Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надежность информационных систем»

**Отчёт по лабораторной работе №7**

Перемежение/деперемежение данных в информационно-вычислительных системах

Студент: Жук С.С.

ФИТ 3 курс 2 группа

Преподаватель: Савельева М.Г.

**Содержание**

[1 Теоретические сведения 3](#_Toc195603203)

[2 Перемежение/деперемежение 7](#_Toc195603204)

[3 Информационное сообщение с различным количеством ошибок 11](#_Toc195603205)

[Вывод 13](#_Toc195603206)

# **1 Теоретические сведения**

Довольно часто распределение ошибок носит взаимозависимый характер. В таких случаях говорят о группах (или пакетах) ошибок.

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.

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 рисунке 1.1.

Изображение выглядит как текст, число, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.1 – Пояснения 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.

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.

Рассмотрим процесс передачи информации с использованием кода Хемминга и блокового перемежителя. Информационный поток (в виде отдельных информационных слов длиной 4 бита) на входе кодера Хемминга (используется код (7, 4)) имеет вид, как показано на рисунке 1.2. Записываем информационное слово:

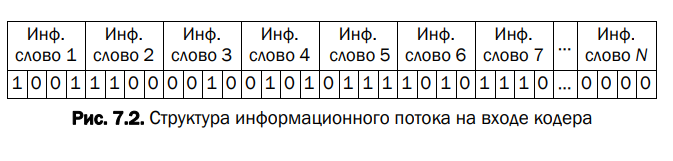


Рисунок 1.2 – Структура информационного потока на входе кодера

На выходе кодера сообщение будет иметь вид, представленный на рисунке 1.3. Считаем проверочную матрицу, проверочные биты, добавляем их и формируем кодовые слова:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.3 – Последовательность символов сообщения на выходе кодера

Используется таблица перемежения размером 7×7, которая после заполнения закодированными сообщениями будет иметь вид, показанный на рисунке 1.4. Строим матрицу перемежения, просто построчно записываем. Одно кодовое слово – одна строка.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.4 – Вид и содержание матрицы перемежения

После перемежения сообщение соответствует последовательности, изображенной на рисунке 1.5. Дальше кодовая комбинация читается по столбцам и записывается в строку:

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.5 – Закодированное сообщение после перемежения

Предположим, что в процессе передачи информации по каналу возник пакет ошибок (выделено черным) длиной 7 битов (рисунок 1.6).

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.6 – Передаваемое сообщение, содержащее группу ошибок

Сообщение на выходе канала записывается по столбцам в матрицу (тех же размеров) деперемежения (рисунок 1.7). Опять строим матрицу, но это уже деперемежение, записываем кодовые комбинации по столбцам:

Изображение выглядит как текст, снимок экрана, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.7 – Вид и содержание матрицы деперемежения

Из матрицы деперемежения двоичные символы сообщения считываются по строкам и поступают на декодер кода Хемминга (рисунок 1.8). Записываем матрицу деперемежения в одну строку по строкам:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.8 – Ошибки разнесены по всему сообщению

После деперемежения пакет ошибок преобразован в одиночные (формально – независимые) ошибки кратности 1 для каждой из кодовых комбинаций кода Хемминга. Такие ошибки код в состоянии исправить. Информационный поток на выходе декодера кода Хемминга имеет вид в соответствии с рисунком 1.9. Исправили ошибки, убрали избыточные биты, получили сообщение:

Изображение выглядит как текст, Шрифт, снимок экрана, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1.9 – Информационный поток на выходе декодера кода Хемминга

# **2 Перемежение/деперемежение**

Циклические коды – это особый класс линейных блоковых кодов, где сдвиг всех битов вправо с переносом последнего в начало даёт допустимый код. Одним из эффективных способов обнаружения и исправления ошибок при передаче данных является использование циклических кодов с заданным порождающим полиномом.

