МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность Информационные системы и технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7 НА ТЕМУ:**

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

Ф.И.О.

Трофимчук Михаил Витальевич

Минск 2022

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

**Теоретические сведения**

Проанализированные и исследованные нами коды, как и большинство других кодов, которые были разработаны для увеличения надежности каналов передачи и хранения информации, наиболее эффективны, когда возникающие ошибки статистически независимы, т. е. вероятность передачи (хранения в памяти) любого символа одинакова. Однако 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; cм., например, [9−10, 19−22]).

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.

Рассмотренный метод блочного перемежения применяется в SM. К числу других используемых на практике относятся следующие методы перемежения/деперемежения: псевдослучайный, S-типа (применяется в турбо-кодировании: CDMA (Codе Division Multiple Access) – стандарт беспроводной связи множественного доступа с кодовым разделением каналов и др.); циклически-сдвиговый; сверточный; случайный; диагональный; многошаговый [5, 23, 24].

Перемежение (перестановка) символов также является основой некоторых классов криптографических методов [1], которые мы будем анализировать с практической точки зрения в другой части курса.

**Ход работы**

Разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется  
блочный перемежитель/деперемежитель.

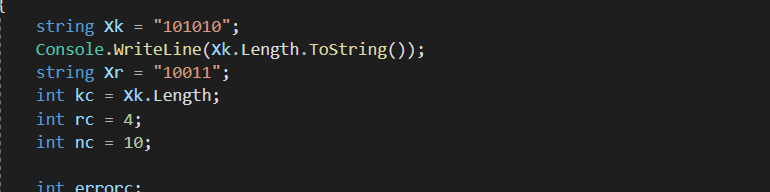
****

Рисунок 1 – Параметры кода

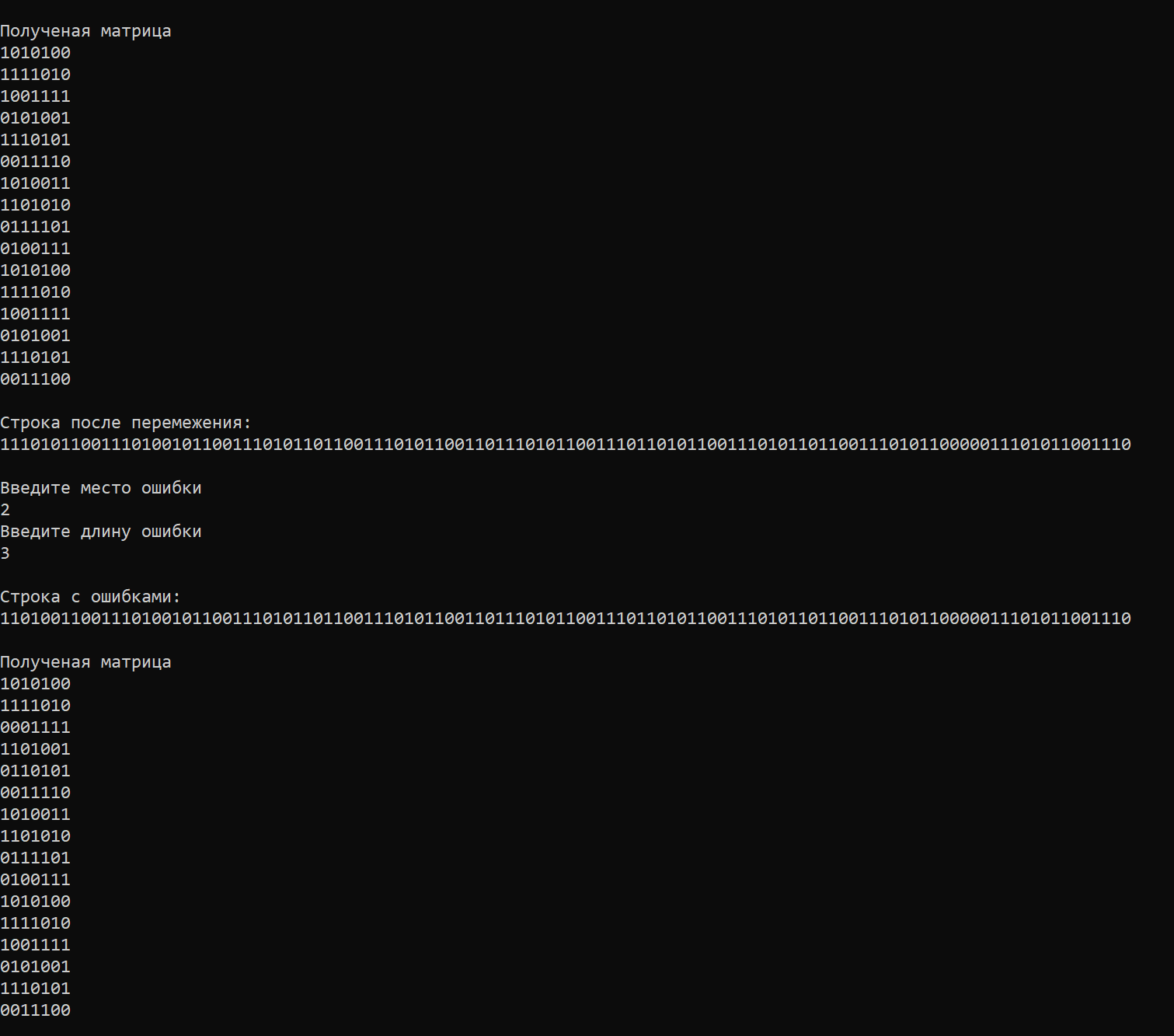


Рисунок 2 – Результат перемежения/деперемежения

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  namespace SevenLab  {  public static class Extensions  {  public static IEnumerable<IEnumerable<T>> Split<T>(this T[] arr, int size)  {  return arr.Select((s, i) => arr.Skip(i \* size).Take(size)).Where(a => a.Any());  }  }  class Program  {  static void Main(string[] args)  {  string Xk = "101010";  Console.WriteLine(Xk.Length.ToString());  string Xr = "10011";  int kc = Xk.Length;  int rc = 4;  int nc = 10;  int errorc;  int[] masXk = new int[kc];  StrInMas(masXk, Xk);  int[] masXr = new int[Xr.Length];  StrInMas(masXr, Xr);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  Console.WriteLine("Входная строка: " + Xk);  Console.WriteLine("Порождающий полином: " + Xr);  Console.WriteLine("k = {0}, r = {1}, n = {2}", kc, rc, nc);  Console.WriteLine("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");  int[,] generationMatrix = new int[kc, nc];  CreateGenerationMatrix(generationMatrix, masXr, kc, nc);  Console.WriteLine("\nПорождающая матрица");  OutMatrix(generationMatrix, kc, nc);  CreateCanonicalMatrix(generationMatrix, kc, nc);  Console.WriteLine("\nКаноническая матрица");  OutMatrix(generationMatrix, kc, nc);  int[,] checkMatrixc = new int[nc, rc];  CreateCheckMatrix(checkMatrixc, generationMatrix, kc, nc);  Console.WriteLine("\nПроверочная матрица");  OutMatrix(checkMatrixc, nc, rc);  //6.2  int[] masXn = new int[nc];  Shift(masXn, masXk, rc);  //2.  