МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра САПР

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Студент гр. 2301 _____ Комиссаров П.Е.

Преподаватель: _____ Пестерев Д.О.

Санкт-Петербург
2024

1. Экспериментальная часть с результатами исследования эффективности различных компрессоров

На рисунках 1,2 представлены таблицы, в которых приведены результаты сжатия различных файлов с помощью разных видов компрессоров.

Сжимались файлы: фотографии в row-формате (черно-белое фото, фото в оттенках серого, цветное изображение), текст на русском языке и текст enwik7

Все изображения имели одинаковое разрешение, а именно 600x600 пикселей

	image_bw	image_grey	image_color (rgb)	russian text	enwik7
Huffman algorithm (HA)	0,092419712	0,709375305	0,577338234	0,319616839	0,5508096
Arithmetic coding (AC)	0,0913	0,832122	0,77551233	0,55223	0,891123
Run-length encoding (RLE)	0,035442968	1,007761345	1,045439477	1,013273607	1.0837639
BWT + RLE	0,001011905	1,003012388	1.011234432	0,735932534	0,737342
BWT + MTF + HA	0,008305648	0,087155248	0,5307296	0,324722941	0,3114632
BWT + MTF + AC	-	-	-	-	-
BWT + MTF + RLE + HA	0,008305648	0,087155248	0,5307296	0,360737216	0,31278666
BWT + MTF + RLE + AC	-	-	-	-	-
LZ77	0,018516058	1,203676905	0,95492933	0,773670645	0,9864998
LZ77 + HA	0,092419712	0,354141897	0,577338234	0,296213553	0,534648
LZ77 + AC	0,02223	0,67645566	0,991233	0,66304	0,6743805

Рисунок 1 – Таблица коэффициента сжатия

На рисунке 1 представлена таблица, в которой приведены коэффициенты сжатия файлов с помощью различных видов компрессоров. Значения были получены путем деления размеров сжатого файла на исходный.

Обозначение цветов:

0,087155248 - Лучший коэффициент сжатия для каждого вида файлов

1,003012388 - Плохое сжатие

BWT + MTF + HA - Лучший алгоритм сжатия/ Хороший коэффициент сжатия

0,991233 - Файл сжимался

\- - Алгоритм не готов

На рисунке 2 представлена таблица, в которой приведены разница в размерах исходного и сжатого файлов. Значения в таблице соответствуют – (размер исходного файла/размер сжатого файла)

	image_bw	image_grey	image_color (rgb)	russian text	enwik7
Huffman algorithm (HA)	353/33	401/285	1208/698	2852/911	9766/5379
Arithmetic coding (AC)	353/34	401/227	1208/937	2852/1550	9766/8702
Run-length encoding (RLE)	353/13	401/759	1208/1263	2852/2890	9766/10584
BWT + RLE	353/1	401/403	1208/1211	2852/2099	9766/7201
BWT + MTF + HA	353/3	401/35	1208/642	2852/927	9766/3042
BWT + MTF + AC	-	-	-	-	-
BWT + MTF + RLE + HA	353/3	401/35	1208/642	2852/1029	9766/3042
BWT + MTF + RLE + AC	-	-	-	-	-
LZ77	353/25	401/450	1208/1153	2852/1997	9766/9634
LZ77 + HA	353/8	401/175	1208/762	2852/844	9766/6176
LZ77 + AC	353/7	401/271	1208/1200	2852/1891	9766/6586

Рисунок 2 – Таблица изменения размеров файлов

Из таблиц можно заметить, что самыми эффективным компрессорами являются гибридные компрессоры BWT-MTF-HA и BWT-MTF-RLE-HA.

– Это возникает из-за того, что BWT+MTF обеспечивают текст большими повторами символов. А HA+RLE успешно сжимают повторяющиеся символы.

– Лишь, по теории, алгоритм BWT-MTF-RLE-HA должен работать быстрее, чем BWT-MTF-HA. Но на практике этого доказать не удалось. Возможно нужно провести больше тестов на разных файлах.

Из негибридных компрессоров выделяются Алгоритм Хаффмана, Арифметическое кодирование, которые успешно сжимает все представленные файлы.

Худший показатель сжатия получен у RLE, сжать получилось лишь черно-белое изображение.

– Это возникает из-за того, что RLE ищет на своем пути повторяющиеся подряд символы. Этот метод актуален лишь для чб фото (так как там используется только 2 цвета). Все остальные файлы сохранили практически исходный размер

Также посредственный показатель сжатия получился у LZ77. Этот алгоритм становится одним из самых эффективных в совокупности с алгоритмом Хаффмана

Также в таблицах отсутствуют данные о работе двух компрессоров, в основе которых лежит Арифметическое кодирование

Это вызвано тем, что алгоритм плохо работает с длинными строками текста (из-за уязвимости в арифметическом кодировании).

Арифметическое кодирование текста производится на питоне с помощью типа данных decimal (с точность вычислений 100 знаков после запятой). Числа на выходе зачастую получаются длиннее самого текстового отрезка, поэтому кодирование становится неэффективным. Иной реализации я пока что не придумал.

1.1 Графики

Рассмотрим зависимость коэффициента сжатия от размера буфера алгоритма LZ77. Проверка будет совершенная на текстовом файле с русским тестом (таким же, как в таблице)

Полученные данные представлены на Рисунке 3

bufer	300	400	500	600	700	800	900	1000	1100	1200
ratio	0.956463	0.94779	0.94263	0.93936	0.9372	0.93587	0.93481	0.9340	0.93377	0.93354

Рисунок 3 – таблица зависимости сжатия от размера буфера LZ77

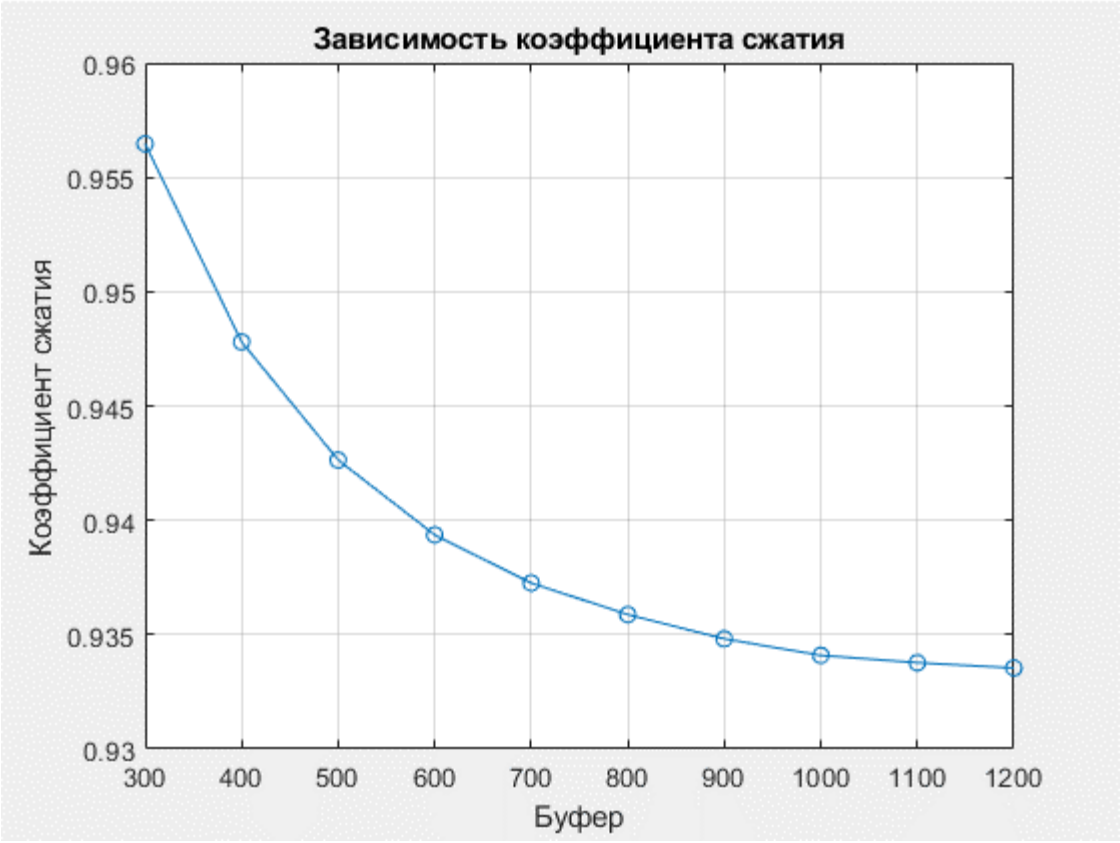

Рисунок 4 – График зависимости размера сжатия файла от размера буфера

Из графика видно, что сжатие становится более эффективным с увеличением буфера для RLE. Но с увеличением буфера, коэффициент сжатия меняется все меньше. Можно сделать вывод, что в какой-то момент времени размер буфера не будет влиять на изменение размера файла.

Но сжатие должно быть более качественное, чем приведено на графике и таблице. Скорее всего в коде происходит неэффективная кодировка.

Также исследуем зависимость коэффициента сжатия от размера строки, подаваемой на вход BWT.

Полученные данные представлены на рисунке 5

length	1000	5000	15000	30000	50000	75000	100000	125000	150000
ratio	0.998	0.98251	0.97084	0.961789	0.962726	0.95073	0.95427	0.955972	0.94557

Рисунок 5 - Зависимость коэффициента сжатия от размера строки

Рисунок 6 – График зависимости коэффициента сжатия от размера строки

Из графика видно, что с увеличением длины подаваемого файла, коэффициент сжатия файла с помощью BWT становится лучше.

