Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Перемежение/деперемежение данных в информационно-вычислительных системах**

Студент: Дрозд А. И.

ФИТ 3 курс 2 группа

Преподаватель:

Нистюк Ольга Александровна

Минск 2024

**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**

Циклические коды – это семейство помехоустойчивых кодов, одной из разновидностей которых являются коды Хемминга. Основные свойства ЦК:

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 Yn = 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. 1.1.

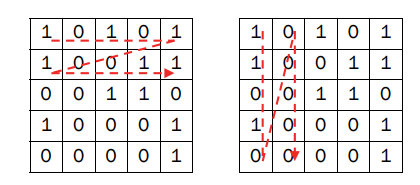


Рисунок 1.1 – Пояснение принципа блочного перемежения

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**

Вариант 3



Рисунок 2.1 – Варианты заданий

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, представленной на рисунке 2.1.

На основе моего варианта (3 вариант), необходимо реализовать процессы перемежения/деперемежения на основе итеративного кода, изучаемого нами ранее. Для этого первоначально нам необходимо разработать метод, который будет подсчитывать горизонтальные паритеты для каждого информационного слова. Данный метод показан в листинге 2.2.

|  |
| --- |
| static void CalculateParities(int[,] matrix)  {  int[] vertParitys = new int[matrix.GetLength(1)];  for (int j = 0; j < matrix.GetLength(1); j++)  {  for (int i = 0; i < matrix.GetLength(0); i++)  {  vertParitys[j] += matrix[i, j];  }  vertParitys[j] %= 2;  }  Console.WriteLine("Вертикальные паритеты:");  for (int j = 0; j < vertParitys.Length; j++)  {  Console.Write(vertParitys[j] + " ");  }  Console.WriteLine();  int[] horizontParitys = new int[matrix.GetLength(0)];  for (int i = 0; i < matrix.GetLength(0); i++)  {  for (int j = 0; j < matrix.GetLength(1); j++)  {  horizontParitys[i] += matrix[i, j];  }  horizontParitys[i] %= 2;  }  Console.WriteLine("Горизонтальные паритеты:");  for (int i = 0; i < horizontParitys.Length; i++)  {  Console.Write(horizontParitys[i] + " ");  }  Console.WriteLine();  } |

Листинг 2.2 – Нахождение групп паритетов

Полученное сообщение с паритетами записывается в новую матрицу. Запись происходит по строкам, в то время как передача информации происходит по столбцам.

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.

Для выполнения этого задания были созданы функции для исправления ошибок. Было сгенерировано случайное местоположение пакета ошибок каждой из трех заданных длин (4, 5, 6) для 30 случайных ситуаций. Для каждой ситуации выполнялись операции перемежения, деперемежения и исправления ошибок.

Для генерации данных групп (пакетов) ошибок используется программная реализация, показанная в листинге 2.3.

|  |
| --- |
| static void GenerateErrorPackets(int[,] matrix, int packetLength)  {  Random random = new Random();  int errorCount = packetLength switch  {  3 => 3,  5 => 5,  6 => 6,  \_ => throw new ArgumentException("Неподдерживаемая длина пакета"),  };  int startIndex = random.Next(0, matrix.GetLength(0) - packetLength + 1);  StringBuilder errorString = new StringBuilder();  for (int i = 0; i < packetLength; i++)  {  if (i < errorCount)  errorString.Append(random.Next(2));  else  errorString.Append(matrix[startIndex + i, 0]);  }  for (int i = 0; i < packetLength; i++)  {  matrix[startIndex + i, 0] = int.Parse(errorString[i].ToString());  }  Console.WriteLine($"Сообщение с пакетом ошибок длиной {packetLength}:");  for (int i = 0; i < matrix.GetLength(0); i++)  {  for (int j = 0; j < matrix.GetLength(1); j++)  {  Console.Write(matrix[i, j] + " ");  }  Console.WriteLine();  }  Console.WriteLine("Строка с ошибками:");  Console.WriteLine(MatrixToString(matrix));  string messageWithError = MatrixToString(matrix);  string[] blocksWithError = SplitIntoBlocks(messageWithError, 7);  for (int b = 0; b < blocksWithError.Length; b++)  {  int[,] blockMatrixWithError = new int[3, 2];  FillMatrixFromBlock(blockMatrixWithError, blocksWithError[b]);  CalculateParities(blockMatrixWithError);  }  } |

Листинг 2.3 – Генерация пакетов ошибок

Для исправления ошибок необходимо снова прочитать матрицу по строкам, затем считаем паритеты, используя метод из листинга 2.2. Посчитав паритеты можем сравнить их с паритетами, высчитанными в изначальной строке. Выяснив, где ошибки нужно отбросить избыточные биты, чтоб осталось исходное сообщение. На листинге 2.4 представлены методы, с помощью которых переводим матрицу в строки.

