Учреждение образования

«Белорусский государственный технологический университет»

**Отчеты по лабораторным работам**

по дисциплине: «Защита информации и надежность информационных систем»

Студента 3 курса 7 группы ФИТ

Воликова Дмитрия Анатольевича

Содержание

[Лабораторная работа №1: Элементы теории информации. Параметры и характеристики дискретных информационных систем 4](#_Toc166758514)

[Лабораторная работа №2: Элементы теории информации. Информативность данных в различных кодировках 8](#_Toc166758515)

[Лабораторная работа №3: Избыточное кодирование данных в информационных системах. Код хемминга 11](#_Toc166758516)

[Лабораторная работа №4. Избыточное кодирование данных в информационных системах. Итеративные коды 16](#_Toc166758517)

[Лабораторная работа №5. Избыточное кодирование данных в ИС 22](#_Toc166758518)

[Лабораторная работа №6. Перемежение/Деперемежение Данных В Информационно-Вычислительных Системах. 29](#_Toc166758519)

[Лабораторная работа №7. Сжатие/Распаковка Данных Методом Барроуза – Уилера 33](#_Toc166758520)

[Лабораторная работа №8. Сжатие/Распаковка данных на основе статистических методов 38](#_Toc166758521)

[Лабораторная работа №9. Сжатие/распаковка данных методом Лемпеля – Зива 43](#_Toc166758522)

[Лабораторная работа №10. Сжатие/распаковка данных арифметическим методом 51](#_Toc166758523)

[Листинг 56](#_Toc166758524)

[Лабораторная работа №1 56](#_Toc166758525)

[Лабораторная работа №2 59](#_Toc166758526)

[Лабораторная работа №3 65](#_Toc166758527)

[Лабораторная работа №4 72](#_Toc166758528)

[Лабораторная работа №5 79](#_Toc166758529)

[Лабораторная работа №6 86](#_Toc166758530)

[Лабораторная работа №7 92](#_Toc166758531)

[Лабораторная работа № 8 95](#_Toc166758532)

[Лабораторная работа №9 103](#_Toc166758533)

[Лабораторная работа №10 107](#_Toc166758534)

# Лабораторная работа №1: Элементы теории информации. Параметры и характеристики дискретных информационных систем

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

**Задачи:**

1. Закрепить теоретические знания по основам теории информации
2. Разработать приложение для расчета и анализа параметров и информативных характеристик дискретных ИС
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**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.

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.

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.

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).

Энтропию алфавита А = {a i} по К. Шеннону рассчитывают по следующей формуле:

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

Сообщение Хk, которое состоит из k символов, должно характеризоваться определенным количеством информации I(Хk):

Если вероятность ошибки в ДСК отлична от 0 (р > 0), переданное сообщение может содержать ошибки: Хk ≠ Y k . Количество информации в таком сообщении при его передаче по ДСК будет определяться не энтропией двоичного алфавита (в соответствии с выражением (2.3)), а эффективной энтропией Hе(A) алфавита или пропускной способностью канала:

где H(Y | X) – условная энтропия:

**Практическое задание**

Создать приложение для расчета и анализа параметров и информативных характеристик дискретных ИС, с помощью которого:

а) рассчитать энтропию указанных преподавателем алфавитов: один – на латинице, другой – на кириллице (по формуле (2.1) перейти от частоты появления каждого символа алфавита к соответствующей вероятности); в качестве входного может быть принят произвольный электронный текстовый документ на основе соответствующего алфавита; частоты появления символов алфавитов оформить в виде гистограмм (можно воспользоваться приложением MS Excel);

Код программы можно увидеть в листинге А.

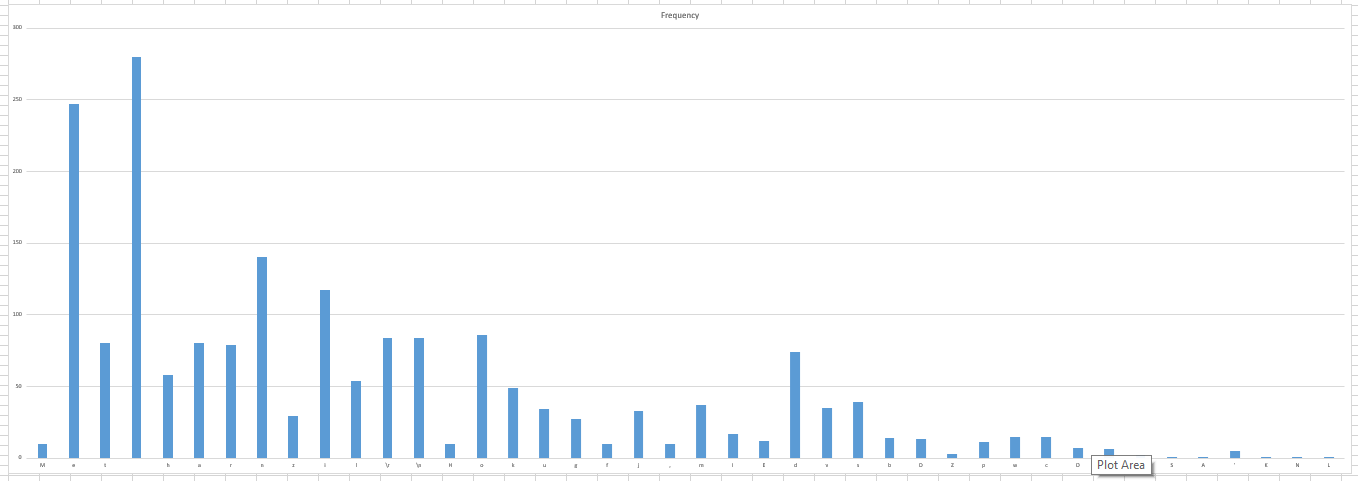


Рисунок 1.1 – Гистограмма латинского языка

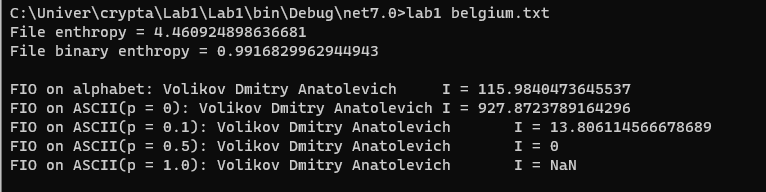


Рисунок 1.2 – Вывод программы латинского языка

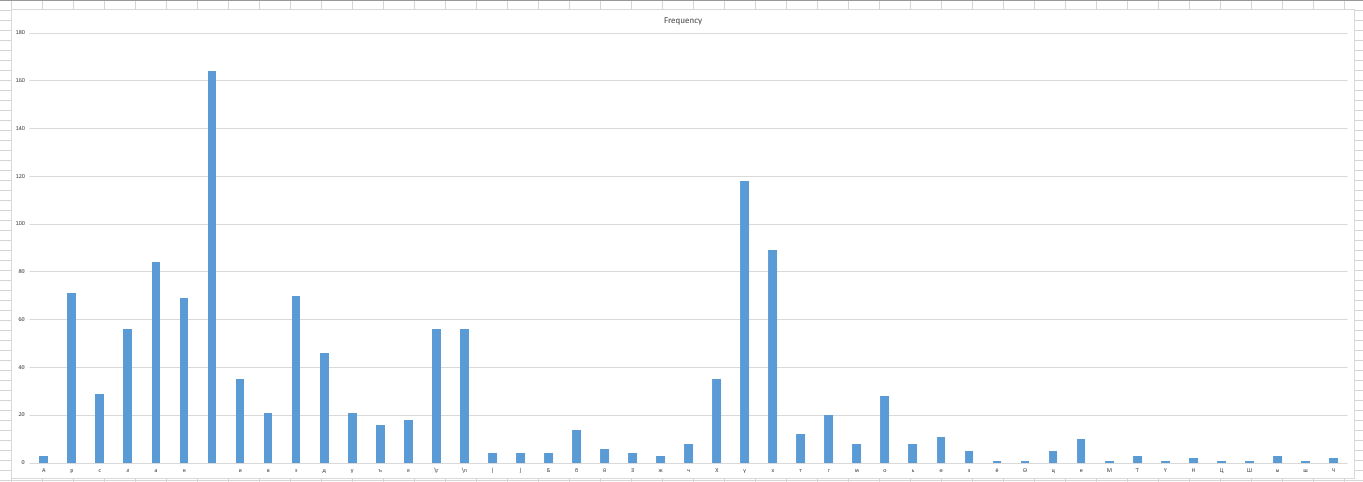


Рисунок 1.3 – Гистограмма кириллицы

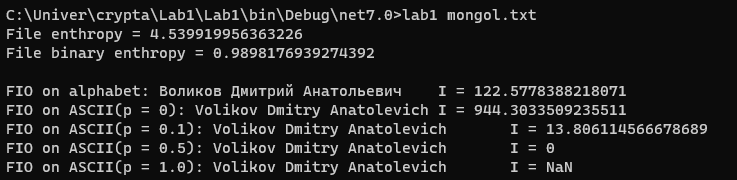


Рисунок 1.4 – Вывод программы на кириллице

б) для входных документов, представленных в бинарных кодах, определить энтропию бинарного алфавита;

в) используя значения энтропии алфавитов, полученных в пунктах (а) и (б), подсчитать количество информации в сообщении, состоящем из собственных фамилии, имени и отчества (на основе исходного алфавита – (а) и в кодах ASCII – (б)); объяснить полученный результат;

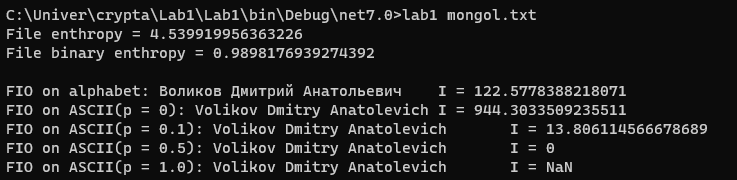
г) выполнить задание пункта (в) при условии, что вероятность ошибочной передачи единичного бита сообщения составляет: 0,1; 0,5; 1,0.

Рисунок 1.5 – задание Г

**Вывод:** в ходе этой лабораторной работы я изучил такие понятия как алфавит, энтропия алфавита, полезная энтропия, условная энтропия, а также количество информации. В ходе работы была написана программа, которая составляет алфавит на основе файла. На основе этого алфавита и были проведены все расчёты.

# Лабораторная работа №2: Элементы теории информации. Информативность данных в различных кодировках

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

**Задачи:**

1. Закрепить теоретические знания по взаимной конвертации данных, представленных в кодах ASCII и base64.
2. Разработать приложение для конвертации произвольного документа в формат base64 и обратно.
3. Исследовать энтропийные характеристики используемых в конвертерах алфавитов.
4. Изучить особенности практической реализации операции XOR над данными, представленными в разных форматах.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретические сведения**

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):

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).

Процесс кодирования представляет группу из 24 последовательных битов в форме строки из 4 символов. Обработка выполняется слева направо, а 24-битная исходная группа образуется конкатенацией трех 8-битных групп (байтов). Данные 24 бита после этого трактуются как 4 сцепленных группы по 6 битов, каждая из которых транслируется в один символ алфавита base64.

Каждая 6-битная группа используется в качестве индекса массива из 64 печатных символов. Символы алфавита, соответствующие индексу, помещаются в выходную строку [6]. Кодирование base64 с безопасным алфавитом используется для представления URL и имен файлов.

**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 «=».

**Практическое задание**

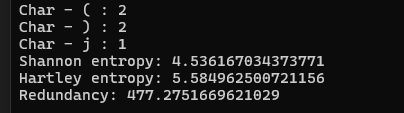
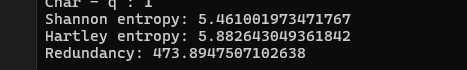
1. Создать собственное приложение (приветствуется!) или воспользоваться Base64-онлайн-кодировщиком, с помощью которого конвертировать произвольный документ (а) на латинице (можно использовать документ из лабораторной работы № 1) в документ (б) формата base64.
2. С помощью приложения, созданного в лабораторной работе № 1, получить распределение частотных свойств алфавитов по документам (а) и (б). Вычислить энтропию Хартли и Шеннона, а также избыточность алфавитов. Объяснить полученный результат.

Рисунок 2.1 – Избыточность исходного алфавита

Рисунок 2.2 – Избыточнось алфавита Base64

1. 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?

Рисунок 2.3 – Результат операции Dmitry^Volikov^Volikov

**Вывод:** в ходе этой лабораторной работы я изучил понятие избыточности, написал программу для подсчёта избыточности алфавита. Также изучил Base64 кодирование, написал программу, которая высчитывает base64 формат входного алфавита и сохраняет его в новый файл.

# Лабораторная работа №3: Избыточное кодирование данных в информационных системах. Код хемминга

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

**Задачи:**

1. Закрепить теоретические знания по использованию методов помехоустойчивого кодирования для повышения надежности передачи и хранения в памяти компьютера двоичных данных.
2. Разработать приложение для кодирования/декодирования двоичной информации кодом Хемминга с минимальным кодовым расстоянием 3 или 4.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**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, хактеризующее 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 значении (результате) вследствие неправильно запрограммированного алгоритма или неправильно составленной программы.

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

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

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

К. Шеннон сформулировал теорему для случая передачи дискретной информации по каналу связи с помехами, утверждающую, что вероятность ошибочного декодирования принимаемых сигналов может быть обеспечена сколь угодно малой путем выбора соответствующего способа кодирования сигналов. В теореме Шеннона не говорится о том, как нужно строить необходимые помехоустойчивые коды. Однако в ней указывается на принципиальную возможность кодирования, при котором может быть обеспечена сколь угодно высокая надежность передачи.

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

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

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

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

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

Матрица Н, называемая проверочной, равна

В коде Хемминга с минимальным кодовым расстоянием dmin = 3 проверочная матрица Н имеет классический вид и состоит из двух подматриц: P’ размером k×r и I размером r×r соответственно.

В последнем выражении I – единичная матрица порядка r (r×r).

Количество r избыточных (проверочных) символов кодового слова определяется из следующей простой логической цепи рассуждений.