2. Весь написанный код

2.1. Арифметическое кодирование

```
  using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace arifmetic
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }
   internal class Program
   {
      static double arifm_code(string str, double[] percent, char[] chars, out int length_error)            //ПЕРЕМЕННАЯ ДЛЯ ОШИБКИ ERROR
      {
         double[] intervals = new double[percent.Length + 1];

         double left_border = 0;            //границы
         double right_border = 1;

         double interval = 0;          //переменная для создания интервалов

         for (int i = 1; i < percent.Length + 1; i++)
         {
            interval += percent[i - 1];
            intervals[i] = interval;              //создание переменной интервалов
         }

         char to_find = ' ';               //кодируемый символ
         int index = 0;

         int j = 0;
         length_error = 0;

         foreach (char ch_find in str)
         {
            j++;
            double part_length = right_border - left_border;           //длина интервала
            to_find = ch_find;                                 //символ, который необходимо найти
            index = Array.IndexOf(chars, to_find);
            left_border += intervals[index] * part_length;             //новые границы
            right_border = left_border + percent[index] * part_length;
            if (left_border == right_border)                                          //ЕСЛИ ГРАНИЦЫ СОВПАЛИ, ТО КОДИРОВКА
ПРЕРЫВАЕТСЯ
            {
               length_error = j;                                                  //ВОЗВРАЩАЕТ ИНДЕКС, ГДЕ ПРЕРВАЛАСЬ КОДИРОВКА
               break;
            }
         }

         double result = (left_border + right_border) / 2;
         return result;
      }
```

```
static void text_alphabet_from_line(out char[] chars, out double[] percent, string line)
{
    int count = 0;
    int count_symbols = 0;

    List<char> alphabet = new List<char>();

    foreach (char ch in line)               //создание алфавита
    {
        for (int i = 0; i < alphabet.Count; i++)
            if (ch != alphabet[i])
                count++;

        if (count == alphabet.Count)       //если символ не встретился за прогон, добавляем
            alphabet.Add(ch);
        count = 0;

    }
    chars = new char[alphabet.Count];
    for (int i = 0; i < alphabet.Count; i++)        //переписываем алфавит в массив
        chars[i] = alphabet[i];
    Array.Sort(chars);                              //сортируем массив

    double[] appearance = new double[alphabet.Count];
    percent = new double[alphabet.Count];

    foreach (char ch in line)               //создание массива повторок
        for (int i = 0; i < alphabet.Count; i++)
            if (ch == alphabet[i])
            {
                appearance[i] += 1;
                count_symbols += 1;
                break;
            }
    for (int i = 0; i < alphabet.Count; i++)
        percent[i] += appearance[i] / count_symbols;        //считаем процент появления
}
static void Main(string[] args)
{
    string text_russian = "rusTolstoy.txt";
    string text_russian_out = "rusTolstoy_out.txt";

    string text_enwik7 = "enwik7.txt";
    string text_enwik7_out = "enwik7_out.txt";

    string text_gray = "Gray_image.txt";
    string text_gray_out = "Gray_image_out.txt";

    string text_bw = "BW_image.txt";
    string text_bw_out = "BW_image_out.txt";

    string text_color = "Color_image.txt";
    string text_color_out = "Color_image_out.txt";
    string lines;

    int error = 0;
    char[] chars = new char[1];                 //массив символов
    double result = 0;
    double[] percent = new double[1];               //массив вероятностей
    List<int> length = new List<int>();
    string text_result = "";


    Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
    using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))   //чтение построчно
    using (StreamWriter writer = new StreamWriter(text_russian_out))
        while ((lines = reader.ReadLine()) != null)
            while (lines.Length != 0)
                if (lines.Length > 0)                                                   //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                {
                    text_alphabet_from_line(out chars, out percent, lines);
                    int step = Math.Min(lines.Length, 15);
```

```
                result = arifm_code(lines.Substring(0, step), percent, chars, out error);                    //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ СТРОКЕ
                if (error != 0)                                                    //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                {
                   int nothing;                                                    //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ БУДЕТ)
                   text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));    //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                   result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                   text_result = result.ToString("F20");
                   text_result = text_result.Substring(2, text_result.Length - 2);
                   text_result = text_result.TrimEnd('0');
                   long number = long.Parse(text_result);                          // Преобразование строки в целое число
                   text_result = number.ToString("X");
                   writer.Write(text_result);
                   text_result = "";
                   lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";              //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                }
                else                                                               //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                {
                   text_result = result.ToString("F20");
                   text_result = text_result.Substring(2, text_result.Length - 2);
                   text_result = text_result.TrimEnd('0');
                   long number = long.Parse(text_result);                          // Преобразование строки в целое число
                   text_result = number.ToString("X");
                   writer.Write(text_result);
                   text_result = "";
                   lines = lines.Length > 15 ? lines.Substring(15) : "";
                }
             }
        result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");
        using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_enwik7_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                          //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                       text_alphabet_from_line(out chars, out percent, lines);
                       int step = Math.Min(lines.Length, 15);
                       result = arifm_code(lines.Substring(0, step), percent, chars, out error);                    //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ СТРОКЕ
                       if (error != 0)                                             //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                       {
                          int nothing;                                             //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ БУДЕТ)
                          text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));    //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                          result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                          text_result = result.ToString("F20");
                          text_result = text_result.Substring(2, text_result.Length - 2);
                          text_result = text_result.TrimEnd('0');
                          long number = long.Parse(text_result);                   // Преобразование строки в целое число
                          text_result = number.ToString("X");
                          writer.Write(text_result);

                          text_result = "";
                          lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";              //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                       }
                       else                                                        //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                       {
```

```
                    text_result = result.ToString("F20");
                    text_result = text_result.TrimEnd('0');
                    text_result = text_result.Substring(2, text_result.Length - 2);
                    long number = long.Parse(text_result);                    // Преобразование строки в целое число
                    text_result = number.ToString("X");
                    writer.Write(text_result);
                    lines = lines.Length > 15 ? lines.Substring(15) : "";
                    text_result = "";

                }
            }
        result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");









        Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");
        using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_gray_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                   //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                        text_alphabet_from_line(out chars, out percent, lines);
                        int step = Math.Min(lines.Length, 15);
                        result = arifm_code(lines.Substring(0, step), percent, chars, out error);          //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ СТРОКЕ
                        if (error != 0)                                     //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                        {
                            int nothing;                                    //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ БУДЕТ)
                            text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));    //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                            result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                    // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
                            text_result = "";
                            lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";             //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                        }
                        else                                                //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                        {
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                    // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
                            text_result = "";
                            lines = lines.Length > 15 ? lines.Substring(15) : "";
                        }
                    }
        result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");
```

```
Console.WriteLine("=============================================================================================================\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
        using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_bw_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                       //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                        text_alphabet_from_line(out chars, out percent, lines);
                        int step = Math.Min(lines.Length, 15);
                        result = arifm_code(lines.Substring(0, step), percent, chars, out error);                   //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ СТРОКЕ
                        if (error != 0)                                         //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                        {
                            int nothing;                                        //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ БУДЕТ)
                            text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));     //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                            result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                  // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
                            text_result = "";
                            lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";             //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                        }
                        else                                                    //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                        {
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                  // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
                            text_result = "";
                            lines = lines.Length > 15 ? lines.Substring(15) : "";
                        }
                    }
        result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
        using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_color_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                       //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                        text_alphabet_from_line(out chars, out percent, lines);
                        int step = Math.Min(lines.Length, 15);
                        result = arifm_code(lines.Substring(0, step), percent, chars, out error);                   //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ СТРОКЕ
                        if (error != 0)                                         //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                        {
```

```
                    int nothing;                                                    //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ
БУДЕТ)
                    text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));     //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО
СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                    result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                    text_result = result.ToString("F20");
                    text_result = text_result.Substring(2, text_result.Length - 2);
                    text_result = text_result.TrimEnd('0');
                    long number = long.Parse(text_result);                          // Преобразование строки в целое число
                    text_result = number.ToString("X");
                    writer.Write(text_result);
                    text_result = "";
                    lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";             //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА
ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                }
                else                                                        //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                {
                    text_result = result.ToString("F20");
                    text_result = text_result.Substring(2, text_result.Length - 2);
                    text_result = text_result.TrimEnd('0');
                    long number = long.Parse(text_result);                          // Преобразование строки в целое число
                    text_result = number.ToString("X");
                    writer.Write(text_result);
                    text_result = "";
                    lines = lines.Length > 15 ? lines.Substring(15) : "";
                }
            }
        result = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");


        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();

    }
  }
}
```

2.2 BWT+MTF+AC

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace bwt_mtf_ac
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }

   internal class Program
   {

      static double arifm_code(string str, double[] percent, char[] chars, out int length_error)            //ПЕРЕМЕННАЯ ДЛЯ ОШИБКИ ERROR
      {
         double[] intervals = new double[percent.Length + 1];

         double left_border = 0;            //границы
         double right_border = 1;

         double interval = 0;          //переменная для создания интервалов

         for (int i = 1; i < percent.Length + 1; i++)
         {
            interval += percent[i - 1];
            intervals[i] = interval;               //создание переменной интервалов
         }

         char to_find = ' ';               //кодируемый символ
         int index = 0;

         int j = 0;
         length_error = 0;

         foreach (char ch_find in str)
         {
            j++;
            double part_length = right_border - left_border;            //длина интервала
            to_find = ch_find;                                  //символ, который необходимо найти
            index = Array.IndexOf(chars, to_find);
            left_border += intervals[index] * part_length;              //новые границы
            right_border = left_border + percent[index] * part_length;
            if (left_border == right_border)                                          //ЕСЛИ ГРАНИЦЫ СОВПАЛИ, ТО КОДИРОВКА
ПРЕРЫВАЕТСЯ
            {
               length_error = j;                                                  //ВОЗВРАЩАЕТ ИНДЕКС, ГДЕ ПРЕРВАЛАСЬ КОДИРОВКА
               break;
            }
         }

         double result = (left_border + right_border) / 2;
         return result;
      }
      static void text_alphabet_from_line(out char[] chars, out double[] percent, string line)
```

```
{
    int count = 0;
    int count_symbols = 0;

    List<char> alphabet = new List<char>();

    foreach (char ch in line)               //создание алфавита
    {
        for (int i = 0; i < alphabet.Count; i++)
            if (ch != alphabet[i])
                count++;

        if (count == alphabet.Count)        //если символ не встретился за прогон, добавляем
            alphabet.Add(ch);
        count = 0;

    }
    chars = new char[alphabet.Count];
    for (int i = 0; i < alphabet.Count; i++)        //переписываем алфавит в массив
        chars[i] = alphabet[i];
    Array.Sort(chars);                              //сортируем массив

    double[] appearance = new double[alphabet.Count];
    percent = new double[alphabet.Count];

    foreach (char ch in line)               //создание массива повторок
        for (int i = 0; i < alphabet.Count; i++)
            if (ch == alphabet[i])
            {
                appearance[i] += 1;
                count_symbols += 1;
                break;
            }
    for (int i = 0; i < alphabet.Count; i++)
        percent[i] += appearance[i] / count_symbols;        //считаем процент появления
}


static List<char> alphabet_func(string str) //создание алфавита
{
    List<char> chars = new List<char>();
    for (int i = 0; i < str.Length; i++)
    {
        if (!chars.Contains(str[i]))        //добавляем уникальные буквы в алфавит
            chars.Add(str[i]);
    }
    chars.Sort((a, b) => a.CompareTo(b));   //сортировка
    return chars;
}

static string Move_to_Front(string str, List<char> alphabet)
{
    string result = "";
    int index;
    char symbol;
    char char_element;
    char removed_char;
    for (int i = 0; i < str.Length; i++)
    {
        symbol = str[i];                    //символ в строке
        index = alphabet.IndexOf(symbol);   //индекс символа в алфавите
        char_element = (char)(index + 40);          //индекс символа без служебных символов
        result += char_element.ToString();          //записываем символ в форматированную строку
        removed_char = alphabet[index];             //используемый символ в алфавите

        alphabet.RemoveAt(index);       //ставим символ в начало алфавита
        alphabet.Insert(0, removed_char);
    }
    return result;
}
static BitArray to_bit(BitArray resoult, string compressed)
{
    int index = 0;
```

```
            foreach (char chars in compressed)
            {
                if (chars == '1')
                    resoult[index] = true;
                else if (chars == '0')
                    resoult[index] = false;
                index++;
            }
            if (index < resoult.Length)
            {
                BitArray cut = new BitArray(index);         //обрезка лишней части строки
                for (int i = 0; i < index; i++)
                    cut[i] = resoult[i];
                resoult = cut;
            }
            return resoult;
        }

        static void Main(string[] args)
        {
            string text_enwik7 = "enwik7.txt";
            string text_enwik7_bwt = "enwik7_bwt.txt";
            string text_enwik7_out = "enwik7_out.txt";

            string text_gray = "Gray_image.txt";
            string text_gray_bwt = "Gray_image_bwt.txt";
            string text_gray_out = "Gray_image_out.txt";

            string text_bw = "BW_image.txt";
            string text_bw_bwt = "BW_image_bwt.txt";
            string text_bw_out = "BW_image_out.txt";

            string text_color = "Color_image.txt";
            string text_color_bwt = "Color_image_bwt.txt";
            string text_color_out = "Color_image_out.txt";


            string text_russian = "rusTolstoy.txt"; //"graph.txt"
            string text_russian_bwt = "rusTolstoy_bwt.txt";
            string text_russian_out = "rusTolstoy_out.txt";




            Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");

Console.WriteLine("=============================================================================================================
=====");
            Console.WriteLine("Начался BWT...");
            BWTFast bwt = new BWTFast();
            string text_in_line = "";
            string line = "";
            using (StreamWriter res = new StreamWriter(text_russian_bwt))
            {
                using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
                {
                    while ((line = reader.ReadLine()) != null)
                    {
                        text_in_line += line;
                    }
                }
                bwt.GetBWT(text_in_line);
                res.Write(text_in_line);
            }
            Console.WriteLine("BWT -- готов, переходим к MTF...");

            List<char> alphabet_move_to_front = new List<char>();
            int index = 0;
            line = "";
            string MTF_result = "";

            while (index < text_in_line.Length)
            {
```

```
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к Haffman...");


        double result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");


    }
  }
}
```

2.3 BWT+MTF+HAFF

```
using bwt_;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;

namespace bwt_mtf_haff
{
    public class checking_compression
    {
        public static double get_checking_compression(string compressed_text, string main_text)
        {
            FileInfo compressed_text_info = new FileInfo(compressed_text);
            FileInfo main_text_info = new FileInfo(main_text);

            long compressed_text_size = compressed_text_info.Length;
            long main_text_size = main_text_info.Length;

            double compress_ratio = (double)compressed_text_size / main_text_size;

            return compress_ratio;
        }
    }
    class Node : IComparable<Node> //узел дерева
    {
        public char? Symbol { get; set; } //символ
        public int Frequency { get; set; } // вероятность
        public Node Left { get; set; } //левый ребенок
        public Node Right { get; set; } //правый ребенок
```

```
    public Node(char symbol, int frequency)
    {
        Symbol = symbol;
        Frequency = frequency;
    }

    public Node(int frequency, Node left, Node right) //узел который при слиянии 2х других
    {
        Frequency = frequency;
        Left = left;
        Right = right;
    }

    public int CompareTo(Node other) //сравнение по вероятности
    {
        return Frequency.CompareTo(other.Frequency);
    }
}
class PriorityQueue<T> where T : IComparable<T> //очередь с приориитетом с помощью бинарной кучи
{
    private List<T> heap;

    public int Count { get { return heap.Count; } }

    public PriorityQueue()
    {
        heap = new List<T>();
    }

    public void Enqueue(T item) //добавление нового элемента в кучу
    {
        heap.Add(item); //добавляем элемент в конец
        int i = Count - 1;
        while (i > 0) //поднимаем наверх кучи пока элемент больше родителя
        {
```

```
        int parent = (i - 1) / 2;
        if (heap[parent].CompareTo(item) <= 0)
            break;
        heap[i] = heap[parent];
        i = parent;
    }
    heap[i] = item; //добавляем элемент на нудное место
}

public T Dequeue() //удаление и возвращение минимального элемента из кучи
{
    if (Count == 0)
        throw new InvalidOperationException("Queue is empty");//куча пуста
    T item = heap[0];
    int i = Count - 1;
    T last = heap[i];
    heap.RemoveAt(i); //удаляем последний в куче элемент

    if (i > 0) //нужно перестроить кучу если в ней есть ещё элементы
    {
        int parent = 0;
        while (true)
        {
            int child = parent * 2 + 1; //индекс ребенка
            if (child >= i) //если его нет то стоп
                break;
            if (child + 1 < i && heap[child + 1].CompareTo(heap[child]) < 0) //выбираем если правый меньше левого
                child++;
            if (last.CompareTo(heap[child]) <= 0)
                break;
            heap[parent] = heap[child];
            parent = child;
        }
        heap[parent] = last;
    }
```