Для построения генераторной матрицы циклического кода используется порождающий полином, представленный в виде строки битов. На основе этого полинома формируется порождающая матрица размером *k* × *n*, где первая строка содержит сам полином, выровненный по левому краю. Каждая последующая строка получается циклическим сдвигом предыдущей влево. Такой способ построения соответствует свойству цикличности кода. Кодовые слова можно безопасно передавать по каналу связи с возможностью обнаружения и исправления одиночных ошибок, а также некоторых множественных.

Для начала опишем функцию, которая разбивает входную строку на фрагменты фиксированной длины. Если последний блок короче размера, он дополняется нулями, чтобы обеспечить одинаковую длину для всех сегментов. Код представлен в листинге 2.1.

|  |
| --- |
| const splitMessage = (msg, size) => {  const result = [];  for (let i = 0; i < msg.length; i += size) {  result.push(msg.slice(i, i + size).padEnd(size, '0'));  }  return result; }; |

Листинг 2.1 – Функция для разбития на блоки информационного сообщения

Следующим шагом будет описана функция, которая выполняет перестановку элементов в матрице, извлекая их по столбцам, а не строкам. Фактически, она транспонирует данные, формируя новую последовательность. Функция продемонстрирована в листинге 2.2.

|  |
| --- |
| const interleave = (matrix) => {  const rows = matrix.length;  const cols = matrix[0].length;  let result = '';  for (let c = 0; c < cols; c++) {  for (let r = 0; r < rows; r++) {  result += matrix[r][c];  }  }  return result; }; |

Листинг 2.2 – Функция для чтения по столбцам матрицы перемежения

Далее опишем функцию, которая выполняет обратную операцию к interleave: она восстанавливает первоначальную матрицу из переставленной последовательности. Сначала создается пустая матрица, последовательно заполняются элементы, проходя по столбцам, а внутри каждого столбца по строкам. В конце каждая строка преобразуется в строку и возвращается в виде массива. Программная реализация функции показана в листинге 2.3.

|  |
| --- |
| const deinterleave = (str, rows, cols) => {  const matrix = *Array*.from({ length: rows }, () => *Array*(cols).fill('0'));  let index = 0;  for (let c = 0; c < cols; c++) {  for (let r = 0; r < rows; r++) {  matrix[r][c] = str[index++];  }  }  return matrix.map(row => row.join('')); }; |

Листинг 2.3 – Функция деления многочленов по модулю 2

Следующая функция случайным образом выбирает группу битов в строке и инвертирует их. Сначала преобразуем строку битов в массив. Далее выбираем случайное начальное положение ошибки. Инвертируем несколько подряд идущих битов. Программная реализация функции представлена в листинге 2.4.

|  |
| --- |
| const addRandomGroupErrors = (bitStr, groupLen) => {  const bits = bitStr.split('');  const start = *Math*.floor(*Math*.random() \* (bits.length - groupLen + 1));  for (let i = 0; i < groupLen; i++) {  bits[start + i] = bits[start + i] === '0' ? '1' : '0';  }  return bits.join(''); }; |

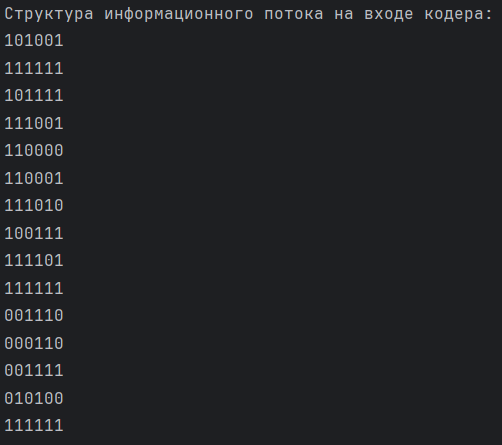
Листинг 2.4 – Функция для построения канонической порождающей матрицы

Эти фрагменты кода иллюстрируют часть алгоритма кодирования перемежения/деперемежения с использованием циклического кода для защиты данных от ошибок при их передаче. Циклический код добавляет проверочные биты в исходное сообщение, чтобы оно стало исправляемым в случае ошибок.