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

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

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

**Важно запомнить, что в силу выражений ненулевой синдром всегда равен сумме по модулю 2 тех вектор столбцов матрицы Н, номера которых соответствуют номерам ошибочных битов в слове Yn.**

**Практическое задание**

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

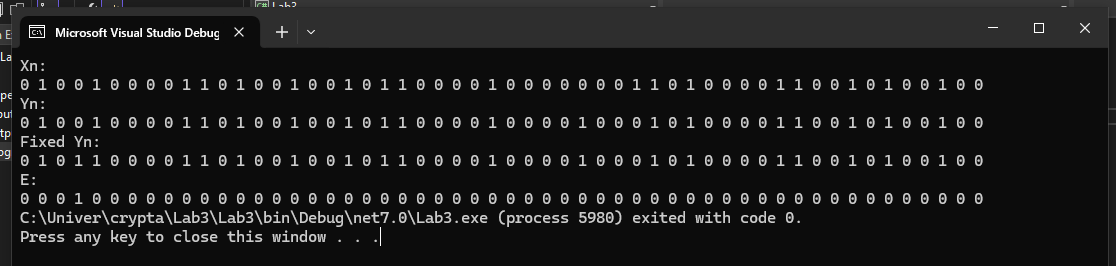


Рисунок 3.1 – Результат программы с количеством ошибок 2

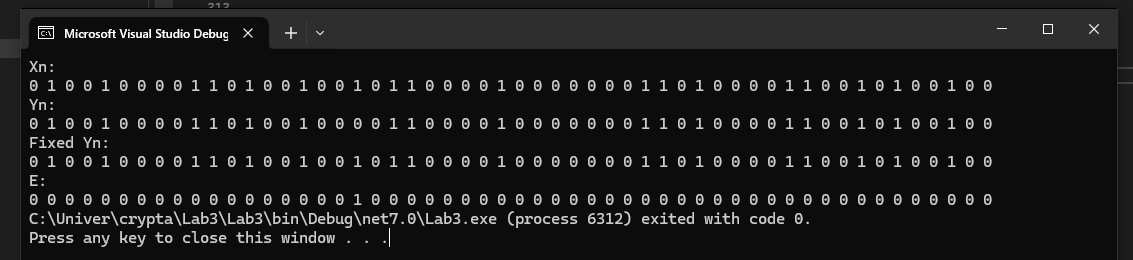


Рисунок 3.2 – Результат программы с количеством ошибок 1

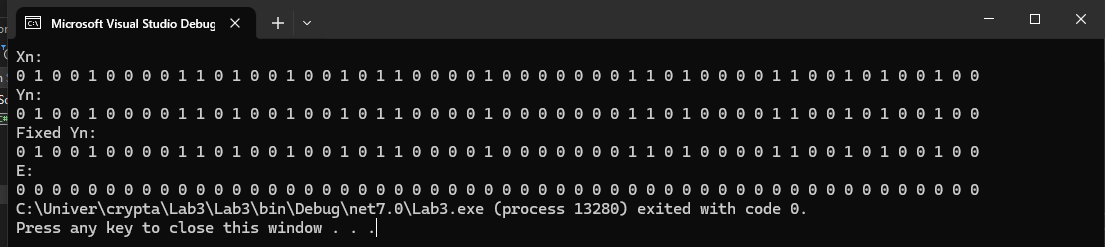


Рисунок 3.3 – Результат программы с количеством ошибок 0

**Вывод**: в ходе этой лабораторной я изучил понятие помехоустойчивого кодирования. Принцип передачи информации с помощью кодеров и декодеров на основе кода Хэмминга.

# Лабораторная работа №4. Избыточное кодирование данных в информационных системах. Итеративные коды

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

**Задачи:**

1. Закрепить теоретические знания по использованию итеративных кодов для повышения надежности передачи и хранения в памяти компьютера двоичных данных.
2. Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**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 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 информа-ционных 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).

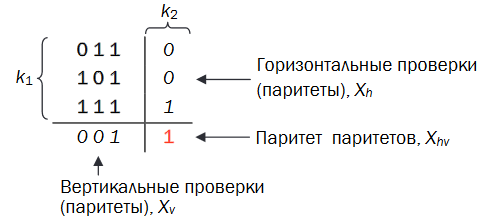


Рисунок 4.1 – Пояснение к принципу формирования избыточных символов итеративного кода

В соответствии с рис. 4.1 кодовое слово будет иметь следующий вид: Xn = 0111011110010011. Как видно, избыточные символы (называемые также паритетами) в приведенном кодовом слове в принятом порядке (Xh, Xv, X hv) записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ Xhv (паритет паритетов) равен сумме по модулю 2 символов информационного слова Xk, а также проверочных символов Xv и Xh.

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

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

Декодирование начинают сразу, не ожидая поступления всего блока информации. Проверка соответствия избыточных символов полученного слова (Y r = Y h, Y v , Y hv либо Y r = Y h, Y v) при декодировании позволяет обнаружить любое нечетное число искаженных символов, расположенных в одной строке или в одном столбце. Формально такое декодирование осуществляется сравнением принятых

(Yh, Yv, Yhv) и вновь вычисленных (Y’h, Y’v, Y’hv) для полученного слова паритетов. В упрощенной форме это показано на рис. 5.2. Определение местоположения одиночной ошибки по строке указывает на наличие ошибки в этой строке матрицы, а проверка по столбцу – конкретный символ (рис. 5.2, а).

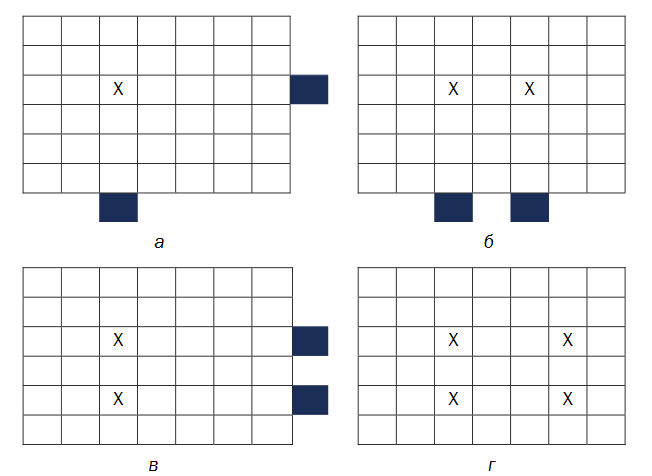


Рисунок 4.2 – Пояснение к принципу обнаружения местоположения ошибочных битов в принятом сообщении: а–г – варианты местоположения ошибок

Однако этим кодом не могут быть установлены местоположения многократных ошибок, имеющих четное число искаженных символов как по строкам, так и по столбцам (рис. 4.2, б, в). Простейшая необнаруживаемая ошибка содержит четыре искаженных символа, расположенных в вершинах прямоугольника или квадрата (рис. 4.2, г). Это происходит из-за того, что четность (паритет) по строкам и по столбцам матрицы не нарушается. Полезную информацию о кодировании и декодировании информации итеративным кодом можно найти в источниках [10, 11].

**Практическое задание**

Вариант 2.

k = 20, k1 = { 4, 5 }, k2 = { 2, 10 }.

Количество групп паритетов: 2, 3.

0 ошибок:



Рисунок 4.3 – Результат работы при 0 ошибок

1 ошибка:



Рисунок 4.4 – Результат работы при 1 ошибке

2 ошибки:



Рисунок 4.5 – Результат работы при 2 ошибках

Вывод: в ходе этой лабораторной работы я изучил принципы итеративного кодирования и построил алгоритм, который вычисляет поизицию ошибки и исправляет её.

# Лабораторная работа №5. Избыточное кодирование данных в ИС

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

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и использованию ЦК для повышения надежности передачи и хранения в памяти компьютера двоичных данных, для контроля интегральности файлов информации.
2. Разработать приложение для кодирования/декодирования двоичной информации циклическим кодом.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**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).

**Операции кодирования и декодирования**

Рассматриваемые операции сводятся к известным процедурам умножения и деления двоичных чисел либо соответствующих этим числам полиномов. Действия с кодовыми словами производятся по правилам арифметики по модулю 2. Следует помнить, что **вычитание равносильно сложению.** Это следует из простых рассуждений: из равенства х^z − 1 = 0 получаем хz = 1. Прибавив к левой и правой частям по единице, имеем х^z + 1 = 1 + 1 = 0. Таким образом, вместо двучлена х^z − 1 можно ввести х^z + 1 или 1 + х^z, из чего следует также, что х^z + х^z = х^z(1 + 1) = 0.

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

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

здесь Bj(X) = Xn – кодовое слово; Aj(X) = Xk – информационное слово.

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

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

при нулевом остатке: R(X) = 0.

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

При этом следует обратить внимание на важную деталь: умножение полинома на х приводит к сдвигу членов полинома на один разряд влево, а при умножении на хr – на r разрядов влево с заменой r младших разрядов полинома на нули. Деление же полинома на х приводит к соответствующему сдвигу членов полинома вправо с уменьшением показателей членов на 1. Такой сдвиг требует дописать справа r проверочных символов к исходной кодовой комбинации Аi(Х) после умножения ее на хr.

**Кодирование информационного слова**

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

**Декодирование принятого сообщения по синдрому**

Основная операция: принятое кодовое слово (Y n) нужно поделить на порождающий полином, который использовался при кодировании. Если Y n принадлежит коду, т. е. слово не искажено помехами, то остаток от деления (синдром) будет нулевым. Ненулевой остаток свидетельствует о наличии ошибок в принятой кодовой комбинации: Y n ≠ Хn.

Деление принятого кодового слова на G(X) формально запишем в следующем виде:

где S r − остаток от деления (Y n) / (G(X)), или синдром.

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

Наряду с полиномиальным способом задания кода, структуру построения кода можно определить с помощью матричного представления. При этом в ряде случаев проще реализуется построение кодирующих и декодирующих устройств ЦК. Здесь нам следует вернуться к упомянутым матрицам и представлению ЦК с их помощью.

Порождающая матрица G циклического кода имеет в качестве строк векторы:

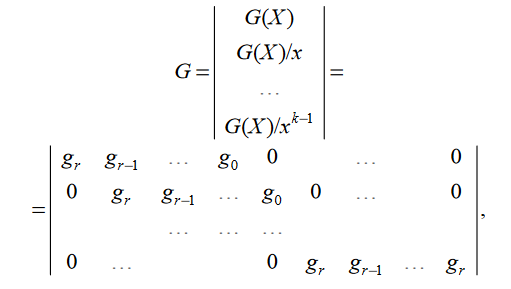


Рисунок 5.1 – Порождающая матрица

где G(X) – порождающий полином ЦК; g0, ..., g r – коэффициенты полинома.

Справедливо:

С учетом этого проверочная матрица H кода строится на основе полинома, полученного в результате следующей операции:

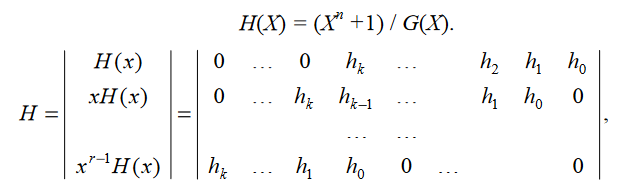


Рисунок 5.2 – Проверочная матрица

**Практическое задание**

Вариант 2.

r = 4

G(X) = x4 + x + 1

0 ошибок:

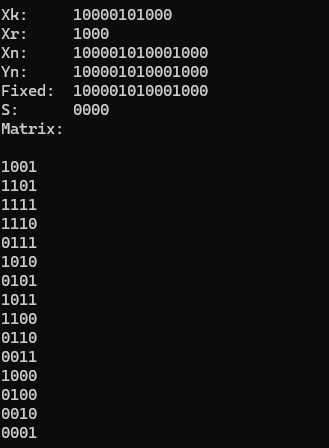


Рисунок 5.3 – Результат программы при кол-ве ошибок 0

1 ошибка:

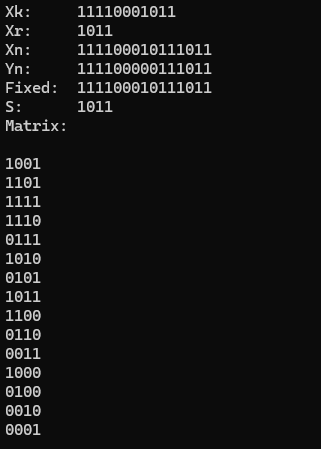


Рисунок 5.4 – Результат программы при кол-ве ошибок 1

2 ошибки:

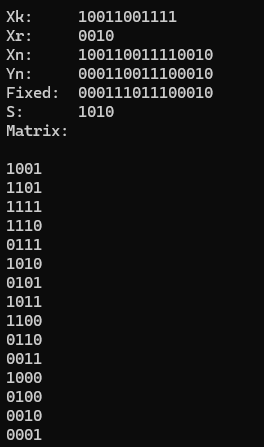


Рисунок 5.5 – Результат программы при кол-ве ошибок 2

**Вывод:** в ходе этой лабораторной работы я изучил основы кодирования с использованием циклических кодов. Мною была реализована программа, которая генерирует случайное информационное слово длиною k и при помощи операций над полиномами рассчитывает оишбочную позицию и исправляет одиночную ошибку.

# Лабораторная работа №6. Перемежение/Деперемежение Данных В Информационно-Вычислительных Системах.

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

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов перемежения/деперемежения двоичных данных в информационных системах.
2. Разработать приложение для реализации метода перемежения/деперемежения символов в сообщениях на основе двоичного алфавита.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретические сведения**

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).

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.

Например, если Хn = 1011111, а Y n = 1000011, то длина пакета ошибок составляет 3 бита (ошибочные биты подчеркнуты).

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

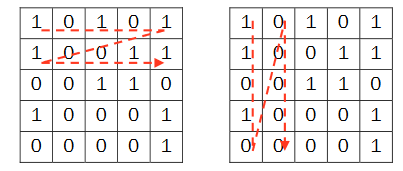


Рисунок 6.1 - Пояснение принципа блочного перемежения

Передаваемая последовательность (1010110011…) делится на блоки по 5 битов. Каждый блок записывается в отдельную строку таблицы по порядку. Сообщение для передачи или хранения формируется при считывании символов из таблицы по столбцам: 11010000001… . Деперемежение производится в обратной последовательности. Для данного примера глубина перемежения (разница между позициями одного и того же символа до и после перемежения) равна 4: например, второй символ после перемежения станет шестым. Особенностью является неизменная позиция первого символа.

В общем случае выбор глубины перемежения зависит от двух факторов. С одной стороны, чем больше расстояние между соседними символами, тем большей длины пакет ошибок может быть исправлен. С другой стороны, чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше задержка сигнала.

Для борьбы с длинными пакетами ошибок желательно увеличивать размеры таблицы. Однако это приводит к увеличению задержки в отправке и декодировании сообщения.

**Практическое задание**

1. Необходимо разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель. По желанию студент может использовать иной. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы.



