быть легко проанализированы и оптимизированы с использованием методов линейной алгебры;
* циклические коды могут быть спроектированы так, чтобы иметь высокий уровень возможности исправления ошибок, что делает их полезными для приложений, где целостность данных имеет решающее значение, например, в телекоммуникациях или цифровом хранилище.

В целом, циклические коды являются мощным инструментом для исправления ошибок, который обладает многими преимуществами по сравнению с другими типами кодов. Их простота, эффективность и высокий уровень возможности исправления ошибок делают их хорошо подходящими для широкого спектра применений. Именно поэтому они и были выбраны мною для их исследования и программной реализации.

# 2 Разработка программного средства

## 2.1 Обоснование выбора языка и среды разработки

Python имеет следующие преимущества для разработки канала связи на основе циклических кодов:

* простой и понятный синтаксис, что делает его более читаемым и понятным для разработчиков;
* большая стандартная библиотека, содержащая многократно используемые коды практически для любой задачи, что позволяет ускорить разработку;
* объектно-ориентированная архитектура, которая позволяет создавать более сложные программы и использовать наследование и полиморфизм;
* динамическая типизация и автоматическое управление памятью, что упрощает процесс разработки и уменьшает количество ошибок;
* полная интроспекция и механизм обработки исключений, что позволяет быстро находить и исправлять ошибки;
* поддержка различных операционных систем, таких как Windows, macOS, Linux и Unix.

В целом, Python является эффективным, простым в изучении и использовании языком программирования, который имеет множество преимуществ для разработки канала связи на основе циклических кодов.

VS Code – это бесплатный редактор исходного кода, который имеет множество преимуществ. Некоторые из них включают в себя:

* легкость в установке и использовании на разных операционных системах;
* большое количество плагинов и расширений, включая расширение "Python", которое позволяет редактировать, отлаживать и тестировать код;
* встроенный отладчик, который упрощает процесс отладки кода, в частности Python кода;
* интеграция с системой контроля версий Git, что позволяет управлять исходным кодом проекта;
* поддержка виртуальных сред, которые позволяют создавать изолированные среды для различных проектов;
* интерактивные вычисления на Jupyter Notebooks.

В целом, VS Code является отличным выбором для разработки благодаря своей легкости в использовании, большому количеству плагинов и расширений, а также интеграции с системой контроля версий Git.

## 2.2 Архитектура программного средства

Архитектура программы соответствует функциональной. Функциональная архитектура представляет собой набор функций и их подфункций, определяющих преобразования, осуществляемые системой при выполнении своего назначения.

Функциональная архитектура программного средства для разработки канала связи на основе циклических кодов включает в себя следующие функциональные блоки:

* блок кодирования, который отвечает за преобразование исходных данных в закодированные данные;
* блок декодирования, который отвечает за преобразование закодированных данных обратно в исходные данные;
* блок обработки ошибок, который отвечает за обнаружение и исправление ошибок в закодированных данных;
* блок интерфейса пользователя, который отвечает за взаимодействие с пользователем и отображение результатов работы программного средства.

Каждый из этих функциональных блоков был реализован отдельными наборами чистых функций. Чистая функция – это функция, которая при одинаковых значениях аргументов всегда возвращает одинаковые значения и не имеет наблюдаемых побочных эффектов.

В целом, функциональная архитектура программного средства для разработки канала связи на основе циклических кодов была спроектирована таким образом, чтобы обеспечить эффективное кодирование и декодирование данных, а также обработку ошибок, связанных с передачей данных. Кроме того, она обеспечивает удобный интерфейс для взаимодействия с пользователем.

## 2.3 Блок-схемы кода и алгоритма циклических кодов

Блок-схема алгоритма кодирования циклическими кодами представлена на рисунке 1.

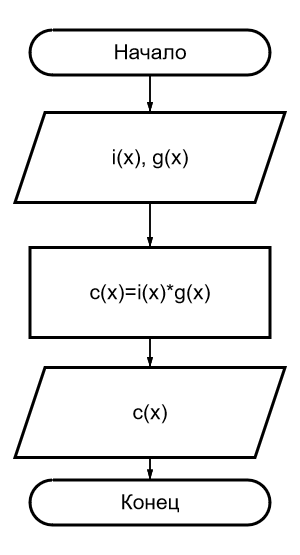


Рисунок 1 – Блок-схема алгоритма кодирования

Блок-схема алгоритма декодирования с вылавливанием ошибок представлена на рисунке 2.