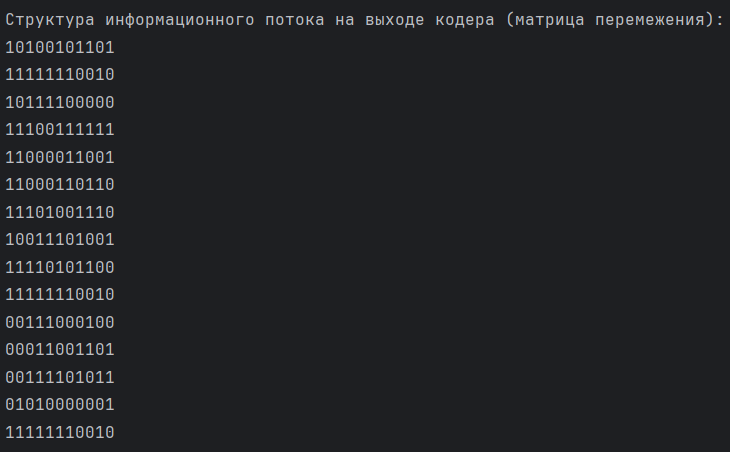
В результате получим закодированную последовательность слов с помощью циклического кода, матрицу перемежения, состоящую из данной последовательности, закодированное сообщение после перемежения, сообщение с ошибками, матрицу деперемежения, построенную на основе информационного сообщения с ошибками. Результат показан на рисунке 2.1.