```
            return item;
        }

    }
    class HuffmanCoding
    {
        public static Dictionary<char, string> Encode(string str)
        {
            Dictionary<char, int> freq = frequencyMap(str); //словарь с вероятностью и символами
            PriorityQueue<Node> priorityQueue = priority_queue(freq); //очередб с приоритетом

            while (priorityQueue.Count > 1) //строим дерево склдывая 2 детей с наимельшей вероятностью
            {
                Node left = priorityQueue.Dequeue();
                Node right = priorityQueue.Dequeue();
                Node parent = new Node(left.Frequency + right.Frequency, left, right);
                priorityQueue.Enqueue(parent);
            }

            Node root = priorityQueue.Dequeue(); //извлекаем корень
            Dictionary<char, string> encodingMap = encodeMap(root); //теперь словарь не с вероятностями а с кодами

            return encodingMap;
        }

        public static string compress(string str, Dictionary<char, string> encode) // получаем строку по кодам хаффмана
        {
            return string.Concat(str.Select(c => encode[c]));
        }

        public static string Decompress(string str, Dictionary<char, string> encode)
        {
            Dictionary<string, char> decodingMap = encode.ToDictionary(pair => pair.Value, pair => pair.Key); //теперь каждому коду соответствует буква, а не наоборот
            string decoded = "";
```

```
    string currentCode = "";

    foreach (char bit in str)
    {
        currentCode += bit; //дописывает 0 или 1
        if (decodingMap.ContainsKey(currentCode)) //если в словаре есть такой символ то записываем его в расшифрованную строку
        {
            decoded += decodingMap[currentCode];
            currentCode = "";
        }
    }

    return decoded;
}

private static Dictionary<char, int> frequencyMap(string str)
{
    Dictionary<char, int> frequencyMap = new Dictionary<char, int>();

    foreach (char c in str) //проходим по каждому элементу и смотрим сколько он встечается в тексте
    {
        if (frequencyMap.ContainsKey(c))
            frequencyMap[c]++;
        else
            frequencyMap[c] = 1;
    }

    return frequencyMap;
}

private static PriorityQueue<Node> priority_queue(Dictionary<char, int> freq)
{
    PriorityQueue<Node> priorityQueue = new PriorityQueue<Node>();

    foreach (var entry in freq)
```

```
        {
            priorityQueue.Enqueue(new Node(entry.Key, entry.Value)); //просто добавляем в очередь с приоритетом все пары символ - вероятность в очередь
        }

        return priorityQueue;
    }

    private static Dictionary<char, string> encodeMap(Node root) //словарь на основе дерева хаффмана
    {
        Dictionary<char, string> encodingMap = new Dictionary<char, string>();
        encodeMap_Tree(root, "", encodingMap);
        return encodingMap;
    }

    private static void encodeMap_Tree(Node node, string code, Dictionary<char, string> encode) //рекурсивно из дерево берем коды
    {
        if (node.Symbol.HasValue)
        {
            encode[node.Symbol.Value] = code;
        }
        else
        {
            encodeMap_Tree(node.Left, code + "0", encode);
            encodeMap_Tree(node.Right, code + "1", encode);
        }
    }
}
internal class Program
{
    static List<char> alphabet_func(string str) //создание алфавита
    {
        List<char> chars = new List<char>();
        for (int i = 0; i < str.Length; i++)
        {
            if (!chars.Contains(str[i]))        //добавляем уникальные буквы в алфавит
```

```
            chars.Add(str[i]);
        }
        chars.Sort((a, b) => a.CompareTo(b));   //сортировка
        return chars;
    }

    static string Move_to_Front(string str, List<char> alphabet)
    {
        string result = "";
        int index;
        char symbol;
        char char_element;
        char removed_char;
        for (int i = 0; i < str.Length; i++)
        {
            symbol = str[i];                    //символ в строке
            index = alphabet.IndexOf(symbol);   //индекс символа в алфавите
            char_element = (char)(index + 40);         //индекс символа без служебных символов
            result += char_element.ToString();         //записываем символ в форматированную строку
            removed_char = alphabet[index];            //используемый символ в алфавите

            alphabet.RemoveAt(index);       //ставим символ в начало алфавита
            alphabet.Insert(0, removed_char);
        }
        return result;
    }
    static BitArray to_bit(BitArray resoult, string compressed)
    {
        int index = 0;
        foreach (char chars in compressed)
        {
            if (chars == '1')
                resoult[index] = true;
            else if (chars == '0')
                resoult[index] = false;
```

```
            index++;
        }
        if (index < resoult.Length)
        {
            BitArray cut = new BitArray(index);         //обрезка лишней части строки
            for (int i = 0; i < index; i++)
                cut[i] = resoult[i];
            resoult = cut;
        }
        return resoult;
    }

    static void Main(string[] args)
    {




        string text_enwik7 = "enwik7.txt";
        string text_enwik7_bwt = "enwik7_bwt.txt";
        string text_enwik7_out = "enwik7_out.txt";

        string text_gray = "Gray_image.txt";
        string text_gray_bwt = "Gray_image_bwt.txt";
        string text_gray_out = "Gray_image_out.txt";

        string text_bw = "BW_image.txt";
        string text_bw_bwt = "BW_image_bwt.txt";
        string text_bw_out = "BW_image_out.txt";

        string text_color = "Color_image.txt";
        string text_color_bwt = "Color_image_bwt.txt";
        string text_color_out = "Color_image_out.txt";


        string text_russian = "rusTolstoy.txt"; //"graph.txt"
```

```
        string text_russian_bwt = "rusTolstoy_bwt.txt";
        string text_russian_out = "rusTolstoy_out.txt";

        Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
Console.WriteLine("=============================================================================================================");
        Console.WriteLine("Начался BWT...");
        BWTFast bwt = new BWTFast();
        string text_in_line = "";
        string line = "";
        using (StreamWriter res = new StreamWriter(text_russian_bwt))
        {
            using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            }
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("BWT -- готов, переходим к MTF...");

        List<char> alphabet_move_to_front = new List<char>();
        int index = 0;
        line = "";
        string MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
```

```
            index += step;


        }

        Console.WriteLine("MTF -- готов, переходим к Haffman...");
        string result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_russian_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                //text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);


                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }


        double result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================");

        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================\n\n");
```

```
        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");

Console.WriteLine("============================================================================================================");

        Console.WriteLine("Начался BWT...");

        bwt = new BWTFast();

        text_in_line = "";

        line = "";

        using (StreamWriter res = new StreamWriter(text_enwik7_bwt))

        {

            using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))

                while ((line = reader.ReadLine()) != null)

                {

                    text_in_line += line;

                }

            bwt.GetBWT(text_in_line);

            res.Write(text_in_line);

        }

        Console.WriteLine("BWT -- готов, переходим к MTF...");


        alphabet_move_to_front = new List<char>();

        index = 0;

        line = "";

        MTF_result = "";


        while (index < text_in_line.Length)

        {

            int step = Math.Min(120, text_in_line.Length - index);

            line = text_in_line.Substring(index, step);

            alphabet_move_to_front = alphabet_func(line);

            MTF_result += Move_to_Front(line, alphabet_move_to_front);

            index += step;
```

```
        }

        Console.WriteLine("MTF -- готов, переходим к Haffman...");
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_enwik7_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);
Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");
Console.WriteLine("=============================================================================================================\n\n");
```

```
        Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");

Console.WriteLine("=============================================================================================================");

        Console.WriteLine("Начался BWT...");

        bwt = new BWTFast();

        text_in_line = "";

        line = "";

        using (StreamWriter res = new StreamWriter(text_gray_bwt))

        {

            using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))

                while ((line = reader.ReadLine()) != null)

                {

                    text_in_line += line;

                }

            bwt.GetBWT(text_in_line);

            res.Write(text_in_line);

        }

        Console.WriteLine("BWT -- готов, переходим к MTF...");


        alphabet_move_to_front = new List<char>();

        index = 0;

        line = "";

        MTF_result = "";


        while (index < text_in_line.Length)

        {

            int step = Math.Min(120, text_in_line.Length - index);

            line = text_in_line.Substring(index, step);

            alphabet_move_to_front = alphabet_func(line);

            MTF_result += Move_to_Front(line, alphabet_move_to_front);

            index += step;


        }
```

```
        Console.WriteLine("MTF -- готов, переходим к Haffman...");
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_gray_out, FileMode.Create, FileAccess.Write))
        {

           index = 0;
           while (index < result_haff.Length)
           {
              int step = Math.Min(120, result_haff.Length - index);
              text_in_line = result_haff.Substring(index, step);
              text_in_line = line;
              var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
              string compressed = HuffmanCoding.compress(text_in_line, encode_map);
              BitArray res = new BitArray(compressed.Length * 8);
              res = to_bit(res, compressed);
              byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
              res.CopyTo(bytes, 0);

              writer.Write(bytes, 0, bytes.Length);
              index += step;
           }
        }

        result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================
======");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");
```

```
        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");

Console.WriteLine("==========================================================================================================");
        Console.WriteLine("Начался BWT...");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        using (StreamWriter res = new StreamWriter(text_bw_bwt))
        {
            using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("BWT -- готов, переходим к MTF...");

        alphabet_move_to_front = new List<char>();
        index = 0;
        line = "";
        MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к Haffman...");
```

```
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_bw_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("=============================================================================================
======");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");









        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
```

```
Console.WriteLine("=============================================================================================================");
            Console.WriteLine("Начался BWT...");
            bwt = new BWTFast();
            text_in_line = "";
            line = "";
            using (StreamWriter res = new StreamWriter(text_color_bwt))
            {
                using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
                    while ((line = reader.ReadLine()) != null)
                    {
                        text_in_line += line;
                    }
                bwt.GetBWT(text_in_line);
                res.Write(text_in_line);
            }
            Console.WriteLine("BWT -- готов, переходим к MTF...");

            alphabet_move_to_front = new List<char>();
            index = 0;
            line = "";
            MTF_result = "";

            while (index < text_in_line.Length)
            {
                int step = Math.Min(120, text_in_line.Length - index);
                line = text_in_line.Substring(index, step);
                alphabet_move_to_front = alphabet_func(line);
                MTF_result += Move_to_Front(line, alphabet_move_to_front);
                index += step;

            }

            Console.WriteLine("MTF -- готов, переходим к Haffman...");
            result_haff = MTF_result;
```

```
        using (FileStream writer = new FileStream(text_color_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("=============================================================================================
======");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");



        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();

    }
}
```

2.4 BWT+MTF+RLE+HAFF

```
using bwt;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;

namespace bwt_mtf_haff
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }
   class Node : IComparable<Node> //узел дерева
   {
      public char? Symbol { get; set; } //символ
      public int Frequency { get; set; } // вероятность
      public Node Left { get; set; } //левый ребенок
      public Node Right { get; set; } //правый ребенок

      public Node(char symbol, int frequency)
      {
         Symbol = symbol;
         Frequency = frequency;
      }

      public Node(int frequency, Node left, Node right) //узел который при слиянии 2х других
      {
         Frequency = frequency;
         Left = left;
         Right = right;
      }

      public int CompareTo(Node other) //сравнение по вероятности
      {
         return Frequency.CompareTo(other.Frequency);
      }
   }
   class PriorityQueue<T> where T : IComparable<T> //очередь с приориитетом с помощью бинарной кучи
   {
      private List<T> heap;

      public int Count { get { return heap.Count; } }

      public PriorityQueue()
      {
         heap = new List<T>();
      }

      public void Enqueue(T item) //добавление нового элемента в кучу
      {
         heap.Add(item); //добавляем элемент в конец
         int i = Count - 1;
         while (i > 0) //поднимаем наверх кучи пока элемент больше родителя
         {
```

```
            int parent = (i - 1) / 2;
            if (heap[parent].CompareTo(item) <= 0)
                break;
            heap[i] = heap[parent];
            i = parent;
        }
        heap[i] = item; //добавляем элемент на нудное место
    }

    public T Dequeue() //удаление и возвращение минимального элемента из кучи
    {
        if (Count == 0)
            throw new InvalidOperationException("Queue is empty");//куча пуста
        T item = heap[0];
        int i = Count - 1;
        T last = heap[i];
        heap.RemoveAt(i); //удаляем последний в куче элемент

        if (i > 0) //нужно перестроить кучу если в ней есть ещё элементы
        {
            int parent = 0;
            while (true)
            {
                int child = parent * 2 + 1; //индекс ребенка
                if (child >= i) //если его нет то стоп
                    break;
                if (child + 1 < i && heap[child + 1].CompareTo(heap[child]) < 0) //выбираем если правый меньше левого
                    child++;
                if (last.CompareTo(heap[child]) <= 0)
                    break;
                heap[parent] = heap[child];
                parent = child;
            }
            heap[parent] = last;
        }

        return item;
    }

}
class HuffmanCoding
{
    public static Dictionary<char, string> Encode(string str)
    {
        Dictionary<char, int> freq = frequencyMap(str); //словарь с вероятностью и символами
        PriorityQueue<Node> priorityQueue = priority_queue(freq); //очередб с приоритетом

        while (priorityQueue.Count > 1) //строим дерево склдывая 2 детей с наимельшей вероятностью
        {
            Node left = priorityQueue.Dequeue();
            Node right = priorityQueue.Dequeue();
            Node parent = new Node(left.Frequency + right.Frequency, left, right);
            priorityQueue.Enqueue(parent);
        }

        Node root = priorityQueue.Dequeue(); //извлекаем корень
        Dictionary<char, string> encodingMap = encodeMap(root); //теперь словарь не с вероятностями а с кодами

        return encodingMap;
    }

    public static string compress(string str, Dictionary<char, string> encode) // получаем строку по кодам хаффмана
    {
        return string.Concat(str.Select(c => encode[c]));
    }

    public static string Decompress(string str, Dictionary<char, string> encode)
    {
        Dictionary<string, char> decodingMap = encode.ToDictionary(pair => pair.Value, pair => pair.Key); //теперь каждому коду соответствует
буква, а не наоборот
        string decoded = "";
        string currentCode = "";
```