Рисунок 6.2 – Вариант 2

Исходное сообщение:



Рисунок 6.3 – Исходное сообщение (случайная генерация)

Избыточное сообщение:



Рисунок 6.4 – Избыточное сообщение (цкилический код)

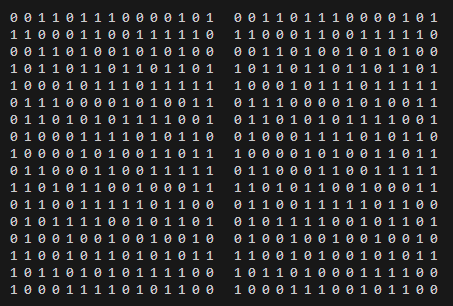


Рисунок 6.5 – Матрицы перемежения до и после появления блочных ошибок

Декодирование и сравенение исходного и выходного сообщений:

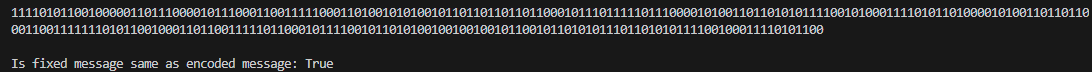


Рисунок 6.6 – Выходное сообщение

Прогон алгоритма по различносу кол-ву ошибок даёт следующий результат:

1. 3 ошибки – 56% успеха
2. 4 ошибки – 52% успеха
3. 5 ошибок – 49% успеха

**Вывод**: в ходе этой лабораторной работы я изучил принцип работы блочного перемежения. На основе циклических кодов и алгоритмов, которые я описал в прошлой лабораторной работе, была построена программа перемежения/деперемежения и протестирована на различном размере групповых ошибок.

# Лабораторная работа №7. Сжатие/Распаковка Данных Методом Барроуза – Уилера

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

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/ разархивации) данных на основе метода Барроуза − Уилера (Burrows-Wheeler transform, BWT).
2. Разработать приложение для реализации метода Барроуза − Уилера.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретическая часть**

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. Исходя из этого, далее дадим краткую характеристику данному классу методов преобразования информации.

Сжатие информации является одним из способов ее кодирования.

Развитие методов сжатия данных имеют длинную историю, которая началась задолго до появления компьютеров и компьютерных сетей.

В основе сжатия данных, как одна из первопричин, лежит избыточность, что являлось предметом исследования в лабораторной работе № 3.

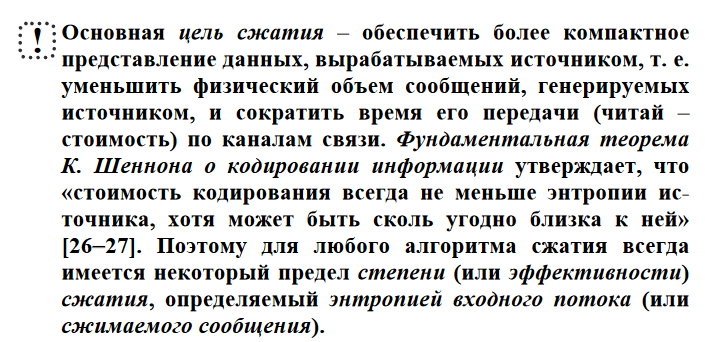


Рисунок 7.1. – Основная цель сжатия

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:

1. Степень сжатия (R)
2. Скорость сжатия
3. Качество сжатия

Степень сжатия R обычно оценивается следующим образом:

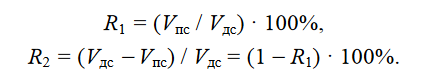


Рисунок 7.2 – Формулы степени сжатия

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

Что касается третьей из приведенных технических характеристик (качества сжатия), то она показывает, по существу, совместимость данного метода с другими. Это важно, если принять во внимание то обстоятельство, что практически все современные архиваторы основаны на использовании нескольких разных методов сжатия (кодирования).

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

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

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

В настоящее время можно встретить достаточно большое число архивирующих и сжимающих утилит, большинство из которых доступны для некоммерческого использования. Поддержка популярных форматов файловых архивов начинает включаться в разные утилиты и программы. Часто используемые форматы становятся стандартными форматами архивов (zip, arj, rar, ha, pak, cab и др.).

**Метод Барроуза – Уилера**

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

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

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

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

Обратимость преобразования, т. е. возможность извлечения М из BWT(М), i основана на рекурсивности преобразования. Вспомним, что общим для рекурсивных функций, рекурсивных алгоритмов и рекуррентных последовательностей является то, что для вычисления очередного значения функции, получения очередной реализации алгоритма, определения очередного члена последовательности необходимо обращаться к предшествующим их значениям, вычисленным раньше.

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.

После k шагов матрица W2 будет получена. Зная значение числа z, получаем входной блок М сообщения. Как видим, последующий вид матрицы определяется предыдущим ее состоянием.

Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы. Входной блок данных может иметь произ-вольную длину.
2. С помощью приложения выполнить прямое и обратное преобразование своей фамилии

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

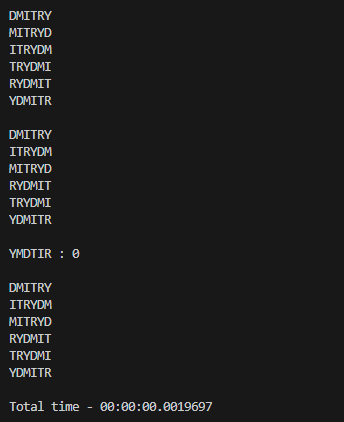


Рисунок 7.3 – Результат работы программы по имени

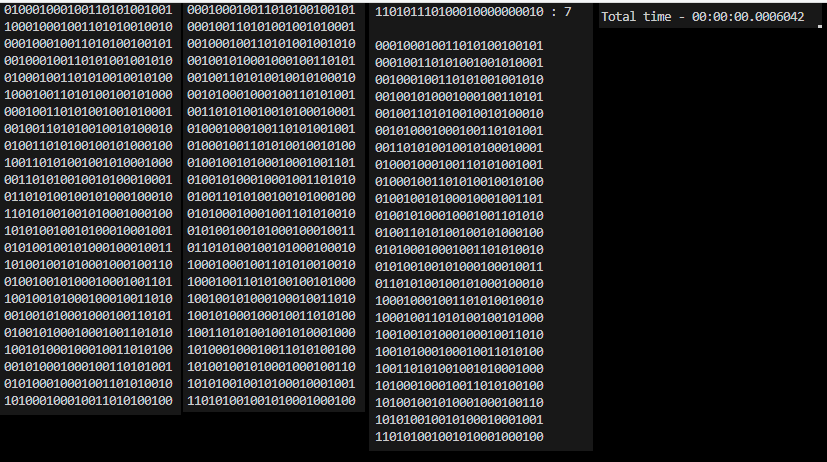


Рисунок 7.4 – Результат работы программы на основе ASCII кодов

**Вывод:** в ходе этой лабораторной работы я изучил принципы сжатия данных. На основе метода Барроуза-Уилера написал программу прямого и обратного преобразований.

# Лабораторная работа №8. Сжатие/Распаковка данных на основе статистических методов

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

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе методов Шеннона − Фано и Хаффмана.
2. Разработать приложение для реализации методов Шеннона − Фано и Хаффмана
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретические сведения**

Мы неоднократно подчеркивали, что каждый из естественных языков обладает избыточностью. Среди европейских языков белорусский и русский обладают одним из самых высоких уровней избыточности. Об этом можно судить по размерам русского перевода английского текста. Обычно он примерно на 20−30% больше.

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 !!!**

**Метод Шеннона – Фано**

Код Шеннона – Фано не является оптимальным (обеспечивает минимальную избыточность) в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона – Фано, и все они могут дать различные результаты.

Итак, необходимо выполнить следующие действия:

1. Подсчитать вероятностные параметры символов алфавита А = {ai} (реализуется статическая версия алгоритма);
2. Отсортировать – обычно в порядке убывания (не возрастания, т. е. могут иметь место повторяющиеся значения) вероятностей р(аi); р(аi) – вероятность появления в сжимаемом сообщении на произвольной позиции символа аi алфавита, т. е. создать таблицу символов алфавита, на основе которого генерируется сжимаемое сообщение;
3. каждому символу отсортированного множества поставить в соответствие бинарный код, для чего это множество (таблица) символов делится на две группы таким образом, чтобы каждая из групп имела приблизительно одинаковую суммарную частоту (вероятность). Очевидно, на первом шаге такая суммарная вероятность в каждой из групп должна быть максимально близка к 0,5. Первому из полученных подмножеств устанавливается первый символ бинарного кода: 0, второй − 1 (или наоборот). Для вычисления следующих битов кодов данная процедура повторяется рекурсивно для каждого из полученных на текущем шаге подмножеств, в котором содержится больше одного символа. Получим таблицу, в которой длина кодовых комбинаций меняется от минимального (lmin) до максимального (lmax) значений.

**Алгоритм прямого преобразования**: необходимо выполнить одну операцию: заменить символы входного сообщения соответствующими бинарными кодами.

**Алгоритм обратного преобразования**: на входе – сообщение в виде бинарной последовательности.

Шаг 1. Анализируются lmin начальных бинарных символов: осуществляется поиск в таблице соответствующего совпадения. Если такое будет найдено, то на выходе будет символ исходного алфавита с совпадающим кодом. После этого процедура повторяется, т. е. анализируются очередные lmin символов. Если не найдено в таблице совпадения, переходим к шагу 2.

Шаг 2. Длина анализируемой последовательности увеличивается на 1 бит: lmin + 1. Осуществляется поиск совпадающей бинарной комбинации такой же длины в таблице. Если такая комбинация существует, на выходе распаковщика формируется соответствующий символ исходного алфавита, если нет – длина анализируемой последовательности увеличивается еще на один бит, и т. д.

По различным причинам при анализе очередной последовательности длиной lmax совпадение в таблице может быть не найдено. Для нейтрализации подобных коллизий архиваторы содержат средства контроля ошибок с помощью корректирующих кодов.

**Метод Хаффмана**

Метод основан на алгоритме оптимального префиксного кодирования алфавита: исходный алгоритм Хаффмана является оптимальным для посимвольного кодирования с известным входным распределением вероятностей, т. е. для отдельного кодирования несвязанных символов в таком потоке данных. Отличается от метода Шеннона – Фано лишь в части кодирования символов исходного алфавита.

В данном случае бинарные коды создаются на основе дерева, ветви которого обозначаются бинарными символами.

Бинарным кодом символа исходного алфавита будет последовательность обозначений ветвей дерева от корня до листа, соответствующего этому символу. В основе бинарного кода лежит следующее положение.

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

Построение дерева начинается с сортирования символов исходного алфавита в порядке убывания (не возрастания).

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

**Практическая часть**

**Метод Шеннона – Фано**

За основу в программе я взял словарь, который хранит в себе значения частоты появления символа и его строковое представление, рассчитанное по методу Шеннона – Фано.

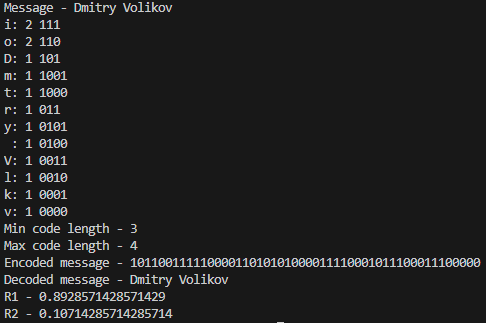


Рисунок 8.1 – Результат программы по методу Шеннона – Фано

**Метод Хаффмана**

За основу в программе я взял всё тот же словарь, который хранит в себе значения частоты появления символа и его строковое представление, рассчитанное по методу Хаффмана.

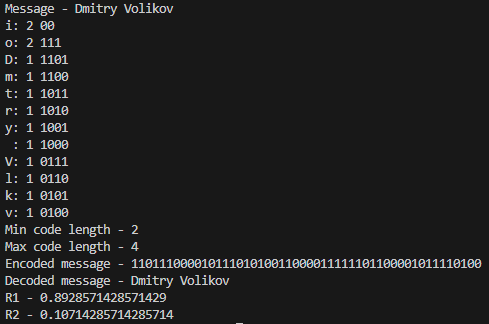


Рисунок 8.2 – Результат программы по методу Хаффмана

**Вывод**: в ходе этой лабораторной работы я закрепил свои знания об алгоритмах статистического сжатия (алгоритмы Хаффмана и Шеннона – Фано). Мною были написаны 2 программы, которые на основе входного сообщения вычисляют частоты появления каждого символа в сообщении и рассчитывают представление этих символов в бинарном представлении. Такой способ позволяет нам значительно уменьшить длину передаваемого сообщения по сравнению с кодами ASCII.

# Лабораторная работа №9. Сжатие/распаковка данных методом Лемпеля – Зива

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

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе метода Лемпеля − Зива.
2. Разработать приложение для реализации метода Лемпеля – Зива
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

**Теоретическая часть**

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

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

Суть метода LZ77 (как и последующих его модификаций) состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также скользящим словарем (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 − (p − 1) , St − (p − 1)+1, St − (p − 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

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.

**Примеры реализации метода**

**Пример 1**. Используется алфавит А = {0,1,2,3}, N = 4, принимаем: длина словаря n1 = 15, длина буфера данных (кодирования) n2 = 13; для обозначения p и q используется четверичная система счисления. Тогда длина кодовой комбинации на каждом шаге составляет

Входной поток S = 2000302013020130313031303130313333333.

Вспомним соответствие между числами в десятичной и четверичной системах счисления:

010 = 004, 110 = 014, 210 = 024, 310 = 034, 410 = 104, 510 = 114 и т. д.

**Прямое преобразование**. Шаг 1: состояние буфера отображает рис. 9.1

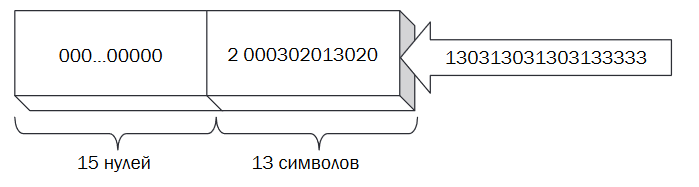


Рисунок 9.1 – Состояние буфера на первом шаге сжатия сообщения S

Анализируем 1-й символ в буфере кодирования на предмет соответствия (наличия) такого же символа или нескольких символов в словаре. Таких символов нет, следовательно, p1 = 0. Длина повторения q1 = 0. Таким образом, имеем следующую триаду: (p1, q1, s1) = (00 00 2)4 (нижний индекс справа от знака равенства означает основание системы счисления).

Шаг 2: состояние буфера отображает рис. 10.2.

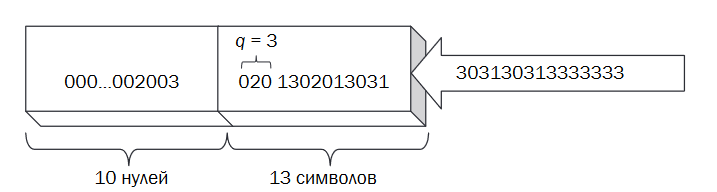


Рисунок 9.2 – Состояние буфера на втором шаге сжатия сообщения S

Находим повторение (000), длина этого повторения q2 = 3. Поскольку в словаре нулевые символы записываются с 1-й по 14-ю позицию (для упрощения индексация ведется слева направо), то индексом р2 (началом повторения) может быть выбрано любое число от 1 до 12: 1 ≤ р2 ≤ 12; выбираем р2 = 6. Итак, получим следующую триаду: (p2, q2, s2) = (6, 3, 3) = (12 03 3)4.

