Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.04 – «Программная инженерия»

Дисциплина: «Защита информации»

Профиль: «Разработка программно-информационных систем»

Семестр 5

ОТЧЕТ

по лабораторной работе №6

Тема: «Методы сжатия информации»

Выполнил: студент группы АСУ-19-1б

Шеретов М.А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: старший преподаватель

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2022

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по применению различных методов сжатия информации. Получить сравнительную характеристику сжатия информации, используя различные комбинации методов. Сделать вывод.

**ЗАДАНИЕ**

Выполнить первое сжатие файла способом кодирования серий (RLE), и повторное, используя алгоритм Хаффмана.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Сжатие информации - проблема, имеющая достаточно давнюю историю, гораздо более давнюю, нежели история развития вычислительной техники, которая (история) обычно шла параллельно с историей развития проблемы кодирования и шифровки информации.

Все алгоритмы сжатия оперируют входным потоком информации, минимальной единицей которой является бит, а максимальной - несколько бит, байт или несколько байт.

Целью процесса сжатия, как правило, есть получение более компактного выходного потока информационных единиц из некоторого изначально некомпактного входного потока при помощи некоторого их преобразования.

Основными техническими характеристиками процессов сжатия и результатов их работы являются:

* степень сжатия (compressrating) или отношение (ratio) объемов исходного и результирующего потоков;
* скорость сжатия - время, затрачиваемое на сжатие некоторого объема информации входного потока, до получения из него эквивалентного выходного потока;
* качество сжатия - величина, показывающая на сколько сильно упакован выходной поток, при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Все способы сжатия можно разделить на две категории: обратимое и необратимое сжатие.

Под необратимым сжатием подразумевают такое преобразование входного потока данных, при котором выходной поток, основанный на определенном формате информации, представляет, с некоторой точки зрения, достаточно похожий по внешним характеристикам на входной поток объект, однако отличается от него объемом.

Степень сходства входного и выходного потоков определяется степенью соответствия некоторых свойств объекта (т.е. сжатой и несжатой информации в соответствии с некоторым определенным форматом данных), представляемого данным потоком информации.

Такие подходы и алгоритмы используются для сжатия, например данных растровых графических файлов с низкой степенью повторяемости байтов в потоке. При таком подходе используется свойство структуры формата графического файла и возможность представить графическую картинку приблизительно схожую по качеству отображения (для восприятия человеческим глазом) несколькими (а точнее n) способами. Поэтому, кроме степени или величины сжатия, в таких алгоритмах возникает понятие качества, т.к. исходное изображение в процессе сжатия изменяется, то под качеством можно понимать степень соответствия исходного и результирующего изображения, оцениваемая субъективно, исходя из формата информации. Для графических файлов такое соответствие определяется визуально, хотя имеются и соответствующие интеллектуальные алгоритмы и программы. Необратимое сжатие невозможно применять в областях, в которых необходимо иметь точное соответствие информационной структуры входного и выходного потоков. Данный подход реализован в популярных форматах представления видео и фото информации, известных как JPEG и JFIF алгоритмы и JPG и JIF форматы файлов.

Обратимое сжатие всегда приводит к снижению объема выходного потока информации без изменения его информативности, т.е. - без потери информационной структуры.

Более того, из выходного потока, при помощи восстанавливающего или декомпрессирующего алгоритма, можно получить входной, а процесс восстановления называется декомпрессией или распаковкой и только после процесса распаковки данные пригодны для обработки в соответствии с их внутренним форматом.

Перейдем теперь непосредственно к алгоритмическим особенностям обратимых алгоритмов и рассмотрим важнейшие теоретические подходы к сжатию данных, связанные с реализацией кодирующих систем и способы сжатия информации.

**Сжатие способом кодирования серий (RLE)**

Наиболее известный простой подход и алгоритм сжатия информации обратимым путем - это кодирование серий последовательностей (RunLengthEncoding - RLE).