```
            foreach (char bit in str)
            {
                currentCode += bit; //дописывает 0 или 1
                if (decodingMap.ContainsKey(currentCode)) //если в словаре есть такой символ то записываем его в расшифрованную строку
                {
                    decoded += decodingMap[currentCode];
                    currentCode = "";
                }
            }

            return decoded;
        }

        private static Dictionary<char, int> frequencyMap(string str)
        {
            Dictionary<char, int> frequencyMap = new Dictionary<char, int>();

            foreach (char c in str) //проходим по каждому элементу и смотрим сколько он встечается в тексте
            {
                if (frequencyMap.ContainsKey(c))
                    frequencyMap[c]++;
                else
                    frequencyMap[c] = 1;
            }

            return frequencyMap;
        }

        private static PriorityQueue<Node> priority_queue(Dictionary<char, int> freq)
        {
            PriorityQueue<Node> priorityQueue = new PriorityQueue<Node>();

            foreach (var entry in freq)
            {
                priorityQueue.Enqueue(new Node(entry.Key, entry.Value)); //просто добавляем в очередь с приоритетом все пары символ - вероятность
в очередь
            }

            return priorityQueue;
        }

        private static Dictionary<char, string> encodeMap(Node root) //словарь на основе дерева хаффмана
        {
            Dictionary<char, string> encodingMap = new Dictionary<char, string>();
            encodeMap_Tree(root, "", encodingMap);
            return encodingMap;
        }

        private static void encodeMap_Tree(Node node, string code, Dictionary<char, string> encode) //рекурсивно из дерево берем коды
        {
            if (node.Symbol.HasValue)
            {
                encode[node.Symbol.Value] = code;
            }
            else
            {
                encodeMap_Tree(node.Left, code + "0", encode);
                encodeMap_Tree(node.Right, code + "1", encode);
            }
        }
    }
    internal class Program
    {
        static List<char> alphabet_func_mtf(string str) //создание алфавита
        {
            List<char> chars = new List<char>();
            for (int i = 0; i < str.Length; i++)
            {
                if (!chars.Contains(str[i]))        //добавляем уникальные буквы в алфавит
                    chars.Add(str[i]);
            }
            chars.Sort((a, b) => a.CompareTo(b));   //сортировка
            return chars;
```

```
}

static string Move_to_Front(string str, List<char> alphabet)
{
    string result = "";
    int index;
    char symbol;
    char char_element;
    char removed_char;
    for (int i = 0; i < str.Length; i++)
    {
        symbol = str[i];                    //символ в строке
        index = alphabet.IndexOf(symbol);   //индекс символа в алфавите
        char_element = (char)(index + 40);         //индекс символа без служебных символов
        result += char_element.ToString();         //записываем символ в форматированную строку
        removed_char = alphabet[index];            //используемый символ в алфавите

        alphabet.RemoveAt(index);       //ставим символ в начало алфавита
        alphabet.Insert(0, removed_char);
    }
    return result;
}
static BitArray to_bit_haff(BitArray resoult, string compressed)
{
    int index = 0;
    foreach (char chars in compressed)
    {
        if (chars == '1')
            resoult[index] = true;
        else if (chars == '0')
            resoult[index] = false;
        index++;
    }
    if (index < resoult.Length)
    {
        BitArray cut = new BitArray(index);         //обрезка лишней части строки
        for (int i = 0; i < index; i++)
            cut[i] = resoult[i];
        resoult = cut;
    }
    return resoult;
}

static void rle(string line, ref string result)
{

    char current_char = '0';
    string temp_str = "";


    int count_repeat = 1;               //счетчик повторных символов
    int index_in_str = 0;               //индекс прохода по строке

    while (index_in_str < line.Length - 2)          //идем по строе
    {
        current_char = line[index_in_str];          //текущий символ
        if (current_char == line[index_in_str + 1]) //если символ повтояется
        {
            count_repeat++;
            index_in_str++;
        }
        else
        {
            if (count_repeat == 1)                  //если символ ни разу не повторился
            {
                temp_str += line[index_in_str];
                index_in_str++;
                while (line[index_in_str] != line[index_in_str - 1] && index_in_str != line.Length - 1)
                {
                    temp_str += line[index_in_str];     //пока символы не повторяются
                    index_in_str++;
                }
                index_in_str--;                         //вычитаем индекс символа, который повторился
```

```
                temp_str = temp_str.Substring(0, temp_str.Length - 1);  //вырезаем из строки символ
                result+=('&');
                result += (temp_str);     //вставляем строку неповторяющихся символов между разделителями
                result += ('&');
                temp_str = "";
            }
            else
            {
                result += ((char)(count_repeat));     //записываем символ повторок
                result += (current_char);             //записываем текущий символ
                count_repeat = 1;
                index_in_str++;
            }
        }

    }

    result += ((char)(count_repeat));
    result += (current_char);


}

static void Main(string[] args)
{



    string text_enwik7 = "enwik7.txt";
    string text_enwik7_bwt = "enwik7_bwt.txt";
    string text_enwik7_out = "enwik7_out.txt";

    string text_gray = "Gray_image.txt";
    string text_gray_bwt = "Gray_image_bwt.txt";
    string text_gray_out = "Gray_image_out.txt";

    string text_bw = "BW_image.txt";
    string text_bw_bwt = "BW_image_bwt.txt";
    string text_bw_out = "BW_image_out.txt";

    string text_color = "Color_image.txt";
    string text_color_bwt = "Color_image_bwt.txt";
    string text_color_out = "Color_image_out.txt";


    string text_russian = "rusTolstoy.txt"; //"graph.txt"
    string text_russian_bwt = "rusTolstoy_bwt.txt";
    string text_russian_out = "rusTolstoy_out.txt";

    Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");

Console.WriteLine("=============================================================================================================
=====");
    Console.WriteLine("Начался BWT...");
    BWTFast bwt = new BWTFast();
    string text_in_line = "";
    string line = "";
    using (StreamWriter res = new StreamWriter(text_russian_bwt))
    {
        using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
        {
            while ((line = reader.ReadLine()) != null)
            {
                text_in_line += line;
            }
        }
        bwt.GetBWT(text_in_line);
        res.Write(text_in_line);
    }
    Console.WriteLine("BWT -- готов, переходим к MTF...");

    List<char> alphabet_move_to_front = new List<char>();
    int index = 0;
```

```
        line = "";
        string MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func_mtf(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к RLE...");
        string result_rle = "";
        rle(MTF_result, ref result_rle);
        Console.WriteLine("RLE -- готов, переходим к HAFF...");
        string result_haff = result_rle;
        using (FileStream writer = new FileStream(text_russian_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                //text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit_haff(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        double result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================
======");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
======\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");

Console.WriteLine("=============================================================================================
======");
        Console.WriteLine("Начался BWT...");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        using (StreamWriter res = new StreamWriter(text_enwik7_bwt))
        {
            using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("BWT -- готов, переходим к MTF...");
```

```
        alphabet_move_to_front = new List<char>();
        index = 0;
        line = "";
        MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func_mtf(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к RLE...");
         result_rle = "";
        rle(MTF_result, ref result_rle);
        Console.WriteLine("RLE -- готов, переходим к HAFF...");
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_enwik7_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit_haff(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");

Console.WriteLine("=============================================================================================================");
        Console.WriteLine("Начался BWT...");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        using (StreamWriter res = new StreamWriter(text_gray_bwt))
        {
            using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("BWT -- готов, переходим к MTF...");
```

```
        alphabet_move_to_front = new List<char>();
        index = 0;
        line = "";
        MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func_mtf(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к RLE...");
        result_rle = "";
        rle(MTF_result, ref result_rle);
        Console.WriteLine("RLE -- готов, переходим к HAFF...");
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_gray_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit_haff(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");

Console.WriteLine("=============================================================================================================");
        Console.WriteLine("Начался BWT...");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        using (StreamWriter res = new StreamWriter(text_bw_bwt))
        {
            using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
```

```
        Console.WriteLine("BWT -- готов, переходим к MTF...");

        alphabet_move_to_front = new List<char>();
        index = 0;
        line = "";
        MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func_mtf(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к RLE...");
        result_rle = "";
        rle(MTF_result, ref result_rle);
        Console.WriteLine("RLE -- готов, переходим к HAFF...");
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_bw_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit_haff(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine("Начался BWT...");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        using (StreamWriter res = new StreamWriter(text_color_bwt))
        {
            using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
```

```
        }
        Console.WriteLine("BWT -- готов, переходим к MTF...");

        alphabet_move_to_front = new List<char>();
        index = 0;
        line = "";
        MTF_result = "";

        while (index < text_in_line.Length)
        {
            int step = Math.Min(120, text_in_line.Length - index);
            line = text_in_line.Substring(index, step);
            alphabet_move_to_front = alphabet_func_mtf(line);
            MTF_result += Move_to_Front(line, alphabet_move_to_front);
            index += step;

        }

        Console.WriteLine("MTF -- готов, переходим к RLE...");
        result_rle = "";
        rle(MTF_result, ref result_rle);
        Console.WriteLine("RLE -- готов, переходим к HAFF...");
        result_haff = MTF_result;
        using (FileStream writer = new FileStream(text_color_out, FileMode.Create, FileAccess.Write))
        {

            index = 0;
            while (index < result_haff.Length)
            {
                int step = Math.Min(120, result_haff.Length - index);
                text_in_line = result_haff.Substring(index, step);
                text_in_line = line;
                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit_haff(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

        result = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");


        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();

    }
  }
}
```

2.5 BWT+RLE

```
using bwt_rle;
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using static System.Net.Mime.MediaTypeNames;

namespace bwt_rle1
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }
   internal class Program
   {
      static void rle(string text, string text_out)
      {
         using (StreamReader reader = new StreamReader(text, Encoding.UTF8)) //чтение построчно
         using (StreamWriter writer = new StreamWriter(text_out, false))
         {
            char current_char = '0';
            string line;
            string temp_str = "";

            while ((line = reader.ReadLine()) != null) // пптриааа
            {
               int count_repeat = 1;              //счетчик повторных символов
               int index_in_str = 0;              //индекс прохода по строке

               while (index_in_str < line.Length - 2)          //идем по строе
               {
                  current_char = line[index_in_str];          //текущий символ
                  if (current_char == line[index_in_str + 1]) //если символ повтояется
                  {
                     count_repeat++;
                     index_in_str++;
                  }
                  else
                  {
                     if (count_repeat == 1)                   //если символ ни разу не повторился
                     {
                        temp_str += line[index_in_str];
                        index_in_str++;
                        while (line[index_in_str] != line[index_in_str - 1] && index_in_str != line.Length - 1)
                        {
                           temp_str += line[index_in_str];     //пока символы не повторяются
                           index_in_str++;
                        }
                        index_in_str--;                        //вычитаем индекс символа, который повторился
                        temp_str = temp_str.Substring(0, temp_str.Length - 1);  //вырезаем из строки символ
                        writer.Write('&');
                        writer.Write(temp_str);     //вставляем строку неповторяющихся символов между разделителями
                        writer.Write('&');
                        temp_str = "";
                     }
```

```
                else
                {
                    writer.Write((char)(count_repeat));     //записываем символ повторок
                    writer.Write(current_char);             //записываем текущий символ
                    count_repeat = 1;
                    index_in_str++;
                }
            }

        }

        writer.Write((char)(count_repeat));
        writer.Write(current_char);
        writer.Write('\n');
    }
  }
}

static void Main(string[] args)
{

    string text_russian = "rusTolstoy.txt";
    string text_russian_bwt = "rusTolstoy_bwt.txt";
    string text_russian_out = "rusTolstoy_out.txt";

    string text_enwik7 = "enwik7.txt";
    string text_enwik7_bwt = "enwik7_bwt.txt";
    string text_enwik7_out = "enwik7_out.txt";

    string text_gray = "Gray_image.txt";
    string text_gray_bwt = "Gray_image_bwt.txt";
    string text_gray_out = "Gray_image_out.txt";

    string text_bw = "BW_image.txt";
    string text_bw_bwt = "BW_image_bwt.txt";
    string text_bw_out = "BW_image_out.txt";

    string text_color = "Color_image.txt";
    string text_color_bwt = "Color_image_bwt.txt";
    string text_color_out = "Color_image_out.txt";


    double result = 0;

    Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
    BWTFast bwt = new BWTFast();
    string text_in_line = "";
    string line = "";
    Console.WriteLine("Код работает...");
    using (StreamWriter res = new StreamWriter(text_russian_bwt))
    {
        using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
        {
            while ((line = reader.ReadLine()) != null)
            {
                text_in_line += line;
            }
        }
        Console.WriteLine("Текст считался, началось бвт...");
        bwt.GetBWT(text_in_line);
        res.Write(text_in_line);

    }
    Console.WriteLine("Начался рле...");
    rle(text_russian_bwt, text_russian_out);
    result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================
=====");
    Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");
```

```
        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        Console.WriteLine("Код работает...");
        using (StreamWriter res = new StreamWriter(text_enwik7_bwt))
        {
            using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            }
            Console.WriteLine("Текст считался, началось бвт...");
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("Начался рле...");
        rle(text_enwik7_bwt, text_enwik7_out);
        result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        Console.WriteLine("Код работает...");
        using (StreamWriter res = new StreamWriter(text_gray_bwt))
        {
            using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            }
            Console.WriteLine("Текст считался, началось бвт...");
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("Начался рле...");
        rle(text_gray_bwt, text_gray_out);
        result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
=====\n\n");


