Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надежность информационных систем»

**Отчёт по лабораторной работе №5**

Избыточное кодирование данных в информационных системах.

Итеративные коды

Студент: Жук С.С.

ФИТ 3 курс 2 группа

Преподаватель: Савельева М.Г.

Минск 2025

**Содержание**

[1 Теоретические сведения 3](#_Toc194998887)

[2 Информационное и кодовое слово, Z-матрица 5](#_Toc194998888)

[3 Информационное сообщение с различным количеством ошибок 7](#_Toc194998889)

[3.1 Одиночная ошибка 7](#_Toc194998890)

[3.2 Двойная ошибка 9](#_Toc194998891)

[Вывод 11](#_Toc194998892)

# **1 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 либо композиционными, являются одними из самых простых (с точки зрения аппаратной реализации) избыточных кодов, позволяющих исправлять ошибки в информационных словах.

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

Кодовые слова записываются в виде таблицы. Проверочные символы вычисляются исходя из того, что строки и столбцы должны содержать четное (нечетное) число единиц. На рисунке 1.1 представлена матрица, поясняющая принцип формирования избыточных символов итеративного кода.

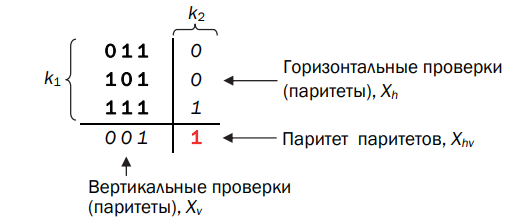


Рисунок 1.1 – Матрица для итеративного кода

Избыточные символы (паритеты) в кодовом слове в принятом порядке (*Xh*, *Xv*, *Xhv*) записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ *Xhv* (паритет паритетов) равен сумме по модулю 2 символов информационного слова *Xk*, а также проверочных символов *Xv* и *Xh*.

Поскольку двумерная матрица формируется как комбинация двух кодов простой четности (по каждому измерению), каждый из которых характеризуется минимальным кодовым расстоянием *d*min = 2, то полученный итеративный код (*r* = *k*1 + *k*2) будет характеризоваться минимальным кодовым расстоянием, равным произведению *d*min по строкам и по столбцам, т. е. 4.

Использование символа *Xhv* обеспечивает минимальное кодовое расстояние такого итеративного кода *d*min (*r* = *k*1 + *k*2 + 1) на единицу больше. В этом легко обнаруживается сходство кода с кодом Хемминга при *d*min = 4.

Декодирование начинают сразу, не ожидая поступления всего блока информации. Проверка соответствия избыточных символов полученного слова (*Yr*= *Yh*, *Yv*, Yhv либо *Yr*= *Yh*, *Yv*) при декодировании позволяет обнаружить любое нечетное число искаженных символов, расположенных в одной строке или в одном столбце. Формально такое декодирование осуществляется сравнением принятых (*Yh*, *Yv*, *Yhv*) и вновь вычисленных (*Y*’*h*, *Y*’*v*, *Y*’*hv*) для полученного слова паритетов. Определение местоположения одиночной ошибки по строке указывает на наличие ошибки в этой строке матрицы, а проверка по столбцу – конкретный символ.

Однако этим кодом не могут быть установлены местоположения многократных ошибок, имеющих четное число искаженных символов как по строкам, так и по столбцам. Простейшая необнаруживаемая ошибка содержит четыре искаженных символа, расположенных в вершинах прямоугольника или квадрата. Это происходит из-за того, что четность (паритет) по строкам и по столбцам матрицы не нарушается.

Принято считать рассматриваемый код многомерным, если количество измерений, по которым вычисляются и анализируются паритеты, не менее 3. Таким образом, простейшим многомерным линейным итеративным кодом является код трехмерный.

Пример реализации трехмерного случая иллюстрирует рис. 1.2. Дополнительно к двум кодам на основе кодов простой четности (по вертикали и горизонтали) избыточные символы вычисляются по диагонали: *Xd*.

Приведенную на рис. 1.2 трехмерную структуру итеративных кодов можно дополнить достаточно большим числом разнообразных проверок на четность по диагоналям, тем самым получив набор кодов с высокими корректирующими возможностями.