Суть методов данного подхода состоит в замене цепочек или серий повторяющихся байтов или их последовательностей на один кодирующий байт и счетчик числа их повторений.

Например:

44 44 44 11 11 11 11 01 33 FF 22 22 - исходная последовательность

03 44 04 11 00 03 01 03 FF 02 22 - сжатая последовательность

Первый байт указывает сколько раз нужно повторить следующий байт

Если первый байт равен 00, то затем идет счетчик, показывающий сколько за ним следует неповторяющихся данных.

Данные методы, как правило, достаточно эффективны для сжатия растровых графических изображений (BMP, PCX, TIF, GIF), т.к. последние содержат достаточно много длинных серий повторяющихся последовательностей байтов.

Недостатком метода RLE является достаточно низкая степень сжатия.

**Алгоритм Хаффмана**

Сжимая файл по алгоритму Хаффмана первое что мы должны сделать - это необходимо прочитать файл полностью и подсчитать сколько раз встречается каждый символ из расширенного набора ASCII.

Если мы будем учитывать все 256 символов, то для нас не будет разницы в сжатии текстового и EXE файла.

После подсчета частоты вхождения каждого символа, необходимо просмотреть таблицу кодов ASCII и сформировать бинарное дерево.

Пример:

Мы имеем файл длинной в 100 байт и имеющий 6 различных символов в себе . Мы подсчитали вхождение каждого из символов в файл и получили следующее :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Символ | A | B | C | D | E | F |
| Число вхождений | 10 | 20 | 30 | 5 | 25 | 10 |

Теперь мы берем эти числа и будем называть их частотой вхождения для каждого символа.

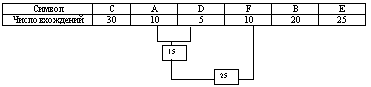
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Символ | C | E | B | F | A | D |
| Число вхождений | 30 | 25 | 20 | 10 | 10 | 5 |

Мы возьмем из последней таблицы 2 символа с наименьшей частотой. В нашем случае это D (5) и какой либо символ из F или A (10), можно взять любой из них например A.

Сформируем из "узлов" D и A новый "узел", частота вхождения для которого будет равна сумме частот D и A :

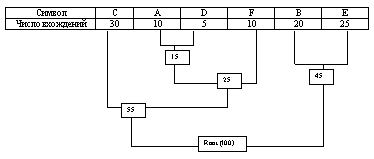
arch_003.gif (850 bytes)

Номер в рамке - сумма частот символов D и A. Теперь мы снова ищем два символа с самыми низкими частотами вхождения. Исключая из просмотра D и A и рассматривая вместо них новый "узел" с суммарной частотой вхождения. Самая низкая частота теперь у F и нового "узла". Снова сделаем операцию слияния узлов :



Рассматриваем таблицу снова для следующих двух символов ( B и E ).

Мы продолжаем в этот режим пока все "дерево" не сформировано, т.е. пока все не сведется к одному узлу.



Теперь когда наше дерево создано, мы можем кодировать файл . Мы должны всегда начинать из корня ( Root ). Кодируя первый символ (лист дерева С) Мы прослеживаем вверх по дереву все повороты ветвей и если мы делаем левый поворот, то запоминаем 0-й бит, и аналогично 1-й бит для правого поворота. Так для C, мы будем идти влево к 55 ( и запомним 0 ), затем снова влево (0) к самому символу . Код Хаффмана для нашего символа C - 00. Для следующего символа ( А ) у нас получается - лево,право,лево,лево , что выливается в последовательность 0100. Выполнив выше сказанное для всех символов получим

C = 00 ( 2 бита )  
A = 0100 ( 4 бита )  
D = 0101 ( 4 бита )  
F = 011 ( 3 бита )  
B = 10 ( 2 бита )  
E = 11 ( 2 бита )

При кодировании заменяем символы на данные последовательности.