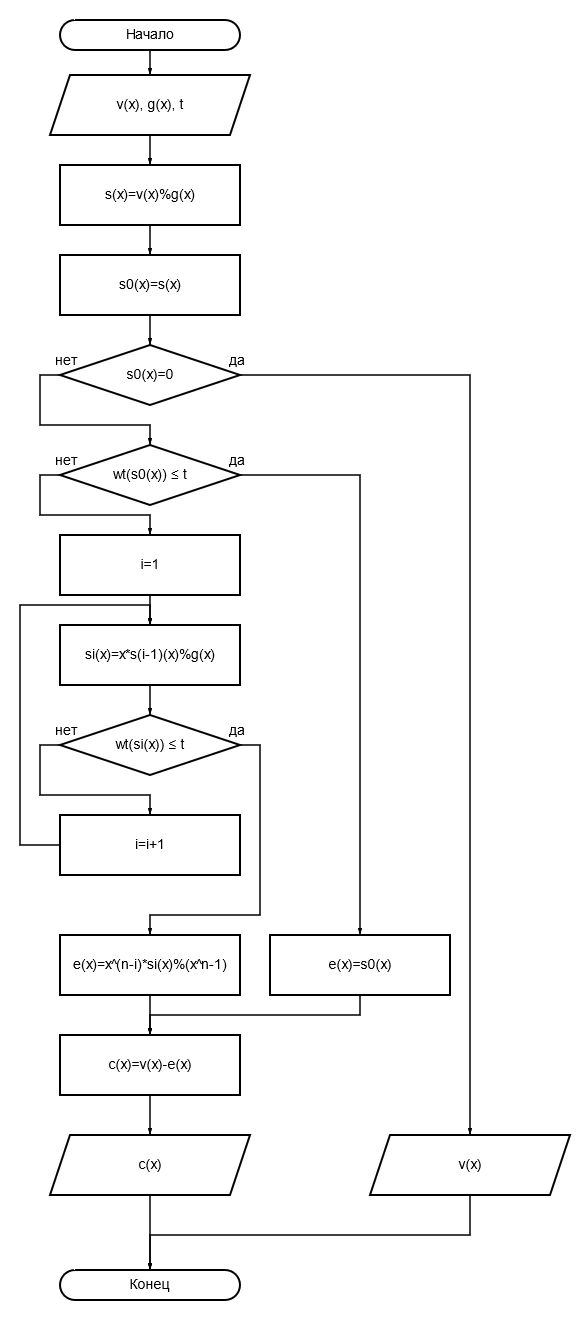


Рисунок 2 – Блок-схема алгоритма декодирования с вылавливанием ошибок

Общий алгоритм работы программы на рисунке 3.

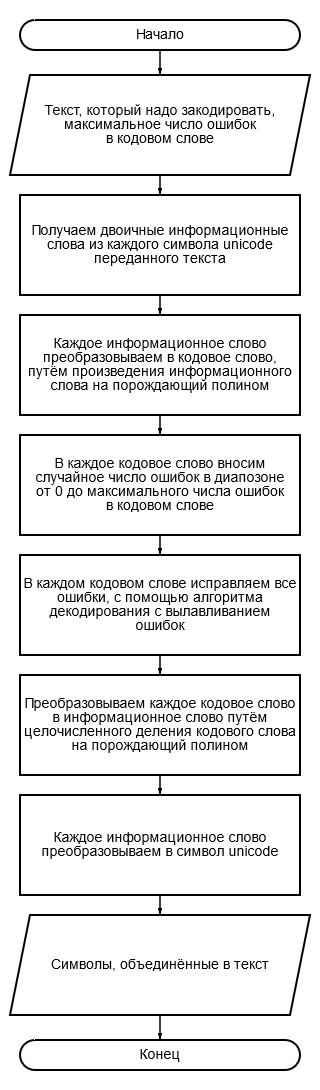


Рисунок 3 – Общий алгоритм работы программы

Данные рисунки отражают алгоритмы и общую работу программы.

## 2.4 Основные методы и классы

Основными функциями программы являются:

1. get\_inf\_words – данная функция преобразует переданный в качестве аргумента текст в массив информационных слов и возвращает этот массив, параметрами данной функции являются: текст, длина информационного слова при исправлении ошибок и длина информационных слов при преобразовании информационного слова в символ.
2. get\_code\_words – данная функция преобразует информационные слова в кодовые путём произведения информационного слова и порождающего полинома и возвращает эти кодовые слова в виде вложенных в массив массивов битов, параметрами данной функции являются: информационные слова, порождающий полином и длина кодовых слов.
3. get\_inf\_word\_from\_code\_word – данная функция преобразует кодовое слово в информационное слово путём целочисленного деления кодового слова на порождающий полином и возвращает это информационное слово в виде массива битов, параметрами данной функции являются: кодовое слово, порождающий полином и длина информационного слова при исправлении ошибок.
4. make\_mistake\_in\_vector – данная функция вносит случайное допустимое число ошибок в векторе и возвращает этот вектор в виде массива битов, параметрами этой функции являются: вектор и максимальное число ошибок, которые можно допустить в этом векторе.
5. correct\_mistake\_in\_code\_word – эта функция при помощи алгоритма с вылавливанием ошибок исправляет ошибки в кодовом слове и возвращает исправленное слово без ошибок, параметрами данной функции являются: кодовое слово с ошибками, порождающий полином, длина кодовых слов и максимальное число исправляемых ошибок этим кодом.

## 2.5 Демонстрация программного средства

Интерфейс окна ввода данных программы представлен на рисунке 4.

