МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение   
высшего образования

«ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ЭНЕРГЕТИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ В.И. ЛЕНИНА»

Кафедра программного обеспечения компьютерных систем

**Пояснительная записка к курсовому проекту**

«Статический алгоритм Хаффмана»

Дисциплина: Алгоритмы и структуры данных

Выполнил студент группы 3-41 Н.В. Ткачёв

подпись, дата

Проверил доктор техн. наук Е.Р. Пантелеев

подпись, дата

Оценка:

Иваново 2020

**Задание**

Написать кодер, использующий статический алгоритм Хаффмана. Исследовать его эффективность (степень сжатия) в зависимости от типа и размера сжимаемых файлов.

**Допущения**

Список принятых допущений на этапе разработки прототипа программы:

* 1. Проблема передачи модели кодирования.
     + Допущение: кодер и декодер используют общую оперативную память, где хранится модель кодирования.
     + Способ решения проблемы после снятия допущения: передача вектора длин кодов в заголовке выходного потока кодера.

1. Проблема определения конца файла для декодера:
   * + Допущение: кодер и декодер используют общую оперативную память, где хранится длина входного потока кодера.
     + Способ решения проблемы после снятия допущения: включить в алфавит входного потока кодера служебный символ eof (endoffile) с частотой 0.
2. Проблема организации чтения и записи кодов переменной длины:
   * + Допущение: 1 бит выходного потока кодера - это 1 байт.
     + Способ решения проблемы после снятия допущения: применить структуру данных блочного типа для организации записи и чтения кодов переменной длины.

**Содержание**

[Описание алгоритма 4](#_Toc40700094)

[1. Таблицы частот 4](#_Toc40700095)

[2. Модель кодирования 4](#_Toc40700096)

[3. Кодирование 5](#_Toc40700097)

[4. Декодирование 6](#_Toc40700098)

[Разработка быстрого прототипа 6](#_Toc40700099)

[Скриншоты работы приложения 6](#_Toc40700100)

[Выводы 6](#_Toc40700101)

[Разработка оптимизированной версии 7](#_Toc40700102)

[План отмены допущений 7](#_Toc40700103)

[UML диаграмма активности 7](#_Toc40700104)

[Добавление служебного символа eof 8](#_Toc40700105)

[Передача модели кодирования 8](#_Toc40700106)

[Добавление структуры блочного типа 8](#_Toc40700107)

[Скриншоты работы приложения 9](#_Toc40700108)

[Обратимость 11](#_Toc40700109)

[Выводы 13](#_Toc40700110)

[Руководства пользователя 14](#_Toc40700111)

[Испытания для проверки эффективности алгоритма 18](#_Toc40700112)

[Вывод 19](#_Toc40700113)

[Список литературы 20](#_Toc40700114)

[Приложение 1. Листинг быстрого прототипа 21](#_Toc40700115)

[Приложение 2. Листинг оптимизированной версии 24](#_Toc40700116)

# Описание алгоритма

Суть статического алгоритма Хаффмана заключается в том, что символы почти любого алфавита имеют разную вероятность появления в тексте. Итак, раз разные символы имеют разные вероятности появления, следовательно, если кодировать их не все по 8 бит, как они хранятся в ASCII формате, а длину кода наиболее часто встречаемых уменьшить за счет увеличения длины кода редких символов, можно хорошо сжать исходный текст.

В алгоритме можно выделить четыре последовательные операции:

1. [Построение таблицы частот](http://hypertest.ispu.ru/module.php?sid=3995!e21c86a304d9c9890966539415ae65fc&mid=2295)
2. [Построение модели](http://hypertest.ispu.ru/module.php?sid=3995!e21c86a304d9c9890966539415ae65fc&mid=2296) кодирования
3. [Кодирование](http://hypertest.ispu.ru/module.php?sid=3995!e21c86a304d9c9890966539415ae65fc&mid=2297)
4. [Декодирование](http://hypertest.ispu.ru/module.php?sid=3995!e21c86a304d9c9890966539415ae65fc&mid=2298)

## 1. Таблицы частот

Первый этап при кодировании информации статическим алгоритмом Хаффмана является построение таблицы частот.

**Таблица частот** – таблица, в которой каждый символ входного потока имеет вес, который может быть равен либо вероятности, либо количеству вхождений этого символа в ожидаемое сообщение.

Таблица частот сортируется по убыванию.

В качестве примера построим таблицу частот для строчки: **абракадабра**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **А** | **Б** | **Р** | **К** | **Д** |
| **5** | **2** | **2** | **1** | **1** |

## 2. Модель кодирования

Модель кодирования представлена в виде дерева Хаффмана. Дерево Хаффмана строится на основе [таблицы частот](http://hypertest.ispu.ru/module.php?sid=3995!e21c86a304d9c9890966539415ae65fc&mid=2295). По дереву Хаффмана строится таблица кодов.

**Дерево Хаффмана** – [двоичное дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE), листьями которого являются символы алфавита, в каждой вершине которого хранится частота символа (для листа) или сумма частот ее двух детей (будем называть это число весом вершины).

Дерево Хаффмана (рис.1) для строчки с текстом: **абракадабра**

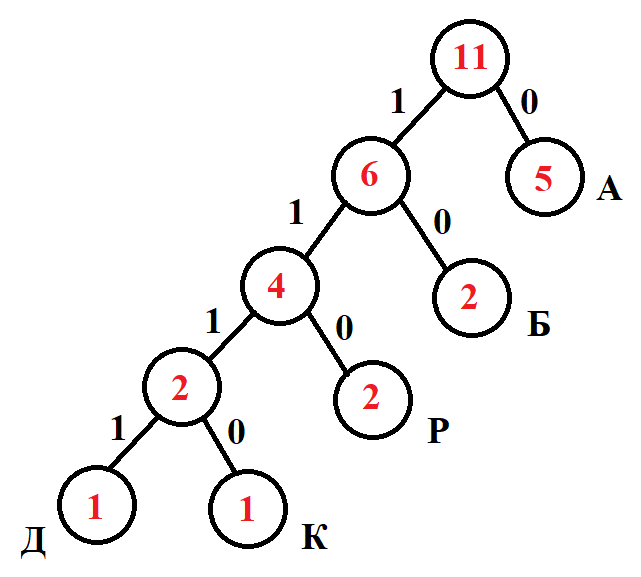


Рисунок 1 – Дерево Хаффмана

## 3. Кодирование

Работа кодера включает в себя несколько этапов:

1. Производится частотный анализ входного потока. Впоследствии по результатам анализа строится частотная таблица;
2. Строится модель кодирования. Этот этап включает в себя следующие действия:
   1. Упорядочивание вектора частот по убыванию.
   2. Построение бинарного дерева, листьями которого являются два крайних элемента вектора частот, при этом листья взвешены частотами, а корень взвешен суммами частот дочерних вершин.
   3. Замена частоты двух последних элементов их суммой.
   4. Восстановление упорядоченности списка по возрастанию.

Следует отметить, что символам алфавита входного сообщения соответствуют листья дерева. Так как дерево является бинарным, есть возможность разметить его рёбра нулями и единицами. Кодом символа для Хаффмана является цепочка двоичных значений разметки от корня дерева к листу, соответствующему символу. Однозначность декодирования обеспечивается тем, то ни один из кодов не является префиксом другого.

В качестве примера возьмём строчку: **абракадабра**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **А** | **Б** | **Р** | **К** | **Д** |
| **0** | **10** | **110** | **1110** | **1111** |

1. Кодирование. Пока не достигнут конец входного потока, выполняются следующие действия:
   1. Чтение символа входного потока.
   2. Отправка кода Хаффмана для символа в выходной поток.

## 4. Декодирование

Работа декодера включает в себя несколько этапов:

1. Декодер использует модель кодирования, построенную кодером.
2. Пока не конец входного потока, выполнять:
   1. Считать байт. Добавить к буферу последний бит этого байта.
   2. Если буфер найден в таблице кодов, то отправить символ в выходной поток и обнулить буфер.

# Разработка быстрого прототипа

## Скриншоты работы приложения

В качестве примера возьмем строчку: **абракадабра**. Результаты построения таблицы частот, кодирование символов представлено на рис. 2.

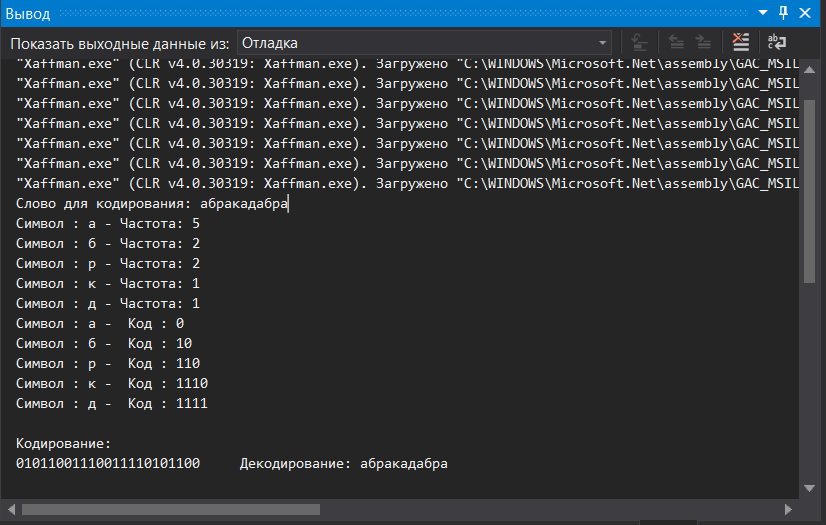


Рисунок 2 – Результаты работы программы

## Выводы

На практике убедились в работоспособности статического алгоритма Хаффмана. На этапе создания быстрого прототипа, не ставились перед собой цели добиться сжатия файла, так как кодировали один бит одним байтом.