Передвигаем сообщение в окне на q2 + 1 = 3 + 1 = 4 позиции.

Шаг 3: состояние буфера отображает рис. 9.3

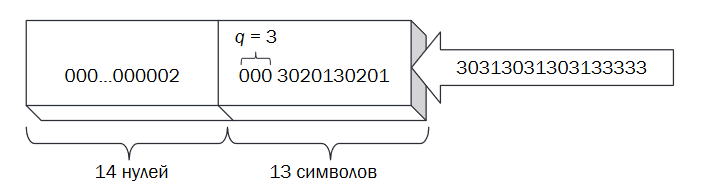


Рисунок 9.3 – Состояние буфера на третьем шаге сжатия сообщения S

В этой ситуации наиболее длинный повтор – 020, т. е. q3 = 3, р3 = 10. Передвигаем сообщение в окне на q3 + 1 = 4 позиции. И получаем триаду: (p3, q3, s3) = (10, 3, 1) = (22 03 1)4.

Шаг 4: состояние буфера отображает рис. 10.4

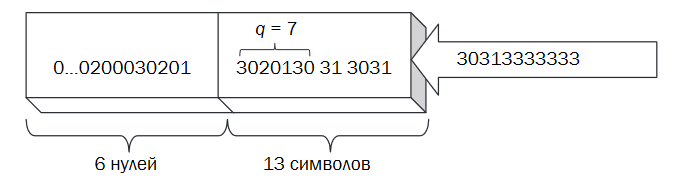


Рисунок 9.4 – Состояние буфера на четвертом шаге сжатия сообщения S

В этой ситуации наиболее длинный повтор – 3020130. Значит, q4 = 7, р4 = 11. Передвигаемся на q4 + 1 = 8 и получаем триаду: (p4 , q4 , s4 ) = (11, 7, 3) = (23 13 3)4 . И т. д.

**Обратное преобразование**. Имеем на входе «запакованное» сообщение: 00002 12033 22031 23133 30301 02013 32103… .

В исходном состоянии в окне (используется одно скользящее окно) записывают 15 нулей (такой выбрана длина окна). Результат анализа – символы исходного сообщения на основе выбранного алфавита.

Шаг 1: анализируем первые 5 символов (вспомним, l(с) = 5): 00002, следовательно, q1 = 0 и р1 = 0. В результате в младший (крайний справа) разряд окна записывается лишь символ 3 (s1 = 3).

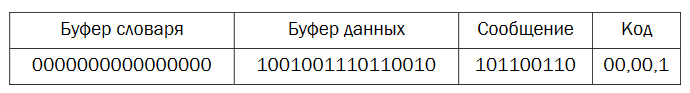
Шаг 2: анализируем следующие 5 символов: 12033, т. е. р2 = 6, q2 = 3. Поскольку в окне содержатся только нули, за исключением последнего разряда (15, индексация ведется как и в случае прямого преобразования), то мы можем сделать вывод, что наш повтор – это следующие три символа: 000. К ним дописывается еще последний символ из анализируемой триады (s2 = 3). Таким образом, после двух шагов начальные символы распакованного сообщения будут такими: 20003. В остальных (начальных: с первого по десятый) разрядах окна будут нули.

Шаг 3: анализируем третью триаду сжатого сообщения: 22031. Следовательно, р3 = 10, q3 = 3. Разряды с десятого (р3 = 10) по двенадцатый (q3 = 3) окна содержат символы 020. Эти разряды допишутся справа к существующим символам окна и дополнительно к ним допишется символ 1 (s3 = 1). Таким образом, в окне после этого будет записано: 000000200030201. И т. д.

**Пример 2**. Рассмотрим преобразование входного потока бинарного типа: S = 1001001110110010101100110. Размер словаря и буфера данных равны 16.

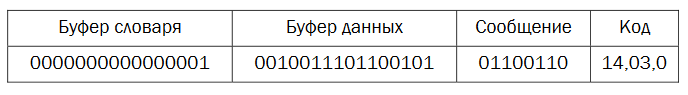
**Прямое преобразование**. Буфер словаря изначально заполнен нулями. Буфер данных содержит первые 16 символов исходного сообщения. Так как размер буфера словаря и буфера данных имеет двухзначное число, то при формировании триады необходимо для записи значений р и q отводить 2 знака.

Шаг 1:



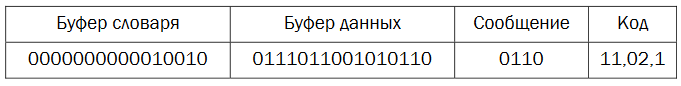
Из буфера данных берем первый слева символ «1» и осуществляем поиск данного символа в буфере словаря. Символ в словаре отсутствует. Формируем триаду с началом повторяющей последовательности (р), равным 0, длиной повторяющихся символов (q), равной 0, и следующим за повторяющейся последовательностью символом (c) «1». Сдвигаем окна на q + 1 позицию вправо.

Шаг 2:

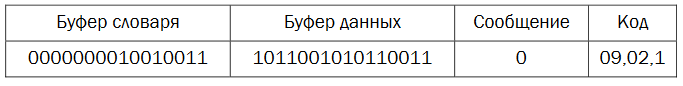


Из буфера данных берем символ «0» и осуществляем поиск данного символа в буфере словаря. Символ в словаре имеется. Увеличиваем длину анализируемой последовательности на 1. Осуществляем поиск «00» в буфере словаря. Такая последовательность имеется. Увеличиваем длину последовательности на 1. Осуществляем поиск «001» в буфере словаря. Вновь последовательность имеется в буфере словаря. Опять увеличиваем длину последовательности на 1. Осуществляем поиск «0010» в буфере словаря. Такого сочетания символов в словаре не обнаружено. Формируем триаду с началом повторяющей последовательности (р), равным 14, длиной повторяющихся символов (q), равной 3, и следующим за повторяющейся последовательностью символом (c) «0». Сдвигаем окна на q + 1 позицию вправо.

Шаг 3:



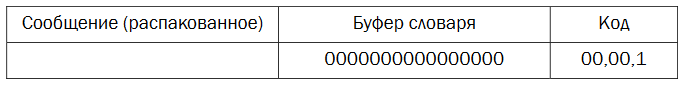
Шаг 4:



Конечным результатом сжатия будет набор полученных триад (кодов): (00,00,1)(14,03,0)(11,02,1)(09,02,1)(06,05,1)(09,06,1)(02,01, …).

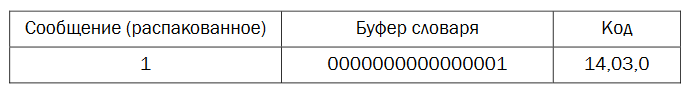
**Обратное преобразование**. В процессе восстановления используются только набор триад (кодов) и буфер. Принимаем размер буфера и его начальное содержимое такое же, как и при сжатии.

Шаг 1:



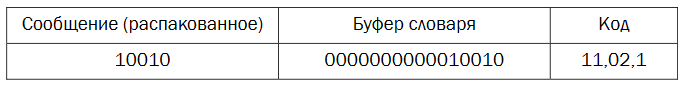
Анализируется триада (00,00,1). Начало повторяющейся последовательности (p) и ее длина (q) равны 0 – повторений нет. Добавляем к словарю символ «1» и сдвигаем окно словаря на q + 1 позицию вправо.

Шаг 2:



Анализируется триада (14,03,0). Повторяющаяся последовательность (p) начинается с четырнадцатой позиции и длиной (q), равной 3. Добавляем к словарю повторяющуюся последовательность «001» и символ «0». Сдвигаем окно словаря на q + 1 позицию вправо.

Шаг 3:



Анализируется триада (11,02,1). Повторяющаяся последовательность (p) начинается с одиннадцатой позиции и длиной (q), равной 2. Добавляем к словарю повторяющуюся последовательность «01» и символ «1». Сдвигаем окно словаря на q + 1 позицию вправо. И т. д.

Как видно из приведенных примеров, алгоритм LZ77 обладает следующими очевидными недостатками:

1. невозможностью кодирования подстрок, отстоящих одна от другой на расстояние, большее длины словаря;
2. длина кодируемой подстроки ограничена размером буфера.

Однако если увеличивать размер словаря, то это приведет к увеличению времени преобразования и длины кодовых последовательностей.

Алгоритмы LZ-формата используются в тех случаях, когда требуется универсальное сжатие. Например, в известном стандарте модема V.42bis, протоколе передачи данных ZModem, форматах GIF, TIFF, ARC и других прикладных программах. Некоторые алгоритмы данного класса используются в дисковых утилитах сжатия типа DoubleSpace и Stacker, графических форматах типа PNG, а также в универсальных утилитах архивирования и сжатия, включая ZIP, GZIP и LHA.

**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 выполнения операций сжатия/распаковки.
4. Результаты оформить в виде отчета по установленным правилам.

Оценим время сжатия и распаковки для алгоритма, изменяя размеры окон n1 и n2:

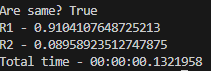


Рисунок 9.5 – Результат программы при n1 = 20 и n2 = 20

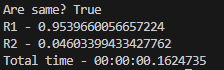


Рисунок 9.6 – Результат программы при n1 = 16 и n2 = 16

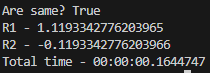


Рисунок 9.7 – Результат программы при n1 = 10 и n2 = 10

**Вывод**: в ходе лабораторной работы я закрепил свои знания об алгоритме сжатия Лемпеля – Зива (известен как LZ77 или LZ78), работающий на основе скользящего окна. Мною была написана программа, которая получает текст из файла, преобразует его в бинарный вид и кодирует. Были проведены оценки сжатия и распаковки, а также времени, затраченного на работу алгоритма.

# Лабораторная работа №10. Сжатие/распаковка данных арифметическим методом

**Цель:** приобретение практических навыков использования арифметического метода для сжатия/распаковки данных.

**Задачи:**

1. Закрепить теоретические знания по алгебраическому описанию и использованию и использованию арифметических методов сжатия/распаковки (архивации/разархивации) данных.
2. Разработать приложение для реализации арифметических методов
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.\

**Теоретическая часть**

При арифметическом сжатии (кодировании) текст представляется вещественными числами в интервале от 0 до 1. По мере анализа текста отображающий его интервал уменьшается, а количество битов для его представления возрастает. Очередные символы текста сокращают величину интервала, исходя из значений соответствующих вероятностей.

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

Таким образом, как и во всех энтропийных алгоритмах, исходной является информация о частоте встречаемости каждого символа алфавита.

*Алгоритмы* прямого и обратного преобразований базируются на операциях с *«рабочим отрезком».*

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

**Описание примера**

Дана последовательность X k = «молоко». Рассчитываем статистику:

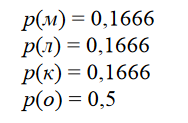


Рисунок 10.1 – Статистика вероятностей символов в сообщении

Шаг 0: строим основной рабочий отрезок, как показано на рис. 10.2.

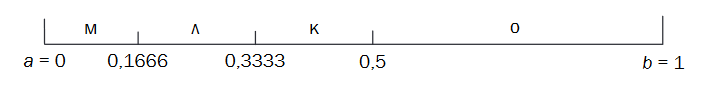


Рисунок 10.2 – Разметка и точки основного рабочего отрезка

Как показано на рис. 11.1, при инициализации алгоритма a = 0, b = 1 (a0 = 0, b0 = 1).

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

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

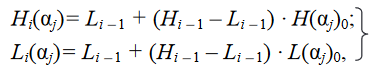


Рисунок 10.3 – Формулы расчёта текущих верхней и нижних границ

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

Таким образом, итогом сжатия входной последовательности «молоко» будет число 0,107060185 (для упрощения дальнейших вычислительных операций округлим его до 0,1071).

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

1. о значении числа, являющегося итогом сжатия сообщения (в нашем случае 0,1071);
2. количестве символов в сжатом сообщении;
3. вероятностных параметрах всех символов исходного сообщения (таблица вероятностей).

Как и при сжатии, вначале необходимо начальный рабочий отрезок [0; 1) разбить на интервалы, длины которых равны вероятностям появления соответствующих символов, т. е. создать рабочий отрезок, полностью соответствующий рис. 10.2.

Анализ будем проводить с использованием конкретных данных, взятых из последнего примера.

Итак, в качестве исходного участка для обратного преобразования принимается исходный участок для прямого преобразования с одинаковыми точками его разбиения. Ниже он повторяется.

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

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



Рисунок 10.4 – Формула расчёта кода для символа

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

Так как нам известно, что символов в сообщении было 6, то на последнем шаге необходимо только определить недостающий символ. Таким символом становится символ «о». Полученное сообщение «молоко» и есть восстановленная («распакованная») последовательность.

**Практическое задание**

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 № 1 такой конкатенацией будет последовательность «летоисчислениевремяпрепровождение».