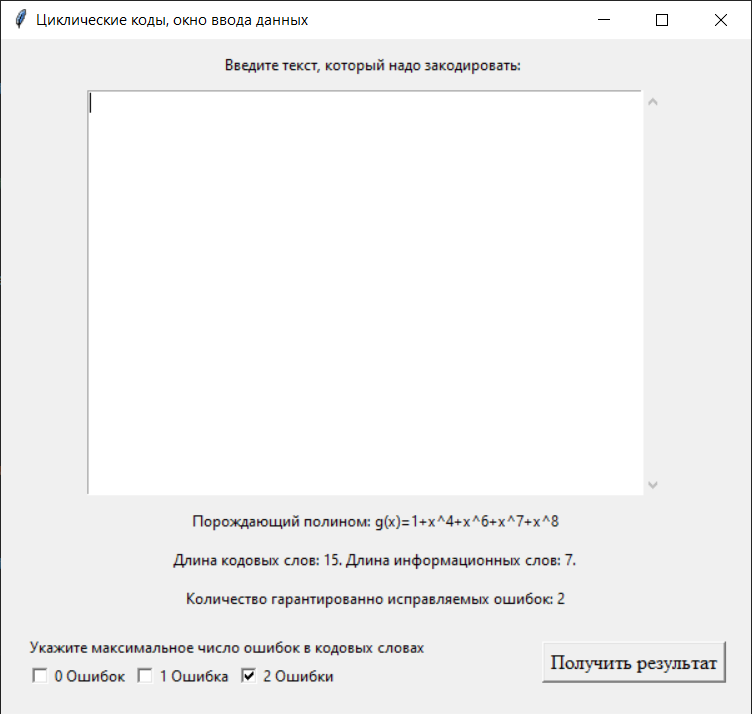


Рисунок 4 – Интерфейс окна ввода данных программы

Интерфейс окна ввода данных программы включает следующие элементы:

* текстовое поле, для ввода текста, который нужно закодировать (Рисунок 5);

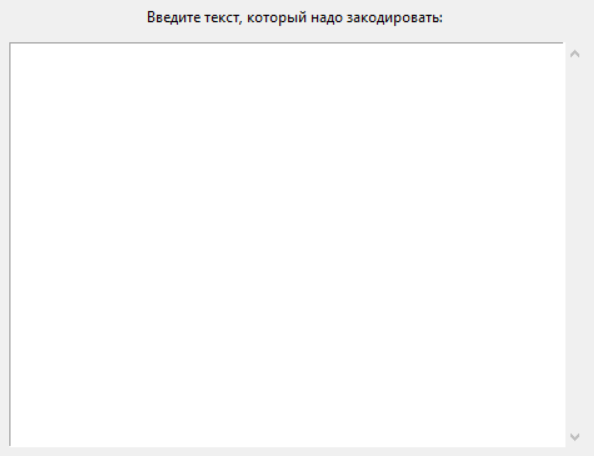


Рисунок 5 – Текстовое поле, для ввода текста, который нужно закодировать

* параметры кода (Рисунок 6);

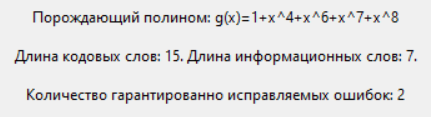


Рисунок 6 – Параметры кода

* checkboxes, которые определяют максимальное допустимое число ошибок в кодовых словах (Рисунок 7);

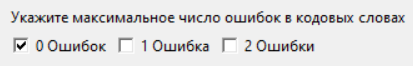


Рисунок 7 – Чекбоксы, которые определяют максимальное допустимое число ошибок в кодовых словах

* кнопка, которая запускает алгоритмы кодирования и декодирования программы (Рисунок 8).



Рисунок 8 – Кнопка, которая запускает алгоритмы

кодирования и декодирования программы

Интерфейс окна вывода данных программы представлен на рисунке 9.