        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        Console.WriteLine("Код работает...");
        using (StreamWriter res = new StreamWriter(text_bw_bwt))
        {
            using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
            {
                while ((line = reader.ReadLine()) != null)
```

```
                {
                    text_in_line += line;
                }
            }
            Console.WriteLine("Текст считался, началось бвт...");
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("Начался рле...");
        rle(text_bw_bwt, text_bw_out);
        result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
        bwt = new BWTFast();
        text_in_line = "";
        line = "";
        Console.WriteLine("Код работает...");
        using (StreamWriter res = new StreamWriter(text_color_bwt))
        {
            using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line += line;
                }
            }
            Console.WriteLine("Текст считался, началось бвт...");
            bwt.GetBWT(text_in_line);
            res.Write(text_in_line);
        }
        Console.WriteLine("Начался рле...");
        rle(text_color_out, text_color);
        result = checking_compression.get_checking_compression(text_color_bwt, text_color_out);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");


        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();

    }
  }
}
```

2.6 Алгоритм Хаффмана

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Text;
using System.Collections;

class Program
{
   static void Main()
   {

      string text_russian = "rusTolstoy.txt";
      string text_russian_out = "rusTolstoy_out.txt";

      string text_enwik7 = "enwik7.txt";
      string text_enwik7_out = "enwik7_out.txt";

      string text_gray = "Gray_image.txt";
      string text_gray_out = "Gray_image_out.txt";

      string text_bw = "BW_image.txt";
      string text_bw_out = "BW_image_out.txt";

      string text_color = "Color_image.txt";
      string text_color_out = "Color_image_out.txt";

      Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
      string line;
      string text_in_line = "";
      using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
         using (FileStream writer = new FileStream(text_russian_out, FileMode.Create, FileAccess.Write))
            while ((line = reader.ReadLine()) != null)
               if (line != "")
               {
                  text_in_line = line;
                  var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                  string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                  BitArray res = new BitArray(compressed.Length * 8);
                  res = toBit(res, compressed);
                  byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                  res.CopyTo(bytes, 0);

                  writer.Write(bytes, 0, bytes.Length);
               }

      double resoult = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================
=====");
      Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("=============================================================================================
=====\n\n");


      Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");
      line ="";
       text_in_line = "";
      using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))
      using (FileStream writer = new FileStream(text_enwik7_out, FileMode.Create, FileAccess.Write))
         while ((line = reader.ReadLine()) != null)
            if (line != "")
            {
               text_in_line = line;
               var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
               string compressed = HuffmanCoding.compress(text_in_line, encode_map);
               BitArray res = new BitArray(compressed.Length * 8);
               res = toBit(res, compressed);
               byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
```

```
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
            }

        resoult = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("=============================================================================================
=====\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))
        using (FileStream writer = new FileStream(text_gray_out, FileMode.Create, FileAccess.Write))
            while ((line = reader.ReadLine()) != null)
                if (line != "")
                {
                    text_in_line = line;
                    var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                    string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                    BitArray res = new BitArray(compressed.Length * 8);
                    res = toBit(res, compressed);
                    byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                    res.CopyTo(bytes, 0);

                    writer.Write(bytes, 0, bytes.Length);
                }

        resoult = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("=============================================================================================
=====\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
        using (FileStream writer = new FileStream(text_bw_out, FileMode.Create, FileAccess.Write))
            while ((line = reader.ReadLine()) != null)
                if (line != "")
                {
                    text_in_line = line;
                    var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                    string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                    BitArray res = new BitArray(compressed.Length * 8);
                    res = toBit(res, compressed);
                    byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                    res.CopyTo(bytes, 0);

                    writer.Write(bytes, 0, bytes.Length);
                }

        resoult = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {resoult}");
```

```
Console.WriteLine("=============================================================================================
=====\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
        using (FileStream writer = new FileStream(text_color_out, FileMode.Create, FileAccess.Write))
            while ((line = reader.ReadLine()) != null)
                if (line != "")
                {
                    text_in_line = line;
                    var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                    string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                    BitArray res = new BitArray(compressed.Length * 8);
                    res = toBit(res, compressed);
                    byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                    res.CopyTo(bytes, 0);

                    writer.Write(bytes, 0, bytes.Length);
                }

        resoult = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("=============================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("=============================================================================================
=====\n\n");

        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();
    }
    static BitArray toBit(BitArray res, string compressed)
    {
        int index = 0;
        foreach (char chars in compressed)
        {
            if (chars == '1')
            {
                res[index] = true;
            }
            else if (chars == '0')
            {
                res[index] = false;
            }
            index++;
        }
        if (index < res.Length)
        {
            BitArray cut = new BitArray(index);
            for (int i = 0; i < index; i++)
            {
                cut[i] = res[i];
            }
            res = cut;
        }
        return res;
    }
}

class Node : IComparable<Node> //узел дерева
{
    public char? Symbol { get; set; } //символ
    public int Frequency { get; set; } // вероятность
    public Node Left { get; set; } //левый ребенок
    public Node Right { get; set; } //правый ребенок
```

```
    public Node(char symbol, int frequency)
    {
        Symbol = symbol;
        Frequency = frequency;
    }

    public Node(int frequency, Node left, Node right) //узел который при слиянии 2х других
    {
        Frequency = frequency;
        Left = left;
        Right = right;
    }

    public int CompareTo(Node other) //сравнение по вероятности
    {
        return Frequency.CompareTo(other.Frequency);
    }
}

class HuffmanCoding
{
    public static Dictionary<char, string> Encode(string str)
    {
        Dictionary<char, int> freq = frequencyMap(str); //словарь с вероятностью и символами
        PriorityQueue<Node> priorityQueue = priority_queue(freq); //очередб с приоритетом

        while (priorityQueue.Count > 1) //строим дерево склдывая 2 детей с наимельшей вероятностью
        {
            Node left = priorityQueue.Dequeue();
            Node right = priorityQueue.Dequeue();
            Node parent = new Node(left.Frequency + right.Frequency, left, right);
            priorityQueue.Enqueue(parent);
        }

        Node root = priorityQueue.Dequeue(); //извлекаем корень
        Dictionary<char, string> encodingMap = encodeMap(root); //теперь словарь не с вероятностями а с кодами

        return encodingMap;
    }

    public static string compress(string str, Dictionary<char, string> encode) // получаем строку по кодам хаффмана
    {
        return string.Concat(str.Select(c => encode[c]));
    }

    public static string Decompress(string str, Dictionary<char, string> encode)
    {
        Dictionary<string, char> decodingMap = encode.ToDictionary(pair => pair.Value, pair => pair.Key); //теперь каждому коду соответствует буква, а не наоборот
        string decoded = "";
        string currentCode = "";

        foreach (char bit in str)
        {
            currentCode += bit; //дописывает 0 или 1
            if (decodingMap.ContainsKey(currentCode)) //если в словаре есть такой символ то записываем его в расшифрованную строку
            {
                decoded += decodingMap[currentCode];
                currentCode = "";
            }
        }

        return decoded;
    }

    private static Dictionary<char, int> frequencyMap(string str)
    {
        Dictionary<char, int> frequencyMap = new Dictionary<char, int>();

        foreach (char c in str) //проходим по каждому элементу и смотрим сколько он встечается в тексте
        {
            if (frequencyMap.ContainsKey(c))
                frequencyMap[c]++;
```

```
            else
                frequencyMap[c] = 1;
        }

        return frequencyMap;
    }

    private static PriorityQueue<Node> priority_queue(Dictionary<char, int> freq)
    {
        PriorityQueue<Node> priorityQueue = new PriorityQueue<Node>();

        foreach (var entry in freq)
        {
            priorityQueue.Enqueue(new Node(entry.Key, entry.Value)); //просто добавляем в очередь с приоритетом все пары символ - вероятность в
очередь
        }

        return priorityQueue;
    }

    private static Dictionary<char, string> encodeMap(Node root) //словарь на основе дерева хаффмана
    {
        Dictionary<char, string> encodingMap = new Dictionary<char, string>();
        encodeMap_Tree(root, "", encodingMap);
        return encodingMap;
    }

    private static void encodeMap_Tree(Node node, string code, Dictionary<char, string> encode) //рекурсивно из дерево берем коды
    {
        if (node.Symbol.HasValue)
        {
            encode[node.Symbol.Value] = code;
        }
        else
        {
            encodeMap_Tree(node.Left, code + "0", encode);
            encodeMap_Tree(node.Right, code + "1", encode);
        }
    }
}

class PriorityQueue<T> where T : IComparable<T> //очередь с приориитетом с помощью бинарной кучи
{
    private List<T> heap;

    public int Count { get { return heap.Count; } }

    public PriorityQueue()
    {
        heap = new List<T>();
    }

    public void Enqueue(T item) //добавление нового элемента в кучу
    {
        heap.Add(item); //добавляем элемент в конец
        int i = Count - 1;
        while (i > 0) //поднимаем наверх кучи пока элемент больше родителя
        {
            int parent = (i - 1) / 2;
            if (heap[parent].CompareTo(item) <= 0)
                break;
            heap[i] = heap[parent];
            i = parent;
        }
        heap[i] = item; //добавляем элемент на нудное место
    }

    public T Dequeue() //удаление и возвращение минимального элемента из кучи
    {
        if (Count == 0)
            throw new InvalidOperationException("Queue is empty");//куча пуста
        T item = heap[0];
        int i = Count - 1;
```

```
        T last = heap[i];
        heap.RemoveAt(i); //удаляем последний в куче элемент

        if (i > 0) //нужно перестроить кучу если в ней есть ещё элементы
        {
            int parent = 0;
            while (true)
            {
                int child = parent * 2 + 1; //индекс ребенка
                if (child >= i) //если его нет то стоп
                    break;
                if (child + 1 < i && heap[child + 1].CompareTo(heap[child]) < 0) //выбираем если правый меньше левого
                    child++;
                if (last.CompareTo(heap[child]) <= 0)
                    break;
                heap[parent] = heap[child];
                parent = child;
            }
            heap[parent] = last;
        }

        return item;
    }

}
public class checking_compression
{
    public static double get_checking_compression(string compressed_text, string main_text)
    {
        FileInfo compressed_text_info = new FileInfo(compressed_text);
        FileInfo main_text_info = new FileInfo(main_text);

        long compressed_text_size = compressed_text_info.Length;
        long main_text_size = main_text_info.Length;

        double compress_ratio = (double)compressed_text_size / main_text_size;

        return compress_ratio;
    }
}
```

2.7 LZ77

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using static System.Net.Mime.MediaTypeNames;

namespace Lz77
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }
   internal class Program
   {
      public struct triplet
      {
         public int prev;           // Смещение назад в строке (отступ влево)
         public int length;         // Длина подстроки
         public char next;          // Символ, следующий за подстрокой

         // Конструктор для инициализации структуры triplet
         public triplet(int prev, int length, char next)
         {
            this.prev = prev;
            this.length = length;
            this.next = next;
         }
         public void print_node()
         {
            Console.WriteLine($"Нода: ({prev},{length},{next})");
            Console.WriteLine($"Отступить влево на {prev} символов; Длина подстроки: {length}; Символ справа: {next}\n");
         }
      }

      // Функция для проверки, является ли объект триплетом с одним символом
      static public bool bool_triplet_one_symbol(object obj)
      {
         return obj is triplet_one_symbol;
      }

      // Структура для хранения информации о триплете с одним символом
      public struct triplet_one_symbol
      {
         public char next;          // Символ справа


         public triplet_one_symbol(char symbol)
         {
            this.next = symbol;
         }
         public void print_node() // для вывода
         {
            Console.WriteLine($"Нода: ({next})");
            Console.WriteLine($"Отступить влево на {0} символов; Длина подстроки: {0}; Символ справа: {next}\n");
         }
      }
```

```
static List<object> lz77(string str, int bufer_size)
{
    List<object> triplet_list = new List<object>();
    string bufer = "";
    int pos = 0;
    int prev_symbols_triplet = 0;
    int length_triplet = 0;
    char next_symbol_triplet = '\0';

    while (pos < str.Length)                //идем по строке
    {
        // Получаем текущий буфер
        bufer = str.Substring(Math.Max(pos - bufer_size, 0), pos - Math.Max(pos - bufer_size, 0));
        int index_substr_bufer = bufer.Length;
        int length_substring = 0;

        // Ищем самую длинную повторяющуюся подстроку в буфере
        for (int i = 1; i < bufer.Length + 1; i++)
        {
            if ((bufer.LastIndexOf(str.Substring(pos, i)) != -1))
            {
                // Если нашли, пытаемся найти подстроку на элемент больше
                index_substr_bufer = bufer.LastIndexOf(str.Substring(pos, i));
                length_substring = i;
            }
            else
                break;
        }

        // Сохраняем сдвиг и длину подстроки
        prev_symbols_triplet = bufer.Length - index_substr_bufer;
        length_triplet = length_substring;

        int pos_repeat = pos;
        int length_substring_repeat = length_substring;
        if (length_substring != 0)
        {
            // Проверяем повторки, если они есть, записываем как можно длиннее
            while (str[pos_repeat] == str[pos_repeat + length_substring])
            {
                pos_repeat++;
                length_substring_repeat++;