# Разработка оптимизированной версии

## План отмены допущений

1. Включить в таблицу частот входного потока кодера служебный символ eof (endoffile) с частотой 0 для обозначения конца файла.
2. Передача от кодера декодеру таблицу частот в заголовке закодированного файла.
3. Применить структуру данных блочного типа для организации записи и чтения кодов переменной длины.

## UML диаграмма активности

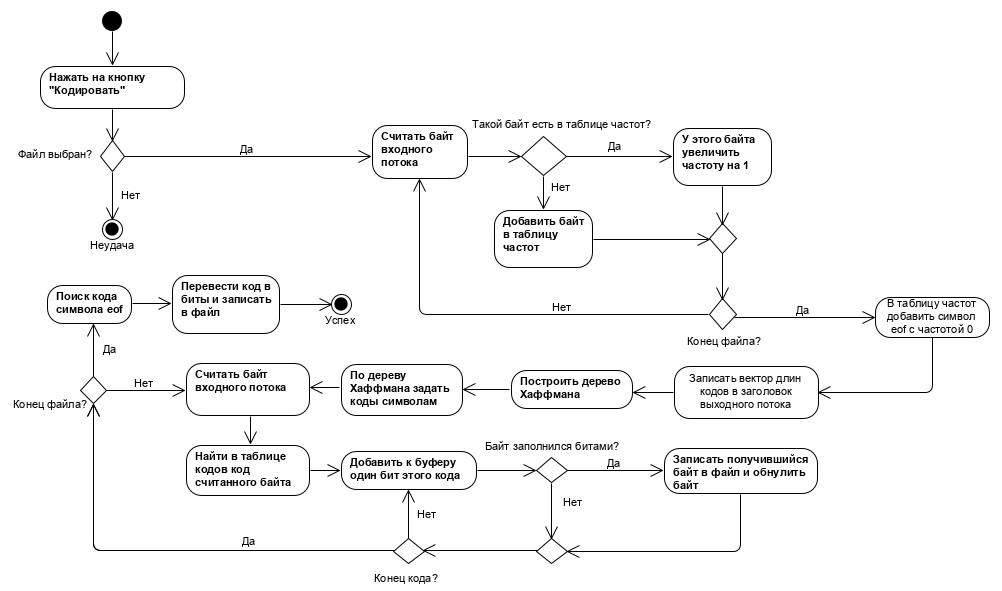


Рисунок 3 – Оптимизированная версия кодера

## Добавление служебного символа eof

После формирования вектора частот включаем в него служебный символ eof с частотой 0.

FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read); for (int i = 0; i < fileStream.Length; i++) //Создание вектора частот

{

byte b = (byte)fileStream.ReadByte(); //Считывание строчки посимвольно

if (!tables.Exists(j => j.symbol == b)) //Проверка на наличие символа в таблице

{ //Если символа в таблице нет, добавляем новый символ в таблицу с частотой = 1

Node t = new Node(b, 1);

tables.Add(t);

} //Если символ найден в таблице, увеличиваем его частоту на 1

else tables.Find(j => j.symbol == b).weight++;

}

fileStream.Close(); //Закрытие файла

Sortirovka(tables); //Упорядочевание вектора частот

Node eof = new Node(null, 0); //Добавление eof

tables.Add(eof);

## Передача модели кодирования

Модель кодирования в заголовке сжатого файла будем передавать таким образом:

Запишем в выходной файл от 0 до 255 частоты каждого байта (если байт есть в векторе частот, то запишем в выходной файл частоту, соответствующую его частоте в формате int, если такого байта нет, то 0).

FileStream fileStream = new FileStream(h, FileMode.Create, FileAccess.Write);

for (int i = 0; i < 256; i++) //Проходим по всем байтам

{

if (tables.Find((x) => x.symbol == i) != null) //Есть ли такой байт в таблице частот

{ //Если записываем его частоту в формате int в заголовок файла

byte[] intBytes = BitConverter.GetBytes(tables.Find((x) => x.symbol == i).weight);

for (int j = 0; j < 4; j++)

fileStream.WriteByte(intBytes[j]);

}

else //Если нет записываем частоту 0 в формате int

{

for (int j = 0; j < 4; j++)

fileStream.WriteByte(0);

}

}

fileStream.Close();

## Добавление структуры блочного типа

Для записи и чтения кодов переменной длины будем использовать отдельный **класс bitOperation.** Он содержит в себе значение текущего бита, на который указывает указатель.

public int bitNumber; //Номер бита

public bitOperation()

{

bitNumber = 0;

}

Также, данный класс содержит 2 функции для работы с битами:  
public void setbit(ref byte b, char code) – функция устанавливающая бит  
public void readbit(byte b, ref char code) – функция для считывания бита



Рисунок 4 – Структура байта

Ниже представлен фрагмент кода, для кодирования по битам.

FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);

byte newb = 0;

for (int i = 0; i < fileStream.Length; i++)

{

byte b = (byte)fileStream.ReadByte(); //Считывание строчки побайтово

for (int j = 0; j < modelCodes.Find((x) => x.symbol == b).code.Length; j++) //Цикл по длине кода символа

{

bitOperation.setbit(ref newb, modelCodes.Find(x => x.symbol == b).code[j]); //У байта устанавливаем биты согласно коду символа

if (bitOperation.bitNumber == 0) //Если байт заполнился, то записываем его в файл

fileStream1.WriteByte(newb);

}

}

Ниже представлен фрагмент кода, для декодирования по битам.

while (!finish) //Выполнять, пока не найден символ eof

{

byte b = (byte)fileStream.ReadByte(); //Считываем 1 байт

do

{

bitOperation.readbit(b, ref c); //Считываем биты байта

cod += c; //Заполняем код битами

if (modelCodes.Find((x) => x.code == cod) != null) //Есть такой код

{

if (modelCodes.Find(x => x.code == cod).symbol != null) //Символ не eof

{

fileStream1.WriteByte((byte)modelCodes.Find(x => x.code == cod).symbol); //Записываем символ в файл

cod = ""; //Обнуляем код

}

else finish = true; //Конец закодированной строчки найден

}

} while (bitOperation.bitNumber != 0); //Конец байта

}

## Скриншоты работы приложения

Для кодирования была выбрана картинка с расширением png (см. рисунок 5). Размер файла представлен на рисунке 6.

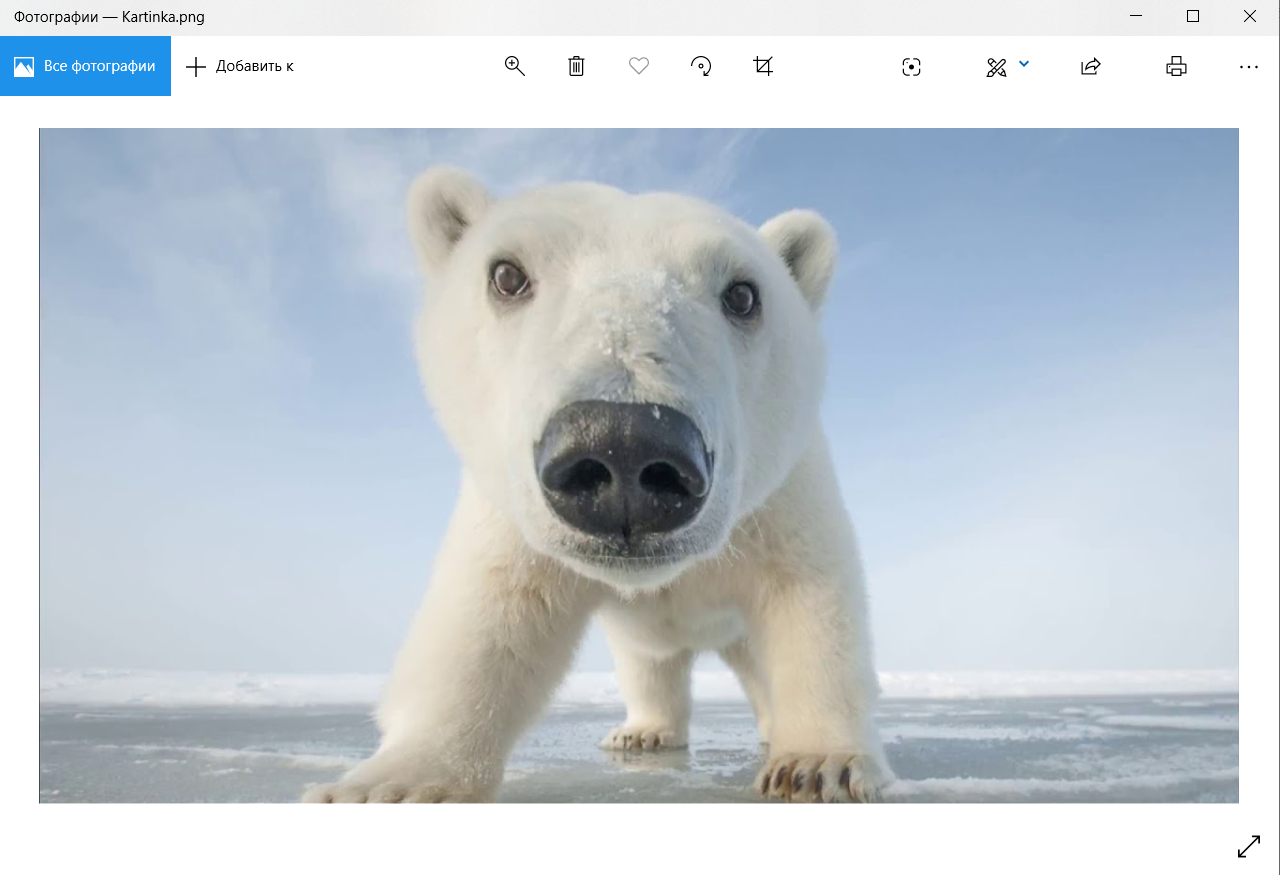


Рисунок 5 – Исходный файл для кодирования

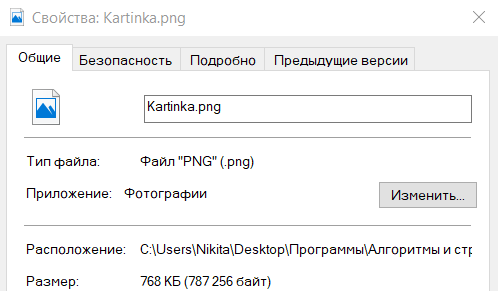


Рисунок 6 – Размер файла Kartinka.png

После кодирования был создан файл с расширением xuf. Размер файла представлен на рисунке 7.