Console.WriteLine("\nДеление");  SearchResidue(masXn, masXr);  Console.WriteLine("Остаток (S-синдром):");  OutMass(masXn);  Console.WriteLine("\n");  Console.WriteLine("Итоговая строка:");  Shift(masXn, masXk, rc);  OutMass(masXn);  Console.WriteLine();  int[] save = new int[nc];  int prop = 0;  foreach (int menuitem in masXn)  {  save[prop++] = menuitem;  }  int altero = 0;  int[] alter = new int[110];  for (int i = 0; i < 11; i++)  {  foreach (var item in masXn)  {  alter[altero++] = item;  }  }  List<int> list = alter.ToList<int>();  list.Add(0);  list.Add(0);  // You can convert it back to an array if you would like to  masXn = list.ToArray();  Alternation(masXn);  Console.WriteLine("\nСтрока после перемежения: ");  OutMas(masXn);  int error;  int errorLenght;  try  {  Console.WriteLine("\n\nВведите место ошибки");  error = Convert.ToInt32(Console.ReadLine());  Console.WriteLine("Введите длину ошибки");  errorLenght = Convert.ToInt32(Console.ReadLine());  for (int i = error; i < (error + errorLenght); i++)  {  masXn[i] = (masXn[i] + 1) % 2;  }  }  catch { }  Console.WriteLine("\nСтрока с ошибками: ");  OutMas(masXn);    ReAlternation(masXn);  Console.WriteLine("\nСтрока после re:перемежения: ");  List<int> listt = new List<int>(masXn);  listt.RemoveAt(masXn.Length - 1);  masXn = listt.ToArray();  List<int> listt2 = new List<int>(masXn);  listt2.RemoveAt(masXn.Length - 1);  masXn = listt2.ToArray();  OutMas(masXn);  int jojo = 0;  int[] errorw = new int[110];  foreach (var item in masXn)  {  errorw[jojo++] = item;  }  //var nChunks = 11;  //var totalLength = masXn.Count();  //var chunkLength = (int)Math.Ceiling(totalLength / (double)nChunks);  //var parts = Enumerable.Range(0, 10)  // .Select(i => masXn.Skip(i \* chunkLength).Take(chunkLength));  int size = 10;  var arrays = masXn.Split(size);  Console.WriteLine();    foreach (var array in arrays)  {  Console.Write(String.Join("", array));  }    int[] itog = new int[110];  int ite = 0;  #region  int[] m1 = new int[10];  int[] m2 = new int[10];  int[] m3 = new int[10];  int[] m4 = new int[10];  int[] m5 = new int[10];  int[] m6 = new int[10];  int[] m7 = new int[10];  int[] m8 = new int[10];  int[] m9 = new int[10];  int[] m10 = new int[10];  int[] m11 = new int[10];  m1 = arrays.ElementAt(0).ToArray();  m2 = arrays.ElementAt(1).ToArray();  m3 = arrays.ElementAt(2).ToArray();  m4 = arrays.ElementAt(3).ToArray();  m5 = arrays.ElementAt(4).ToArray();  m6 = arrays.ElementAt(5).ToArray();  m7 = arrays.ElementAt(6).ToArray();  m8 = arrays.ElementAt(7).ToArray();  m9 = arrays.ElementAt(8).ToArray();  m10 = arrays.ElementAt(9).ToArray();  m11 = arrays.ElementAt(10).ToArray();  Console.WriteLine("Попааааааааа");  foreach (int item in m1)  {  Console.Write(item);  }  foreach (int item in m2)  {  Console.Write(item);  }  foreach (int item in m3)  {  