            }
            if (pos_repeat - pos > 1) // Если повторка 1, то это не повтор
            {
                pos = pos_repeat;
                length_triplet = length_substring_repeat;
            }
        }

        // Записываем следующий символ
        next_symbol_triplet = str[pos + length_substring];

        pos += length_substring + 1;

        // Добавляем триплет в список
        if (prev_symbols_triplet == 0 && length_triplet == 0)
        {
            triplet_one_symbol triplet = new triplet_one_symbol(next_symbol_triplet);
            triplet_list.Add(triplet);

        }
        else
        {
            triplet triplet = new triplet(prev_symbols_triplet, length_triplet, next_symbol_triplet);
            triplet_list.Add(triplet);

        }

    }
    return triplet_list; // возвращаем список триплетов
```

```
    }

    static public string listToStr(List<object> list_result)
    {
        string res = "";
        int count = 0;
        foreach (object list in list_result)
        {
            if (bool_triplet_one_symbol(list))
            {
                triplet_one_symbol triplet = (triplet_one_symbol)list;
                res += triplet.next;
                count += 1;
            }
            else
            {
                triplet triplet = (triplet)list;
                res += triplet.prev;
                res += triplet.length;
                res += triplet.next;
            }
        }
        return res;
    }

    static void Main(string[] args)
    {
        string text_russian = "rusTolstoy.txt";
        string text_russian_out = "rusTolstoy_out.txt";

        string text_enwik7 = "enwik7.txt";
        string text_enwik7_out = "enwik7_out.txt";

        string text_gray = "Gray_image.txt";
        string text_gray_out = "Gray_image_out.txt";

        string text_bw = "BW_image.txt";
        string text_bw_out = "BW_image_out.txt";

        string text_color = "Color_image.txt";
        string text_color_out = "Color_image_out.txt";


        Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
        string line;
        string text_in_line = "";
        using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
        {
            using (StreamWriter writer = new StreamWriter(text_russian_out, false))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "́";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    writer.Write(text_in_line);
                }

        }

        double result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");
         line = "";
```

```
 text_in_line = "";
using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))
{
    using (StreamWriter writer = new StreamWriter(text_enwik7_out, false))
        while ((line = reader.ReadLine()) != null)
        {
            text_in_line = line + "";//
            List<object> list_Res = lz77(text_in_line, 5);
            text_in_line = listToStr(list_Res);
            writer.Write(text_in_line);
        }

}

 result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=============================================================================================
======");
Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
======\n\n");




Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");
line="";
text_in_line = "";
using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))
{
    using (StreamWriter writer = new StreamWriter(text_gray_out, false))
        while ((line = reader.ReadLine()) != null)
        {
            text_in_line = line + "";//
            List<object> list_Res = lz77(text_in_line, 5);
            text_in_line = listToStr(list_Res);
            writer.Write(text_in_line);
        }

}

result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================
======");
Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
======\n\n");




Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
line = "";
text_in_line = "";
using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
{
    using (StreamWriter writer = new StreamWriter(text_bw_out, false))
        while ((line = reader.ReadLine()) != null)
        {
            text_in_line = line + "";//
            List<object> list_Res = lz77(text_in_line, 5);
            text_in_line = listToStr(list_Res);
            writer.Write(text_in_line);
        }

}

result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("=============================================================================================
```

```
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("==========================================================================================
=====\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
        {
            using (StreamWriter writer = new StreamWriter(text_color_out, false))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "̑";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    writer.Write(text_in_line);
                }

        }

        result = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("==========================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("==========================================================================================
=====\n\n");



        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();
    }
  }
}
```

2.8 LZ77+Арифметическое кодирование

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;

namespace arifmetic
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }
   internal class Program
   {
      public struct triplet
      {
         public int prev;           // Смещение назад в строке (отступ влево)
         public int length;         // Длина подстроки
         public char next;          // Символ, следующий за подстрокой

         // Конструктор для инициализации структуры triplet
         public triplet(int prev, int length, char next)
         {
            this.prev = prev;
            this.length = length;
            this.next = next;
         }
         public void print_node()
         {
            Console.WriteLine($"Нода: ({prev},{length},{next})");
            Console.WriteLine($"Отступить влево на {prev} символов; Длина подстроки: {length}; Символ справа: {next}\n");
         }
      }

      // Функция для проверки, является ли объект триплетом с одним символом
      static public bool bool_triplet_one_symbol(object obj)
      {
         return obj is triplet_one_symbol;
      }

      // Структура для хранения информации о триплете с одним символом
      public struct triplet_one_symbol
      {
         public char next;          // Символ справа


         public triplet_one_symbol(char symbol)
         {
            this.next = symbol;
         }
         public void print_node() // для вывода
         {
            Console.WriteLine($"Нода: ({next})");
            Console.WriteLine($"Отступить влево на {0} символов; Длина подстроки: {0}; Символ справа: {next}\n");
         }
      }
```

```
static List<object> lz77(string str, int bufer_size)
{
    List<object> triplet_list = new List<object>();
    string bufer = "";
    int pos = 0;
    int prev_symbols_triplet = 0;
    int length_triplet = 0;
    char next_symbol_triplet = '\0';

    while (pos < str.Length)                //идем по строке
    {
        // Получаем текущий буфер
        bufer = str.Substring(Math.Max(pos - bufer_size, 0), pos - Math.Max(pos - bufer_size, 0));
        int index_substr_bufer = bufer.Length;
        int length_substring = 0;

        // Ищем самую длинную повторяющуюся подстроку в буфере
        for (int i = 1; i < bufer.Length + 1; i++)
        {
            if ((bufer.LastIndexOf(str.Substring(pos, i)) != -1))
            {
                // Если нашли, пытаемся найти подстроку на элемент больше
                index_substr_bufer = bufer.LastIndexOf(str.Substring(pos, i));
                length_substring = i;
            }
            else
                break;
        }

        // Сохраняем сдвиг и длину подстроки
        prev_symbols_triplet = bufer.Length - index_substr_bufer;
        length_triplet = length_substring;

        int pos_repeat = pos;
        int length_substring_repeat = length_substring;
        if (length_substring != 0)
        {
            // Проверяем повторки, если они есть, записываем как можно длиннее
            while (str[pos_repeat] == str[pos_repeat + length_substring])
            {
                pos_repeat++;
                length_substring_repeat++;

            }
            if (pos_repeat - pos > 1) // Если повторка 1, то это не повтор
            {
                pos = pos_repeat;
                length_triplet = length_substring_repeat;
            }
        }

        // Записываем следующий символ
        next_symbol_triplet = str[pos + length_substring];

        pos += length_substring + 1;

        // Добавляем триплет в список
        if (prev_symbols_triplet == 0 && length_triplet == 0)
        {
            triplet_one_symbol triplet = new triplet_one_symbol(next_symbol_triplet);
            triplet_list.Add(triplet);

        }
        else
        {
            triplet triplet = new triplet(prev_symbols_triplet, length_triplet, next_symbol_triplet);
            triplet_list.Add(triplet);

        }

    }
    return triplet_list; // возвращаем список триплетов
```

```
}

static public string listToStr(List<object> list_result)
{
    string res = "";
    int count = 0;
    foreach (object list in list_result)
    {
        if (bool_triplet_one_symbol(list))
        {
            triplet_one_symbol triplet = (triplet_one_symbol)list;
            res += triplet.next;
            count += 1;
        }
        else
        {
            triplet triplet = (triplet)list;
            res += triplet.prev;
            res += triplet.length;
            res += triplet.next;
        }
    }
    return res;
}



static double arifm_code(string str, double[] percent, char[] chars, out int length_error)            //ПЕРЕМЕННАЯ ДЛЯ ОШИБКИ ERROR
{
    double[] intervals = new double[percent.Length + 1];
    length_error=0;
    double left_border = 0;            //границы
    double right_border = 1;

    double interval = 0;          //переменная для создания интервалов

    for (int i = 1; i < percent.Length + 1; i++)
    {
        interval += percent[i - 1];
        intervals[i] = interval;              //создание переменной интервалов
    }

    char to_find = ' ';                //кодируемый символ
    int index = 0;

    int j = 0;
    length_error = 0;

    foreach (char ch_find in str)
    {
        j++;
        double part_length = right_border - left_border;            //длина интервала
        to_find = ch_find;                                          //символ, который необходимо найти
        index = Array.IndexOf(chars, to_find);
        left_border += intervals[index] * part_length;              //новые границы
        right_border = left_border + percent[index] * part_length;
        if (left_border == right_border)                                        //ЕСЛИ ГРАНИЦЫ СОВПАЛИ, ТО КОДИРОВКА ПРЕРЫВАЕТСЯ
        {
            length_error = j;                                                   //ВОЗВРАЩАЕТ ИНДЕКС, ГДЕ ПРЕРВАЛАСЬ КОДИРОВКА
            break;
        }
    }

    double result = (left_border + right_border) / 2;
    return result;
}
static void text_alphabet_from_line(out char[] chars, out double[] percent, string line)
{
    int count = 0;
    int count_symbols = 0;

    List<char> alphabet = new List<char>();
```

```
    foreach (char ch in line)               //создание алфавита
    {
        for (int i = 0; i < alphabet.Count; i++)
            if (ch != alphabet[i])
                count++;

        if (count == alphabet.Count)       //если символ не встретился за прогон, добавляем
            alphabet.Add(ch);
        count = 0;

    }
    chars = new char[alphabet.Count];
    for (int i = 0; i < alphabet.Count; i++)        //переписываем алфавит в массив
        chars[i] = alphabet[i];
    Array.Sort(chars);                              //сортируем массив

    double[] appearance = new double[alphabet.Count];
    percent = new double[alphabet.Count];

    foreach (char ch in line)               //создание массива повторок
        for (int i = 0; i < alphabet.Count; i++)
            if (ch == alphabet[i])
            {
                appearance[i] += 1;
                count_symbols += 1;
                break;
            }
    for (int i = 0; i < alphabet.Count; i++)
        percent[i] += appearance[i] / count_symbols;        //считаем процент появления
}
static void Main(string[] args)
{
    string text_russian = "rusTolstoy.txt";
    string text_russian_lz77 = "text_russian_lz77.txt";
    string text_russian_out = "rusTolstoy_out.txt";

    string text_enwik7 = "enwik7.txt";
    string text_enwik7_lz77 = "text_enwik7_lz77.txt";
    string text_enwik7_out = "enwik7_out.txt";

    string text_gray = "Gray_image.txt";
    string text_gray_lz77 = "text_gray_lz77.txt";
    string text_gray_out = "Gray_image_out.txt";

    string text_bw = "BW_image.txt";
    string text_bw_lz77 = "text_bw_lz77.txt";
    string text_bw_out = "BW_image_out.txt";

    string text_color = "Color_image.txt";
    string text_color_lz77 = "text_color_lz77.txt";
    string text_color_out = "Color_image_out.txt";
    string lines;

    int error = 0;
    char[] chars = new char[1];                 //массив символов
    double result = 0;
    double[] percent = new double[1];               //массив вероятностей
    List<int> length = new List<int>();
    string text_result = "";
    string line;
    string text_in_line = "";

    Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");

    using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
    {
        using (StreamWriter writer = new StreamWriter(text_russian_lz77, false))
            while ((line = reader.ReadLine()) != null)
            {
                text_in_line = line + "\n";//
                List<object> list_Res = lz77(text_in_line, 5);
                text_in_line = listToStr(list_Res);
```

```
                writer.Write(text_in_line);
            }

        }
        Console.WriteLine("lz77 отработал, переходим к arifmetic...");
        using (StreamReader reader = new StreamReader(text_russian_lz77, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_russian_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                           //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                        text_alphabet_from_line(out chars, out percent, lines);
                        int step = Math.Min(lines.Length, 15);
                        result = arifm_code(lines.Substring(0, step), percent, chars, out error);                   //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ СТРОКЕ
                        if (error != 0)                                             //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                        {
                            int nothing;                                            //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ БУДЕТ)
                            text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));     //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                            result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                  // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
                            text_result = "";
                            lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";             //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                        }
                        else                                                        //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                        {
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                  // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
                            text_result = "";
                            lines = lines.Length > 15 ? lines.Substring(15) : "";
                        }
                    }
        result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");

        using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))
        {
            using (StreamWriter writer = new StreamWriter(text_enwik7_lz77, false))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "\n";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    writer.Write(text_in_line);
                }

        }
        Console.WriteLine("lz77 отработал, переходим к arifmetic...");
        using (StreamReader reader = new StreamReader(text_enwik7_lz77, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_enwik7_out))
            while ((lines = reader.ReadLine()) != null)
```

```
            while (lines.Length != 0)
                if (lines.Length > 0)                                           //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                {
                    text_alphabet_from_line(out chars, out percent, lines);
                    int step = Math.Min(lines.Length, 15);
                    result = arifm_code(lines.Substring(0, step), percent, chars, out error);                   //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ
СТРОКЕ
                    if (error != 0)                                             //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                    {
                        int nothing;                                            //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ
БУДЕТ)
                        text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));     //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО
СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                        result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                        text_result = result.ToString("F20");
                        text_result = text_result.Substring(2, text_result.Length - 2);
                        text_result = text_result.TrimEnd('0');
                        long number = long.Parse(text_result);                          // Преобразование строки в целое число
                        text_result = number.ToString("X");
                        writer.Write(text_result);

                        text_result = "";
                        lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";             //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ,
ТО ОНА ПУСТАЯ)
                    }
                    else                                                        //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                    {

                        text_result = result.ToString("F20");
                        text_result = text_result.TrimEnd('0');
                        text_result = text_result.Substring(2, text_result.Length - 2);
                        long number = long.Parse(text_result);                          // Преобразование строки в целое число
                        text_result = number.ToString("X");
                        writer.Write(text_result);
                        lines = lines.Length > 15 ? lines.Substring(15) : "";
                        text_result = "";

                    }
                }
        result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=============================================================================================
======");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================
======\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");

        using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))
        {
            using (StreamWriter writer = new StreamWriter(text_gray_lz77, false))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    writer.Write(text_in_line);
                }

        }
        Console.WriteLine("lz77 отработал, переходим к arifmetic...");
        using (StreamReader reader = new StreamReader(text_gray_lz77, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_gray_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                       //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                        text_alphabet_from_line(out chars, out percent, lines);
```

```
                int step = Math.Min(lines.Length, 5);
                result = arifm_code(lines.Substring(0, step), percent, chars, out error);                    //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ
СТРОКЕ
                if (error != 0)                                                 //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                {
                   int nothing;                                                 //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ
БУДЕТ)
                   text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));      //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО
СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                   result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                   text_result = result.ToString("F20");
                   text_result = text_result.Substring(2, text_result.Length - 2);
                   text_result = text_result.TrimEnd('0');
                   long number = long.Parse(text_result);                       // Преобразование строки в целое число
                   text_result = number.ToString("X");
                   writer.Write(text_result);
                   text_result = "";
                   lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";              //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ,
ТО ОНА ПУСТАЯ)
                }
                else                                                            //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                {
                   text_result = result.ToString("F20");
                   text_result = text_result.Substring(2, text_result.Length - 2);
                   text_result = text_result.TrimEnd('0');
                   long number = long.Parse(text_result);                       // Преобразование строки в целое число
                   text_result = number.ToString("X");
                   writer.Write(text_result);
                   text_result = "";
                   lines = lines.Length > 15 ? lines.Substring(15) : "";
                }
             }
        result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=============================================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================
=====\n\n");



        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
        using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
        {
           using (StreamWriter writer = new StreamWriter(text_bw_lz77, false))
              while ((line = reader.ReadLine()) != null)
              {
                 text_in_line = line + "";//
                 List<object> list_Res = lz77(text_in_line, 5);
                 text_in_line = listToStr(list_Res);
                 writer.Write(text_in_line);
              }

        }
        Console.WriteLine("lz77 отработал, переходим к arifmetic...");
        using (StreamReader reader = new StreamReader(text_bw_lz77, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_bw_out))
           while ((lines = reader.ReadLine()) != null)
              while (lines.Length != 0)
                 if (lines.Length > 0)                                          //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                 {
                    text_alphabet_from_line(out chars, out percent, lines);
                    int step = Math.Min(lines.Length, 15);
                    result = arifm_code(lines.Substring(0, step), percent, chars, out error);                    //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ
СТРОКЕ
                    if (error != 0)                                             //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                    {
                       int nothing;                                             //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ
БУДЕТ)
                       text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));      //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО
СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
```

```
                    result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                    text_result = result.ToString("F20");
                    text_result = text_result.Substring(2, text_result.Length - 2);
                    text_result = text_result.TrimEnd('0');
                    long number = long.Parse(text_result);                    // Преобразование строки в целое число
                    text_result = number.ToString("X");
                    writer.Write(text_result);
                    text_result = "";
                    lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";          //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ,
ТО ОНА ПУСТАЯ)
                }
                else                                                  //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                {
                    text_result = result.ToString("F20");
                    text_result = text_result.Substring(2, text_result.Length - 2);
                    text_result = text_result.TrimEnd('0');
                    long number = long.Parse(text_result);                    // Преобразование строки в целое число
                    text_result = number.ToString("X");
                    writer.Write(text_result);
                    text_result = "";
                    lines = lines.Length > 15 ? lines.Substring(15) : "";
                }
            }
        result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("==========================================================================================
=====");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("==========================================================================================
=====\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
        using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
        {
            using (StreamWriter writer = new StreamWriter(text_color_lz77, false))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "́";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    writer.Write(text_in_line);
                }

        }
        Console.WriteLine("lz77 отработал, переходим к arifmetic...");
        using (StreamReader reader = new StreamReader(text_color_lz77, Encoding.UTF8))   //чтение построчно
        using (StreamWriter writer = new StreamWriter(text_color_out))
            while ((lines = reader.ReadLine()) != null)
                while (lines.Length != 0)
                    if (lines.Length > 0)                                 //ПОКА СТРОКА НЕ ЗАКОНЧИЛАСЬ
                    {
                        text_alphabet_from_line(out chars, out percent, lines);
                        int step = Math.Min(lines.Length, 15);
                        result = arifm_code(lines.Substring(0, step), percent, chars, out error);          //ПРОВЕРКА КОДИРОВКИ НА ВСЕЙ
СТРОКЕ
                        if (error != 0)                                   //ЕСЛИ СТРОКА НЕ ЗАКОДИЛАСЬ (ОШИБКА)
                        {
                            int nothing;                                  //ПЕРЕМЕННАЯ, КУДА ЗАПИШЕТСЯ ОШИБКА(КОТОРОЙ НЕ
БУДЕТ)
                            text_alphabet_from_line(out chars, out percent, lines.Substring(0, error - 1));      //ОБРАБОТКА ОБРЕЗАННОЙ СТРОКИ (ДО
СИМВОЛА, ГДЕ ВОЗНИКНЕТ ОШИБКА КОДИРОВАНИЯ)
                            result = arifm_code(lines.Substring(0, error - 1), percent, chars, out nothing);
                            text_result = result.ToString("F20");
                            text_result = text_result.Substring(2, text_result.Length - 2);
                            text_result = text_result.TrimEnd('0');
                            long number = long.Parse(text_result);                    // Преобразование строки в целое число
                            text_result = number.ToString("X");
                            writer.Write(text_result);
```

```
                    text_result = "";
                    lines = lines.Length > error - 1 ? lines.Substring(error - 1) : "";          //ОБРЕЗКА СТРОКИ (ЕСЛИ СТРОКА ЗАКОНЧИЛАСЬ, ТО ОНА ПУСТАЯ)
                }
                else                                                    //ЕСЛИ СТРОКА ОБРАБОТАЛАСЬ, ТО СОХРАНЯЕМ РЕЗУЛЬТАТЫ
                {
                    text_result = result.ToString("F20");
                    text_result = text_result.Substring(2, text_result.Length - 2);
                    text_result = text_result.TrimEnd('0');
                    long number = long.Parse(text_result);                  // Преобразование строки в целое число
                    text_result = number.ToString("X");
                    writer.Write(text_result);
                    text_result = "";
                    lines = lines.Length > 15 ? lines.Substring(15) : "";
                }
            }
        result = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("=============================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=============================================================================================================\n\n");


        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();



    }

}
}
```