**ХОД РАБОТЫ**

Окно программы представлено на рисунке 1.

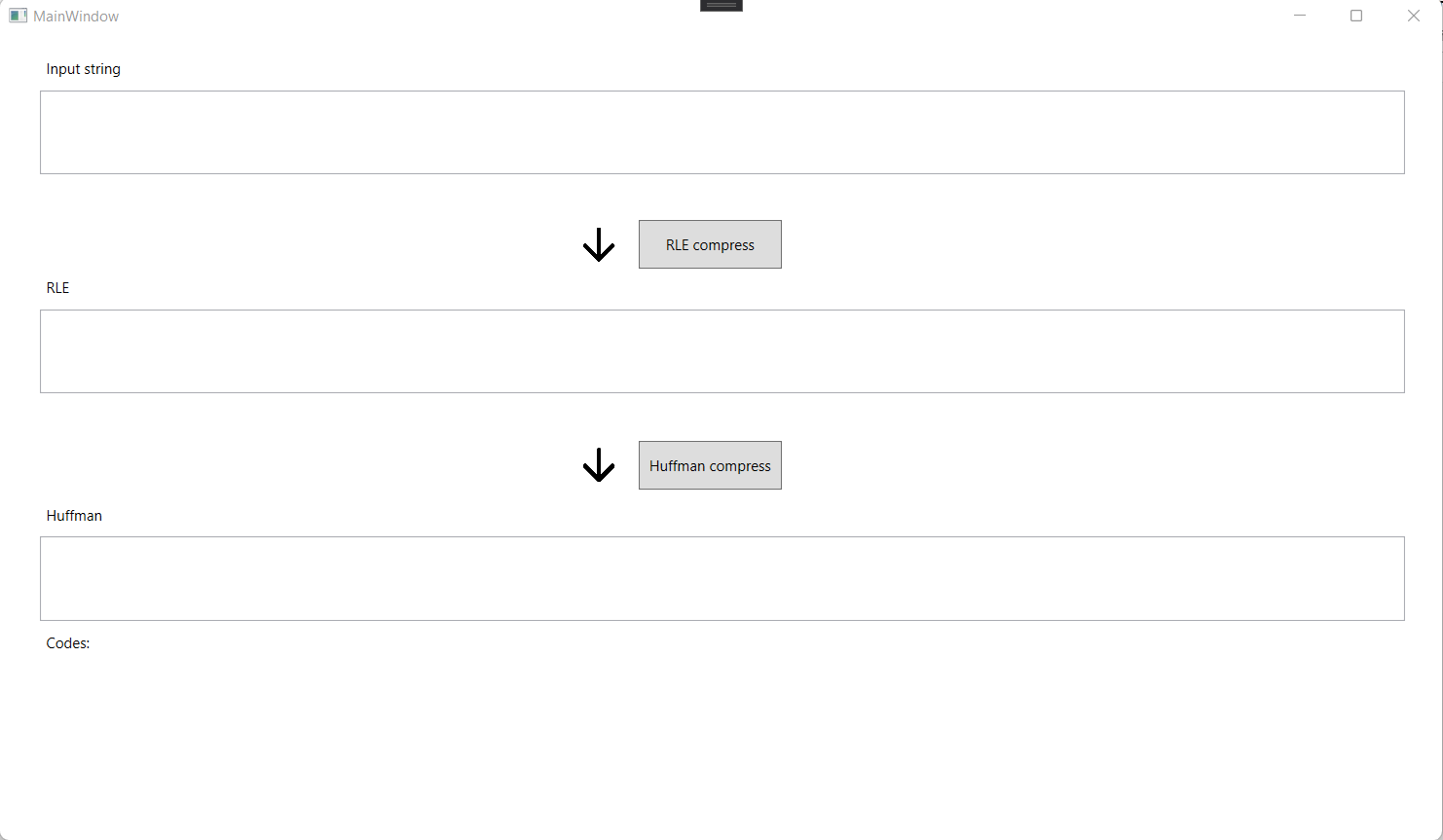


Рисунок 1. Окно программы

В первое поле вводится первоначальная строка, которую необходимо сжать. При нажатии на кнопку RLE compress строка будет преобразована в массив байт и сжата в соответствии с алгоритмом RLE. Таким образом будет получен новый массив байт, который запишется в поле RLE. При этом, если повторяющихся подряд символов больше 255 (число значений байта), то будет выполнен переход к следующей паре (то есть счетчик сбросится и кол-во символов будет считаться заново).

После этого можно нажать на кнопку Huffman compress. Произойдет сжатие с помощью алгоритма Хаффмана. Будет создано бинарное дерево с кодами для каждого байта (или символа). Далее каждый байт будет заменен на код и записан в поле Huffman. Коды символов будут представлены ниже под полем.

Также, после серии сжатий будет выведена информация об объеме информации (в битах) на каждой стадии сжатия (до сжатия, RLE, алг. Хаффмана) и во сколько раз уменьшился объем информации.

Примеры работы программы представлены на рисунках 2 и 3.