Console.Write(item);  }  foreach (int item in m4)  {  Console.Write(item);  }  foreach (int item in m5)  {  Console.Write(item);  }  foreach (int item in m6)  {  Console.Write(item);  }  foreach (int item in m7)  {  Console.Write(item);  }  foreach (int item in m8)  {  Console.Write(item);  }  foreach (int item in m9)  {  Console.Write(item);  }  foreach (int item in m10)  {  Console.Write(item);  }  foreach (int item in m11)  {  Console.Write(item);  }  #endregion  Console.WriteLine("Попааааааааа");  int[] itogus = new int[10];  // SearchError(m1, masXk, checkMatrixc, rc);  //SearchError(m2, masXk, checkMatrixc, rc);  //SearchError(m3, masXk, checkMatrixc, rc);  //SearchError(m4, masXk, checkMatrixc, rc);  //SearchError(m5, masXk, checkMatrixc, rc);  //SearchError(m6, masXk, checkMatrixc, rc);  //SearchError(m7, masXk, checkMatrixc, rc);  //SearchError(m8, masXk, checkMatrixc, rc);  //SearchError(m9, masXk, checkMatrixc, rc);  //SearchError(m10, masXk, checkMatrixc, rc);  //SearchError(m11, masXk, checkMatrixc, rc);  itogus = SearchError(m1, masXk, checkMatrixc, rc)  .Concat(SearchError(m2, masXk, checkMatrixc, rc))  .Concat(SearchError(m3, masXk, checkMatrixc, rc))  .Concat(SearchError(m4, masXk, checkMatrixc, rc))  .Concat(SearchError(m5, masXk, checkMatrixc, rc))  .Concat(SearchError(m6, masXk, checkMatrixc, rc))  .Concat(SearchError(m7, masXk, checkMatrixc, rc))  .Concat(SearchError(m8, masXk, checkMatrixc, rc))  .Concat(SearchError(m9, masXk, checkMatrixc, rc))  .Concat(SearchError(m10, masXk, checkMatrixc, rc))  .Concat(SearchError(m11, masXk, checkMatrixc, rc))  .ToArray();  Console.WriteLine();  Console.WriteLine("Итооог");  foreach (var item in itogus)  {  Console.Write(item.ToString());  }  Console.WriteLine();  foreach (var item in errorw)  {  Console.Write(item);  }  Console.WriteLine("Итооог");  Console.WriteLine(); Console.WriteLine();  foreach (var item in itogus)  {  Console.Write(item.ToString());  }  //foreach (var item in parts.ElementAt(0))  //{  // first[ite++] = item;  // //itog[u++] = item;  //}  //Console.WriteLine("\nfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");  //foreach (var item in first)  //{  // Console.Write( item.ToString());  //}  ////SearchError(first, masXk, checkMatrixc, rc);  //int opp = 0;  //foreach (var item in parts.ElementAt(1))  //{  // second[opp++] = item;  // //itog[u++] = item;  //}  //Console.WriteLine("\nfffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff");  //foreach (var item in second)  //{  // Console.Write(item.ToString());  //}  // SearchError(secondso, masXk, checkMatrixc, rc);  // SearchError(second, masXk, checkMatrixc, rc);  //Array.Clear(first, 0, first.Length);  //ite = 0;  // SearchError(first, masXk, checkMatrixc, rc);  //SearchError(masXn, masXk, checkMatrixc, rc);  Console.WriteLine("\n\nСтрока после исправления ошибок: ");  OutMas(save);  //RemoveCheckBits(masK2, masXn, checkMatrixc);  Console.WriteLine("\n\nСтрока после удаления проверочных бит: ");  