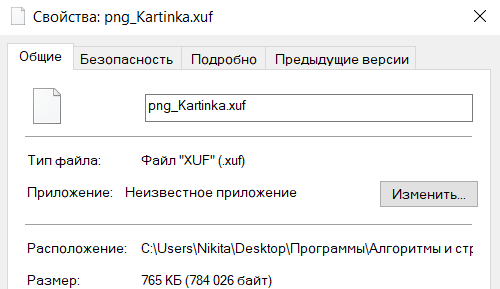


Рисунок 7 – Размер кодированного файла

Файл после декодирования представлен на рисунке 8.

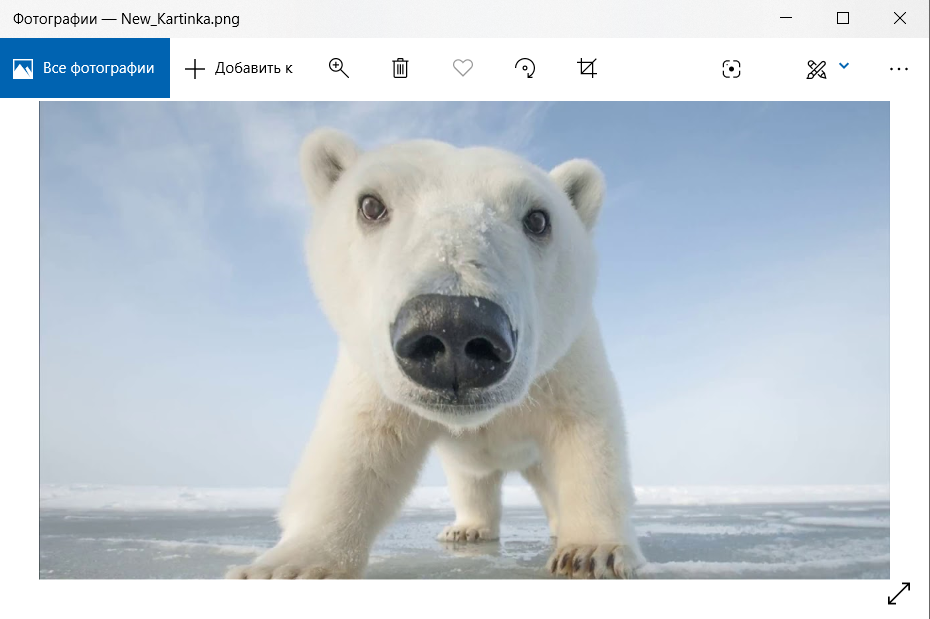


Рисунок 8 – Файл после декодирования

## Обратимость

Для проверки корректности сжатия приложением данных мы закодировали exe файл. В результате декодирования мы получили то же рабочее приложение, что и кодировали. Сбоев в работе не наблюдалось. Это означает, что сжатие осуществляется корректно. Результат представлен на рисунке 9-12.