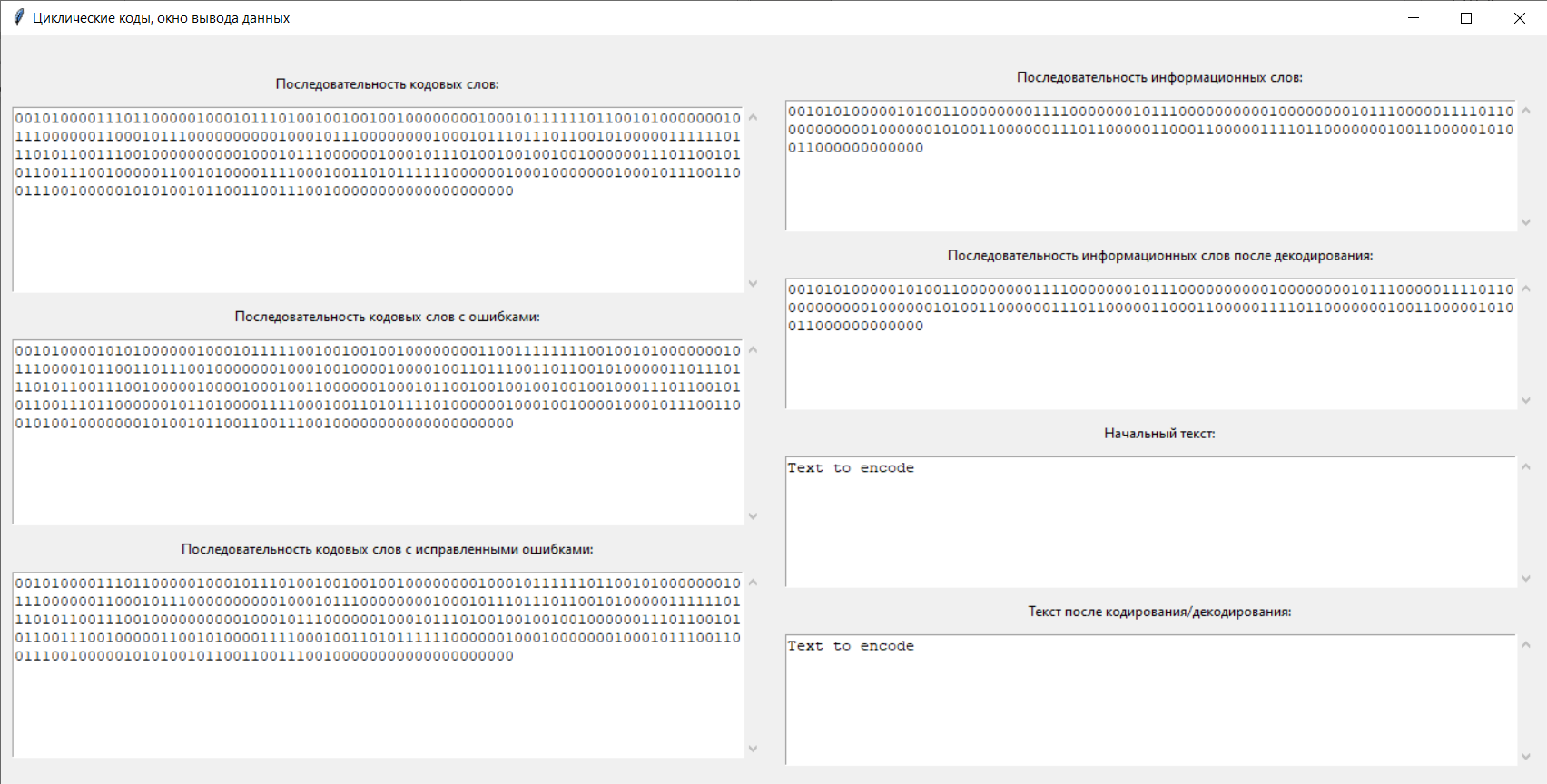


Рисунок 9 – Интерфейс окна вывода данных программы

Интерфейс окна вывода данных программы включает следующие элементы:

* последовательность кодовых слов (Рисунок 10);

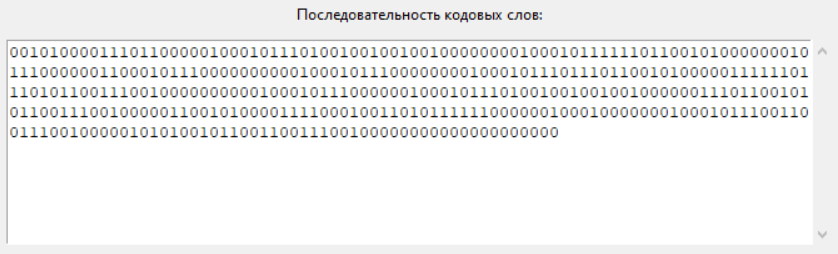


Рисунок 10 – Последовательность кодовых слов

* последовательность кодовых слов с внесёнными случайным образом ошибками (Рисунок 11);

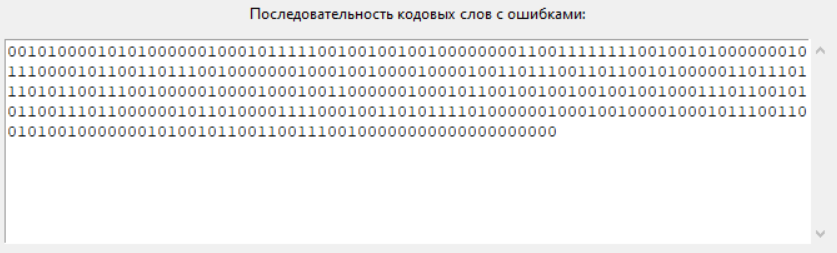


Рисунок 11 – Последовательность кодовых слов с внесёнными случайным образом ошибками

* последовательность кодовых слов с исправленными ошибками (Рисунок 12);

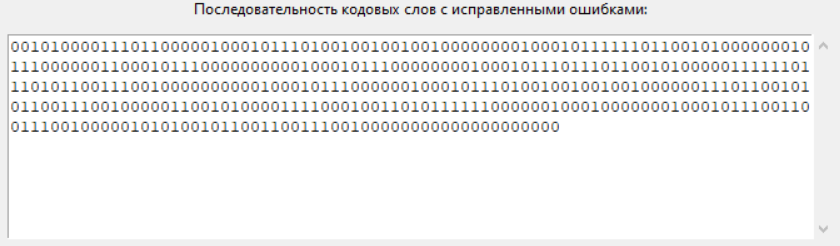


Рисунок 12 – Последовательность кодовых слов с исправленными ошибками

* последовательность информационных слов (Рисунок 13);