Рисунок 1.2 – Принцип формирования избыточных символов для линейного итеративного кода с диагональными проверками

# **2 Информационное и кодовое слово, Z-матрица**

Итеративные коды — это класс линейных кодов, для которых процесс кодирования и декодирования основывается на итеративных алгоритмах, таких как декодирование с использованием алгоритма вероятностного или детерминированного подхода. В отличие от циклических кодов, которые строятся на основе сдвигов битов и порождающих полиномов, итеративные коды используют более сложные методы для улучшения эффективности и обнаружения ошибок.

Процесс построения порождающей матрицы для итеративного кода отличается тем, что может включать использование различных схем кодирования, таких как коды с контрольными суммами или коды на основе декодирования с повторениями. Порождающая матрица может быть как фиксированной, так и адаптируемой в зависимости от условий передачи, где процесс кодирования и декодирования может быть повторён несколько раз для улучшения результатов.

Опишем функцию, которая генерирует случайные двоичные значения. В этом контексте матрица заполняется нулями, а затем первая строка заполняется битами, которые определяются порождающим полиномом. Остальные строки генерируются с помощью циклического сдвига. Функция продемонстрирована в листинге 2.1.

|  |
| --- |
| const generateBinary = (n) => {  return *Array*.from({ length: n }, () => *Math*.round(*Math*.random())); }; |

Листинг 2.1 – Функция для генерации случайных двоичных значений

Далее опишем функцию, которая выполняет вычисление проверочных битов для заданной матрицы. Она принимает матрицу данных matrix, массив групп groups, а также размеры матрицы *k*1 и *k*2. Результатом работы функции является массив проверочных битов. Внутри функции создается пустой массив parityBits, который будет хранить проверочные биты. Затем начинается цикл, который выполняется дважды. В первой итерации суммируются элементы каждой колонки матрицы, и результат подвергается операции по модулю 2 с помощью функции mod2. Эти проверочные биты добавляются в массив parityBits. Во второй итерации суммируются все элементы каждой строки матрицы, и также применяется операция по модулю 2, после чего результаты добавляются в parityBits.

Если в массиве groups присутствует число 3, то вычисляются дополнительные проверочные биты, которые соответствуют диагоналям матрицы, идущим слева направо. Для каждой диагонали вычисляется сумма элементов, затем применяется операция по модулю 2, и результаты добавляются в parityBits. Если в массиве groups присутствует число 4, то выполняется аналогичная операция для диагоналей, но они идут справа налево. Эти результаты также добавляются в parityBits.

Если в groups содержится число 5, то обрабатывается группа из 5 подряд идущих элементов в каждой строке. Для каждого такого блока вычисляется их сумма по модулю 2, и полученные результаты добавляются в массив parityBits. В конце функция возвращает массив parityBits, который содержит все вычисленные проверочные биты.

Следующим шагом была разработана функция, которая служит для вычисления синдромов четности по строкам и столбцам матрицы. Она принимает на вход матрицу и возвращает объект с двумя массивами: первый содержит четность суммы элементов каждой строки по модулю 2, а второй включает четность суммы элементов каждого столбца по модулю 2. Для этого в функции используется метод для строк, который суммирует все элементы строки и затем вычисляет остаток от деления на 2, и метод для столбцов, который суммирует все элементы столбца и также берет остаток от деления на 2. Функция показана на рисунке 2.2.

|  |
| --- |
| const calculateParitySums = (matrix) => {  const rowParitySums = matrix.map(row => row.reduce((sum, val) => sum + val, 0) % 2);  const columnParitySums = matrix[0].map((\_, colIdx) => matrix.reduce((sum, row) => sum + row[colIdx], 0) % 2);  return { rowParitySums, columnParitySums }; }; |

Листинг 2.2 – Функция для вычисления синдромов четности

Далее опишем функцию, которая создает новую матрицу, которая содержит результаты XOR для синдромов четности строк и столбцов двух матриц. Внутри этой функции сначала рассчитываются четности для каждой из матриц. Затем для каждой строки и каждого столбца выполняется XOR между соответствующими синдромами четности этих матриц. Результаты XOR записываются в последнюю строку и последний столбец новой матрицы, а на пересечении последней строки и последнего столбца записывается итоговый результат XOR для последнего элемента. Функция продемонстрирована на рисунке 2.3.