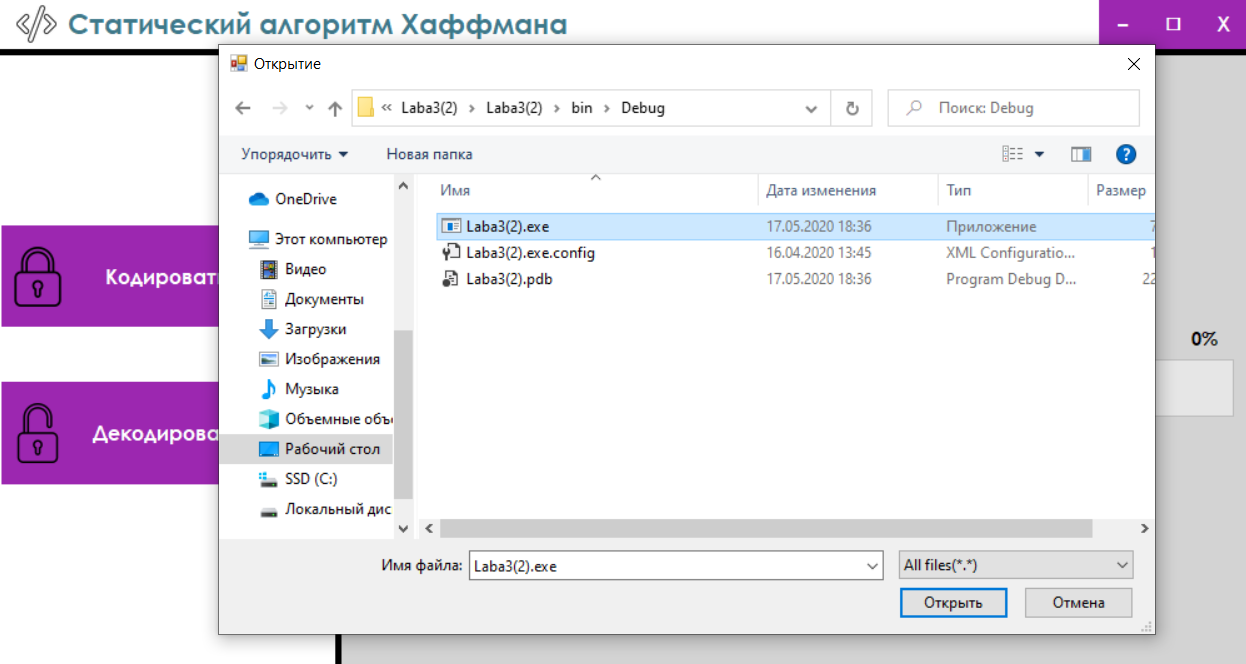


Рисунок 9 – Выбор exe файла



Рисунок 10 – Процесс кодирования

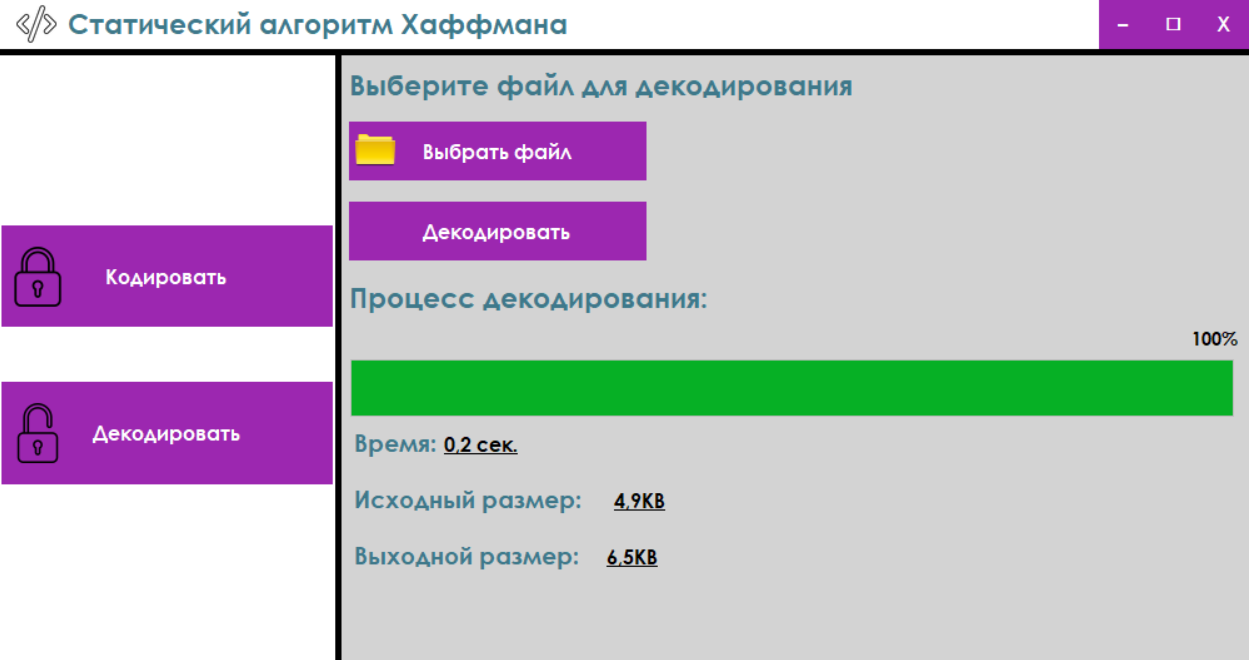


Рисунок 11 – Процесс декодирования

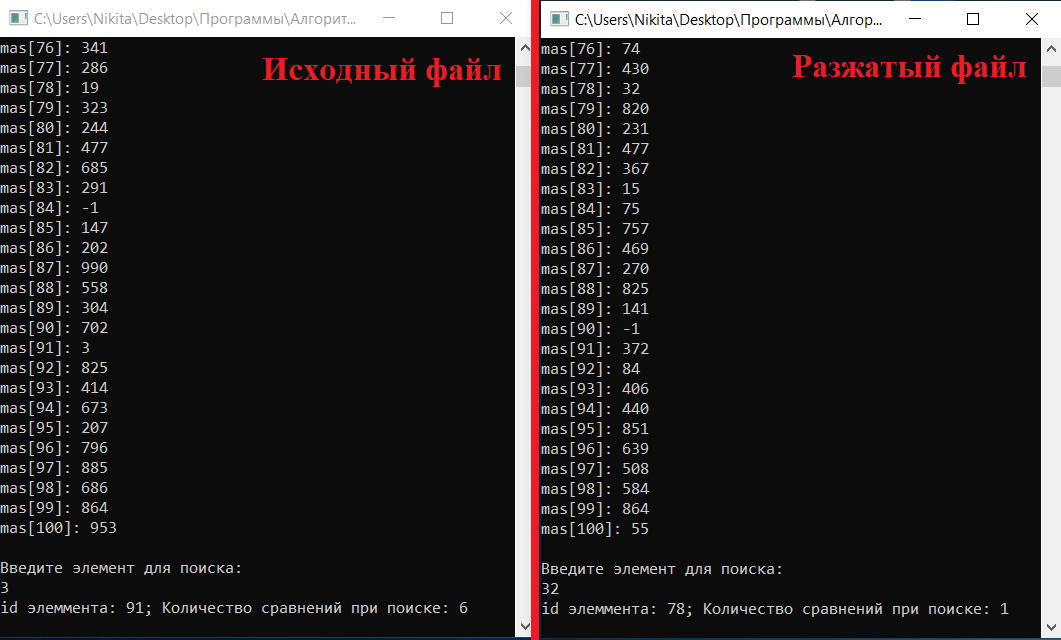


Рисунок 12 – Запуск исходного и разжатого файла

## Выводы

На этапе оптимизации быстрого прототипа, был построен и реализован план отмены допущений. На основе декодирования .exe файла можем сделать вывод, что алгоритм является обратимым, т.к. приложение не потеряло своей функциональности. На практике убедились в работоспособности приложения, после внесения отмены допущений.

# Руководства пользователя

Открывая приложения, мы видим главное окно, представленное на рисунке 13.

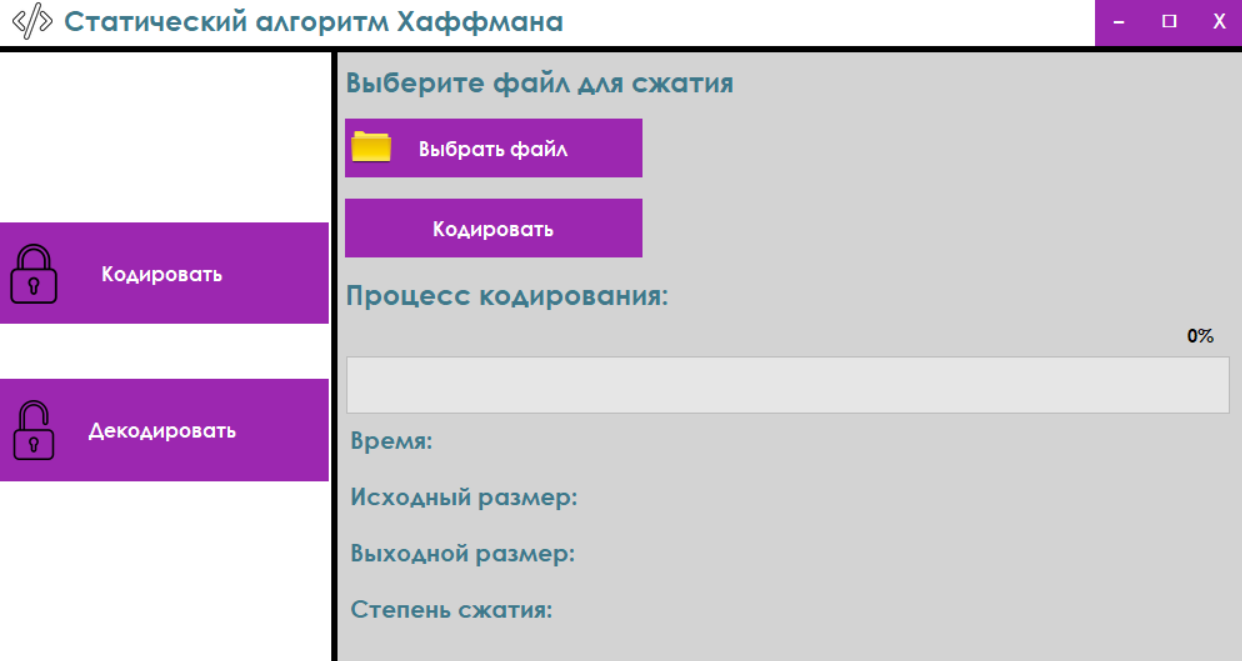


Рисунок 13 – Главное окно приложения

Выбор файла для кодирования осуществляется нажатием на кнопку «Выбрать файл». Далее появляется диалоговое окно с возможностью выбора файла (см. рисунок 14).

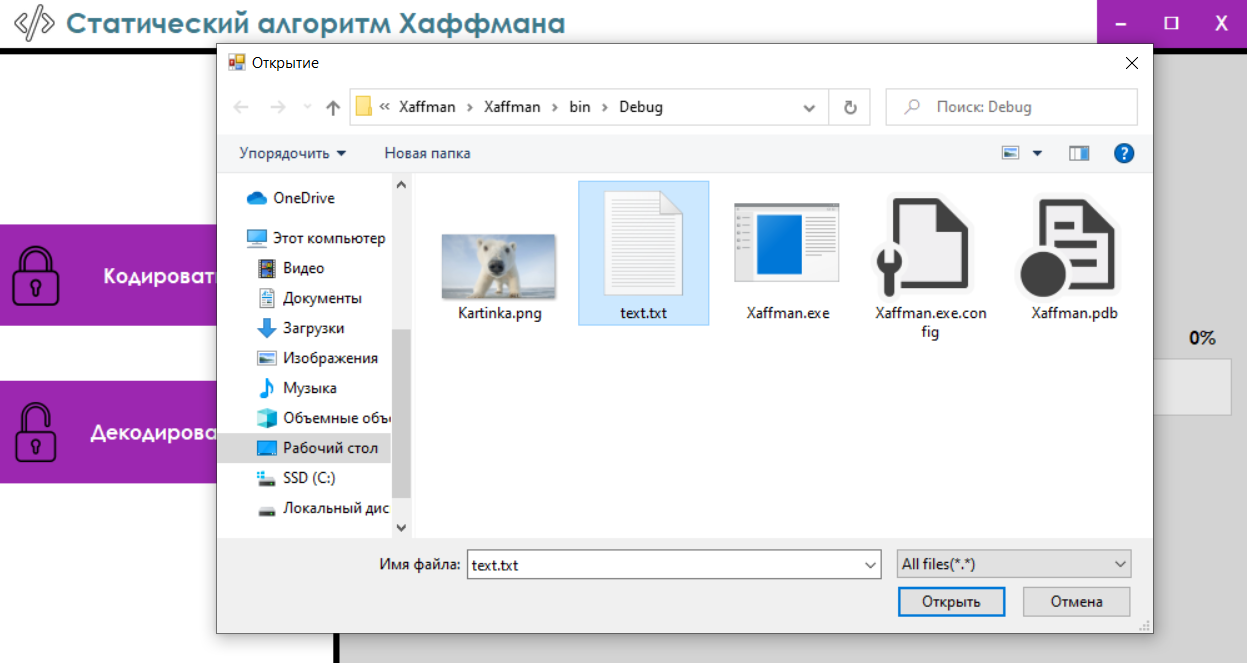


Рисунок 14 – Диалоговое окно выбора файла для кодирования

После выбора файла, нажимаем на кнопку «Кодировать», процесс кодирования представлен на рисунке 15.

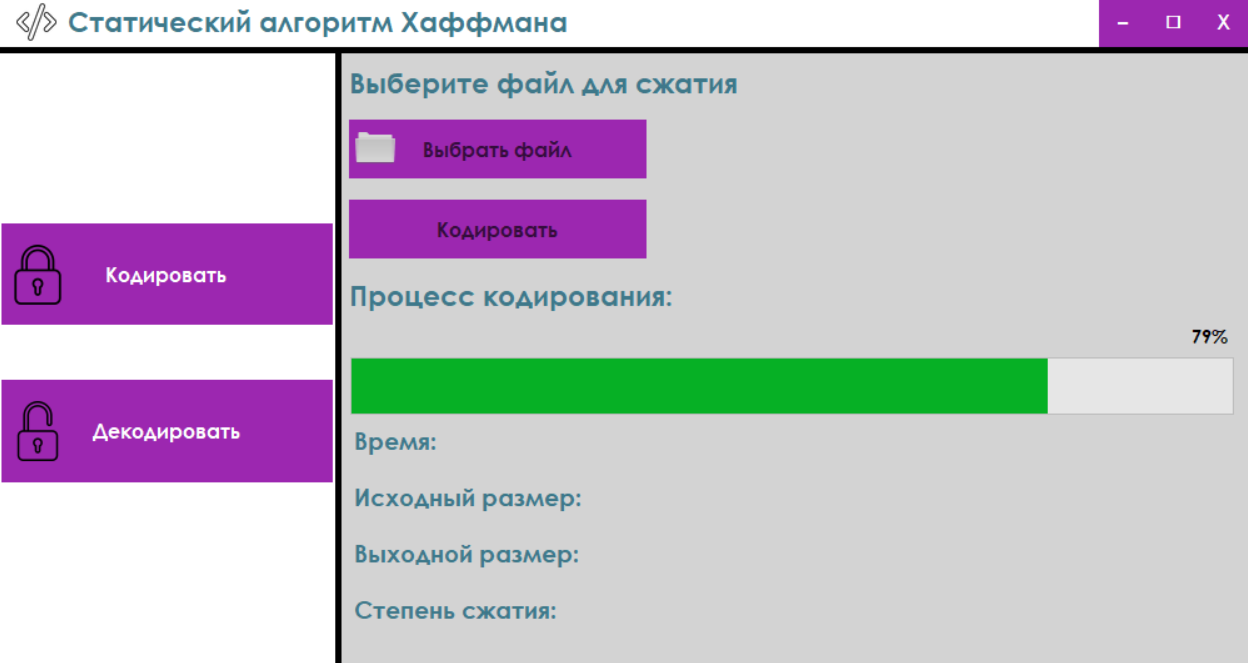


Рисунок 15 – Процесс кодирования

Далее после успешного кодирования, рассчитывается время кодирования и считается степень сжатия файла (см. рисунок 16). Закодированный файл помещается в папку с проектом.