2.9 LZ77+Алгоритм Хаффмана

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;
using static System.Net.Mime.MediaTypeNames;

namespace Lz77
{
   public class checking_compression
   {
      public static double get_checking_compression(string compressed_text, string main_text)
      {
         FileInfo compressed_text_info = new FileInfo(compressed_text);
         FileInfo main_text_info = new FileInfo(main_text);

         long compressed_text_size = compressed_text_info.Length;
         long main_text_size = main_text_info.Length;

         double compress_ratio = (double)compressed_text_size / main_text_size;

         return compress_ratio;
      }
   }
   class Node : IComparable<Node> //узел дерева
   {
      public char? Symbol { get; set; } //символ
      public int Frequency { get; set; } // вероятность
      public Node Left { get; set; } //левый ребенок
      public Node Right { get; set; } //правый ребенок

      public Node(char symbol, int frequency)
      {
         Symbol = symbol;
         Frequency = frequency;
      }

      public Node(int frequency, Node left, Node right) //узел который при слиянии 2х других
      {
         Frequency = frequency;
         Left = left;
         Right = right;
      }

      public int CompareTo(Node other) //сравнение по вероятности
      {
         return Frequency.CompareTo(other.Frequency);
      }
   }
   class PriorityQueue<T> where T : IComparable<T> //очередь с приориитетом с помощью бинарной кучи
   {
      private List<T> heap;

      public int Count { get { return heap.Count; } }

      public PriorityQueue()
      {
         heap = new List<T>();
```

```
    }

    public void Enqueue(T item) //добавление нового элемента в кучу
    {
        heap.Add(item); //добавляем элемент в конец
        int i = Count - 1;
        while (i > 0) //поднимаем наверх кучи пока элемент больше родителя
        {
            int parent = (i - 1) / 2;
            if (heap[parent].CompareTo(item) <= 0)
                break;
            heap[i] = heap[parent];
            i = parent;
        }
        heap[i] = item; //добавляем элемент на нудное место
    }

    public T Dequeue() //удаление и возвращение минимального элемента из кучи
    {
        if (Count == 0)
            throw new InvalidOperationException("Queue is empty");//куча пуста
        T item = heap[0];
        int i = Count - 1;
        T last = heap[i];
        heap.RemoveAt(i); //удаляем последний в куче элемент

        if (i > 0) //нужно перестроить кучу если в ней есть ещё элементы
        {
            int parent = 0;
            while (true)
            {
                int child = parent * 2 + 1; //индекс ребенка
                if (child >= i) //если его нет то стоп
                    break;
                if (child + 1 < i && heap[child + 1].CompareTo(heap[child]) < 0) //выбираем если правый меньше левого
                    child++;
                if (last.CompareTo(heap[child]) <= 0)
                    break;
                heap[parent] = heap[child];
                parent = child;
            }
            heap[parent] = last;
        }

        return item;
    }

}
class HuffmanCoding
{
    public static Dictionary<char, string> Encode(string str)
    {
        Dictionary<char, int> freq = frequencyMap(str); //словарь с вероятностью и символами
        PriorityQueue<Node> priorityQueue = priority_queue(freq); //очередб с приоритетом

        while (priorityQueue.Count > 1) //строим дерево склдывая 2 детей с наимельшей вероятностью
        {
            Node left = priorityQueue.Dequeue();
            Node right = priorityQueue.Dequeue();
            Node parent = new Node(left.Frequency + right.Frequency, left, right);
            priorityQueue.Enqueue(parent);
        }

        Node root = priorityQueue.Dequeue(); //извлекаем корень
        Dictionary<char, string> encodingMap = encodeMap(root); //теперь словарь не с вероятностями а с кодами
```

```
        return encodingMap;
    }

    public static string compress(string str, Dictionary<char, string> encode) // получаем строку по кодам хаффмана
    {
        return string.Concat(str.Select(c => encode[c]));
    }

    public static string Decompress(string str, Dictionary<char, string> encode)
    {
        Dictionary<string, char> decodingMap = encode.ToDictionary(pair => pair.Value, pair => pair.Key); //теперь каждому коду соответствует буква, а не наоборот
        string decoded = "";
        string currentCode = "";

        foreach (char bit in str)
        {
            currentCode += bit; //дописывает 0 или 1
            if (decodingMap.ContainsKey(currentCode)) //если в словаре есть такой символ то записываем его в расшифрованную строку
            {
                decoded += decodingMap[currentCode];
                currentCode = "";
            }
        }

        return decoded;
    }

    private static Dictionary<char, int> frequencyMap(string str)
    {
        Dictionary<char, int> frequencyMap = new Dictionary<char, int>();

        foreach (char c in str) //проходим по каждому элементу и смотрим сколько он встечается в тексте
        {
            if (frequencyMap.ContainsKey(c))
                frequencyMap[c]++;
            else
                frequencyMap[c] = 1;
        }

        return frequencyMap;
    }

    private static PriorityQueue<Node> priority_queue(Dictionary<char, int> freq)
    {
        PriorityQueue<Node> priorityQueue = new PriorityQueue<Node>();

        foreach (var entry in freq)
        {
            priorityQueue.Enqueue(new Node(entry.Key, entry.Value)); //просто добавляем в очередь с приоритетом все пары символ - вероятность в очередь
        }

        return priorityQueue;
    }

    private static Dictionary<char, string> encodeMap(Node root) //словарь на основе дерева хаффмана
    {
        Dictionary<char, string> encodingMap = new Dictionary<char, string>();
        encodeMap_Tree(root, "", encodingMap);
        return encodingMap;
    }
```

```
        private static void encodeMap_Tree(Node node, string code, Dictionary<char, string> encode) //рекурсивно из дерево берем коды
        {
            if (node.Symbol.HasValue)
            {
                encode[node.Symbol.Value] = code;
            }
            else
            {
                encodeMap_Tree(node.Left, code + "0", encode);
                encodeMap_Tree(node.Right, code + "1", encode);
            }
        }
    }
    internal class Program
    {
        static BitArray to_bit(BitArray resoult, string compressed)
        {
            int index = 0;
            foreach (char chars in compressed)
            {
                if (chars == '1')
                    resoult[index] = true;
                else if (chars == '0')
                    resoult[index] = false;
                index++;
            }
            if (index < resoult.Length)
            {
                BitArray cut = new BitArray(index);         //обрезка лишней части строки
                for (int i = 0; i < index; i++)
                    cut[i] = resoult[i];
                resoult = cut;
            }
            return resoult;
        }
        public struct triplet
        {
            public int prev;            // Смещение назад в строке (отступ влево)
            public int length;          // Длина подстроки
            public char next;           // Символ, следующий за подстрокой