|  |
| --- |
| const createZMatrix = (firstHalfMatrix, secondHalfMatrix) => {  const numRows = firstHalfMatrix.length;  const numCols = firstHalfMatrix[0].length;  let zMatrix = *Array*.from({ length: numRows }, () => *Array*(numCols).fill(0));  const { rowParitySums: rowSums1, columnParitySums: colSums1 } = calculateParitySums(firstHalfMatrix);  const { rowParitySums: rowSums2, columnParitySums: colSums2 } = calculateParitySums(secondHalfMatrix);  const colXORResult = rowSums1.map((sum, index) => sum ^ rowSums2[index]);  const rowXORResult = colSums1.map((sum, index) => sum ^ colSums2[index]);  zMatrix[numRows - 1] = rowXORResult;  for (let i = 0; i < numRows; i++) {  zMatrix[i][numCols - 1] = colXORResult[i];  }  zMatrix[numRows - 1][numCols - 1] = rowXORResult[rowXORResult.length - 1] ^ colXORResult[colXORResult.length - 1];  return zMatrix; }; |

Листинг 2.3 – Функция создания Z-матрицы

# **3 Информационное сообщение с различным количеством ошибок**

В данном разделе будут рассмотрены случаи, когда в передаваемом информационном сообщении могут возникать ошибки. Будут показаны три сценария: наличие 0, 1 и 2 ошибок в переданном сообщении. Для каждого из этих случаев будет проанализирован процесс обнаружения ошибок с использованием итеративного кода.

Также здесь опишем функцию, которая добавляет случайные ошибки в сообщение. Сначала входная матрица копируется в новый массив, чтобы не изменять исходную. Затем начинается цикл, количество итераций которого определяется параметром errorCount (по умолчанию – 1). На каждой итерации случайным образом выбирается позиция в матрице (конкретное место – строка и столбец), и значение в этой позиции инвертируется. Это делается с помощью операции, которая меняет 0 на 1 и наоборот. После того как ошибки были добавлены, возвращается измененная копия матрицы. Функция продемонстрирована в листинге 3.1.

|  |
| --- |
| const introduceError = (matrix, k1, k2, errorCount = 1) => {  const clonedMatrix = matrix.map(row => [...row]);  for (let n = 0; n < errorCount; n++) {  const i = *Math*.floor(*Math*.random() \* k1);  const j = *Math*.floor(*Math*.random() \* k2);  clonedMatrix[i][j] = 1 - clonedMatrix[i][j];  }  return clonedMatrix; }; |

Листинг 3.1 – Функция для добавления случайных ошибок

## **3.1 Одиночная ошибка**

Далее опишем функцию, которая предназначена для исправления ошибок в принятой матрице, используя информацию о синдромах четности, полученных и ожидаемых паритетах. Она принимает пять параметров: receivedParity – массив полученных паритетов, expectedParity – массив ожидаемых паритетов, *k*1 – количество строк данных, *k*2 – количество столбцов данных и matrix – сама матрица, в которой нужно найти и исправить ошибки.

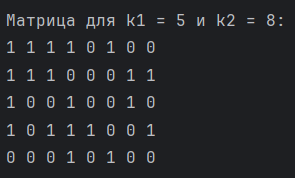
Внутри функции сначала создается копия матрицы, чтобы не изменять исходную, а затем инициализируются два массива: incorrectColumns и incorrectRows, которые будут хранить индексы столбцов и строк, где обнаружены ошибки. Функция проверяет каждый элемент в receivedParity, сравнивая его с соответствующим значением из expectedParity. Если значения не совпадают, это означает, что в данном столбце или строке произошла ошибка.

Затем происходит два этапа проверки: для столбцов и для строк. После этого, если ошибка была обнаружена в одном столбце и одной строке, в соответствующем месте матрицы меняется значение на противоположное (если было 1, становится 0, и наоборот). Если же ошибок больше, функция пытается исправить их на пересечениях всех неправильных строк и столбцов. Функция продемонстрирована в листинге 3.2.

|  |
| --- |
| const findError = (receivedParity, expectedParity, k1, k2, matrix) => {  const correctedMatrix = matrix.map(row => [...row]);  const incorrectColumns = [];  const incorrectRows = [];  for (let n = 0; n < k2; n++) {  if (receivedParity[n] !== expectedParity[n]) {  incorrectColumns.push(n);  }  }  for (let n = k2; n < k1 + k2; n++) {  if (receivedParity[n] !== expectedParity[n]) {  incorrectRows.push(n - k2);  }  }  if (incorrectRows.length === 1 && incorrectColumns.length === 1) {  correctedMatrix[incorrectRows[0]][incorrectColumns[0]] = 1 - correctedMatrix[incorrectRows[0]][incorrectColumns[0]];  }  else if (incorrectRows.length > 1 || incorrectColumns.length > 1) {  for (let i = 0; i < incorrectRows.length; i++) {  for (let j = 0; j < incorrectColumns.length; j++) {  correctedMatrix[incorrectRows[i]][incorrectColumns[j]] = 1 - correctedMatrix[incorrectRows[i]][incorrectColumns[j]];  }  }  }  return correctedMatrix; }; |

Листинг 3.2 – Функция для исправления ошибки

Результат для матрицы 1 показан на рисунке 3.1.