Рисунок 16 – Информация о закодированном файле

Для декодирования файла, необходимо в меню нажать на кнопку «Декодировать» и приложение запустит окно для декодирования (см. рисунок 17).

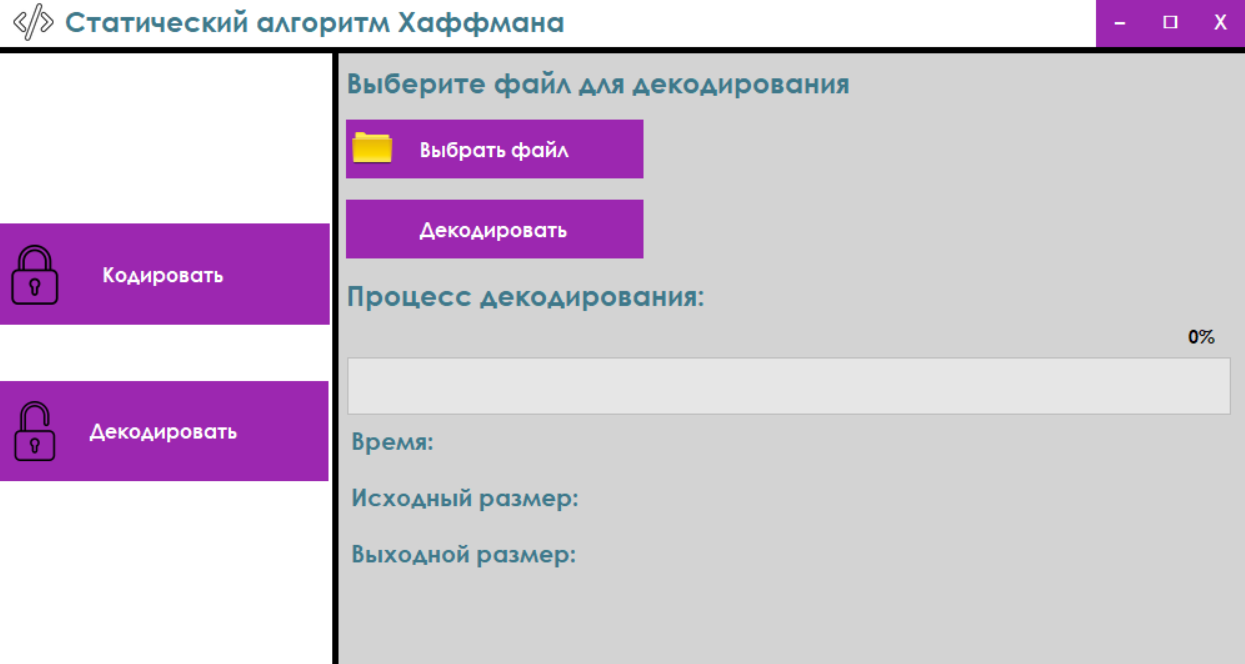


Рисунок 17 – Окно для декодирования файла

Далее для декодирования делаются те же самые действия, что и для кодирования. Файл для декодирования имеет расширение .xuf (см. рисунок 18).

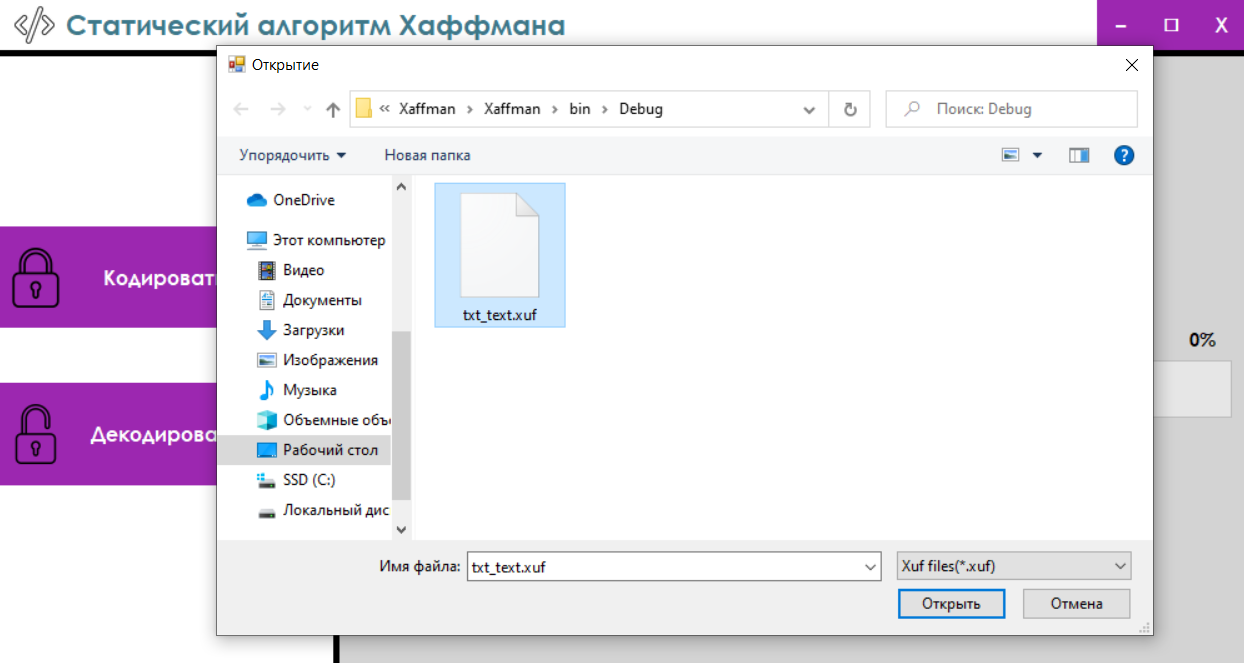


Рисунок 18 – Диалоговое окно выбора файла для декодирования

Декодированный файл, будет иметь то же название файла с добавлением постфикса New и то же расширение, что и исходный файл. Сохранен файл будет в папку с проектом. Прецесс декодирования представлен на рисунке 19.

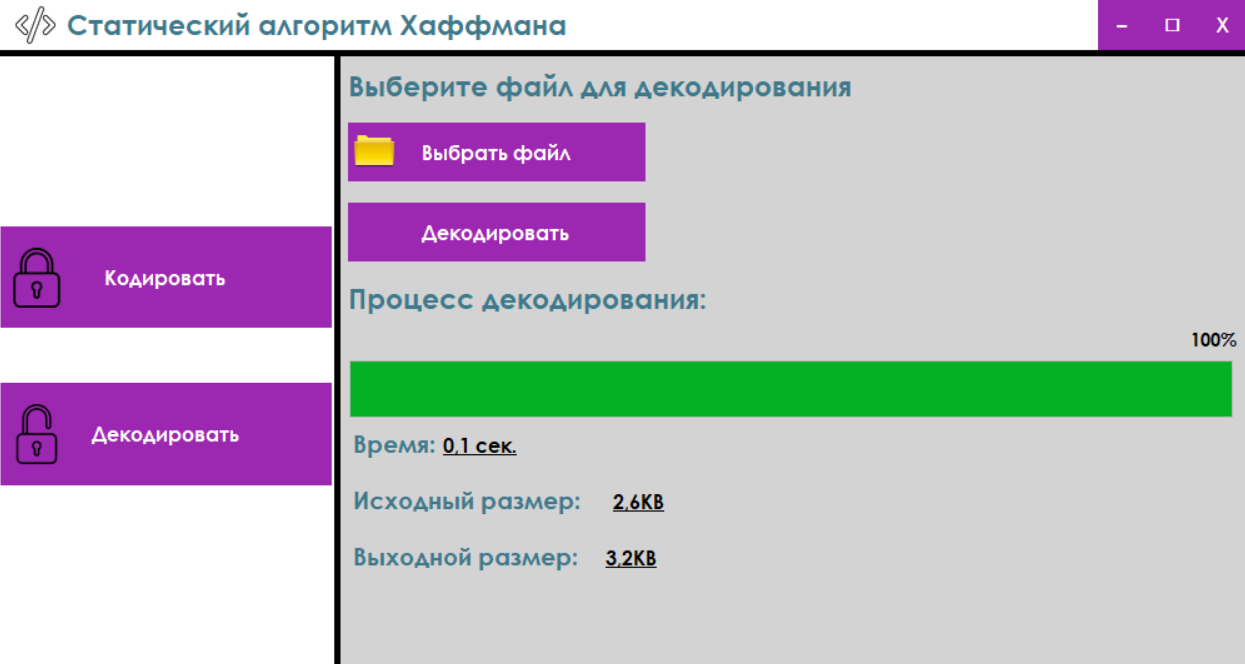


Рисунок 19 – Декодирование файла

В результате работы программы мы получили 2 идентичных файла (см. рисунок 20).