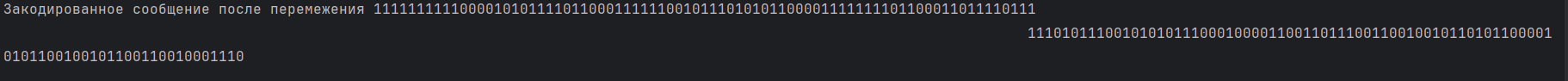


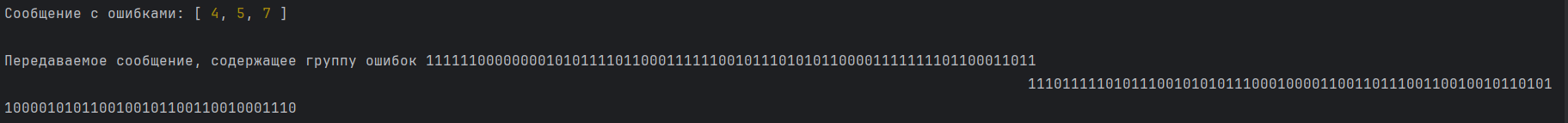
Рисунок 2.1 – Вывод результатов

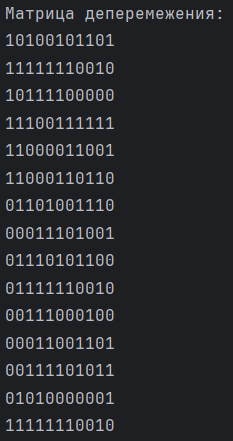


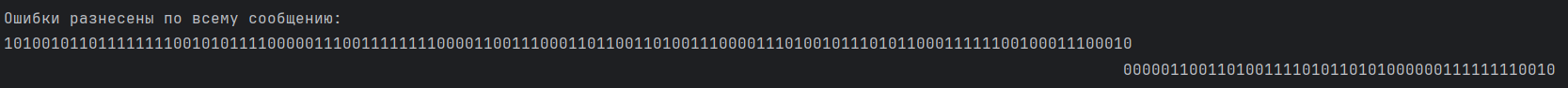












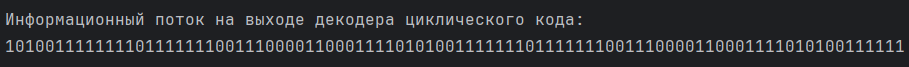


Рисунок 2.2 – Вывод результатов

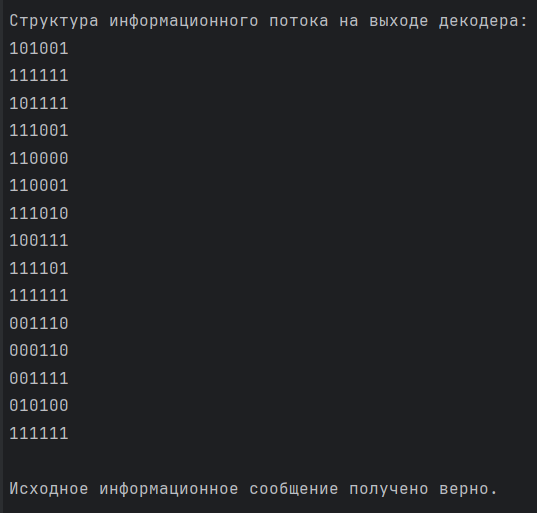


Рисунок 2.3 – Вывод результатов

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

Для выполнения этого задания были созданы функции для исправления ошибок. Было сгенерировано случайное местоположение пакета ошибок каждой из трех заданных длин (4, 5, 7) для 30 случайных ситуаций. Для каждой ситуации выполнялись операции перемежения, деперемежения и исправления ошибок.

После исправления ошибок в сообщении проводилось сравнение передаваемой последовательности и полученной после исправления ошибок. Для каждого случая анализировалась эффективность перемежения/деперемежения, определялся процент исправленной ошибки. Затем вычислялся общий процент исправления ошибок для всех ситуаций каждой длины пакета ошибок. Этот анализ позволил оценить эффективность алгоритмов исправления ошибок. В листинге 3.1 представлена реализация функций.

|  |
| --- |
| const errorGroupSizes = [4, 5, 7]; const trialsPerGroup = 30;  errorGroupSizes.forEach(groupLen => {  let successCount = 0;  for (let t = 0; t < trialsPerGroup; t++) {  const errored = addRandomGroupErrors(interleaved, groupLen);  *console*.log('\nПередаваемое сообщение, содержащее группу ошибок', errored)  const depMatrix = deinterleave(errored, encodedWords.length, n);  *console*.log('\nМатрица деперемежения:');  depMatrix.forEach(word => { *console*.log(`${word}`);});  *console*.log('\nОшибки разнесены по всему сообщению:');  *console*.log(depMatrix.join(''));  const H = makeCheckMatrix(makeCanonicalG(k, n, g), k, n);  const correctedWords = depMatrix.map(code => {  const syndrome = getSyndrome(code, g);  const errVec = findErrorPosition(syndrome, H, n);  const corrected = xorStr(code, errVec);  return corrected.slice(0, k);  });  const recoveredMessage = correctedWords.join('');  *console*.log('\nИнформационный поток на выходе декодера циклического кода:');  *console*.log(recoveredMessage);  const resultWords = splitMessage(recoveredMessage, k);  *console*.log('\nСтруктура информационного потока на выходе декодера:');  resultWords.forEach(word => { *console*.log(`${word}`);});  if (recoveredMessage === originalMessage) {  *console*.log('\nИсходное информационное сообщение получено верно.');  successCount++;  }}  const result = (successCount/trialsPerGroup) \* 100;  *console*.log(`\nОшибка длины ${groupLen}: успешно восстановлено ${result} %`); }); |

Листинг 3.1 – Эффективность перемежения/деперемежения

Изображение выглядит как снимок экрана, Шрифт, текст, типография

Контент, сгенерированный ИИ, может содержать ошибки.





Рисунок 3.1 – Вывод результатов

# **Вывод**

Проведённое исследование показывает, что метод перемежения и деперемежения эффективно защищает данные от групповых ошибок при передаче или хранении информации. Вместо сложных кодов коррекции всё чаще применяют комбинацию перемежения с обычными исправляющими кодами. Среди таких методов блочное перемежение выделяется как простой и надёжный способ.

Тем не менее, при обработке больших объёмов данных скорость работы может заметно снижаться.

Метод перемежения хорошо справляется с ошибками, потому что чаще всего они возникают в соседних битах или в одном байте. В таких случаях их легко обнаружить и исправить с помощью применяемого кода. Однако если ошибки появляются сразу в нескольких местах сообщения, их исправление становится сложнее и может потребовать дополнительных методов.