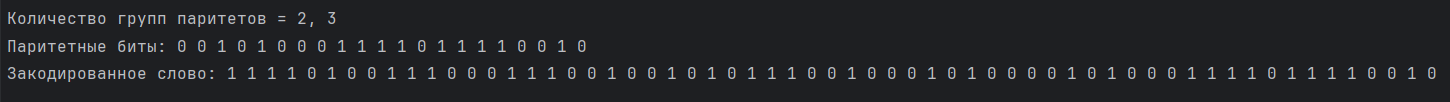


Рисунок 3.1 – Вывод результатов для информационного слова с 1 ошибкой

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 3.2 – Вывод результатов для информационного слова с 1 ошибкой

## **3.2 Двойная ошибка**

Когда в процессе передачи данных не произошло никаких ошибок, переданное сообщение остается целым и корректным. В случае, если мы используем итеративный код, проверочные биты, которые были добавлены в процессе кодирования, будут использоваться для проверки корректности данных. При проверке таких данных с помощью алгоритма декодирования сравниванию избыточные биты и, если они не равны, то добавляем в массивы для хранения корректирующей строки и столбца. Результаты показан на рисунке 3.3 и 3.4.

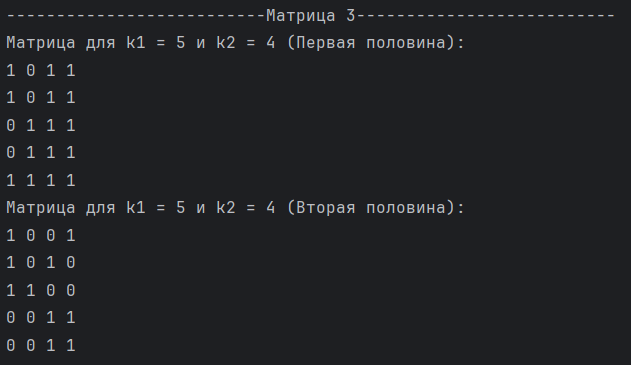
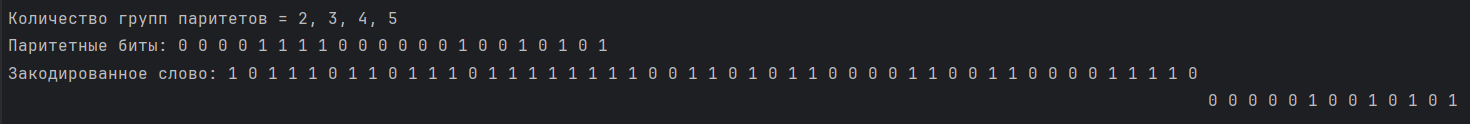


Рисунок 3.3 – Вывод результатов для информационного слова с 2 ошибками



Изображение выглядит как текст, снимок экрана, черный, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

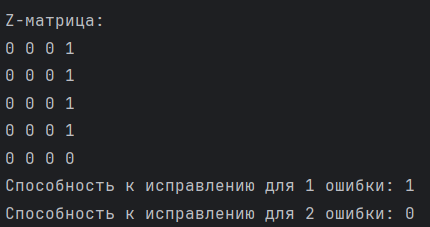


Рисунок 3.4 – Вывод результатов для информационного слова с 2 ошибками

# **Вывод**

В результате выполнения лабораторной работы были изучены основные принципы построения и применения итеративных кодов для обнаружения и исправления ошибок при передаче данных.

Итеративный декодер, благодаря своему методу, может исправить как одиночные, так и некоторые двойные ошибки, что значительно расширяет возможности использования кода в реальных условиях. Однако при внесении более чем двух ошибок корректная декодировка становится невозможной, что подтверждает теоретические ограничения используемого итеративного кода.

Таким образом, в процессе лабораторной работы были реализованы кодирование и декодирование на основе итеративного кода, продемонстрирована эффективность данного метода для исправления как одиночных, так и некоторых двойных ошибок, а также проанализированы его ограничения. При наличии более двух ошибок система может не справиться с их обнаружением или исправлением, что является важным ограничением данного подхода.