Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №7 «Перемежение/деперемежение данных в информационно-вычислительных системах»

Студент: Водчиц Анастасия

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

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

**Задачи**:

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

# **Теоретические сведения**

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.

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.

# **Практические задания**

Разработать собственное приложение, которое позволяет выполнять следующие операции:

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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вариант | Используемый корректирующий код | Длина пакета ошибок | Число столбцов в матрице | Длина сообщения, байт | Длина информационного слова k, бит |
| 3 | Итеративный | 3,5,6 | 6 | 15 | 6 |

Рассмотрим следующую ситуацию:

Сообщение: 12 бит = 2 блока.

Блок 1 (Xk1): 110011

Блок 2 (Xk2): 010101

Итеративный код: k=6.

* Структура: k1=2 строки, k2=3 столбца => k=6 информационных бит.
* Проверочные биты (r=5): 2 по строкам (P1), 3 по столбцам (P2).
* Длина кодового блока: n = k+r = 6+5 = 11 бит.

Перемежитель: 4 столбца, матрица 6x4, перемеженная длина 24 бита.

Пакетная ошибка: b=2 бита.

Кодирование для блока 1: Xk1 = 110011

[ 1 1 0 ]

[ 0 1 1 ]

P1[0] (r0): 1^1^0 = 0

P1[1] (r1): 0^1^1 = 0

P2[0] (c0): 1^0 = 1

P2[1] (c1): 1^1 = 0

P2[2] (c2): 0^1 = 1

Xr1 = 0 0 1 0 1

Xn1 = 110011 00101

Кодирование для блока 2**:** Xk2 = 010101

[ 0 1 0 ]

[ 1 0 1 ]

P1[0] (r0): 0^1^0 = 1

P1[1] (r1): 1^0^1 = 0

P2[0] (c0): 0^1 = 1

P2[1] (c1): 1^0 = 1

P2[2] (c2): 0^1 = 1

Xr2 = 1 0 1 1 1

Xn2 = 010101 10111

Объединенное Xn: 11001100101 01010110111 (22 бита)

Далее построим матрицу для перемежения.

Перемежение изменяет порядок следования бит так, чтобы соседние биты в исходной последовательности оказались разнесены в перемеженной. Это делается для того, чтобы пакетная ошибка (повреждающая несколько бит подряд) в канале связи после деперемежения превратилась в несколько одиночных или малочисленных ошибок, распределенных по разным кодовым блокам.

Матрица 6x4. Записываем биты Xn в матрицу по строкам, слева направо, сверху вниз. Добавляем нули (паддинг) в конце, чтобы заполнить матрицу:

[ 1 1 0 0 ]

[ 1 1 0 0 ]

[ 1 0 1 0 ]

[ 1 0 1 0 ]

[ 1 1 0 1 ]

[ 1 1 0 0 ]

Xn\_perem (чтение по столбцам): 111111 110011 001100 000010 (24 бита)

Моделируем повреждение данных при передаче по каналу. Мы предполагаем, что ошибка происходит *после* перемежения (т.е., в канале). Принимаем Yn\_perem с ошибками в 6 и 7 битах: 111110 010011 001100 000010 (24 бита)

На приемной стороне выполняется обратная операция перемежению, чтобы восстановить исходный порядок следования кодовых блоков перед декодированием. Записываем Yn\_perem в матрицу 6x4 по столбцам (сверху вниз, слева направо):

[ 1 0 0 0 ]

[ 1 1 0 0 ]

[ 1 0 1 0 ]

[ 1 0 1 0 ]

[ 1 1 0 1 ]

[ 0 1 0 0 ]

Читаем Yn по строкам (22 бита): Yn = 1000 1100 1010 1010 1101 01.

Как видим, ошибки разбросаны по разным местам полученного сообщения.

Бит 0 относится к блоку 1 (Xn1) и является информационным битом.

Бит 21 относится к блоку 2 (Xn2) и является проверочным битом.

Итеративный декодер пытается исправить ошибки в принятом блоке Yn, используя избыточность (проверочные биты). Он итеративно вычисляет синдромы (признаки ошибок) и корректирует биты (данных или проверочные).

Разбиваем Yn на блоки:

Yn1 = 10001100101

[ 1 0 0 ]

[ 0 1 1 ]

P1[0] (r0): 1^0^0 = 1

P1[1] (r1): 0^1^1 = 0

P2[0] (c0): 1^0 = 1

P2[1] (c1): 0^1 = 1

P2[2] (c2): 0^1 = 1

Yr1 = 0 0 1 0 1

Yr1’ = 1 0 1 1 1

S = 10 для строк

S = 010 для столбцов

Ошибка в 1 строке, 2 столбце.

Yn2 = 01010110101

[ 0 1 0 ]

[ 1 0 1 ]

P1[0] (r0): 0^1^0 = 1

P1[1] (r1): 1^0^1 = 0

P2[0] (c0): 0^1 = 1

P2[1] (c1): 1^0 = 1

P2[2] (c2): 0^1 = 1

Yr2 = 1 0 1 0 1

Yr2’ = 1 0 1 1 1

S = 00 для строк

S = 010 для столбцов

S = 00010

Ошибка в 4 паритете.

Рассмотрим аналогичные ошибки без перемежения:

Xn: 11001100101 01010110111

Пришло сообщение с ошибками в 6 и 7 битах:

Yn = 11001010101 01010110111

Разбиваем на два блока:

Yn1 = 11001010101

Yn2 = 01010110111

В первом блоке две ошибки, которые итеративный код не умеет исправлять.