Console.WriteLine(Xk);  Console.WriteLine("");  OutMassive(Xk);    }    static int[] Alternation(int[] masN)  {  int[,] matrix = new int[16, 7];  //Получение матрицы  for (int i = 0, m = 0; i < 16; i++)  {  for (int j = 0; j < 7; j++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, 16, 7);  //Перемежение  for (int i = 0, m = 0; i < 7; i++)  {  for (int j = 0; j < 16; j++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }  static int[] ReAlternation(int[] masN)  {  //int r = HemmingLength(k);  int k = 16;  int n = 7;  int[,] matrix = new int[k, n];  //Получение матрицы  for (int j = 0, m = 0; j < n; j++)  {  for (int i = 0; i < k; i++, m++)  {  matrix[i, j] = masN[m];  }  }  Console.WriteLine("\n\nПолученая матрица");  OutMatrix(matrix, k, n);  //RE:Перемежение  for (int j = 0, m = 0; j < k; j++)  {  for (int i = 0; i < n; i++, m++)  {  masN[m] = matrix[j, i];  }  }  return masN;  }      //Поиск синдрома  static int[] Sindrom(int[,] CheckMatrix, int[] mas, int k)  {  int r = HemmingLength(k);  int n = r + k;  int[] sindrom = new int[r];  for (int i = 0, l = 0; i < r; i++, l = 0)  {  for (int j = 0; j < k; j++)  {  if (CheckMatrix[j, i] == 1 && mas[j] == 1) l++;  else sindrom[i] = 0;  }  if (l % 2 == 1) sindrom[i] = 1;  else sindrom[i] = 0;  }  for (int i = 0; i < r; i++)  {  mas[i + k] = sindrom[i];  }  return mas;  }  //Считаем r (кол-во пров. симв.)  static int HemmingLength(int k)  {  int r = (int)(Math.Log(k, 2) + 1.99f);  return r;  }    static void OutMas(int[] mas)  {  for (int i = 0; i < mas.Length; i++)  {  Console.Write(mas[i]);  }  }  //вывод матрицы    public static int[] SearchError(int[] masXn, int[] masXr, int[,] checkMatrix, int r)  {  int n = masXn.Length;  int k = n - r;  int[] masXnSecond = new int[n];  for (int i = 0; i < n; i++)  {  masXnSecond[i] = masXn[i];  }  Console.WriteLine("\nДеление");  SearchResidue(masXnSecond, masXr);  Console.WriteLine();  Console.WriteLine("\n");  Console.WriteLine("\nОстаток:");  OutMass(masXnSecond);  for (int i = 0; i < n; i++)  {  int coincidence = 0;  for (int j = 0; j < r; j++)  {  if (checkMatrix[i, j] == masXnSecond[k + j])  {  coincidence++;  }  }  if (coincidence == r)  {  masXn[i] = (masXn[i] + 1) % 2;  break;  }  }  Console.WriteLine("\nИсправленная строка:");  OutMass(masXn);  return masXn;  }  public static int[] SearchResidue(int[] masXn, int[] masXr)  {  int end = masXn.Length - masXr.Length + 1;  for (int i = 0; i < end; i++)  {  if (masXn[i] == 1)  {  AddingMasMod2(masXn, masXr, i);  OutMass(masXn);  }  }  Console.WriteLine("\n");  return masXn;  }  //Сложение массивов по модулю 2 с опр. позиции  public static int[] AddingMasMod2(int[] mas1, int[] mas2, int pos)  {  int end = pos + mas2.Length;  for (int i = pos; i < end; i++)  {  mas1[i] = (mas1[i] + mas2[i - pos]) % 2;  }  return mas1;  }  //Смещение на массива r  public static int[] Shift(int[] shiftMas, int[] mas, int r)  {  for (int i = 0; i < mas.Length; i++)  {  shiftMas[i] = mas[i];  }  return shiftMas;  }  //Преобразование сторки в массив  public static int[] StrInMas(int[] mas, string str)  {  for (int i = 0; i < str.Length; i++)  {  if (str[i] == 49)  mas[i] = 1;  else mas[i] = 0;  }  return mas;  }  //Создание Порождающей матрицы  static int[,] CreateGenerationMatrix(int[,] generationMatrix, int[] mas, int k, int n)  {  //Заполняем первую строку в проверочной матрице  for (int i = 0; i < n; i++)  {  if (i < mas.Length)  {  generationMatrix[0, i] = mas[i];  }  else  {  generationMatrix[0, i] = 0;  }  }  //Сдвигаем каждую строки вправо от предыдущей  for (int i = 1; i < k; i++)  {  for (int j = 0; j < n - 1; j++)  {  generationMatrix[i, j + 1] = generationMatrix[i - 1, j];  }  generationMatrix[i, 0] = generationMatrix[i - 1, n - 1];  }  return generationMatrix;  }  //Приведение порождающей матрицы к каноническому виду  static int[,] CreateCanonicalMatrix(int[,] generationMatrix, int k, int n)  {  //Перебираем строки для преведению к каноническому виду  for (int i = 0; i < k; i++)  {  int i2 = i + 1;  //Перебираем элементы строки, но только до k-элемента  for (int j = i + 1; j < k; j++)  {  //если мы нашли единицу в строке, то...  if (generationMatrix[i, j] == 1)  {  //перебираем этот столбец, пока не найдем единицу  for (; i2 < k; i2++)  {  bool repeat = false;  //Если нашли, то складываем обе строки  if (generationMatrix[i2, j] == 1)  {  for (int j2 = j - 1; j2 > 0; j2--)  {  //Проверяем, есть ли до этой 1 еще 1, если есть то эту строку пропускаем  if (generationMatrix[i2, j2] == 1)  {  repeat = true;  }  }  if (repeat)  continue;  Console.WriteLine(i + " " + i2);  AddingLinesMatrixMod2(generationMatrix, i, i2, n);  i2++;  break;  }  }  }  }  }  return generationMatrix;  }  //Преобразование канонической матрицы в проверочную  static int[,] CreateCheckMatrix(int[,] checkMatrix, int[,] generationMatrix, int k, int n)  {  int r = n - k;  for (int i = 0; i < k; i++)  {  for (int j = 0; j < r; j++)  {  checkMatrix[i, j] = generationMatrix[i, k + j];  }  }  for (int i = k; i < n; i++)  {  for (int j = 0; j < r; j++)  {  if (j == i - k)  {  checkMatrix[i, j] = 1;  }  else  {  checkMatrix[i, j] = 0;  }  }  }  return checkMatrix;  }  //Сложение строк матрицы  public static int[,] AddingLinesMatrixMod2(int[,] matrix, int str1, int str2, int lengthString)  {  //Console.WriteLine(str1 + " и " + str2);  for (int i = 0; i < lengthString; i++)  {  matrix[str1, i] = (matrix[str1, i] + matrix[str2, i]) % 2;  }  return matrix;  }  //вывод матрицы  public static void OutMatrix(int[,] matrix, int k, int n)  {  for (int i = 0; i < k; i++)  {  for (int j = 0; j < n; j++)  {  Console.Write(matrix[i, j]);  //if (j + 1 == k) Console.Write("|");  }  Console.WriteLine();  }  }  static void OutMassive(string Xk)  {  Console.WriteLine(Xk);  }  //вывод одномерного массива  public static void OutMass(int[] mas)  {  Console.WriteLine();  for (int i = 0; i < mas.Length; i++)  {  //if (i == k) Console.Write("|");  Console.Write(mas[i]);  }  //Console.WriteLine("\n");  }  }  } |