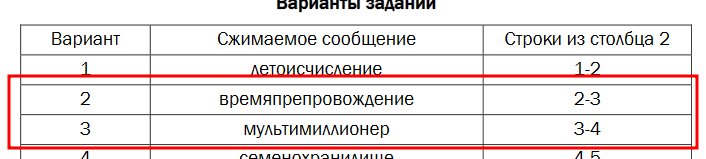


Рисунок 10.5 – Вариант 2

Результат программы №1:

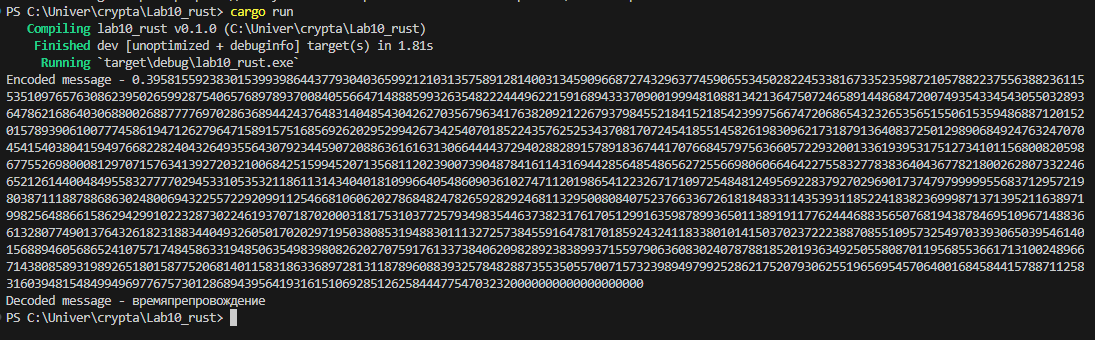


Рисунок 10.6 – Результат программы по строке «времяпрепровождение»

Результат программы №2:

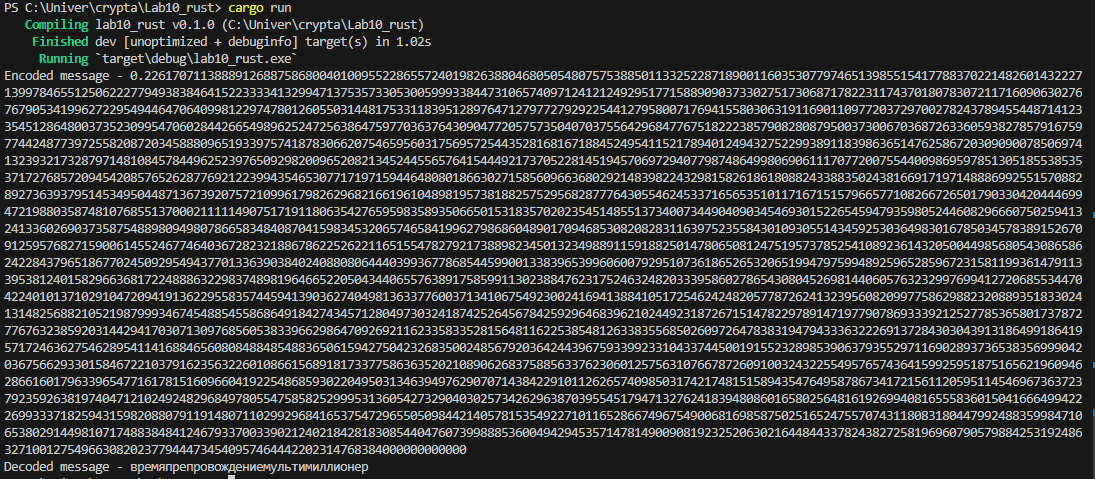


Рисунок 10.7 – Результат программы по строке «времяпрепровождениемультимиллионер»

Вывод: в ходе этой лабораторной работы я изучил очередной алгоритм сжатия данных на основе статистического анализа сообщения. Мною была написана программа, которая может кодировать и декодировать сообщения методом арифметического сжатия.

# Листинг

## Лабораторная работа №1

using System.Text;

namespace lab1

{

class Program

{

public static void Main(string[] args)

{

if (args.Length == 0)

{

Console.Error.WriteLine("Error: alphabet file not specified");

return;

}

string filePath = @"./../../../" + args[0];

string csvFilePath = @"./../../../data.csv";

FileInfo alphabetFile = new(filePath);

if (!alphabetFile.Exists)

{

Console.Error.WriteLine("Error: alphabet file not found");

return;

}

Dictionary<char, int> alphabet = new Dictionary<char, int>();

Dictionary<char, int> binaryAlphabet = new Dictionary<char, int>();

string stringFileContent = File.ReadAllText(filePath);

byte[] binaryFileContent = File.ReadAllBytes(filePath);

StringBuilder sb = new();

foreach (var item in binaryFileContent)

{

sb.Append(Convert.ToString(item, 2).PadLeft(8, '0'));

}

binaryAlphabet['0'] = sb.ToString().Count(c => c == '0');

binaryAlphabet['1'] = sb.ToString().Count(c => c == '1');

foreach (var item in stringFileContent)

{

if (!alphabet.ContainsKey(item))

{

alphabet[item] = stringFileContent.Count(ch => ch == item);

}

}

File.WriteAllText(csvFilePath, "Char;Frequency\n");

foreach (var item in alphabet)

{

string ch = item.Key == '\n' ? "\\n" :

item.Key == '\r' ? "\\r" :

item.Key.ToString();

File.AppendAllText(csvFilePath, ch + ";" + item.Value + '\n');

}

var fileEntropy = calculateEntropy(alphabet, stringFileContent.Length);

var fileBinaryAlphabet = calculateEntropy(binaryAlphabet, sb.Length);

Console.WriteLine("File enthropy = " + fileEntropy);

Console.WriteLine("File binary enthropy = " + fileBinaryAlphabet + "\n");

string alphabetFIO = args[0] == "mongol.txt" ? "Воликов Дмитрий Анатольевич" : "Volikov Dmitry Anatolevich";

string asciiFIO = "Volikov Dmitry Anatolevich";

Console.WriteLine("FIO on alphabet: " + alphabetFIO + "\tI = " + alphabetFIO.Length \* fileEntropy);

Console.WriteLine("FIO on ASCII(p = 0): " + asciiFIO + "\tI = " + asciiFIO.Length \* 8 \* fileEntropy);

Console.WriteLine("FIO on ASCII(p = 0.1): " + asciiFIO + "\tI = " + (1 - calculateConditionalEntropy(0.1)) \* asciiFIO.Length);

Console.WriteLine("FIO on ASCII(p = 0.5): " + asciiFIO + "\tI = " + (1 - calculateConditionalEntropy(0.5)) \* asciiFIO.Length);

Console.WriteLine("FIO on ASCII(p = 1.0): " + asciiFIO + "\tI = " + (1 - calculateConditionalEntropy(1.0)) \* asciiFIO.Length);

}

public static double calculateEntropy(Dictionary<char, int> dict, double length)

{

double result = 0;

foreach (var item in dict)

{

result += Math.Log2(item.Value / length) \* (item.Value / length);

}

return -result;

}

public static double calculateConditionalEntropy(double p)

{

double q = 1 - p;

return -p \* Math.Log2(p) - q \* Math.Log2(q);

}

}

}

## Лабораторная работа №2

using System.Text;

namespace Lab2

{

class Program

{

public static void Main()

{

string fileContent = File.ReadAllText("input.txt");

byte[] fileContentBytes = Encoding.UTF8.GetBytes(fileContent);

string base64EncodedData = new(Base64Encoding(fileContentBytes));

File.WriteAllText("output.txt", base64EncodedData);

Dictionary<char, int> alphabetInput = new();

Dictionary<char, int> alphabetOutput = new();

Console.WriteLine("INPUT ALPHABET PROPERTIES");

foreach (var ch in fileContent)

{

if (!alphabetInput.ContainsKey(ch))

{

alphabetInput[ch] = fileContent.Count(c => c == ch);

}

}

//Console.WriteLine("Chars frequency:");

//PrintCharsFrequency(alphabetInput);

Console.WriteLine($"Shannon entropy: {CalculateShannonEntropy(alphabetInput, fileContent.Length)}");

Console.WriteLine($"Hartley entropy: {CalculateHartleyEntropy(alphabetInput)}");

Console.WriteLine($"Redundancy: {CalculateRedundancy(alphabetInput, fileContent.Length)}\n");

Console.WriteLine("OUTPUT ALPHABET PROPERTIES");

foreach (var ch in base64EncodedData)

{

if (!alphabetOutput.ContainsKey(ch))

{

alphabetOutput[ch] = base64EncodedData.Count(c => c == ch);

}

}

//Console.WriteLine("Chars frequency:");

//PrintCharsFrequency(alphabetOutput);

Console.WriteLine($"Shannon entropy: {CalculateShannonEntropy(alphabetOutput, base64EncodedData.Length)}");

Console.WriteLine($"Hartley entropy: {CalculateHartleyEntropy(alphabetOutput)}");

Console.WriteLine($"Redundancy: {CalculateRedundancy(alphabetOutput, fileContent.Length)}\n");

string name = "Dmitry";

string lastname = "Volikov";

int maxLength = Math.Max(name.Length, lastname.Length);

string asciiName = name.PadRight(maxLength, '\0');

string asciiLastname = lastname.PadRight(maxLength, '\0');

Console.WriteLine("a XOR b XOR b: a = Dmitry, b = Volikov");

Console.WriteLine($"XOR ASCII: {XOR(XOR(asciiName, asciiLastname), asciiLastname)}");

string base64Name = new(Base64Encoding(Encoding.UTF8!.GetBytes(asciiName)));

string base64Lastname = new(Base64Encoding(Encoding.UTF8!.GetBytes(asciiLastname)));

Console.WriteLine($"XOR BASE64: {XOR(XOR(base64Name, base64Lastname), base64Lastname)}");

}

private static string XOR(string name, string lastname)

{

byte[] arr1 = Encoding.ASCII.GetBytes(name);

byte[] arr2 = Encoding.ASCII.GetBytes(lastname);

string result = "";

for (int i = 0; i < arr1.Length; i++)

{

result += (char)(arr1[i] ^ arr2[i]);

}

return result;

}

private static double CalculateShannonEntropy(Dictionary<char, int> dict, double length)

{

double result = 0;

foreach (var item in dict)

{

result += Math.Log2(item.Value / length) \* (item.Value / length);

}

return -result;

}

private static double CalculateHartleyEntropy(Dictionary<char, int> dict)

{

return Math.Log2(dict.Count);

}

private static double CalculateRedundancy(Dictionary<char, int> dict, double length)

{

var shannonEntropy = CalculateShannonEntropy(dict, length);

//var hartleyEntropy = CalculateHartleyEntropy(dict);

return (1 - shannonEntropy / Math.Log2(length)) \* 100;

}

private static void PrintCharsFrequency(Dictionary<char, int> dict)

{

foreach (var item in dict)

{

string ch = item.Key == '\n' ? "\\n" :

item.Key == '\r' ? "\\r" :

item.Key.ToString();

Console.WriteLine($"Char - {ch} : {item.Value}");

}

}

public static char[] Base64Encoding(byte[] data)

{

int length, length2;

int blockCount;

int paddingCount;

length = data.Length;

if ((length % 3) == 0)

{

paddingCount = 0;

blockCount = length / 3;

}

else

{

paddingCount = 3 - (length % 3);

blockCount = (length + paddingCount) / 3;

}

length2 = length + paddingCount;

byte[] source2;

source2 = new byte[length2];

for (int x = 0; x < length2; x++)

{

if (x < length)

{

source2[x] = data[x];

}

else

{

source2[x] = 0;

}

}

byte b1, b2, b3;

byte temp, temp1, temp2, temp3, temp4;

byte[] buffer = new byte[blockCount \* 4];

char[] result = new char[blockCount \* 4];

for (int x = 0; x < blockCount; x++)

{

b1 = source2[x \* 3];

b2 = source2[x \* 3 + 1];

b3 = source2[x \* 3 + 2];

temp1 = (byte)((b1 & 252) >> 2);

temp = (byte)((b1 & 3) << 4);

temp2 = (byte)((b2 & 240) >> 4);

temp2 += temp;

temp = (byte)((b2 & 15) << 2);

temp3 = (byte)((b3 & 192) >> 6);

temp3 += temp;

temp4 = (byte)(b3 & 63);

buffer[x \* 4] = temp1;

buffer[x \* 4 + 1] = temp2;

buffer[x \* 4 + 2] = temp3;

buffer[x \* 4 + 3] = temp4;

}

for (int x = 0; x < blockCount \* 4; x++)

{

result[x] = SixBitToChar(buffer[x]);

}

switch (paddingCount)

{

case 0:

break;

case 1:

result[blockCount \* 4 - 1] = '=';

break;

case 2:

result[blockCount \* 4 - 1] = '=';

result[blockCount \* 4 - 2] = '=';

break;

default:

break;

}

return result;

}

private static char SixBitToChar(byte b)

{

char[] lookupTable = new char[64] {

'A','B','C','D','E','F','G','H','I','J','K','L','M',

'N','O','P','Q','R','S','T','U','V','W','X','Y','Z',

'a','b','c','d','e','f','g','h','i','j','k','l','m',

'n','o','p','q','r','s','t','u','v','w','x','y','z',

'0','1','2','3','4','5','6','7','8','9','+','/'

};

if ((b >= 0) && (b <= 63))

{

return lookupTable[(int)b];

}

else

{

return ' ';

}

}

}

}

## Лабораторная работа №3

using System.Text;

static int fact(int n)

{

if (n == 0)

return 1;

if (n == 1)

return 1;

return fact(n - 1) \* n;

}

static int NewtonBinom(int wt, int r) //wt - min weight of check matrix columns, r - length of check word (binary)

{

return fact(r) / (fact(wt) \* fact(r - wt));

}

static int[,] CalculateH(int k, int r) // H = (P | I), Dmin = 3

{

int counter = 0; // number of filled columns excluding I-matrix

int wt = 2; // weight of matrix columns, do not start with 3 (stack overflow)

int[,] matrix = new int[r,k + r];

while(counter < k)

{

int variationCounter = 0;

int numberOfVariations = NewtonBinom(wt, r);

for (byte i = 1; ;i++)

{

string binaryByte = Convert.ToString(i, 2); // i in binary

int sum = binaryByte.Count(ch => ch.Equals('1')); // weight of i

if (sum == wt)

{

if (binaryByte.Length < r) //add '0'

{

binaryByte = binaryByte.PadLeft(r, '0');

}

for (int j = 0; j < r; j++)

{

matrix[j, counter] = int.Parse(binaryByte[j].ToString());

}

counter++;

variationCounter++;

}

if (counter == k)

{

break;

}

if (variationCounter == numberOfVariations)

{

break;

}

}

wt++;

}

for (int i = k; i < k + r; i++)

{

for (int j = 0; j < r; j++)

{

if (j == i - k)

{

matrix[j, i] = 1;

}

else

{

matrix[j, i] = 0;

}

}

}

return matrix;

}

static int[] CalculateXr(int[,] H, int[] Xk, int r)

{

int[] matrix = new int[r];

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

{

int sum = 0;

for (int j = 0; j < Xk.Length; j++)

{

if (Xk[j] == 1 && Xk[j] == H[i, j])

{

sum += 1;

}

matrix[i] = sum % 2;

}

}

return matrix;

}

static int[] CalculateXn(int[] Xk, int[] Xr, int n)

{

int[] vector = new int[n];

int counter = 0;

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

{

vector[i] = Xk[i];

counter++;

}

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

{

vector[counter] = Xr[i];

counter++;

}

return vector;

}

static int[] CalculateYn(int[] Xn, int errorsCount = 0)

{

if (errorsCount == 1)

{

int[] vector = new int[Xn.Length];

Random random = new();

int index = random.Next(0, Xn.Length - 1);

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

{

if (i == index)

{

vector[i] = Xn[i] == 0 ? 1 : 0;

}

else

{

vector[i] = Xn[i];

}

}

return vector;

}

if (errorsCount == 2)

{

int[] vector = new int[Xn.Length];

Random random = new();

int indexOne = random.Next(0, Xn.Length - 1);

int indexTwo = random.Next(0, Xn.Length - 1);

while (indexOne == indexTwo)