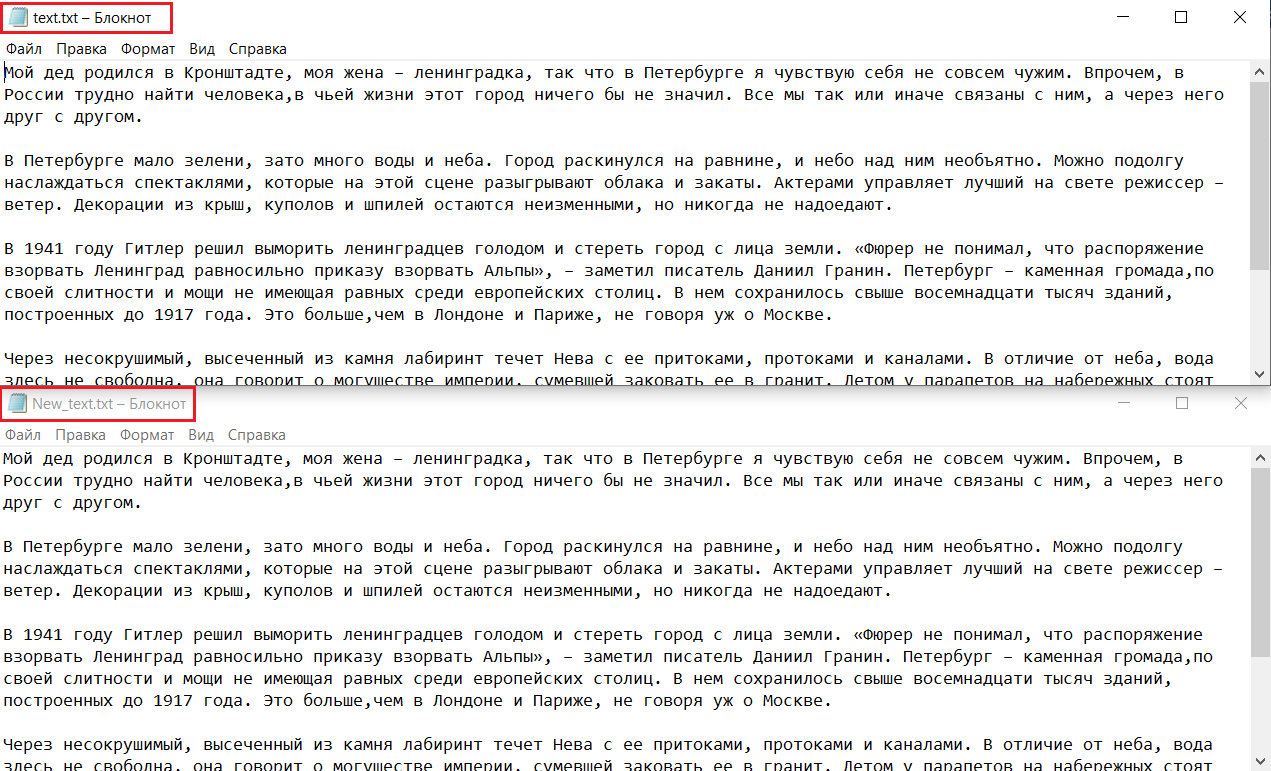


Рисунок 20 – Результат работы декодера

# Испытания для проверки эффективности алгоритма

Для того, чтобы проанализировать эффективность программы сравним степень сжатия различных файлов.

Степень сжатия определим по формуле:



K – степень сжатия;

start – размер исходного файла;

end – размер файла после сжатия.

|  |  |  |  |
| --- | --- | --- | --- |
| **Размер до сжатия** | **Размер после сжатия** | **Тип файла** | **Степень сжатия, %** |
| 1,4 КБ | 1,5 КБ | txt | -9 |
| 8,7 КБ | 6,4 КБ | txt | 26 |
| 1,2 МБ | 647,6 КБ | txt | 47 |
| 1,1 КБ | 1,7 КБ | exe | -58 |
| 6,5 КБ | 4,6 КБ | exe | 24 |
| 16,5 КБ | 11,6 КБ | exe | 29 |
| 1,2 КБ | 1,9 КБ | pdf | -56 |
| 773,4 КБ | 773,9 КБ | pdf | 0 |
| 1,3 МБ | 1,4 МБ | pdf | -1 |
| 1,4 КБ | 1,9 КБ | jpg | -32 |
| 768 КБ | 765 КБ | jpg | 0 |
| 1,6 МБ | 1,6 МБ | jpg | 0 |
| 914 Байт | 1,2 КБ | bmp | -32 |
| 47,6 КБ | 7,3 КБ | bmp | 84 |
| 1,2 МБ | 189,7 КБ | bmp | 84 |

Таблица 1 – Результаты экспериментов

На основе данной таблицы построим гистограмму, на горизонтальной оси которой обозначим тип файла, а по вертикальной – степень сжатия. Результаты представлены на рисунке 21.

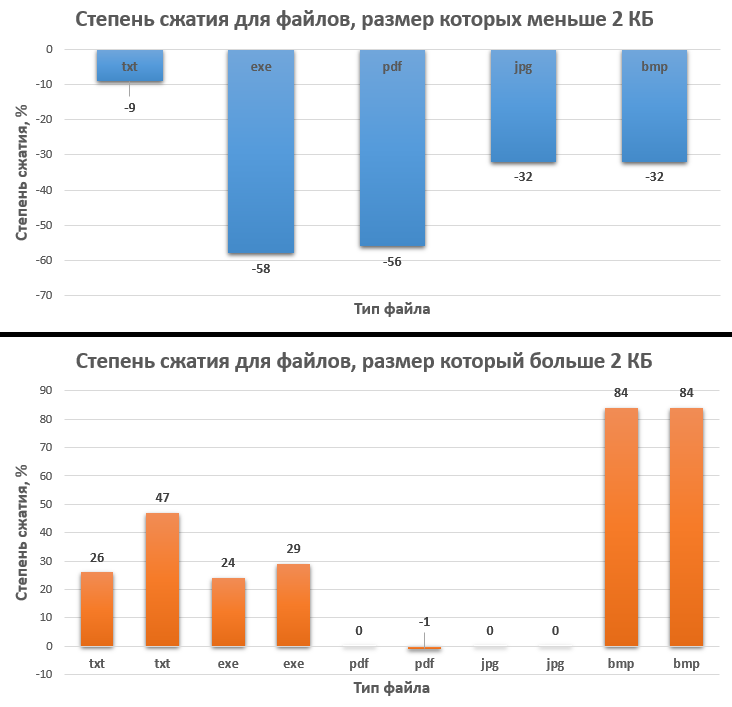


Рисунок 21 – Степени сжатия различных типов файлов

На данной диаграмме мы видим, что некоторые типы файлов показывают хорошие результаты сжатия, а некоторые, наоборот, увеличиваются в размере в результате сжатия. В этом случае степень сжатия является отрицательной. Это связано с тем, что данные типы уже являются сжатыми.

Другой случай увеличения размера, который мы наблюдаем – его малый размер. Это связано с добавлением заголовка к сжатому файлу. Модель кодирования занимается 4\*256 = 1 КБ, сжатого файла, поэтому сжатия файлов меньших размеров нецелесообразно.

# Вывод

На основе анализа эффективности сжатия можем сделать вывод, что для тех типов файлов, которые уже являются сжатыми, и для файлов достаточно малых размеров такой алгоритм сжатия не подходит. В таких случаях размер сжатых файлов в лучшем случае не изменяется, в худшем увеличивается. Для файлов больших размеров и не сжатых типов (bmp) алгоритм Хаффмана достаточно эффективен.

# Список литературы

* Пантелеев, Евгений Рафаилович. Структуры данных и алгоритмы сжатия информации без потерь: методические указания к лабораторной работе по дисциплине «Структуры и алгоритмы обработки данных»; / Е.Р. Пантелеев, П.А. Фомин; Министерство образования Российской Федерации, Ивановский государственный энергетический университет, Каф, программного обеспечения компьютерных систем; под ред. И.Д. Ратмановой.–Иваново: Б.и., 2001.—28 с: ил.
* Алгоритм Хаффмана // algolist.manual.ru URL: <http://algolist.manual.ru/compress/standard/huffman.php> (дата обращения: 12.03.2020).