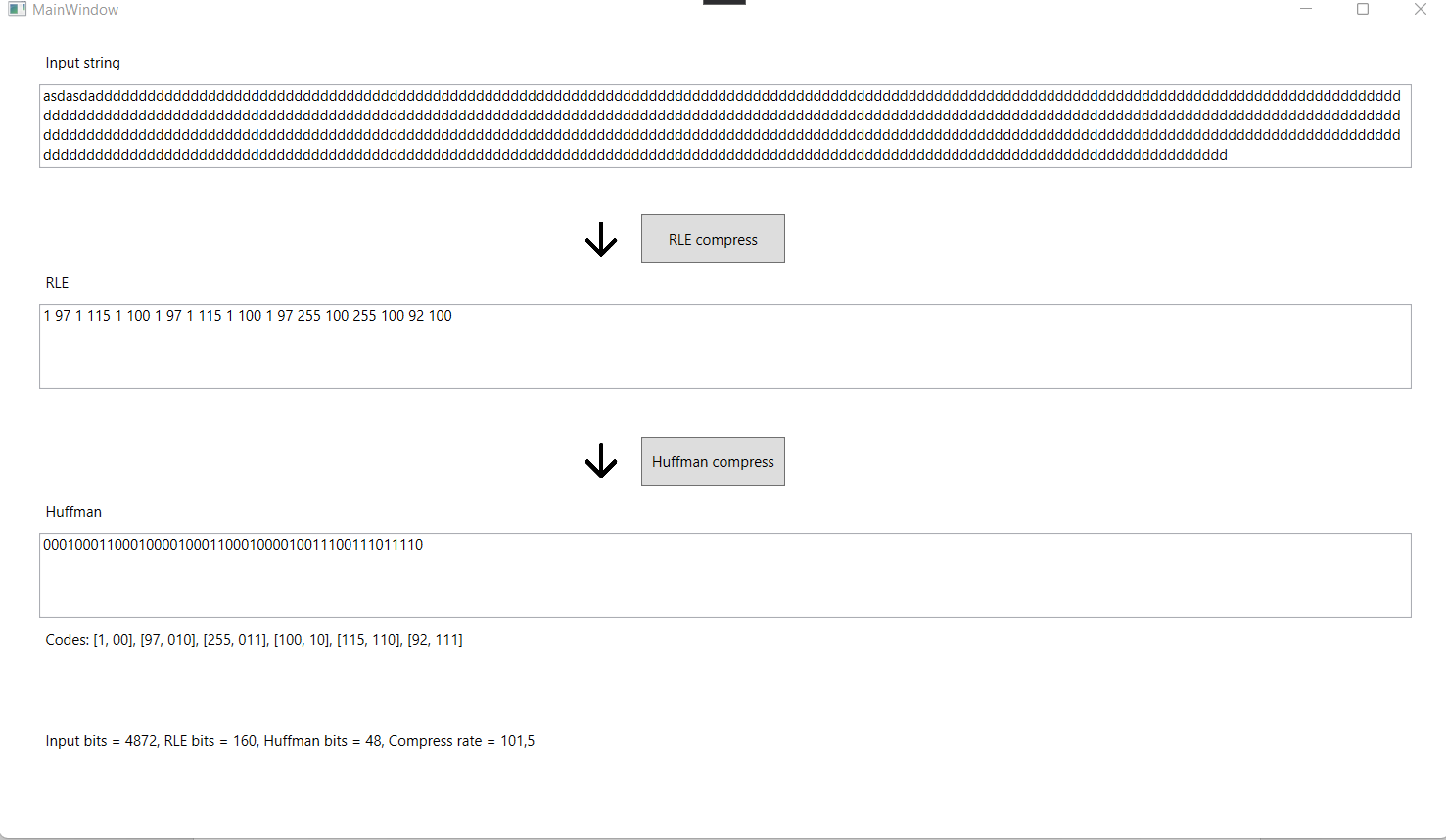


Рисунок 2. Пример работы программы 1

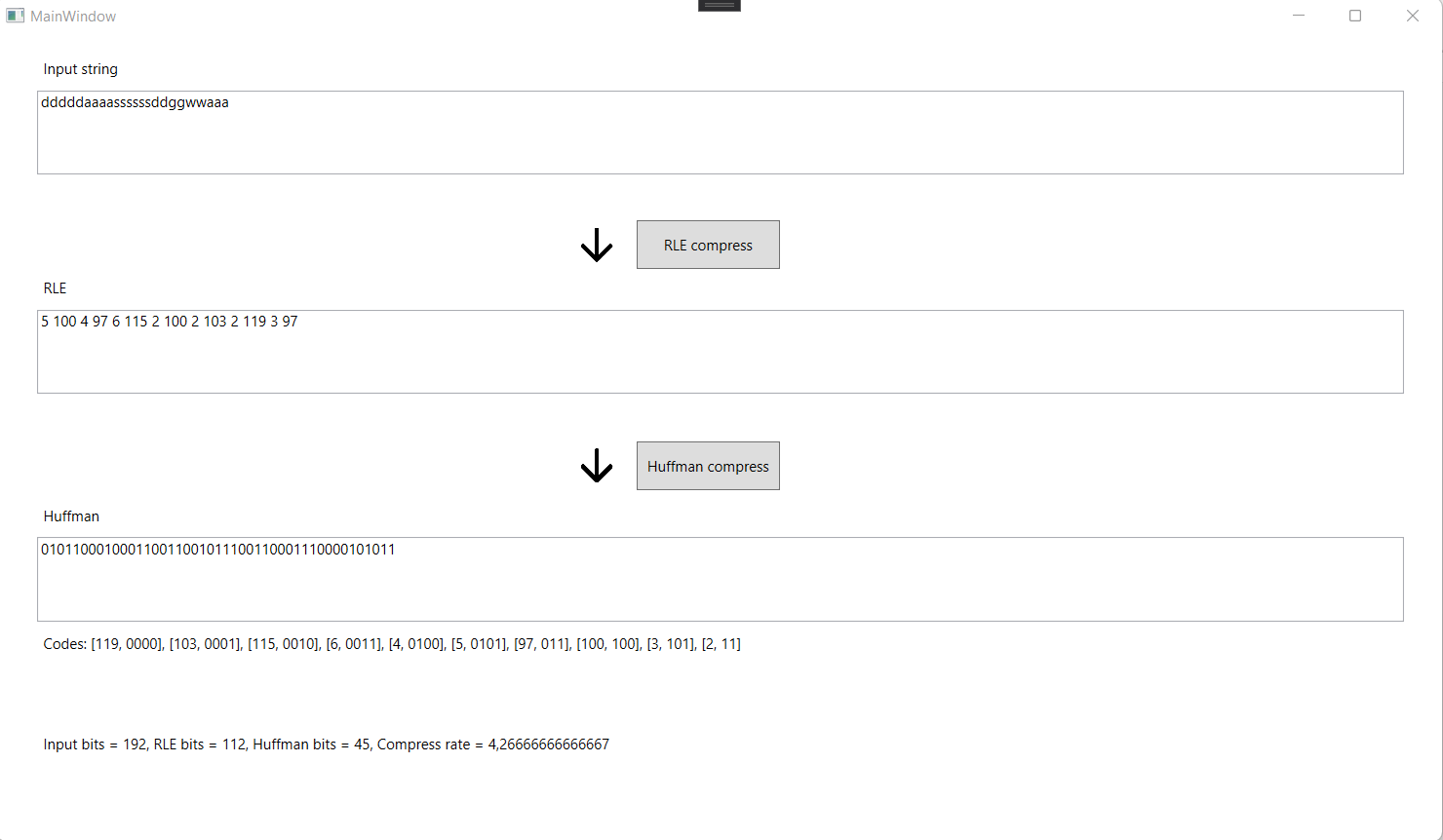


Рисунок 3. Пример работы программы 2

**ПРИЛОЖЕНИЕ А**

**Листинг А1 – Класс RLE**

class RLE

{

public byte[] Compress(byte[] data)

{

List<byte> compressed = new List<byte>();

int i = 0;

int len = data.Length;

while (i <= len-1)

{

byte counter = 1;

byte current = data[i];

var j = i;

while (j < len-1)

{

if (data[j] == data[j + 1] && counter != 255)

{

counter++;

j++;

}

else

{

break;

}

}

compressed.Add(counter);

compressed.Add(current);

i = j + 1;

}

return compressed.ToArray();

}

}

**Листинг А2 – Классы Huffman и HNode**

public class HNode

{

public int probability;

public byte value;

public HNode left;

public HNode right;

public string code = "";