{

indexTwo = random.Next(0, Xn.Length - 1);

}

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

{

if (i == indexOne || i == indexTwo)

{

vector[i] = Xn[i] == 0 ? 1 : 0;

}

else

{

vector[i] = Xn[i];

}

}

return vector;

}

return Xn;

}

static int[] CalculateYk(int[] Yn, int k)

{

int[] vector = new int[k];

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

{

vector[i] = Yn[i];

}

return vector;

}

static int[] CalculateYr(int[] Yn, int r, int k)

{

int[] vector = new int[r];

int counter = 0;

for (int i = k; i < Yn.Length; i++)

{

vector[counter] = Yn[i];

counter++;

}

return vector;

}

static int[] CalculateYrr(int[,] H, int[] Yk, int r)

{

int[] matrix = new int[r];

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

{

int sum = 0;

for (int j = 0; j < Yk.Length; j++)

{

if (Yk[j] == 1 && Yk[j] == H[i, j])

{

sum += 1;

}

matrix[i] = sum % 2;

}

}

return matrix;

}

static int[] CalculateS(int[] Yr, int[] Yrr)

{

int[] vector = new int[Yr.Length];

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

{

vector[i] = Yr[i] == Yrr[i] ? 0 : 1;

}

return vector;

}

static int CalculateErrorPosition(int[,] H, int[] S)

{

int result = -1;

for (int j = 0; j < H.GetLength(1); j++)

{

int sameBits = 0;

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

{

if (H[i, j] == S[i])

{

sameBits++;

}

}

if (sameBits == S.Length)

{

result = j;

break;

}

}

return result;

}

static int[] CalculateE(int n, int mist)

{

int[] vector = new int[n];

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

{

if (i == mist)

{

vector[i] = 1;

}

else

{

vector[i] = 0;

}

}

return vector;

}

static int[] GetFixedMessage(int[] Yn, int[] E)

{

int[] vector = new int[E.Length];

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

{

vector[i] = Yn[i] ^ E[i];

}

return vector;

}

const string inputFilePath = @"./../../../input.txt";

//const string outputFilePath = @"./../../../output.txt";

byte[] binaryFileContent = File.ReadAllBytes(inputFilePath);

int k = binaryFileContent.Length \* 8;

int r = (int)Math.Log2(k) + 1;

int n = k + r;

int[] Xk = new int[k];

StringBuilder sb = new();

binaryFileContent.ToList().ForEach(item => sb.Append(Convert.ToString(item, 2).PadLeft(8, '0')));

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

{

Xk[i] = int.Parse(sb[i].ToString());

}

var H = CalculateH(k, r);

var Xr = CalculateXr(H, Xk, r);

var Xn = CalculateXn(Xk, Xr, n);

var Yn = CalculateYn(Xn);

var Yk = CalculateYk(Yn, k);

var Yr = CalculateYr(Yn, r, k);

var Yrr = CalculateYrr(H, Yk, r);

var S = CalculateS(Yr, Yrr);

var mist = CalculateErrorPosition(H, S);

var E = CalculateE(n, mist);

var Fixed = GetFixedMessage(Yn, E);

## Лабораторная работа №4

static int[] GenerateXk(int k)

{

Random random = new((int)DateTime.Now.Ticks);

int[] word = new int[k];

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

{

word[i] = random.Next() % 2;

}

return word;

}

static int[,] Generate2DMatrix(int[] Xk, int rows, int columns)

{

int[,] matrix = new int[rows, columns];

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

{

for (int j = 0; j < columns; j++)

{

matrix[i, j] = Xk[i \* columns + j];

}

}

return matrix;

}

/\*static int[,,] Generate3DMatrix(int[] Xk, int rows, int columns, int layers)

{

int[,,] matrix = new int[layers, rows, columns];

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

{

for (int j = 0; j < rows; j++)

{

for (int k = 0; k < columns; k++)

{

matrix[i, j, k] = Xk[i \* (rows \* columns) + j \* columns + k];

}

}

}

return matrix;

}

\*/

static int[] CalculateHorizontalParitets(int[,] matrix)

{

int[] result = new int[matrix.GetLength(0)];

for (int i = 0; i < matrix.GetLength(0); i++)

{

int sum = 0;

for (int j = 0; j < matrix.GetLength(1); j++)

{

sum += matrix[i, j];

}

result[i] = sum % 2;

}

return result;

}

static int[] CalculateVerticalParitets(int[,] matrix)

{

int[] result = new int[matrix.GetLength(1)];

for (int i = 0; i < matrix.GetLength(1); i++)

{

int sum = 0;

for (int j = 0; j < matrix.GetLength(0); j++)

{

sum += matrix[j, i];

}

result[i] = sum % 2;

}

return result;

}

static int[] CalculateXn(int[] word, int[] Xh, int[] Xv)

{

return word.Concat(Xh).Concat(Xv).ToArray();

}

static int[] CalculateYn(int[] Xn, int errors = 0)

{

int[] result = new int[Xn.Length];

List<int> errorsPositions = new();

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

{

Random random = new();

int errorPosition = random.Next(0, Xn.Length - 1);

while (errorsPositions.Contains(errorPosition))

{

errorPosition = random.Next(0, Xn.Length - 1);

}

errorsPositions.Add(errorPosition);

}

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

{

if (errorsPositions.Contains(i))

{

result[i] = Xn[i] ^ 1;

}

else

{

result[i] = Xn[i];

}

}

return result;

}

static int[] CalculateYk(int[] Yn, int k)

{

return Yn.ToList().Take(k).ToArray();

}

static int[] CalculateYh(int[] Yn, int k, int k1)

{

return Yn.ToList().Skip(k).Take(k1).ToArray();

}

static int[] CalculateYv(int[] Yn, int k, int k1, int k2)

{

return Yn.ToList().Skip(k + k1).Take(k2).ToArray();

}

(int, int) CalculateErrorPosition(int[] Yh, int[] Yhh, int[] Yv, int[] Yvv)

{

int x = -1;

int y = -1;

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

if (Yh[i] != Yhh[i])

x = i;

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

if (Yv[i] != Yvv[i])

y = i;

return (x, y);

}

int[] CalculateFixedYn(int[] Yk, int[] Yh, int[] Yv, (int, int) error)

{

int[] Yn = Yk.Concat(Yh).Concat(Yv).ToArray();

if (error.Item1 != -1 && error.Item2 == -1)

{

int position = Yk.Length + error.Item1;

Yn[position] = Yn[position] == 1 ? 0 : 1;

}

if (error.Item1 == -1 && error.Item2 != -1)

{

int position = Yk.Length + Yh.Length + error.Item2;

Yn[position] = Yn[position] == 1 ? 0 : 1;

}

if (error.Item1 != -1 && error.Item2 != -1)

{

int position = error.Item1 \* Yv.Length + error.Item2;

Yn[position] = Yn[position] == 1 ? 0 : 1;

}

return Yn;

}

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

int k = 20;

int[] k1s = { 5, 2 };

int[] k2s = { 4, 10 };

int k1 = k1s[1];

int k2 = k2s[1];

var Xk = GenerateXk(k);

var XkMatrix = Generate2DMatrix(Xk, k1, k2);

var Xh = CalculateHorizontalParitets(XkMatrix);

var Xv = CalculateVerticalParitets(XkMatrix);

var Xn = CalculateXn(Xk, Xh, Xv);

var Yn = CalculateYn(Xn, 2);

var Yk = CalculateYk(Yn, k);

var YkMatrix = Generate2DMatrix(Yk, k1, k2);

var Yh = CalculateYh(Yn, k, k1);

var Yv = CalculateYv(Yn, k, k1, k2);

var Yhh = CalculateHorizontalParitets(YkMatrix);

var Yvv = CalculateVerticalParitets(YkMatrix);

var point = CalculateErrorPosition(Yh, Yhh, Yv, Yvv);

var fixedYn = CalculateFixedYn(Yk, Yh, Yv, point);

Console.WriteLine("Xk:");

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

{

Console.Write(Xk[i] + " ");

}

Console.WriteLine();

Console.WriteLine("XkMatrix:");

for (int i = 0; i < XkMatrix.GetLength(0); i++)

{

for (int j = 0; j < XkMatrix.GetLength(1); j++)

{

Console.Write(XkMatrix[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine();

Console.WriteLine("Xh:");

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

{

Console.Write(Xh[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Xv:");

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

{

Console.Write(Xv[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Xn:");

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

{

Console.Write(Xn[i] + " ");

}

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Yk:");

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

{

Console.Write(Yk[i] + " ");

}

Console.WriteLine();

Console.WriteLine("YkMatrix:");

for (int i = 0; i < YkMatrix.GetLength(0); i++)

{

for (int j = 0; j < YkMatrix.GetLength(1); j++)

{

Console.Write(YkMatrix[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine();

Console.WriteLine("Yh:");

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

{

Console.Write(Yh[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Yv:");

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

{

Console.Write(Yv[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Yhh:");

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

{

Console.Write(Yhh[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Yvv:");

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

{

Console.Write(Yvv[i] + " ");

}

Console.WriteLine();

Console.WriteLine($"Error: ({point.Item1};{point.Item2})");

Console.WriteLine("Xn:");

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

{

Console.Write(Xn[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Yn:");

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

{

Console.Write(Yn[i] + " ");

}

Console.WriteLine();

Console.WriteLine("Fixed Yn:");

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

{

Console.Write(fixedYn[i] + " ");

}

Console.WriteLine();

## Лабораторная работа №5

namespace Lab5

{

class Polynom

{

readonly private int degree;

readonly private List<int> bites;

public Polynom(int degree, string bites)

{

this.degree = degree;

this.bites = new List<int>();

foreach (char bit in bites)

{

this.bites.Add(Int32.Parse(bit.ToString()));

}

this.bites = RemoveLeadingZeroes(this.bites);

} // Constructor 1: degree and bites as string

public Polynom(int[] bites)

{

this.bites = RemoveLeadingZeroes(bites.ToList());

degree = this.bites.Count - 1;

} // Constructor 2: int array of bites

public Polynom(int degree) // Constructor 3: degree

{

this.degree = degree;

this.bites = new List<int>();

for (int i = degree; i >= 0; i--)

{

if (i == degree)

{

this.bites.Add(1);

}

else

{

this.bites.Add(0);

}

}

}

public Polynom(List<int> bites)

{

this.bites = bites;

this.degree = bites.Count - 1;

} // Constructor 4: bites

// returns cortege with 2 items: quotinent and remainder polynoms

public (Polynom, Polynom) DivideByPolynom(Polynom polynom)

{

if (degree < polynom.degree)

{

throw new Exception("Error: The degree of the divisor is more than the degree of the dividend");

}

List<int> quotient = new();

List<int> remainder = new();

int counter = 0;

while (counter < bites.Count)

{

remainder.Add(bites[counter]);

if (remainder.Count == polynom.degree + 1)

{

remainder = RemoveLeadingZeroes(remainder);

}

if (remainder.Count == polynom.degree + 1)

{

quotient.Add(1);

for (int i = 0; i < remainder.Count; i++)

{

remainder[i] = remainder[i] ^ polynom.bites[i];

}

remainder = RemoveLeadingZeroes(remainder);

}

else

{

if (quotient.Count != 0)

{

quotient.Add(0);

}

}

counter++;

}

return (new Polynom(quotient.ToArray()), new Polynom(remainder.ToArray()));

}

/\*

\* The method multiplies the original polynomial

\* by a polynomial with one term to the power of the polynomial itself.

\* returns Polynom

\*/

public Polynom MultiplyPolynomByPolynom(Polynom polynom)

{

List<int> result = bites;

List<int> additionalZeroes = new(new int[polynom.degree]);

return new Polynom(result.Concat(additionalZeroes).ToArray());

}

public override string ToString()

{

string result = "";

bites.ForEach(item => result += item.ToString());

return result;

}

private List<int> RemoveLeadingZeroes(List<int> list)

{

List<int> result = new();

list.ForEach(item =>

{

if (result.Count == 0)

{

if (item == 1)

{

result.Add(item);

}

}

else

{

result.Add(item);

}

});

return result;

}

public List<int> GetBites() => this.bites;

}

}

Листинг 5.1 – Polynom класс

namespace Lab5

{

class Program

{

static Polynom GenerateXk(int k)

{

Random random = new((int)DateTime.Now.Ticks);

int[] word = new int[k];

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

{

word[i] = random.Next() % 2;

}

if (word[0] == 0)

{

word[0] = 1;

}

return new Polynom(word);

}

static List<string> GenerateCheckMatrix(int k, int r, Polynom Gx) // string, REALY? yeah, little cheat :)

{

int n = k + r;

List<string> matrix = new();

for (int i = 1; i <= k; i++)

{

var Xnk = new Polynom(n - i);

var Xr = Xnk.DivideByPolynom(Gx).Item2.ToString().PadLeft(r, '0');

matrix.Add(Xr);

}

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

{

string Xr = "1".PadLeft(i + 1, '0').PadRight(r, '0');

matrix.Add(Xr);

}

return matrix;

}

static Polynom CalculateXr(Polynom XkXr, Polynom Gx)

{

var result = XkXr.DivideByPolynom(Gx).Item2.GetBites();

return new Polynom(result);

}

static Polynom CalculateXn(Polynom Xk, Polynom Xr, int r)

{

var XnnBites = Xk.GetBites();

var XrBites = new List<int>();

foreach (var item in Xr.ToString().PadLeft(r, '0'))

{

XrBites.Add(int.Parse(item.ToString()));

}

return new Polynom(XnnBites.Concat(XrBites).ToArray());

}

static Polynom CalculateYn(Polynom Xn, int errors = 0)

{

var bites = Xn.GetBites();

var result = new int[bites.Count];

List<int> errorsPositions = new();

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

{

Random random = new();

int errorPosition = random.Next(0, bites.Count - 1);

while (errorsPositions.Contains(errorPosition))

{

errorPosition = random.Next(0, bites.Count - 1);

}

errorsPositions.Add(errorPosition);

}

for (int i = 0; i < bites.Count; i++)

{

if (errorsPositions.Contains(i))

{

result[i] = bites[i] ^ 1;

}

else

{

result[i] = bites[i];

}

}

return new Polynom(result);

}

static Polynom CalculateFixedYn(Polynom Yn, Polynom S, List<string> CheckMatrix, int r, int k)

{

var bites = Yn.ToString().PadLeft(k + r, '0');

int errorPosition = CheckMatrix.IndexOf(S.ToString().PadLeft(r, '0'));

List<int> fixedBites = new();

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

{

int bit = int.Parse(bites[i].ToString());

if (i == errorPosition)

{

fixedBites.Add(bit ^ 1);

}

else

{

fixedBites.Add(bit);

}

}

return new Polynom(fixedBites);

}

public static void Main()