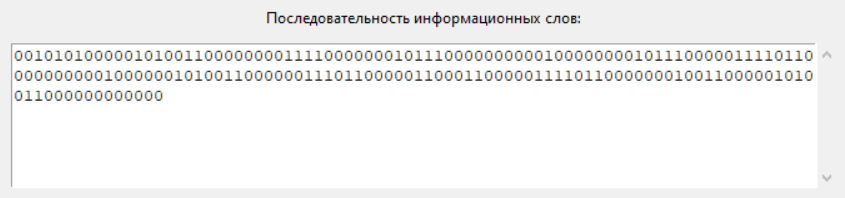


Рисунок 13 – Последовательность информационных слов

* последовательность информационных слов после декодирования (Рисунок 14);

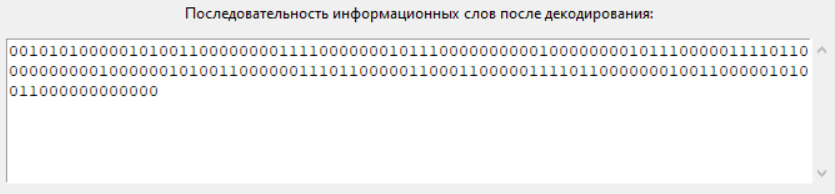


Рисунок 14 – Последовательность информационных слов после декодирования

* начальный текст, введённый пользователем (Рисунок 15);

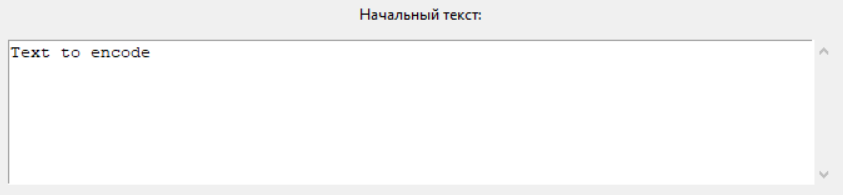


Рисунок 15 – Начальный текст, введённый пользователем

* текст, после кодирования и декодирования (Рисунок 16).

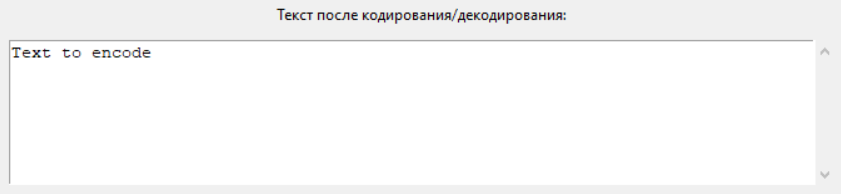


Рисунок 16 – Текст, после кодирования и декодирования

Таким образом был рассмотрен интерфейс всей программы.

# 3 Тестирование программного средства

Для проверки работоспособности программы и правильности выполнения алгоритмов кодирования и декодирования, протестируем программу с исходными данными, представленными на рисунке 17.

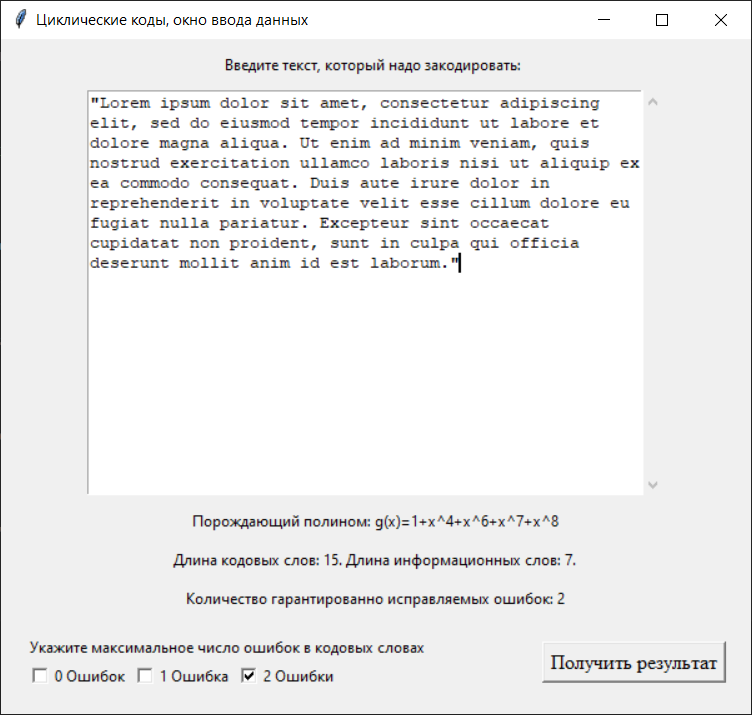


Рисунок 17 – Тестовые исходные данные программы

Результат работы программы, при указанных исходных данных представлен на рисунке 18.



Рисунок 18 – Результат работы программы

Результат кодирования текста представлен на рисунке 19.