public HNode(int probability, byte value, HNode left = null, HNode right = null)

{

this.probability = probability;

this.value = value;

this.left = left;

this.right = right;

}

}

class Huffman

{

private Dictionary<byte, int> Probs(byte[] bytes)

{

var probsMap = new Dictionary<byte, int>();

foreach (var b in bytes)

{

if (probsMap.ContainsKey(b))

{

probsMap[b]++;

}

else

{

probsMap[b] = 1;

}

}

return probsMap;

}

private Dictionary<byte, string> Codes(HNode n, string code, Dictionary<byte, string> codes)

{

string nCode = code + n.code;

if (n.left != null)

{

codes = Codes(n.left, nCode, codes);

}

if (n.right != null)

{

codes = Codes(n.right, nCode, codes);

}

if (n.left == null && n.right == null)

{

codes[n.value] = nCode;

}

return codes;

}

private string Encode(byte[] bytes, Dictionary<byte, string> codes)

{

List<string> enc = new List<string>();

foreach (var b in bytes)

{

enc.Add(codes[b]);

}

return string.Join("", enc);

}

public (string, Dictionary<byte, string>) Compress(byte[] bytes)

{

var probs = Probs(bytes);

List<HNode> nodes = new List<HNode>();

foreach (var p in probs)

{

nodes.Add(new HNode(p.Value, p.Key));

}

while (nodes.Count > 1)

{

nodes.Sort((x, y) => x.probability - y.probability);

var r = nodes[0];

var l = nodes[1];

r.code = "1";

l.code = "0";

var newNode = new HNode(l.probability + r.probability, 0, l, r);

nodes.RemoveAt(0);

nodes.RemoveAt(0);

nodes.Add(newNode);

}

var codes = Codes(nodes[0], "", new Dictionary<byte, string>());

var encoded = Encode(bytes, codes);

return (encoded, codes);