{

int r = 4; //check bites length

int k = 11; //info word length

int n = k + r; //code word length

var Gx = new Polynom(4, "10011"); //Gx = x^4 + x + 1

var Xk = GenerateXk(k);

Console.WriteLine("Xk:\t" + Xk);

var XInRPower = new Polynom(r);

var Xnn = Xk.MultiplyPolynomByPolynom(XInRPower);

var Xr = CalculateXr(Xnn, Gx);

Console.WriteLine("Xr:\t" + Xr.ToString().PadLeft(r, '0'));

var Xn = CalculateXn(Xk, Xr, r);

Console.WriteLine("Xn:\t" + Xn.ToString().PadLeft(n, '0'));

var Yn = CalculateYn(Xn, 2);

Console.WriteLine("Yn:\t" + Yn.ToString().PadLeft(n, '0'));

var CheckMatrix = GenerateCheckMatrix(k, r, Gx);

var S = Yn.DivideByPolynom(Gx).Item2;

var FixedYn = CalculateFixedYn(Yn, S, CheckMatrix, r, k);

Console.WriteLine("Fixed:\t" + FixedYn.ToString().PadLeft(n, '0'));

Console.WriteLine("S:\t" + S.ToString().PadLeft(r, '0'));

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

CheckMatrix.ForEach(item => Console.WriteLine(item));

}

}

}

Листинг 5.2 – Program класс и соновные методы

## Лабораторная работа №6

using System.Text;

namespace Lab6

{

class Program

{

public static List<string> GenerateCheckMatrix(int k, int r, Polynom Gx) // string, REALY? yeah, little cheat :)

{

int n = k + r;

List<string> matrix = new();

for (int i = 1; i <= k; i++)

{

var Xnk = new Polynom(n - i);

var Xr = Xnk.DivideByPolynom(Gx).Item2.ToString().PadLeft(r, '0');

matrix.Add(Xr);

}

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

{

string Xr = "1".PadLeft(i + 1, '0').PadRight(r, '0');

matrix.Add(Xr);

}

return matrix;

}

static int[] GenerateMessage(int bytes, int k)

{

int bitsLength = bytes \* 8;

int legnth = bitsLength + (k - bitsLength % k);

int[] message = new int[legnth];

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

{

Random random = new();

message[i] = random.Next() % 2;

}

return message;

}

public static int[] GenerateEncodedMessage(int[] message, int k, int r, Polynom Gx)

{

List<int> encodedMessage = new();

var XInR = new Polynom(r);

for (int i = 0; i < message.Length / k; i++)

{

var bites = message.Skip(i \* k).Take(k).ToArray();

var Xk = new Polynom(bites);

var XkXinR = Xk.MultiplyPolynomByPolynom(XInR);

var Xr = XkXinR.DivideByPolynom(Gx).Item2;

var XkXr = Xk.ToString().PadLeft(k, '0') + Xr.ToString().PadLeft(r, '0');

foreach (var item in XkXr)

{

encodedMessage.Add(Int32.Parse(item.ToString()));

}

}

return encodedMessage.ToArray();

}

public static int[,] GenerateInterleavingMatrix(int[] encodedMessage, int columns)

{

int rows = encodedMessage.Length / columns;

int[,] matrix = new int[rows, columns];

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

{

for (int j = 0; j < columns; j++)

{

matrix[i, j] = encodedMessage[i \* columns + j];

}

}

return matrix;

}

public static int[,] GenerateDeinterleavingMatrix(int[] encodedMessage, int columns)

{

int rows = encodedMessage.Length / columns;

int[,] matrix = new int[rows, columns];

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

{

for (int j = 0; j < rows; j++)

{

matrix[j, i] = encodedMessage[i \* rows + j];

}

}

return matrix;

}

public static int[] GetInterleavingMessage(int[,] interleavingMatrix)

{

int rows = interleavingMatrix.GetLength(0);

int columns = interleavingMatrix.GetLength(1);

List<int> interleavingMessage = new();

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

{

for (int j = 0; j < rows; j++)

{

interleavingMessage.Add(interleavingMatrix[j, i]);

}

}

return interleavingMessage.ToArray();

}

public static int[] GenerateErrorsInMessage(int[] interleavingMessage, int errors)

{

int[] message = new int[interleavingMessage.Length];

Random random = new();

int startPosition = random.Next() % (message.Length - errors);

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

{

if (i >= startPosition && i - startPosition < errors)

{

message[i] = interleavingMessage[i] ^ 1;

} else

{

message[i] = interleavingMessage[i];

}

}

return message;

}

public static string GenerateFixedMessage(int[] message, Polynom Gx, List<string> checkMatrix, int k, int r)

{

var fixedMessage = new StringBuilder();

for (int i = 0; i < message.Length / (k + r); i++)

{

var bites = message.ToList().Skip(i \* (k + r)).Take(k + r).ToArray();

var polynom = new Polynom(bites);

var result = new StringBuilder(polynom.ToString().PadLeft(k + r, '0'));

//Console.WriteLine(result);

var S = polynom.DivideByPolynom(Gx).Item2.ToString().PadLeft(r, '0');

if (checkMatrix.Contains(S))

{

result[checkMatrix.IndexOf(S)] = result[checkMatrix.IndexOf(S)] == '1' ? '0' : '1';

}

fixedMessage.Append(result.ToString());

}

return fixedMessage.ToString();

}

public static int[] GetNormalMessage(int[,] interleavingMatrix)

{

List<int> message = new();

for (int i = 0; i < interleavingMatrix.GetLength(0); i++)

{

for (int j = 0; j < interleavingMatrix.GetLength(1); j++)

{

message.Add(interleavingMatrix[i, j]);

}

}

return message.ToArray();

}

public static void Main()

{

int l = 15; // message length in bytes

int columns = 15; // columns number in interleaving matrix

int k = 7; // length of info word

int r = 8;

int n = k + r;

int errors = 3;

int ok = 0;

int fail = 0;

Polynom Gx = new(r, "111010001"); // x^8 + x^7 + x6 + x^4 + 1

var checkMatrix = GenerateCheckMatrix(k, r, Gx);

var message = GenerateMessage(l, k);

var encodedMessage = GenerateEncodedMessage(message, k, r, Gx);

var interleavingMatrix = GenerateInterleavingMatrix(encodedMessage, columns);

var interleavingMessage = GetInterleavingMessage(interleavingMatrix);

var interleavingMessageWithErrors = GenerateErrorsInMessage(interleavingMessage, errors);

var interleavingMatrixWithErrors = GenerateDeinterleavingMatrix(interleavingMessageWithErrors, columns);

var normalMessage = GetNormalMessage(interleavingMatrixWithErrors);

var fixedMessage = GenerateFixedMessage(normalMessage, Gx, checkMatrix, k, r);

var encodedMessageString = "";

encodedMessage.ToList().ForEach(item => encodedMessageString += item.ToString());

message.ToList().ForEach(item => Console.Write(item));

Console.WriteLine();

Console.WriteLine();

encodedMessage.ToList().ForEach(item => Console.Write(item));

Console.WriteLine();

Console.WriteLine();

for (int i = 0; i < interleavingMatrix.GetLength(0); i++)

{

for (int j = 0; j < interleavingMatrix.GetLength(1); j++)

{

Console.Write(interleavingMatrix[i,j] + " ");

}

Console.Write("\t");

for (int j = 0; j < interleavingMatrixWithErrors.GetLength(1); j++)

{

Console.Write(interleavingMatrixWithErrors[i, j] + " ");

}

Console.WriteLine();

}

Console.WriteLine();

Console.WriteLine();

fixedMessage.ToList().ForEach(item => Console.Write(item));

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Is fixed message same as encoded message: " + (fixedMessage == encodedMessageString.ToString()));

}

}

}

## Лабораторная работа №7

using System.Diagnostics;

static List<string> GenerateW1(string message)

{

List<string> list = new();

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

{

var str = "";

for (int j = 0; j < message.Length; j++)

{

str += message[(i + j) % message.Length].ToString();

}

list.Add(str);

}

return list;

}

static List<string> GenerateW2(List<string> w1)

{

var list = new List<string>(w1);

list.Sort();

return list;

}

static (string, int) GetCompressedMessage(List<string> w2, string message)

{

string compressedMessage = "";

w2.ForEach(item => compressedMessage += item[item.Length - 1]);

int messagePosition = w2.IndexOf(message);

return (compressedMessage, messagePosition);

}

static List<string> GetDecompressedW2(string message)

{

List<string> list = new();

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

{

list.Add(message[i].ToString());

}

list.Sort();

for (int i = 0; i < message.Length - 1; i++)

{

for (int j = 0; j < message.Length; j++)

{

list[j] = message[j].ToString() + list[j].ToString();

}

list.Sort();

}

return list;

}

static string GetBinaryMessage(string message)

{

string result = "";

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

{

result += Convert.ToString(message[i], 2).PadLeft(8, '0');

}

return result;

}

string message = "DMITRY".ToUpper();

string binaryMessage = GetBinaryMessage("DMI");

var watch = Stopwatch.StartNew();

var W1 = GenerateW1(message);

var W2 = GenerateW2(W1);

var (CompressedMessage, Z) = GetCompressedMessage(W2, message);

var DecompressedW2 = GetDecompressedW2(CompressedMessage);

watch.Stop();

Console.WriteLine("SIMPLE TEXT BWT\n");

for (int i = 0; i < W1.Count; i++)

{

Console.WriteLine(W1[i]);

}

Console.WriteLine();

for (int i = 0; i < W2.Count; i++)

{

Console.WriteLine(W2[i]);

}

Console.WriteLine();

Console.WriteLine(CompressedMessage + " : " + Z);

Console.WriteLine();

for (int i = 0; i < DecompressedW2.Count; i++)

{

Console.WriteLine(DecompressedW2[i]);

}

Console.WriteLine();

Console.WriteLine($"Total time - {watch.Elapsed}");

Console.WriteLine();

Console.WriteLine("BINARY TEXT BWT\n");

var bWatch = Stopwatch.StartNew();

var bW1 = GenerateW1(binaryMessage);

var bW2 = GenerateW2(bW1);

var (bCompressedMessage, bZ) = GetCompressedMessage(bW2, binaryMessage);

var bDecompressedW2 = GetDecompressedW2(bCompressedMessage);

bWatch.Stop();

for (int i = 0; i < bW1.Count; i++)

{

Console.WriteLine(bW1[i]);

}

Console.WriteLine();

for (int i = 0; i < bW2.Count; i++)

{

Console.WriteLine(bW2[i]);

}

Console.WriteLine();

Console.WriteLine(bCompressedMessage + " : " + bZ);

Console.WriteLine();

for (int i = 0; i < bDecompressedW2.Count; i++)

{

Console.WriteLine(bDecompressedW2[i]);

}

Console.WriteLine();

Console.WriteLine($"Total time - {bWatch.Elapsed}");

## Лабораторная работа № 8

**Метод Шеннона – Фано**

static (Dictionary<string, (int, string)>, Dictionary<string, (int, string)>) DivideAlphabet(Dictionary<string, (int, string)> alphabet)

{

var divider = 1;

var realDivider = 1;

int minSum = int.MaxValue;

while (true)

{

var leftSum = alphabet.Take(divider).Sum(item => item.Value.Item1);

var rightSum = alphabet.Skip(divider).Sum(item => item.Value.Item1);

// Why ABS?

// The formula is:

// |Sum(P(Ai)) - Sum(P(Bi))| must be min

// A - left part of alphabet, B - right part of alphabet

var difference = Math.Abs(leftSum - rightSum);

if (difference < minSum)

{

minSum = difference;

realDivider = divider;

}

if (alphabet.Count - divider == 1) {

break;

}

divider++;

}

var leftAlphabet = alphabet.Take(realDivider).ToDictionary(item => item.Key, item => item.Value);

var rightAlphabet = alphabet.Skip(realDivider).ToDictionary(item => item.Key, item => item.Value);

return (leftAlphabet, rightAlphabet);

}

static Dictionary<string, (int, string)> CalculateAlphabet(string message)

{

Dictionary<string, (int, string)> alphabet = new();

foreach (var ch in message)

{

if (!alphabet.ContainsKey(ch.ToString())) {

var newItem = (message.Count(s => s == ch), "");

alphabet.Add(ch.ToString(), newItem);

}

}

var sortedAlphabet = from entry

in alphabet

orderby entry.Value

descending select entry;

return sortedAlphabet.ToDictionary(t => t.Key, t => t.Value);

}

static void CalculateCodes(

Dictionary<string, (int, string)> partOfAlphabet,

Dictionary<string, (int, string)> alphabet

)

{

var (leftAlphabet, rightAlphabet) = DivideAlphabet(partOfAlphabet);

foreach (var item in leftAlphabet) {

alphabet[item.Key] = (alphabet[item.Key].Item1, alphabet[item.Key].Item2 + "1");

}

foreach (var item in rightAlphabet) {

alphabet[item.Key] = (alphabet[item.Key].Item1, alphabet[item.Key].Item2 + "0");

}

if (leftAlphabet.Count > 1) CalculateCodes(leftAlphabet, alphabet);

if (rightAlphabet.Count > 1) CalculateCodes(rightAlphabet, alphabet);

}

static void PrintAlphabet(Dictionary<string, (int, string)> alphabet)

{

foreach (var item in alphabet)

{

Console.WriteLine(item.Key + ": " + item.Value.Item1 + " " + item.Value.Item2);

}

}

static (int, int) GetMinAndMaxCodesLength(Dictionary<string, (int, string)> alphabet)

{

var min = alphabet.Min(item => item.Value.Item2.Length);

var max = alphabet.Max(item => item.Value.Item2.Length);

return (min, max);

}

static string GetEncodedMessage(

Dictionary<string, (int, string)> alphabet,

string message

)

{

var encodedMessage = "";

foreach (var ch in message) {

encodedMessage += alphabet[ch.ToString()].Item2;

}

return encodedMessage;

}

static string GetDecodedMessage(

Dictionary<string, (int, string)> alphabet,

string encodedMessage,

int minCodeLength

)

{

var decodedMessage = "";

var skip = 0;

var length = minCodeLength;

while (true)

{

if (skip == encodedMessage.Length) {

break;

}

var buf = encodedMessage.Substring(skip, length);

var isCharExists = false;

alphabet.ToList().ForEach(item => {

if (item.Value.Item2 == buf) {

isCharExists = true;

}

});

if (!isCharExists) {

length++;

continue;

}

var ch = alphabet.ToList().Single(item => item.Value.Item2 == buf);

decodedMessage += ch.Key;

skip += buf.Length;

length = minCodeLength;

}

return decodedMessage;

}

static double ClaculateR1(string message, string encodedMessage, int alphabetSize)

{

var averageCodeLength = Math.Ceiling(Math.Log2(alphabetSize));

return encodedMessage.Length / (averageCodeLength \* message.Length);