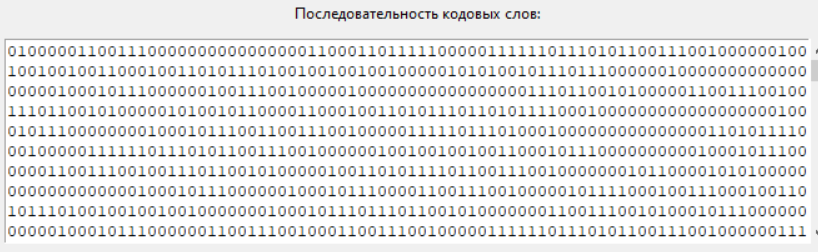


Рисунок 19 – Закодированный текст

На рисунке 20 можно увидеть, что в кодовые слова внесены ошибки.

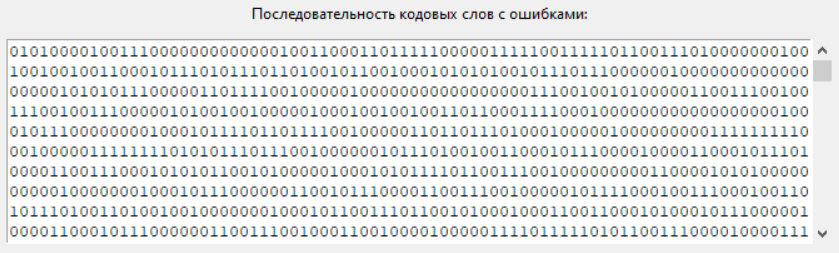


Рисунок 20 – Последовательность кодовых слов, с внесёнными ошибками

На рисунке 21 можно увидеть, что последовательность кодовых слов, с исправленными ошибками, полностью соответствуют начальному закодированному тексту.

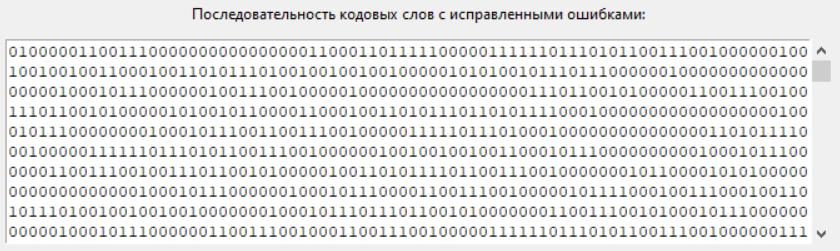


Рисунок 21 – Последовательность кодовых слов, с исправленными ошибками

На рисунке 22 представлен текст после кодирования и декодирования, данный текст полностью совпадает с исходным текстом на рисунке 17.

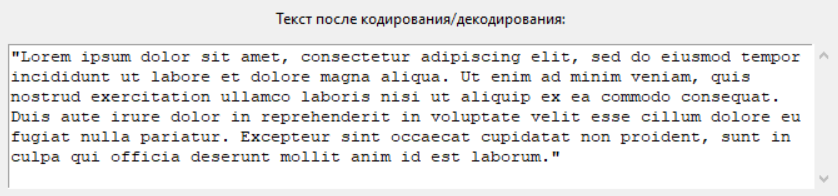


Рисунок 22 – Кодовые слова с внесенными ошибками

Так же проверим работу программы со следующим набором случайных символов:

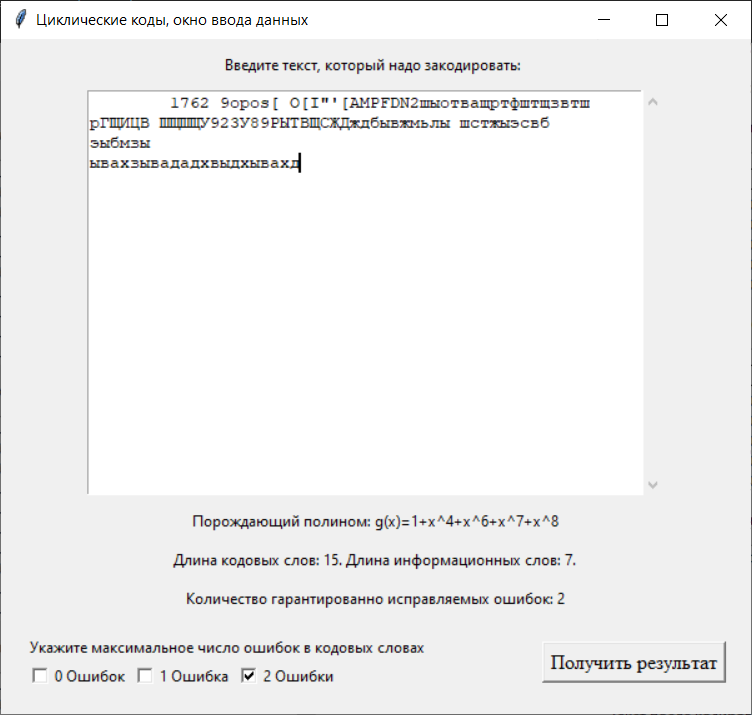


Рисунок 23 – Тестовые исходные данные программы

Результат работы программы, при указанных исходных данных представлен на рисунке 24.



Рисунок 24 – Результат работы программы

Результат кодирования текста представлен на рисунке 25.