Задание 2. Местоположение заданной группы ошибок выбирается (генерируется) случайным образом. Необходимо для группы ошибок каждой длины сгенерировать 30−40 случайных ситуаций. После деперемежения и исправления ошибок в сообщении сравнить передаваемую последовательность и полученную после исправления ошибок. Проанализировать эффективность перемежения/деперемежения.

|  |
| --- |
| **public int[] Interleave(int[] data, bool verbose = false)**  **{**  **if (data.Length != \_originalDataLength)**  **throw new ArgumentException($"Input data length {data.Length} does not match expected length {\_originalDataLength}.", nameof(data));**  **int paddedLength = \_rows \* \_columns;**  **int[,] matrix = new int[\_rows, \_columns];**  **int dataIndex = 0;**  **var interleavedData = new List<int>(paddedLength);**  **if (verbose)**  **{**  **Console.WriteLine("\n--- Перемежение (Interleaving) ---");**  **PrintUtils.PrintVector(data, "Входные данные (до перемежения)");**  **}**  **// Запись по строкам с паддингом**  **for (int i = 0; i < \_rows; i++)**  **{**  **for (int j = 0; j < \_columns; j++)**  **{**  **if (dataIndex < \_originalDataLength) matrix[i, j] = data[dataIndex++];**  **else matrix[i, j] = 0;**  **}**  **}**  **if (verbose)**  **{**  **PrintUtils.PrintMatrix(matrix, $"Матрица перемежителя ({\_rows}x{\_columns}) (Заполнена по строкам)");**  **Console.WriteLine("Чтение данных по столбцам...");**  **}**  **// Чтение по столбцам**  **for (int j = 0; j < \_columns; j++)**  **{**  **for (int i = 0; i < \_rows; i++)**  **{**  **interleavedData.Add(matrix[i, j]);**  **}**  **}**  **int[] result = interleavedData.ToArray();**  **if (verbose)**  **{**  **PrintUtils.PrintVector(result, "Выходные данные (после перемежения)");**  **Console.WriteLine("--- Конец Перемежения ---");**  **}**  **return result;**  **}** |

Листинг 2.1 – Метод перемежения

|  |
| --- |
| **public int[] Deinterleave(int[] interleavedData, bool verbose = false)**  **{**  **int expectedInterleavedLength = \_rows \* \_columns;**  **if (interleavedData.Length != expectedInterleavedLength)**  **throw new ArgumentException($"Input interleaved data length {interleavedData.Length} does not match expected length {expectedInterleavedLength}.", nameof(interleavedData));**  **int[,] matrix = new int[\_rows, \_columns];**  **int dataIndex = 0;**  **var deinterleavedData = new List<int>(\_originalDataLength);**  **if (verbose)**  **{**  **Console.WriteLine("\n--- Деперемежение (Deinterleaving) ---");**  **PrintUtils.PrintVector(interleavedData, "Входные данные (перемеженные, с ошибкой)");**  **}**  **// Запись по столбцам**  **for (int j = 0; j < \_columns; j++)**  **{**  **for (int i = 0; i < \_rows; i++)**  **{**  **matrix[i, j] = interleavedData[dataIndex++];**  **}**  **}**  **if (verbose)**  **{**  **PrintUtils.PrintMatrix(matrix, $"Матрица деперемежителя ({\_rows}x{\_columns}) (Заполнена по столбцам)");**  **Console.WriteLine($"Чтение данных по строкам (до исходной длины {\_originalDataLength})...");**  **}**  **// Чтение по строкам до originalDataLength**  **dataIndex = 0;**  **for (int i = 0; i < \_rows; i++)**  **{**  **for (int j = 0; j < \_columns; j++)**  **{**  **if (dataIndex < \_originalDataLength)**  **{**  **deinterleavedData.Add(matrix[i, j]);**  **dataIndex++;**  **}**  **else break;**  **}**  **if (dataIndex >= \_originalDataLength) break;**  **}**  **int[] result = deinterleavedData.ToArray();**  **if (verbose)**  **{**  **PrintUtils.PrintVector(result, "Выходные данные (деперемеженные, с 'размазанной' ошибкой)");**  **Console.WriteLine("--- Конец Деперемежения ---");**  **}**  **return result;**  **}** |

Листинг 2.2 – Метод деперемежения

Результат испытаний для пакета 3 бита:

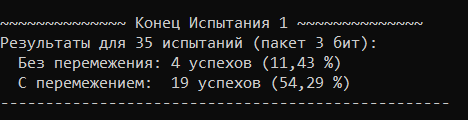


Рисунок 2.1 – Результат испытаний с ошибкой 3 бита

Результат испытаний для пакета 5 бита:

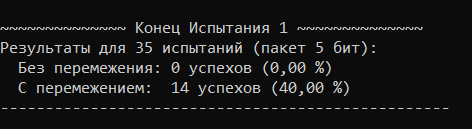


Рисунок 2.2 – Результат испытаний с ошибкой 5 бит

Результат испытаний для пакета 6 бита:

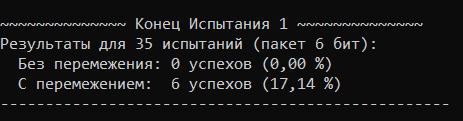


Рисунок 2.3 – Результат испытаний с ошибкой 6 бит

Задание 3. Результаты оформить в виде отчета по установленным правилам.

**Вывод**: в ходе выполнения данной лабораторной работы было проведено исследование влияния перемежения на способность итеративного корректирующего кода (product code k=6, n=11 с проверками по строкам и столбцам) исправлять пакетные ошибки различной длины (b=3, 5, 6 бит). Моделирование осуществлялось путем внесения пакетных ошибок случайной длины и положения в закодированную последовательность данных как без использования перемежителя, так и с его применением (матричный перемежитель с 6 столбцами). Для каждой длины пакета было проведено 35 случайных испытаний.