}

static double ClaculateR2(string message, string encodedMessage, int alphabetSize)

{

var averageCodeLength = Math.Ceiling(Math.Log2(alphabetSize));

return (averageCodeLength \* message.Length - encodedMessage.Length) / (averageCodeLength \* message.Length);

}

var message = "Dmitry Volikov";

var alphabet = CalculateAlphabet(message);

CalculateCodes(alphabet, alphabet);

var (minCodeLength, maxCodeLength) = GetMinAndMaxCodesLength(alphabet);

var encodedMessage = GetEncodedMessage(alphabet, message);

var decodedMessage = GetDecodedMessage(alphabet, encodedMessage, minCodeLength);

var R1 = ClaculateR1(message, encodedMessage, alphabet.Count);

var R2 = ClaculateR2(message, encodedMessage, alphabet.Count);

Console.WriteLine($"Message - {message}");

PrintAlphabet(alphabet);

Console.WriteLine($"Min code length - {minCodeLength}");

Console.WriteLine($"Max code length - {maxCodeLength}");

Console.WriteLine($"Encoded message - {encodedMessage}");

Console.WriteLine($"Decoded message - {decodedMessage}");

Console.WriteLine($"R1 - {R1}");

Console.WriteLine($"R2 - {R2}");

**Метод Хаффмана**

static Dictionary<string, (int, string)> CalculateAlphabet(string message)

{

Dictionary<string, (int, string)> alphabet = new();

foreach (var ch in message)

{

if (!alphabet.ContainsKey(ch.ToString())) {

var newItem = (message.Count(s => s == ch), "");

alphabet.Add(ch.ToString(), newItem);

}

}

var sortedAlphabet = from entry

in alphabet

orderby entry.Value

descending select entry;

return sortedAlphabet.ToDictionary(t => t.Key, t => t.Value);

}

static void CalculateCodes(

Dictionary<string, (int, string)> alphabet,

Dictionary<string, (int, string)> globalAlphabet

) {

if (alphabet.Count == 1) {

return;

}

Dictionary<string, (int, string)> buffer = new(alphabet);

var firstMin = GetMinElementKey(buffer);

buffer.Remove(firstMin.Key);

var secondMin = GetMinElementKey(buffer);

buffer.Remove(secondMin.Key);

foreach (var ch in firstMin.Key) {

globalAlphabet[ch.ToString()] = (globalAlphabet[ch.ToString()].Item1, "0" + globalAlphabet[ch.ToString()].Item2);

}

foreach (var ch in secondMin.Key) {

globalAlphabet[ch.ToString()] = (globalAlphabet[ch.ToString()].Item1, "1" + globalAlphabet[ch.ToString()].Item2);

}

buffer.Add(secondMin.Key + firstMin.Key, (firstMin.Value.Item1 + secondMin.Value.Item1, ""));

CalculateCodes(buffer, globalAlphabet);

}

static KeyValuePair<string, (int, string)> GetMinElementKey(Dictionary<string, (int, string)> alphabet)

{

var minElement = alphabet.First();

foreach (var item in alphabet) {

if (item.Value.Item1 <= minElement.Value.Item1) {

minElement = item;

}

}

return minElement;

}

static (int, int) GetMinAndMaxCodesLength(Dictionary<string, (int, string)> alphabet)

{

var min = alphabet.Min(item => item.Value.Item2.Length);

var max = alphabet.Max(item => item.Value.Item2.Length);

return (min, max);

}

static string GetEncodedMessage(

Dictionary<string, (int, string)> alphabet,

string message

)

{

var encodedMessage = "";

foreach (var ch in message) {

encodedMessage += alphabet[ch.ToString()].Item2;

}

return encodedMessage;

}

static string GetDecodedMessage(

Dictionary<string, (int, string)> alphabet,

string encodedMessage,

int minCodeLength

) {

var decodedMessage = "";

var skip = 0;

var length = minCodeLength;

while (true)

{

if (skip == encodedMessage.Length) {

break;

}

var buf = encodedMessage.Substring(skip, length);

var isCharExists = false;

alphabet.ToList().ForEach(item => {

if (item.Value.Item2 == buf) {

isCharExists = true;

}

});

if (!isCharExists) {

length++;

continue;

}

var ch = alphabet.ToList().Single(item => item.Value.Item2 == buf);

decodedMessage += ch.Key;

skip += buf.Length;

length = minCodeLength;

}

return decodedMessage;

}

static double ClaculateR1(string message, string encodedMessage, int alphabetSize)

{

var averageCodeLength = Math.Ceiling(Math.Log2(alphabetSize));

return encodedMessage.Length / (averageCodeLength \* message.Length);

}

static double ClaculateR2(string message, string encodedMessage, int alphabetSize)

{

var averageCodeLength = Math.Ceiling(Math.Log2(alphabetSize));

return (averageCodeLength \* message.Length - encodedMessage.Length) / (averageCodeLength \* message.Length);

}

static void PrintAlphabet(Dictionary<string, (int, string)> alphabet)

{

foreach (var item in alphabet)

{

Console.WriteLine(item.Key + ": " + item.Value.Item1 + " " + item.Value.Item2);

}

}

var message = "Dmitry Volikov";

var alphabet = CalculateAlphabet(message);

CalculateCodes(alphabet, alphabet);

var (minCodeLength, maxCodeLength) = GetMinAndMaxCodesLength(alphabet);

var encodedMessage = GetEncodedMessage(alphabet, message);

var decodedMessage = GetDecodedMessage(alphabet, encodedMessage, minCodeLength);

var R1 = ClaculateR1(message, encodedMessage, alphabet.Count);

var R2 = ClaculateR2(message, encodedMessage, alphabet.Count);

Console.WriteLine($"Message - {message}");

PrintAlphabet(alphabet);

Console.WriteLine($"Min code length - {minCodeLength}");

Console.WriteLine($"Max code length - {maxCodeLength}");

Console.WriteLine($"Encoded message - {encodedMessage}");

Console.WriteLine($"Decoded message - {decodedMessage}");

Console.WriteLine($"R1 - {R1}");

Console.WriteLine($"R2 - {R2}");

## Лабораторная работа №9

using System.Diagnostics;

using System.Text;

using System.Text.RegularExpressions;

static string GetBinaryMessage(string message)

{

var result = string.Join(

String.Empty,

Encoding.UTF8.GetBytes(message).Select(byt => Convert.ToString(byt, 2).PadLeft(8, '0'))

);

return result;

}

static string EncodeMessage(string message, int n1, int n2)

{

int skip = n2;

string dictionary = new('0', n1);

string buffer = new(message.Take(skip).ToArray());

string encodedMessage = "";

int i = 0; //matched symbols

while (buffer.Length != 0)

{

string str = buffer[i].ToString();

while (dictionary.IndexOf(str) != -1) {

i++;

str += new string(buffer.Skip(i).Take(1).ToArray());

if (new string(buffer.Skip(i).Take(1).ToArray()) == "") {

i--;

break;

}

}

str = new(str.Take(i).ToArray());

var p = !dictionary.Contains(str) ?

0.ToString().PadLeft(n1.ToString().Length, '0') :

dictionary.IndexOf(str).ToString().PadLeft(n1.ToString().Length, '0'); // match position

var offset = i.ToString().PadLeft(n2.ToString().Length, '0');

var s = buffer[i].ToString(); // [offset + 1] char

encodedMessage += p + offset + s;

dictionary = new string(dictionary.Skip(i + 1).ToArray()) + new string(buffer.Take(i + 1).ToArray());

buffer = new string(buffer.Skip(i + 1).ToArray()) + new string(message.Skip(skip).Take(i + 1).ToArray());

skip += i + 1;

i = 0;

}

return encodedMessage;

}

static string DecodeMessage(string encodedMessage, int n1, int n2)

{

var decodedMessage = "";

var buffer = new string('0', n2);

var skip = 0;

while (skip != encodedMessage.Length) {

var part = new string(encodedMessage.Skip(skip).Take(n1.ToString().Length + n2.ToString().Length + 1).ToArray());

var str = "";

var p = int.Parse(part[..n1.ToString().Length]);

var q = int.Parse(part.Substring(n1.ToString().Length, n2.ToString().Length));

var s = part[^1];

if (p == 0 && q == 0) {

str = s.ToString();

}

else {

str = new string(buffer.Skip(p).Take(q).ToArray()) + s;

}

decodedMessage += str;

buffer = new string(buffer.Skip(q + 1).ToArray()) + str;

skip += n1.ToString().Length + n2.ToString().Length + 1;

}

return decodedMessage;

}

static double ClaculateR1(string message, string encodedMessage)

{

var averageCodeLength = Math.Ceiling(Math.Log2(2)); // binary alphabet has 2 chars

return encodedMessage.Length / (averageCodeLength \* message.Length);

}

static double ClaculateR2(string message, string encodedMessage)

{

var averageCodeLength = Math.Ceiling(Math.Log2(2)); // binary alphabet has 2 chars

return (averageCodeLength \* message.Length - encodedMessage.Length) / (averageCodeLength \* message.Length);

}

static string GetMessageFromBinary(string message)

{

var result = Encoding.UTF8.GetString(

Regex.Split(message, "(.{8})")

.Where(binary => !String.IsNullOrEmpty(binary))

.Select(binary => Convert.ToByte(binary, 2))

.ToArray()

);

return result;

}

int n1 = 16;

int n2 = 16;

var fileName = @"input.txt";

var message = File.ReadAllText(fileName);

var binaryMessage = GetBinaryMessage(message);

var watch = Stopwatch.StartNew();

var encodedMessage = EncodeMessage(binaryMessage, n1, n2);

var decodedMessage = DecodeMessage(encodedMessage, n1, n2);

var messageFromBinary = GetMessageFromBinary(decodedMessage);

watch.Stop();

Console.WriteLine(encodedMessage);

Console.WriteLine("Are same? " + (message == messageFromBinary));

var R1 = ClaculateR1(binaryMessage, encodedMessage);

var R2 = ClaculateR2(binaryMessage, encodedMessage);

Console.WriteLine($"R1 - {R1}");

Console.WriteLine($"R2 - {R2}");

Console.WriteLine($"Total time - {watch.Elapsed}");

## Лабораторная работа №10

use std::collections::HashMap;

use bigdecimal::BigDecimal;

fn alphabet(

message: String

) -> Vec<(char, BigDecimal)> {

let mut frequency\_map = HashMap::new();

let length: usize = message.chars().count();

let prec\_size: u64 = u64::try\_from(length.clone()).unwrap();

for c in message.chars() {

let count = frequency\_map.entry(c).or\_insert(BigDecimal::from(0));

\*count += BigDecimal::from(1);

}

let mut res: Vec<\_> = frequency\_map

.iter()

.map(|(char, count)| (\*char, count / BigDecimal::from(length as i64).with\_prec(prec\_size)))

.collect();

res.sort\_by(|(\_, a), (\_, b)| a.cmp(b));

res

}

fn start\_interval(alphabet: Vec<(char, BigDecimal)>) -> Vec<(char, BigDecimal, BigDecimal)> {

let mut interval: Vec<(char, BigDecimal, BigDecimal)> = Vec::new();

let mut low\_boundary: BigDecimal = BigDecimal::from(0);

for item in &alphabet {

if item == &alphabet[0] {

interval.push((alphabet[0].0.clone(), BigDecimal::from(0), alphabet[0].1.clone()));

low\_boundary = alphabet[0].1.clone();

} else {

interval.push((item.0.clone(), low\_boundary.clone(), low\_boundary.clone() + item.1.clone()));

low\_boundary = low\_boundary.clone() + item.1.clone();

}

}

interval

}

fn calculate\_interval(

start\_interval: Vec<(char, BigDecimal, BigDecimal)>,

low\_boundary: BigDecimal,

high\_boundary: BigDecimal

) -> Vec<(char, BigDecimal, BigDecimal)> {

let mut interval: Vec<(char, BigDecimal, BigDecimal)> = Vec::new();

for item in start\_interval {

let new\_low\_boundary = low\_boundary.clone() + (high\_boundary.clone() - low\_boundary.clone()) \* item.1.clone();

let new\_high\_boundary = low\_boundary.clone() + (high\_boundary.clone() - low\_boundary.clone()) \* item.2.clone();

interval.push((item.0, new\_low\_boundary, new\_high\_boundary));

}

interval

}

fn get\_char\_in\_interval(

value: BigDecimal,

start\_interval: Vec<(char, BigDecimal, BigDecimal)>

) -> (BigDecimal, BigDecimal, char) {

let mut low\_boundary: BigDecimal = BigDecimal::from(0);

let mut high\_boundary: BigDecimal = BigDecimal::from(0);

let mut ch: char = ' ';

for item in start\_interval {

if item.1 <= value && value <= item.2 {

low\_boundary = item.1;

high\_boundary = item.2;

ch = item.0;

}

}

return (low\_boundary, high\_boundary, ch);

}

fn encode\_message(

message: String,

start\_interval: Vec<(char, BigDecimal, BigDecimal)>

) -> BigDecimal {

let mut step\_interval: Vec<(char, BigDecimal, BigDecimal)> = start\_interval.clone();

let mut i = 0;

while i < (message.chars().count() - 1) {

let item = step\_interval.iter().find(|item| item.0 == message.chars().nth(i).unwrap()).unwrap();

let low\_boundary: BigDecimal = item.1.clone();

let high\_boundary: BigDecimal = item.2.clone();

step\_interval = calculate\_interval(start\_interval.clone(), low\_boundary, high\_boundary);

i += 1;

}

let result = step\_interval.iter().find(|item| item.0 == message.chars().last().unwrap()).unwrap();

result.1.clone()

}

fn decode\_message(

encoded\_message: BigDecimal,

start\_interval: Vec<(char, BigDecimal, BigDecimal)>,

message\_length: usize

) -> String {

let mut decoded\_message: String = String::new();

let mut step\_value = encoded\_message.clone();

let mut i = 0;

while i < message\_length {

let (low\_boundary, high\_boundary, ch) = get\_char\_in\_interval(step\_value.clone(), start\_interval.clone());

step\_value = (step\_value.clone() - low\_boundary.clone()) / (high\_boundary.clone() - low\_boundary.clone());

decoded\_message.push\_str(&ch.to\_string());

i += 1;

}

decoded\_message

}

fn main() {

let message: &str = "dlkjbhsigpu";

let alphabet = alphabet(message.to\_string());

let start\_interval = start\_interval(alphabet.clone());

let encoded\_message = encode\_message(message.to\_string(), start\_interval.clone());

let decoded\_message = decode\_message(encoded\_message.clone(), start\_interval.clone(), message.chars().count());

println!("Encoded message - {}", encoded\_message);

println!("Decoded message - {}", decoded\_message);

println!("Are messages same: {}", message == decoded\_message.clone());

}