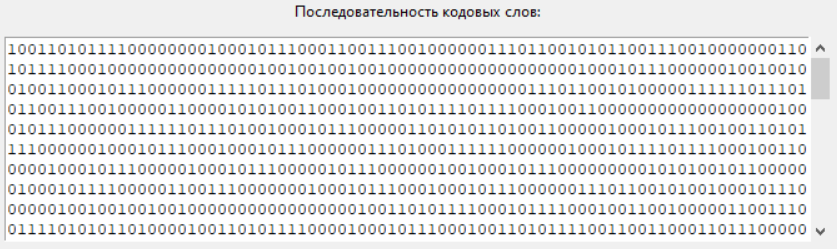


Рисунок 25 – Закодированный текст

На рисунке 26 можно увидеть, что в кодовые слова внесены ошибки.

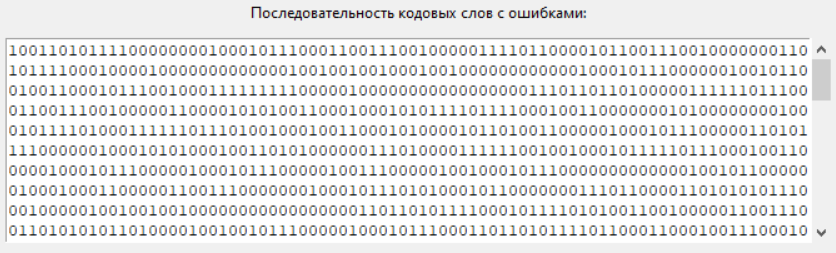


Рисунок 26 – Последовательность кодовых слов, с внесёнными ошибками

На рисунке 27 можно увидеть, что последовательность кодовых слов, с исправленными ошибками, полностью соответствуют начальному закодированному тексту.

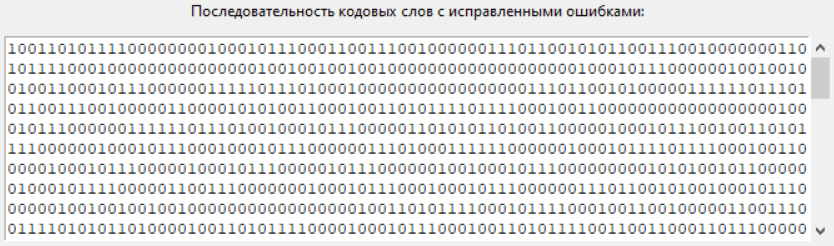


Рисунок 27 – Последовательность кодовых слов, с исправленными ошибками

На рисунке 28 представлен текст после кодирования и декодирования, данный текст полностью совпадает с исходным текстом на рисунке 23.

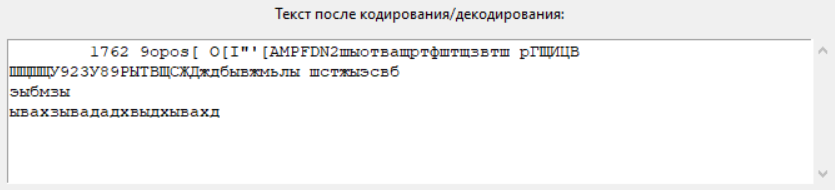


Рисунок 28 – Кодовые слова с внесенными ошибками

Результаты работы программы дают понять, что программа работает исправно.

## Заключение

Помехоустойчивое кодирование является необходимым для защиты информации от помех и ошибок, которые могут возникнуть в процессе передачи и хранения данных. Различные методы и алгоритмы помехоустойчивого кодирования позволяют обеспечить высокую степень защиты от ошибок, а также определить наиболее эффективный метод кодирования для конкретной задачи. В представленной работе был проведен анализ помехоустойчивого кодирования, а также программно реализована модель канала связи на основе циклических коды.

Все поставленные задачи выполнены, а именно:

1. Изучены принципы построения помехоустойчивых кодов с заданными свойствами, методы кодирования и декодирования помехоустойчивых кодов, в частности циклических.
2. Изучены принципы построения циклических кодов и их свойств.
3. Программно реализована модель канала связи на основе циклических кодов.

Помимо задач, по итогу можно сказать, что обозначенная цель работы, которая заключалась в исследовании возможностей циклических кодов и реализации их на практике, была достигнута.

## Перечень использованных информационных ресурсов

1. Золотарев В.В., Овечкин Г.В. Помехоустойчивое кодирование. Методы и алгоритмы. – М.: Горячая линия-Телеком, 2004 – 16-26с.
2. Морелос-Сарагоса Р. Искусство помехоустойчивого кодирования. Методы, алгоритмы, применение. – М.:Техносфера, 2005 – 70с.
3. Базис циклического кода, формирование кодовых комбинаций. [Электронный ресурс] URL: https://scask.ru/q\_book\_pds.php?id=95 (Дата обращения: 02.05.2023).
4. Корректирующие коды [Электронный ресурс] URL: https://www.booksite.ru/fulltext/1/001/008/064/934.htm (Дата обращения: 01.05.2023).
5. Корректирующие коды [Электронный ресурс] URL: http://jstonline.narod.ru/rsw/rsw\_g0/rsw\_g0b0/rsw\_g0b0d.htm (Дата обращения: 01.05.2023).
6. Принципы помехоустойчивого кодирования [Электронный ресурс] URL: https://studfile.net/preview/2688267/page:39/ (Дата обращения: 01.05.2023).
7. Свойства циклических кодов [Электронный ресурс] URL: https://studfile.net/preview/5240128/page:10/ (Дата обращения: 02.05.2023).
8. Циклические коды [Электронный ресурс] URL: http://yourtutor.narod.ru/cyclic/CyclicCodes.htm (Дата обращения: 02.05.2023).
9. Циклический код [Электронный ресурс] URL: <http://peredacha-informacii.ru/tsiklicheskij-kod.html> (Дата обращения: 01.05.2023).
10. Циклическое кодирование [Электронный ресурс] URL: https://studref.com/695480/tehnika/tsiklicheskoe\_kodirovanie (Дата обращения: 02.05.2023).