}

}

**Листинг А3 – Класс MainWindow**

public partial class MainWindow : Window

{

RLE rle = new RLE();

Huffman hf = new Huffman();

int bitsInput = -1;

int bitsRLE = -1;

int bitsHuffman = -1;

public MainWindow()

{

InitializeComponent();

}

private string BytesArrayToString(byte[] bytes)

{

return string.Join(" ", bytes.Select((b) => b.ToString()));

}

private int GetInputBits(string input)

{

return input.Length \* 8;

}

private int GetRLEBits(byte[] input)

{

return input.Length \* 8;

}

private int GetHuffmanBits(string binary)

{

return binary.Length;

}

private void SetCompressRate()

{

double total = -1;

if (bitsInput != -1 && bitsHuffman != -1)

{

total = (double)bitsInput / bitsHuffman;

}

compressRate.Content = $"Input bits = {bitsInput}, RLE bits = {bitsRLE}, Huffman bits = {bitsHuffman}, Compress rate = {total}";

}

private void OnRLEClick(object sender, RoutedEventArgs e)

{

if (InputBox.Text == "") return;

var rleCompressed = rle.Compress(Encoding.ASCII.GetBytes(InputBox.Text));

RLEBox.Text = BytesArrayToString(rleCompressed);

bitsInput = GetInputBits(InputBox.Text);

bitsRLE = GetRLEBits(rleCompressed);

SetCompressRate();

}

private void OnHuffmanClick(object sender, RoutedEventArgs e)

{

if (RLEBox.Text == "") return;

var bytes = RLEBox.Text.Split(' ').Select((s) => byte.Parse(s)).ToArray();

var (hfCompressed, codes) = hf.Compress(bytes);

var asStr = string.Join(", ", codes);

HuffmanBox.Text = hfCompressed;

codesLabel.Content = $"Codes: {asStr}";

bitsHuffman = GetHuffmanBits(hfCompressed);

SetCompressRate();

}

}

**Листинг А4 – Интерфейс**

<Window x:Class="zi6.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:zi6"

mc:Ignorable="d"

Title="MainWindow" Height="700" Width="1200">

<Grid>

<TextBox x:Name="InputBox" HorizontalAlignment="Left" Height="69" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="1121" Margin="33,46,0,0"/>

<Button x:Name="RLEButton" Content="RLE compress" HorizontalAlignment="Left" VerticalAlignment="Top" Width="117" Margin="525,153,0,0" Click="OnRLEClick" Height="40"/>

<Button x:Name="HuffmanButton" Content="Huffman compress" HorizontalAlignment="Left" VerticalAlignment="Top" Width="117" Margin="525,334,0,0" Click="OnHuffmanClick" Height="40"/>

<TextBox x:Name="RLEBox" HorizontalAlignment="Left" Height="69" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="1121" Margin="33,226,0,0"/>

<TextBox x:Name="HuffmanBox" HorizontalAlignment="Left" Height="69" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="1121" Margin="33,413,0,0"/>

<Label Content="Input string" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="33,15,0,0" Width="139"/>

<Label Content="RLE" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="33,195,0,0" Width="139"/>

<Label Content="Huffman" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="33,382,0,0" Width="139"/>

<Label x:Name="codesLabel" Content="Codes: " HorizontalAlignment="Left" VerticalAlignment="Top" Margin="33,487,0,0" Width="1121" Height="69"/>

<Label x:Name="compressRate" Content="" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="33,569,0,0" Width="1121" Height="69"/>

<Image Source="arrow-down.png" HorizontalAlignment="Left" Height="40" VerticalAlignment="Top" Width="42" Margin="472,153,0,0"/>

<Image Source="arrow-down.png" HorizontalAlignment="Left" Height="40" VerticalAlignment="Top" Width="48" Margin="472,334,0,0"/>

</Grid>

</Window>