# Приложение 1. Листинг быстрого прототипа

**Node.cs**

public class Node

{

public string symbol; //Символ

public int weight; //Частота

public string code; //Код символа

public Node parentNode; //Родитель

public Node leftChild; //Левый потомок

public Node rightChild; //Правый потомок

public Node(string symbol, int weight)

{

this.symbol = symbol;

this.weight = weight;

leftChild = rightChild = parentNode = null;

code = "";

}

public Node(Node node1, Node node2) //Соединение 2-х узлов

{

code = "";

parentNode = null;

if (node1.weight >= node2.weight)

{

rightChild = node1;

leftChild = node2;

rightChild.parentNode = leftChild.parentNode = this;

symbol = node1.symbol + node2.symbol;

weight = node1.weight + node2.weight;

}

else if (node1.weight < node2.weight)

{

rightChild = node2;

leftChild = node1;

leftChild.parentNode = rightChild.parentNode = this;

symbol = node2.symbol + node1.symbol;

weight = node2.weight + node1.weight;

}

}

}

**Form1.cs**

public partial class Form1 : Form

{

private List<Node> tables = new List<Node>(); //Таблица частот

private bool flag = false; //Проверка что символ найден

public Form1()

{

InitializeComponent();

}

private void Sortirovka(List<Node> S) //Сортировка

{

Node temp;

for (int i = 0; i < S.Count; i++)

{

for (int j = i + 1; j < S.Count; j++)

{

if (S[i].weight < S[j].weight)

{

temp = S[i];

S[i] = S[j];

S[j] = temp;

}

}

}

}

public void CreatTree(List<Node> nodeList) //Создание дерева

{

while (nodeList.Count > 1)

{

Node node1 = nodeList.Last(); //Берем последний элемент

nodeList.Remove(node1); //Удаляем его из таблицы

Node node2 = nodeList.Last(); //Берем предпоследний элемент

nodeList.Remove(node2); //Удаляем его из таблицы

nodeList.Add(new Node(node1, node2));

Sortirovka(nodeList); //Упорядочивание вектора частот

}

}

public void setCode(string code, Node Nodes) //Задаем коды символам

{

if (Nodes == null)

return;

if (Nodes.leftChild == null && Nodes.rightChild == null)

{

Nodes.code = code;

return;

}

setCode(code + "0", Nodes.leftChild);

setCode(code + "1", Nodes.rightChild);

}

public void PrintInformation(List<Node> nodeList) //Вывод таблица частот

{

foreach (var item in nodeList)

Console.WriteLine("Символ : {0} - Частота: {1}", item.symbol, item.weight);

}

public void PrintfLeafAndCodes(Node nodeList) //Вывод символа и его код

{

if (nodeList == null)

return;

if (nodeList.leftChild == null && nodeList.rightChild == null)

{

Console.WriteLine("Символ : {0} - Код : {1}", nodeList.symbol, nodeList.code);

return;

}

PrintfLeafAndCodes(nodeList.leftChild);

PrintfLeafAndCodes(nodeList.rightChild);

}

public void Koding(string value, Node nodeList) //Кодируем строчку

{

if (nodeList == null)

return;

if (nodeList.symbol == value)

{

textBox2.Text += nodeList.code;

return;

}

Koding(value, nodeList.leftChild);

Koding(value, nodeList.rightChild);

}

public void Raskoding(string value, Node nodeList) //Декодируем строчку

{

if (nodeList == null)

return;

if (nodeList.code == value)

{

textBox3.Text += nodeList.symbol;

flag = true;

return;

}

Raskoding(value, nodeList.leftChild);

Raskoding(value, nodeList.rightChild);

}

private void button1\_Click(object sender, EventArgs e)

{

string st = textBox1.Text; //Считывание строки

for (int i = 0; i < st.Length; i++) //Создание вектора частот

{

string read = Convert.ToChar(st[i]).ToString();

if (!tables.Exists(j => j.symbol == read))

{

Node t = new Node(read, 1);

tables.Add(t);

}

else tables.Find(j => j.symbol == read).weight++;

}

Sortirovka(tables); //Упорядочивание вектора частот

PrintInformation(tables); //Вывод таблицы частот

CreatTree(tables); //Создание дерева кодирования

setCode("", tables[0]); //Кодировка символов

PrintfLeafAndCodes(tables[0]); //Вывод символа и его код

for (int i = 0; i < st.Length; i++)

{

string read = Convert.ToChar(st[i]).ToString();

Koding(read, tables[0]); //Кодируем строчку посимвольно

}

}

private void button2\_Click(object sender, EventArgs e)

{

string st = textBox2.Text; //Считывание строки

string cod = ""; //Код

for (int i = 0; i < st.Length; i++)

{

cod += st[i];

Raskoding(cod, tables[0]);

if (flag)

{

cod = "";

flag = false;

}

}

}

}

# Приложение 2. Листинг оптимизированной версии

**bitOperation.cs**

public class bitOperation

{

public int bitNumber; //Номер бита

public bitOperation()

{

bitNumber = 0;

}

public void setbit(ref byte b, char code)

{

switch(bitNumber)

{

case 0:

{

if (code == '1') b = 1;

else b = 0;

bitNumber++;

}

break;

case 1:

{

if (code == '1') b = (byte)(b | 1 << 1);

bitNumber++;

}

break;

case 2:

{

if (code == '1') b = (byte)(b | 1 << 2);

bitNumber++;

}

break;

case 3:

{

if (code == '1') b = (byte)(b | 1 << 3);

bitNumber++;

}

break;

case 4:

{

if (code == '1') b = (byte)(b | 1 << 4);

bitNumber++;

}

break;

case 5:

{

if (code == '1') b = (byte)(b | 1 << 5);

bitNumber++;

}

break;

case 6:

{

if (code == '1') b = (byte)(b | 1 << 6);

bitNumber++;

}

break;

case 7:

{

if (code == '1') b = (byte)(b | 1 << 7);

bitNumber = 0;

}

break;

}

}

public void readbit(byte b, ref char code)

{

switch(bitNumber)

{

case 0:

{

if ((b & 1) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 1:

{

if ((b & 1 << 1) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 2:

{

if ((b & 1 << 2) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 3:

{

if ((b & 1 << 3) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 4:

{

if ((b & 1 << 4) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 5:

{

if ((b & 1 << 5) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 6:

{

if ((b & 1 << 6) == 0) code = '0';

else code = '1';

bitNumber++;

}

break;

case 7:

{

if ((b & 1 << 7) == 0) code = '0';

else code = '1';

bitNumber = 0;

}

break;

}

}

}

**ModelCode.cs**

public class ModelCode

{

public byte? symbol; //Символ

public string code; //Код символа

public ModelCode(byte? symbol, string code)

{

this.symbol = symbol;

this.code = code;

}

}

**Node.cs**

public class Node

{

public byte? symbol; //Символ

public int weight; //Частота

public string code; //Код символа

public Node parentNode; //Родитель

public Node leftChild; //Левый потомок

public Node rightChild; //Правый потомок

public Node(byte? symbol, int weight)

{

this.symbol = symbol;

this.weight = weight;

leftChild = rightChild = parentNode = null;

code = "";

}

public Node(Node node1, Node node2) //Соединение 2-х узлов

{

code = "";

parentNode = null;

if (node1.weight >= node2.weight)

{

rightChild = node1;

leftChild = node2;

rightChild.parentNode = leftChild.parentNode = this;

symbol = null;

weight = node1.weight + node2.weight;

}

else if (node1.weight < node2.weight)

{

rightChild = node2;

leftChild = node1;

leftChild.parentNode = rightChild.parentNode = this;

symbol = null;

weight = node2.weight + node1.weight;

}

}

}

**ProcessMethod.cs**

public class ProcessMethod

{

private Form1 form;

private bitOperation bitOperation; //Операции с битами

private List<ModelCode> modelCodes; //Модель "символ = код"

private string filename; //Путь к файлу

private string h; //Название, закодированного файла/Декодированного файла

private List<Node> tables; //Таблица частот

public long start\_size, end\_size; //Размер файла до/после

public ProcessMethod(Form1 form)

{

this.form = form;

bitOperation = new bitOperation();

modelCodes = new List<ModelCode>();

filename = h = "";

start\_size = end\_size = 0;

tables = new List<Node>();

}

/\* Получение имени файла \*/

public void Setfilename(string filename)

{

this.filename = filename;

}

/\* Генерация таблицы частот \*/

public void CreateTable()

{

FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);

for (int i = 0; i < fileStream.Length; i++) //Создание вектора частот

{

byte b = (byte)fileStream.ReadByte(); //Считывание строчки посимвольно

if (!tables.Exists(j => j.symbol == b))//Проверка на наличие символа в таблице

{

Node t = new Node(b, 1); //Если символа в таблице нет, добавляем новый символ в таблицу с частотой = 1

tables.Add(t);

}

else tables.Find(j => j.symbol == b).weight++; //Если символ найден в таблице, увеличиваем его частоту на 1

}

fileStream.Close(); //Закрываем файл

Sortirovka(tables); //Упорядочевание вектора частот

Node eof = new Node(null, 0); //Добавление eof

tables.Add(eof);

PrintInformation(tables); //Запись таблицы частот в файл

CreatTree(tables); //Создание дерева кодирования

setCode("", tables[0]); //Кодировка символов

}

/\* Сортировка \*/

private void Sortirovka(List<Node> S)

{

Node temp;

for (int i = 0; i < S.Count; i++)

{

for (int j = i + 1; j < S.Count; j++)

{

if (S[i].weight != S[j].weight)

{

if (S[i].weight < S[j].weight)

{

temp = S[i];

S[i] = S[j];

S[j] = temp;

}

}

else

{

if (S[i].symbol < S[j].symbol)

{

temp = S[i];

S[i] = S[j];

S[j] = temp;

}

}

}

}

}

/\* Создание дерева \*/

private void CreatTree(List<Node> nodeList)

{

while (nodeList.Count > 1) //Дереву нужно 2 листа

{

Node node1 = nodeList.Last(); //Берем последний элемент

nodeList.Remove(node1); //Удаляем его из таблицы

Node node2 = nodeList.Last(); //Берем предпоследний элемент

nodeList.Remove(node2); //Удаляем его из таблицы

nodeList.Add(new Node(node1, node2)); //Создаем новый узел с частотой = сумме частот элементов удаленных из таблицы

Sortirovka(nodeList); //Упорядочевание вектора частот

}

}

/\* Задаем коды символам \*/

private void setCode(string code, Node Nodes)

{

if (Nodes == null)

return;

if (Nodes.leftChild == null && Nodes.rightChild == null) //Поиск листа

{

ModelCode mc = new ModelCode(Nodes.symbol, code);

modelCodes.Add(mc);

Nodes.code = code;

return;