Листинг 1 – Код приложения

Сравнение полученной последовательности после исправления ошибок с передаваемой последовательностью:

Ситуация с группами с тремя ошибками:

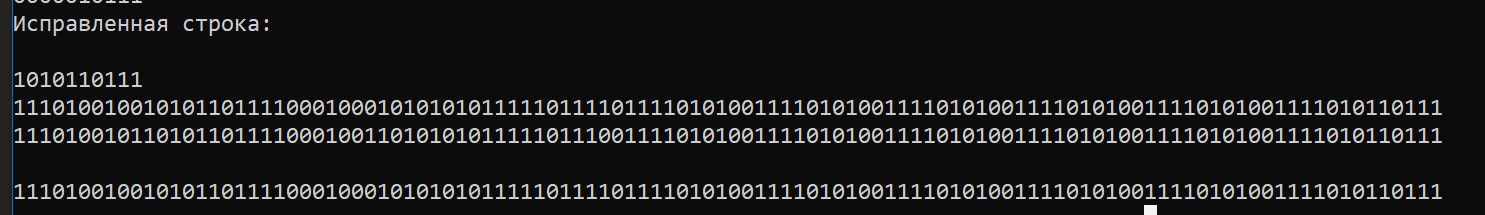


Рисунок 3 – Сравнение с 3 ошибками

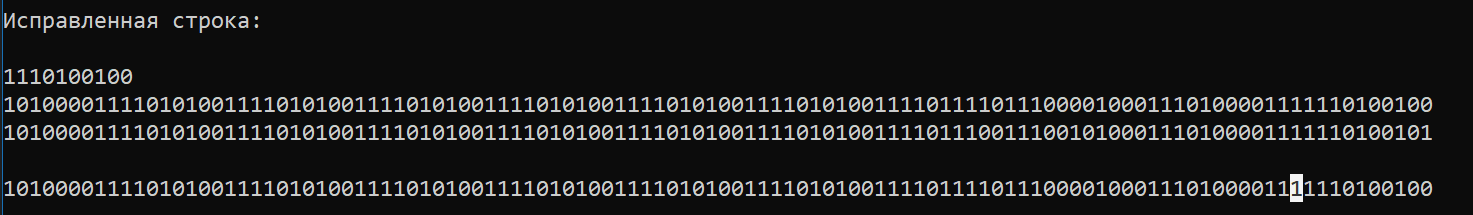


Рисунок 4 – Сравнение с 3 ошибками

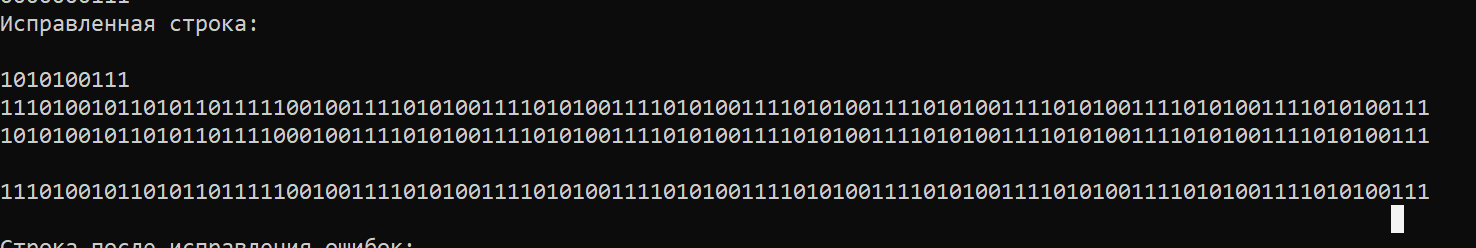


Рисунок 5 – Сравнение с 3 ошибками

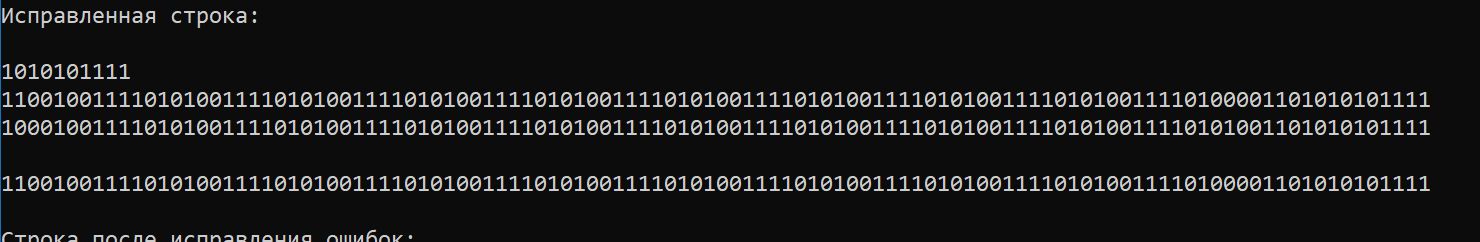
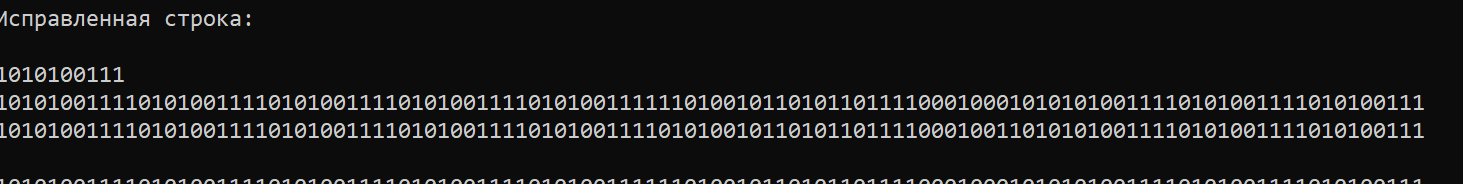


Рисунок 6 – Сравнение с 3 ошибками



Ситуация с группами с пятью ошибками:

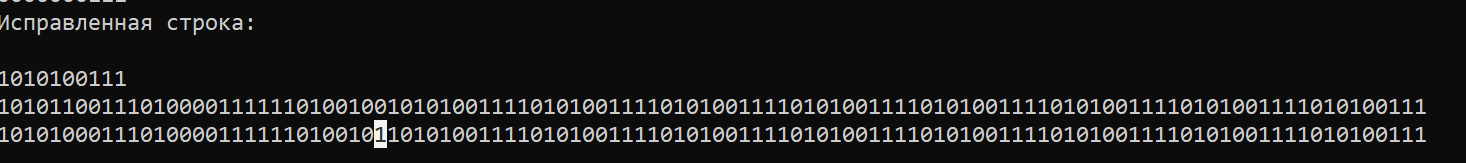


Рисунок 7 – Сравнение с 5 ошибками

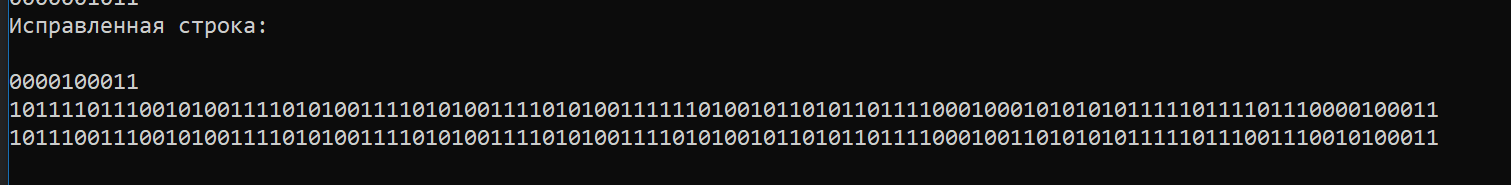


Рисунок 8 – Сравнение с 5 ошибками

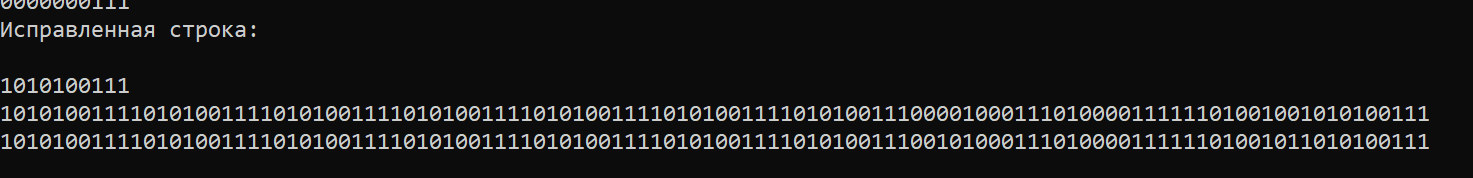


Рисунок 9 – Сравнение с 5 ошибками

Ситуация с группами с семью ошибками:

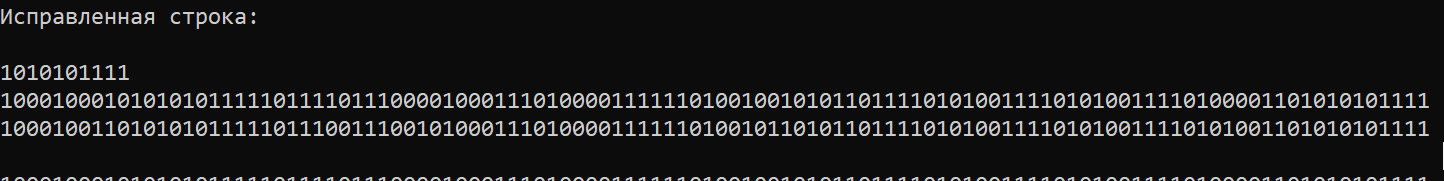


Рисунок 10 – Сравнение с 7 ошибками

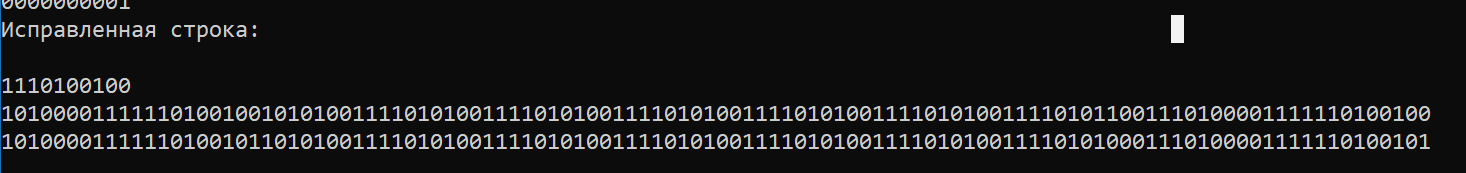


Рисунок 11 – Сравнение с 7 ошибками

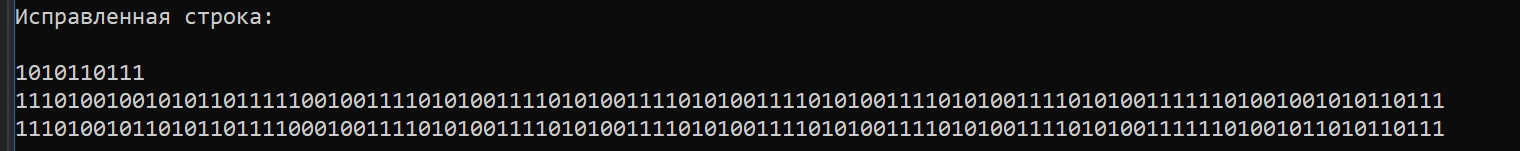


Рисунок 12 – Сравнение с 7 ошибками

**Вывод**: в данной работе было рассмотрено перемежение и деперемежение как варианты кодирования информации. Было установлено, что такой способ должен использоваться в совокупности с другими корректирующими кодами, например Циклическим кодом. Перемежение/деперемежение позволяет разбить исходную строку, на строки нужной длины корректирующего кода, так чтобы можно было исправить как можно больше ошибок.