|  |
| --- |
| static string MatrixToString(int[,] matrix)   {       StringBuilder message = new StringBuilder();       for (int i = 0; i < matrix.GetLength(0); i++)       {           for (int j = 0; j < matrix.GetLength(1); j++)           {               message.Append(matrix[i, j]);           }       }       return message.ToString();   }   static string[] SplitIntoBlocks(string message, int blockSize)   {       int blockCount = message.Length / blockSize;       string[] blocks = new string[blockCount];       for (int b = 0; b < blockCount; b++)       {           blocks[b] = message.Substring(b \* blockSize, blockSize);       }       return blocks;   }     static void FillMatrixFromBlock(int[,] matrix, string block)   {       int k = 0;       for (int i = 0; i < matrix.GetLength(0); i++)       {           for (int j = 0; j < matrix.GetLength(1); j++)           {               matrix[i, j] = block[k++] - '0';               if (k >= block.Length)               {                   return;               }           }       }   } |

Листинг 2.4 – Запись матрицы и разбиение на блоки

В конце отбрасываем контрольные биты и получаем исходное сообщение.

Анализ эффективности метода перемежения/деперемежения данных в данном контексте позволит определить, насколько успешно данный подход справляется с различными видами ошибок и какие факторы могут повлиять на его эффективность. В результате такого исследования можно получить ценные данные о применимости этого метода в реальных ситуациях передачи и хранения информации.

Такой анализ поможет выявить потенциальные ограничения и недостатки метода перемежения/деперемежения данных, а также предложить возможные улучшения или альтернативные подходы для обеспечения более надежной передачи и хранения информации.

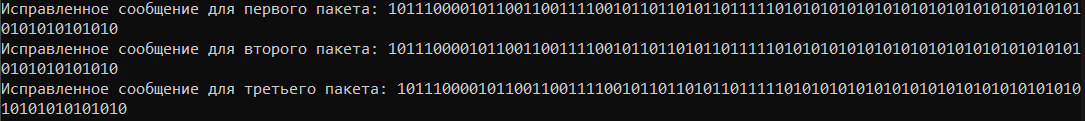
В Приложении А будет представлено несколько ситуаций, чтобы подробно можно было изучить метод перемежения/деперемежения. Исходя из полученной информации можно заметить, что для пакета длиной 3 ошибки исправляются в 80% случаях. В остальных ситуациях ошибки могут проявляться в конечном сообщении.

**Вывод:**

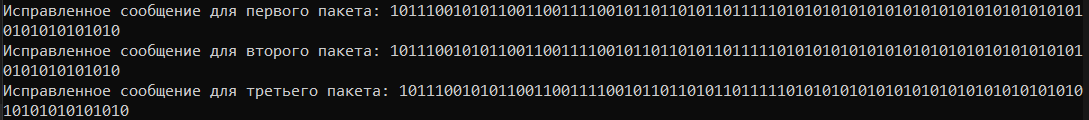
Исследование показывает, что использование метода перемежения/деперемежения данных является эффективным средством для устранения групповых ошибок в каналах передачи и устройствах хранения информации. Вместо использования специализированных кодов коррекции пакетных ошибок, чаще применяется комбинация перемежения/деперемежения с традиционными кодами. Метод блочного перемежения, описанный в тексте, представляет собой один из наиболее простых и эффективных подходов. Но одним из самых главных минусов перемежения и деперемежения является скорость выполнения операции. За счет большой размерности скорость решения заметно замедляется.

**Приложение А**

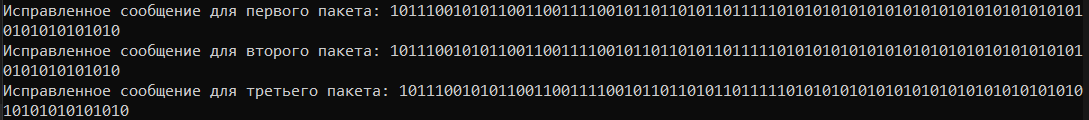
****

****

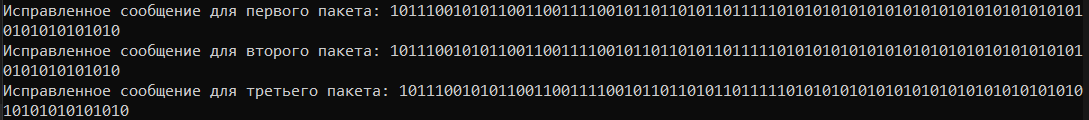
****

****

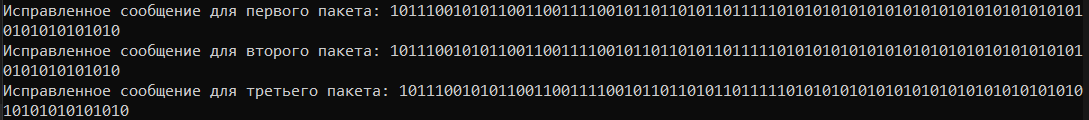
****

****

****

****

****

****