            // Конструктор для инициализации структуры triplet
            public triplet(int prev, int length, char next)
            {
                this.prev = prev;
                this.length = length;
                this.next = next;
            }
            public void print_node()
            {
                Console.WriteLine($"Нода: ({prev},{length},{next})");
                Console.WriteLine($"Отступить влево на {prev} символов; Длина подстроки: {length}; Символ справа: {next}\n");
            }
        }

        // Функция для проверки, является ли объект триплетом с одним символом
        static public bool bool_triplet_one_symbol(object obj)
        {
            return obj is triplet_one_symbol;
        }

        // Структура для хранения информации о триплете с одним символом
        public struct triplet_one_symbol
        {
```

```
    public char next;           // Символ справа


    public triplet_one_symbol(char symbol)
    {
        this.next = symbol;
    }
    public void print_node() // для вывода
    {
        Console.WriteLine($"Нода: ({next})");
        Console.WriteLine($"Отступить влево на {0} символов; Длина подстроки: {0}; Символ справа: {next}\n");
    }
}

static List<object> lz77(string str, int bufer_size)
{
    List<object> triplet_list = new List<object>();
    string bufer = "";
    int pos = 0;
    int prev_symbols_triplet = 0;
    int length_triplet = 0;
    char next_symbol_triplet = '\0';

    while (pos < str.Length)                //идем по строке
    {
        // Получаем текущий буфер
        bufer = str.Substring(Math.Max(pos - bufer_size, 0), pos - Math.Max(pos - bufer_size, 0));
        int index_substr_bufer = bufer.Length;
        int length_substring = 0;

        // Ищем самую длинную повторяющуюся подстроку в буфере
        for (int i = 1; i < bufer.Length + 1; i++)
        {
            if ((bufer.LastIndexOf(str.Substring(pos, i)) != -1))
            {
                // Если нашли, пытаемся найти подстроку на элемент больше
                index_substr_bufer = bufer.LastIndexOf(str.Substring(pos, i));
                length_substring = i;
            }
            else
                break;
        }

        // Сохраняем сдвиг и длину подстроки
        prev_symbols_triplet = bufer.Length - index_substr_bufer;
        length_triplet = length_substring;

        int pos_repeat = pos;
        int length_substring_repeat = length_substring;
        if (length_substring != 0)
        {
            // Проверяем повторки, если они есть, записываем как можно длиннее
            while (str[pos_repeat] == str[pos_repeat + length_substring])
            {
                pos_repeat++;
                length_substring_repeat++;

            }
            if (pos_repeat - pos > 1) // Если повторка 1, то это не повтор
            {
                pos = pos_repeat;
                length_triplet = length_substring_repeat;
            }
        }
```

```
        // Записываем следующий символ
        next_symbol_triplet = str[pos + length_substring];

        pos += length_substring + 1;

        // Добавляем триплет в список
        if (prev_symbols_triplet == 0 && length_triplet == 0)
        {
            triplet_one_symbol triplet = new triplet_one_symbol(next_symbol_triplet);
            triplet_list.Add(triplet);

        }
        else
        {
            triplet triplet = new triplet(prev_symbols_triplet, length_triplet, next_symbol_triplet);
            triplet_list.Add(triplet);

        }

    }
    return triplet_list; // возвращаем список триплетов
}

static public string listToStr(List<object> list_result)
{
    string res = "";
    int count = 0;
    foreach (object list in list_result)
    {
        if (bool_triplet_one_symbol(list))
        {
            triplet_one_symbol triplet = (triplet_one_symbol)list;
            res += triplet.next;
            count += 1;
        }
        else
        {
            triplet triplet = (triplet)list;
            res += triplet.prev;
            res += triplet.length;
            res += triplet.next;
        }
    }
    return res;
}

static void Main(string[] args)
{
    string text_russian = "rusTolstoy.txt";
    string text_russian_out = "rusTolstoy_out.txt";

    string text_enwik7 = "enwik7.txt";
    string text_enwik7_out = "enwik7_out.txt";

    string text_gray = "Gray_image.txt";
    string text_gray_out = "Gray_image_out.txt";

    string text_bw = "BW_image.txt";
    string text_bw_out = "BW_image_out.txt";

    string text_color = "Color_image.txt";
    string text_color_out = "Color_image_out.txt";


    Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
```

```
        string line;
        string text_in_line = "";
        using (StreamReader reader = new StreamReader(text_russian, Encoding.UTF8))
        {
            using (FileStream writer = new FileStream(text_russian_out, FileMode.Create, FileAccess.Write))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    string text = "";
                    int index = 0;
                    while (index < text_in_line.Length)
                    {
                        int step = Math.Min(1000, text_in_line.Length - index);
                        text_in_line = text_in_line.Substring(index, step);

                        var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                        string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                        BitArray res = new BitArray(compressed.Length * 8);
                        res = to_bit(res, compressed);
                        byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                        res.CopyTo(bytes, 0);

                        writer.Write(bytes, 0, bytes.Length);
                        index += step;
                    }

                }

        }

        double result = checking_compression.get_checking_compression(text_russian_out, text_russian);

Console.WriteLine("================================================================================
================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("================================================================================
================\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_enwik7, Encoding.UTF8))
        {
            using (FileStream writer = new FileStream(text_enwik7_out, FileMode.Create, FileAccess.Write))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    int index = 0;
                    while (index < text_in_line.Length)
                    {
                        int step = Math.Min(1000, text_in_line.Length - index);
                        text_in_line = text_in_line.Substring(index, step);

                        var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                        string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                        BitArray res = new BitArray(compressed.Length * 8);
                        res = to_bit(res, compressed);
```

```
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
            }
        }

    }

    result = checking_compression.get_checking_compression(text_enwik7_out, text_enwik7);

Console.WriteLine("=================================================================================
=================");
    Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=================================================================================
=================\n\n");




    Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");
    line = "";
    text_in_line = "";
    using (StreamReader reader = new StreamReader(text_gray, Encoding.UTF8))
    {
       using (FileStream writer = new FileStream(text_gray_out, FileMode.Create, FileAccess.Write))
          while ((line = reader.ReadLine()) != null)
          {
             text_in_line = line + "́";//
             List<object> list_Res = lz77(text_in_line, 5);
             text_in_line = listToStr(list_Res);
             int index = 0;
             while (index < text_in_line.Length)
             {
                int step = Math.Min(1000, text_in_line.Length - index);
                text_in_line = text_in_line.Substring(index, step);

                var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                BitArray res = new BitArray(compressed.Length * 8);
                res = to_bit(res, compressed);
                byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                res.CopyTo(bytes, 0);

                writer.Write(bytes, 0, bytes.Length);
                index += step;
             }
          }

    }

    result = checking_compression.get_checking_compression(text_gray_out, text_gray);

Console.WriteLine("=================================================================================
=================");
    Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=================================================================================
=================\n\n");
```

```
        Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_bw, Encoding.UTF8))
        {
            using (FileStream writer = new FileStream(text_bw_out, FileMode.Create, FileAccess.Write))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "́";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    int index = 0;
                    while (index < text_in_line.Length)
                    {
                        int step = Math.Min(1000, text_in_line.Length - index);
                        text_in_line = text_in_line.Substring(index, step);

                        var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                        string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                        BitArray res = new BitArray(compressed.Length * 8);
                        res = to_bit(res, compressed);
                        byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
                        res.CopyTo(bytes, 0);

                        writer.Write(bytes, 0, bytes.Length);
                        index += step;
                    }
                }

        }

        result = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("================================================================================
=================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("================================================================================
=================\n\n");




        Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
        line = "";
        text_in_line = "";
        using (StreamReader reader = new StreamReader(text_color, Encoding.UTF8))
        {
            using (FileStream writer = new FileStream(text_color_out, FileMode.Create, FileAccess.Write))
                while ((line = reader.ReadLine()) != null)
                {
                    text_in_line = line + "́";//
                    List<object> list_Res = lz77(text_in_line, 5);
                    text_in_line = listToStr(list_Res);
                    int index = 0;
                    while (index < text_in_line.Length)
                    {
                        int step = Math.Min(1000, text_in_line.Length - index);
                        text_in_line = text_in_line.Substring(index, step);

                        var encode_map = HuffmanCoding.Encode(text_in_line); //словарь с кодами
                        string compressed = HuffmanCoding.compress(text_in_line, encode_map);
                        BitArray res = new BitArray(compressed.Length * 8);
                        res = to_bit(res, compressed);
                        byte[] bytes = new byte[(compressed.Length - 1) / 8 + 1];
```

```
                    res.CopyTo(bytes, 0);

                    writer.Write(bytes, 0, bytes.Length);
                    index += step;
                }
            }

        }

        result = checking_compression.get_checking_compression(text_color_out, text_color);

Console.WriteLine("=================================================================================================");
        Console.WriteLine($"Результат сжатия: {result}");

Console.WriteLine("=================================================================================================\n\n");



        Console.WriteLine("\t\t\tПрограмма отработала");
        Console.ReadKey();
    }
  }
}
```

2.10 RLE

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RLE
{
    public class checking_compression
    {
        public static double get_checking_compression(string compressed_text, string
main_text)
        {
            FileInfo compressed_text_info = new FileInfo(compressed_text);
            FileInfo main_text_info = new FileInfo(main_text);

            long compressed_text_size = compressed_text_info.Length;
            long main_text_size = main_text_info.Length;

            double compress_ratio = (double)compressed_text_size / main_text_size;

            return compress_ratio;
        }
    }
    internal class Program
    {
        static void rle(string text, string text_out)
        {
            using (StreamReader reader = new StreamReader(text, Encoding.UTF8)) //чтение
построчно
                using (StreamWriter writer = new StreamWriter(text_out, false))
                {
                    char current_char = '0';
                    string line;
                    string temp_str = "";

                    while ((line = reader.ReadLine()) != null) // пптриааа
                    {
                        int count_repeat = 1;               //счетчик повторных символов
                        int index_in_str = 0;               //индекс прохода по строке

                        while (index_in_str < line.Length - 2)          //идем по строе
                        {
                            current_char = line[index_in_str];          //текущий символ
                            if (current_char == line[index_in_str + 1]) //если символ
повтояется
                            {
                                count_repeat++;
                                index_in_str++;
                            }
                            else
                            {
                                if (count_repeat == 1)                  //если символ ни
разу не повторился
                                {
                                    temp_str += line[index_in_str];
                                    index_in_str++;
                                    while (line[index_in_str] != line[index_in_str - 1] &&
index_in_str != line.Length - 1)
                                    {
```

```
                                    temp_str += line[index_in_str];     //пока символы
не повторяются
                                    index_in_str++;
                                }
                                index_in_str--;                         //вычитаем
индекс символа, который повторился
                                temp_str = temp_str.Substring(0, temp_str.Length - 1);
//вырезаем из строки символ
                                writer.Write('&');
                                writer.Write(temp_str);     //вставляем строку
неповторяющихся символов между разделителями
                                writer.Write('&');
                                temp_str = "";
                            }
                            else
                            {
                                writer.Write((char)(count_repeat));     //записываем
символ повторок
                                writer.Write(current_char);             //записываем
текущий символ
                                count_repeat = 1;
                                index_in_str++;
                            }
                        }

                    }

                    writer.Write((char)(count_repeat));
                    writer.Write(current_char);
                    writer.Write('\n');
                }
            }
        }


        static void Main(string[] args)
        {
            string text_russian = "rusTolstoy.txt";
            string text_russian_out = "rusTolstoy_out.txt";

            string text_enwik7 = "enwik7.txt";
            string text_enwik7_out = "enwik7_out.txt";

            string text_gray = "Gray_image.txt";
            string text_gray_out = "Gray_image_out.txt";

            string text_bw = "BW_image.txt";
            string text_bw_out = "BW_image_out.txt";

            string text_color = "Color_image.txt";
            string text_color_out = "Color_image_out.txt";


            double resoult = 0;

            Console.WriteLine($"\t\t\tРабота с текстом: {text_russian}");
            rle(text_russian, text_russian_out);
            resoult = checking_compression.get_checking_compression(text_russian_out,
text_russian);

Console.WriteLine("================================================================================
========================") ;
            Console.WriteLine($"Результат сжатия: {resoult}");
```

```
Console.WriteLine("================================================================================
========================\n\n");


            Console.WriteLine($"\t\t\tРабота с текстом: {text_enwik7}");
            rle(text_enwik7, text_enwik7_out);
            resoult = checking_compression.get_checking_compression(text_enwik7_out,
text_enwik7);

Console.WriteLine("================================================================================
========================");
            Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("================================================================================
========================\n\n");


            Console.WriteLine($"\t\t\tРабота с текстом: {text_gray}");
            rle(text_gray, text_gray_out);
            resoult = checking_compression.get_checking_compression(text_gray_out,
text_gray);

Console.WriteLine("================================================================================
========================");
            Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("================================================================================
========================\n\n");


            Console.WriteLine($"\t\t\tРабота с текстом: {text_bw}");
            rle(text_bw, text_bw_out);
            resoult = checking_compression.get_checking_compression(text_bw_out, text_bw);

Console.WriteLine("================================================================================
========================");
            Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("================================================================================
========================\n\n");



            Console.WriteLine($"\t\t\tРабота с текстом: {text_color}");
            rle(text_color, text_color_out);
            resoult = checking_compression.get_checking_compression(text_color_out,
text_color);

Console.WriteLine("================================================================================
========================");
            Console.WriteLine($"Результат сжатия: {resoult}");

Console.WriteLine("================================================================================
========================\n\n");

            Console.WriteLine("\t\t\tПрограмма отработала");
            Console.ReadKey();
        }
    }
}
```

3. Ссылка на Гит

гитхаб