## Приложение А

Листинг А.1 – файл main.py

def get\_inf\_words (text, len\_of\_inf\_word,

need\_len\_to\_make\_char\_from\_inf\_word):

inf\_words = [bin(ord(char))[2:] for char in

list(text)]

for i in range(len(inf\_words)):

inf\_words[i] = list(inf\_words[i])

inf\_words[i] = [int(num) for num in

inf\_words[i]]

inf\_words[i].reverse()

inf\_words[i]= make\_vector\_need\_len(inf\_words

[i],need\_len\_to\_make\_char\_from\_inf\_word)

inf\_words[i] = ''.join([str(num) for num in

inf\_words[i]])

inf\_words = [int(char) for char in list(''.join(

inf\_words))]

count\_of\_inf\_word\_need\_len = ceil(len(inf\_words) //

len\_of\_inf\_word) + 1

inf\_words = make\_vector\_need\_len(inf\_words,

len\_of\_inf\_word \* count\_of\_inf\_word\_need\_len)

inf\_words\_need\_len = []

for j in range (count\_of\_inf\_word\_need\_len):

inf\_words\_need\_len.append(inf\_words[j

\*len\_of\_inf\_word:(j+1)\*len\_of\_inf\_word])

return inf\_words\_need\_len

def get\_code\_words(inf\_words, gen\_polynom, n):

code\_words = []

for inf\_word in inf\_words:

code\_word = list(Polynomial(inf\_word) \*

Polynomial(gen\_polynom))

for i in range(len(code\_word)):

if code\_word[i] % 2 == 0:

code\_word[i] = 0

else:

code\_word[i] = 1

while len(code\_word) < n:

code\_word.append(0)

code\_words.append(code\_word)

return code\_words

def get\_inf\_word\_from\_code\_word(code\_word, gen\_polynom,

len\_of\_inf\_word):

inf\_word = list(Polynomial(code\_word) //

Polynomial(gen\_polynom))

for i in range(len(inf\_word)):

if inf\_word[i] % 2 == 0:

inf\_word[i] = 0

else:

inf\_word[i] = 1

inf\_word = make\_vector\_need\_len(inf\_word,

len\_of\_inf\_word)

return inf\_word

def make\_mistake\_in\_vector(vector, num\_of\_errors):

vector\_copy = deepcopy(vector)

num\_memory = []

for i in range(randint(0,num\_of\_errors)):

num = randint(0, len(vector\_copy) - 1)

while num in num\_memory:

num = randint(0, len(vector\_copy) - 1)

num\_memory.append(num)

if (vector\_copy[num] == 0):

vector\_copy[num] = 1

else:

vector\_copy[num] = 0

return vector\_copy

def correct\_mistake\_in\_code\_word(code\_word\_with\_mistakes

gen\_polynom, n, t):

initial\_syndrome = get\_syndrome\_from\_vector

(code\_word\_with\_mistakes, gen\_polynom, n)

initial\_code\_word = []

error\_vector = []

if (sum(initial\_syndrome) == 0):

initial\_code\_word = code\_word\_with\_mistakes

elif (get\_wt(initial\_syndrome) <= t):

error\_vector = initial\_syndrome

initial\_code\_word = get\_binom\_vector([int(num) for

num in list(Polynomial(code\_word\_with\_mistakes) –

Polynomial(error\_vector))])

initial\_code\_word = make\_vector\_need\_len

(initial\_code\_word, n)

else:

for i in range(1,n):

current\_syndrome = get\_binom\_vector(list(

(Polynomial([0,1])\*Polynomial(

initial\_syndrome))%Polynomial(gen\_polynom)))

initial\_syndrome = current\_syndrome

if (get\_wt(current\_syndrome) <= t):

error\_vector1 = list((Polynomial

(get\_vector\_from\_power(n-i))\*Polynomial(

current\_syndrome))%(Polynomial(

get\_vector\_from\_power(n)) + 1))

error\_vector1 = get\_binom\_vector(

error\_vector1)

initial\_code\_word = get\_binom\_vector(

list(Polynomial(code\_word\_with\_mistakes

- Polynomial(error\_vector1)))

initial\_code\_word = make\_vector\_need\_len

(initial\_code\_word, n)

initial\_syndrome = current\_syndrome

break

else:

initial\_syndrome = current\_syndrome

return initial\_code\_word