}

setCode(code + "0", Nodes.leftChild); //Прибавляем к коду символ 0

setCode(code + "1", Nodes.rightChild); //Прибавляем к коду символ 1

}

/\* Запись таблицы частот \*/

public void PrintInformation(List<Node> nodeList)

{

FileInfo fileInf = new FileInfo(filename);

start\_size = fileInf.Length;

h = fileInf.Name;

h = h.Substring(h.LastIndexOf('.') + 1) + "\_" + h.Substring(0, h.LastIndexOf('.') + 1) + "xuf";

FileStream fileStream = new FileStream(h, FileMode.Create, FileAccess.Write);

for (int i = 0; i < 256; i++)

{

if (tables.Find((x) => x.symbol == i) != null) //Если такого байта нет

{

byte[] intBytes = BitConverter.GetBytes(tables.Find((x) => x.symbol == i).weight);

for (int j = 0; j < 4; j++)

fileStream.WriteByte(intBytes[j]); //Записываем частоту байта

}

else //Частоту этого байта записываем 0

{

for (int j = 0; j < 4; j++)

fileStream.WriteByte(0);

}

}

fileStream.Close();

}

/\* Кодируем строчку \*/

public void Koding()

{

FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);

FileStream fileStream1 = new FileStream(h, FileMode.Append, FileAccess.Write);

byte newb = 0;

for (int i = 0; i < fileStream.Length; i++)

{

form.workerCod.ReportProgress((int)((fileStream.Position \* 100 / fileStream.Length)) + 1); //Процесс кодирования

byte b = (byte)fileStream.ReadByte(); //Считывание строчки посимвольно

for (int j = 0; j < modelCodes.Find((x) => x.symbol == b).code.Length; j++)

{

bitOperation.setbit(ref newb, modelCodes.Find(x => x.symbol == b).code[j]);

if (bitOperation.bitNumber == 0)

fileStream1.WriteByte(newb);

}

}

for (int j = 0; j < modelCodes.Find((x) => x.symbol == null).code.Length; j++)

{ //Добавление eof

bitOperation.setbit(ref newb, modelCodes.Find(x => x.symbol == null).code[j]);

if (bitOperation.bitNumber == 0)

fileStream1.WriteByte(newb);

}

if (newb != 0)

fileStream1.WriteByte(newb);

fileStream.Close();

fileStream1.Close();

FileInfo fileInf = new FileInfo(h);

end\_size = fileInf.Length;

bitOperation.bitNumber = 0;

}

/\* Восстанавление таблицы частот \*/

public void RecoveryTable()

{

FileStream fileStream = new FileStream(filename, FileMode.Open, FileAccess.Read);

for (int i = 0; i < 256; i++)

{

byte[] byteN = new byte[4];

for (int j = 0; j < 4; j++)

byteN[j] = (byte)fileStream.ReadByte();

int chastota = BitConverter.ToInt32(byteN, 0);

if (chastota != 0)

{

Node m = new Node((byte)i, chastota);

tables.Add(m);

}

}

Sortirovka(tables);

Node eof = new Node(null, 0); //Добавление eof

tables.Add(eof);

CreatTree(tables); //Создание дерева кодирования

setCode("", tables[0]); //Кодировка символов

Decoding(fileStream);

fileStream.Close();

}

/\* Декодируем строчку \*/

private void Decoding(FileStream fileStream)

{

FileInfo fileInf = new FileInfo(filename);

start\_size = fileInf.Length;

h = fileInf.Name;

h = h.Substring(h.LastIndexOf('\_') + 1, h.LastIndexOf('.') - h.LastIndexOf('\_'));

h += fileInf.Name.Substring(0, fileInf.Name.LastIndexOf('\_'));

h = "New\_" + h;

bool finish = false; //Флаг на проверку конца строчки

FileStream fileStream1 = new FileStream(h, FileMode.Create, FileAccess.Write);

string cod = ""; //Код

char c = '0';

while (!finish)

{

byte b = (byte)fileStream.ReadByte();

if((fileStream.Position \* 100 / fileStream.Length) == 100)

form.workerDecod.ReportProgress((int)((fileStream.Position \* 100 / fileStream.Length))); //Процесс декодирования

else form.workerDecod.ReportProgress((int)((fileStream.Position \* 100 / fileStream.Length + 1)));

do

{

bitOperation.readbit(b, ref c); //Считывание побитам

cod += c;

if (modelCodes.Find((x) => x.code == cod) != null)

{

if (modelCodes.Find(x => x.code == cod).symbol != null)

{

fileStream1.WriteByte((byte)modelCodes.Find(x => x.code == cod).symbol);

cod = "";

}

else finish = true; //Конец закодированной строчки найден

}

} while (bitOperation.bitNumber != 0);

}

fileStream1.Close();

FileInfo fileInf1 = new FileInfo(h);

end\_size = fileInf1.Length;

}

}

**Form1.cs**

public partial class Form1 : Form

{

private ProcessMethod xuffmanCod; //Статический алгоритм Хаффмана – кодер

private ProcessMethod xuffmanDecod; //Статический алгоритм Хаффмана – декодер

private string filename; //Путь к файлу

private long start\_size, end\_size; //Размер файла до/после

private DateTime start, end;

public Form1()

{

InitializeComponent();

openFileDialogCod.Filter = "All files(\*.\*)|\*.\*";

openFileDialogDecod.Filter = "Xuf files(\*.xuf)|\*.xuf";

decod.Visible = false;

start\_size = end\_size = 0;

}

/\* Окно для кодирования \*/

private void button1\_Click(object sender, EventArgs e)

{

decod.Visible = false;

}

/\* Кнопка для кодирования \*/

private void koding\_Click(object sender, EventArgs e)

{

timer1.Enabled = true;

workerCod.RunWorkerAsync();

}

/\* Окно для декодирования \*/

private void button2\_Click(object sender, EventArgs e)

{

decod.Visible = true;

}

/\* Кнопка для декодирования \*/

private void dekoding\_Click(object sender, EventArgs e)

{

timer2.Enabled = true;

workerDecod.RunWorkerAsync();

}

/\* Выбрать файл для кодирования \*/

private void fileCod\_Click(object sender, EventArgs e)

{

if (openFileDialogCod.ShowDialog() == DialogResult.Cancel)

return;

filename = openFileDialogCod.FileName; //Получаем путь выбранного файла

}

/\* Выбрать файл для декодирования \*/

private void fileDecod\_Click(object sender, EventArgs e)

{

if (openFileDialogDecod.ShowDialog() == DialogResult.Cancel)

return;

filename = openFileDialogDecod.FileName; //Получаем путь выбранного файла

}

/\* Поток для кодера \*/

private void workerCod\_DoWork(object sender, DoWorkEventArgs e)

{

xuffmanCod = new ProcessMethod(this);

xuffmanCod.Setfilename(filename);

start = DateTime.Now;

xuffmanCod.CreateTable();

xuffmanCod.Koding(); //Кодирование текста

end = DateTime.Now;

start\_size = xuffmanCod.start\_size;

end\_size = xuffmanCod.end\_size;

}

/\* Поток для декодера \*/

private void workerDecod\_DoWork(object sender, DoWorkEventArgs e)

{

xuffmanDecod = new ProcessMethod(this);

xuffmanDecod.Setfilename(filename);

start = DateTime.Now;

xuffmanDecod.RecoveryTable();

end = DateTime.Now;

start\_size = xuffmanDecod.start\_size;

end\_size = xuffmanDecod.end\_size;

}

/\* Прогресс для кодера \*/

private void workerCod\_ProgressChanged(object sender, ProgressChangedEventArgs e)

{

progressBar1.Value = e.ProgressPercentage;

lblprogress.Text = progressBar1.Value.ToString() + "%";

}

/\* Прогресс для декодера \*/

private void workerDecod\_ProgressChanged(object sender, ProgressChangedEventArgs e)

{

progressBar2.Value = e.ProgressPercentage;

lblprogress1.Text = progressBar2.Value.ToString() + "%";

}

private String BytesToString(long byteCount)

{

string[] suf = { "Byt", "KB", "MB", "GB", "TB", "PB", "EB" };

if (byteCount == 0)

return "0" + suf[0];

long bytes = Math.Abs(byteCount);

int place = Convert.ToInt32(Math.Floor(Math.Log(bytes, 1024)));

double num = Math.Round(bytes / Math.Pow(1024, place), 1);

return (Math.Sign(byteCount) \* num).ToString() + suf[place];

}

/\* Закрыть окно \*/

private void close\_Click(object sender, EventArgs e)

{

this.Close();

}

private bool cl = false; //Флаг для проверки в каком сейчас режиме окно

/\* Масштабировать окно \*/

private void deploy\_Click(object sender, EventArgs e)

{

if (!cl)

WindowState = FormWindowState.Maximized;

else

WindowState = FormWindowState.Normal;

cl = !cl;

}

private void timer1\_Tick(object sender, EventArgs e)

{

if (progressBar1.Value == 100)

{

TimeSpan span = end - start;

timeCod.Text = Math.Round(span.TotalSeconds, 1).ToString() + " cек.";

label8.Text = BytesToString(start\_size);

label9.Text = BytesToString(end\_size);

label10.Text = ((start\_size - end\_size) \* 100 / start\_size).ToString() + " %";

timer1.Enabled = false;

}

}

private void timer2\_Tick(object sender, EventArgs e)

{

if (progressBar2.Value == 100)

{

TimeSpan span = end - start;

label14.Text = Math.Round(span.TotalSeconds, 1).ToString() + " cек.";

label13.Text = BytesToString(start\_size);

label12.Text = BytesToString(end\_size);

timer2.Enabled = false;

}

}

/\* Свернуть окно \*/

private void button4\_Click(object sender, EventArgs e)

{

this.WindowState = FormWindowState.Minimized;

}

}