**Министерство образования и науки РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования «Ивановский государственный энергетический университет имени В.И. Ленина»**

**Кафедра программного обеспечения компьютерных систем**

**Алгоритмы сжатия**

Пояснительная записка к курсовому проекту

по дисциплине: «Алгоритмы и структуры данных»

**Выполнила студентка гр. 3-41\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Варфоломеева А. А.**

**Проверили**

**каф. ПОКС \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пантелеев Е.Р.**

**Иваново, 2019**

Оглавление

[**Вариант 13** 3](#_Toc7370838)

[**Описание алгоритма** 3](#_Toc7370839)

[**Разработка быстрого прототипа** 4](#_Toc7370840)

[**Список принятых допущений на этапе разработки прототипа программы:** 4](#_Toc7370841)

[**Алгоритм записи байта в файл** 4](#_Toc7370842)

[**UML диаграмма активности** 5](#_Toc7370843)

[**Результаты испытаний** 6](#_Toc7370844)

[**Вывод** 10](#_Toc7370845)

[**Разработка оптимизированной версии** 11](#_Toc7370846)

[**План отмены допущений** 11](#_Toc7370847)

[**UML диаграмма активности** 11](#_Toc7370848)

[**Результаты испытаний** 11](#_Toc7370849)

[**Эффективность** 13](#_Toc7370850)

[**Вывод** 14](#_Toc7370851)

[**Разработка оконного интерфейса** 15](#_Toc7370852)

[**Приложение 1. Листинг быстрого прототипа.** 16](#_Toc7370853)

[**Приложение 2. Листинг оптимизированной версии** 23](#_Toc7370854)

# **Вариант 13**

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

# **Описание алгоритма**

Этот метод, также как и метод Хаффмана, строит кодировочное дерево, однако алгоритм его построения отличается:

1. Символы исходного алфавита упорядочиваются по невозрастанию вероятностей. В результате получаем последовательность (a i1 ,:,a iN ), в которой для всех j = 1, 2,:, N – 1 выполняется соотношение a ij >= a ij+1 .
2. Переменной-счётчику t присваивается значение 1.
3. Рассматриваемую последовательность разбиваем на M групп (в частности, две), не меняя порядка следования символов, так, чтобы суммарные вероятности символов в каждой группе были примерно одинаковы и максимально близки к 1/M. Получаем совокупность подпоследовательностей G 1 = (ai1 , ai2 , : ,ai k1 ),

G 2 = (a i k1+1 , a i k1+2 , : ,a ik2 ), …G m .

1. Формируем t-й символ кодовых слов. Всем символам из подпоследовательности G S приписываем символ bS, s=1,2, …, M.
2. Переменная-счётчик увеличивается на 1.
3. Просматриваем все группы - подпоследовательности. Если некоторая группа G S состоит из одного символа a j то для этого a j процесс построения кодового слова считается законченным. Для каждой из этих групп, содержащих по 2 и более символов, выполняем действия, соответствующие Шагу 3 и Шагу 4. В результате получаем очередные t-е символы кодовых слов. После просмотра всех групп, осуществляется переход к Шагу 5.

Процесс работы алгоритма заканчивается, когда все группы будут содержать ровно по одному символу исходного алфавита.

Следует заметить, что дерево Шеннона-Фано строится несколько быстрее и проще, чем в методе Хаффмана, но дает меньшую степень сжатия.

# **Разработка быстрого прототипа**

## **Список принятых допущений на этапе разработки прототипа программы:**

1. Проблема передачи модели кодирования.

Допущение: хранение модели кодирования в оперативной памяти.

Способ решения: передача модели кодирования кодером декодеру в заголовке сжатого файла.

1. Проблема обнаружения конца файла декодером.

Допущение: хранение длины входного потока кодера в оперативной памяти.

Способ решения: длину входного потока потребуется передавать в заголовке файла или использовать служебный символ признака конца файла.

**Алгоритм записи байта в файл**

После того, как мы создали бинарный файл, в который будем записывать полученные коды, мы создаем две переменные: int b=0, count =0. Первая представляет собой код в десятичной системе счисления, а вторая, количество битов в собираемом байте.

Для каждого символа входного потока мы проходим цикл по всему алфавиту, упорядоченному по частоте появления символа. Алфавит представлен классом symbol, в котором помимо самого символа содержится частота его появления и строка, в которой записан код.

Если символ алфавита и символ входного потока совпадает, мы попадаем в новый цикл: для каждого символа строки кода этого символа в байт записывается код этого символа, а значение переменной count увеличивается на 1. Далее, если мы собрали полный байт, то есть значение count==8, мы выбрасываем b, предварительно переконвертировав его в байтовый формат, в файл, а значения count и b обнуляем. Если же байт еще не полный, мы проводим битовый сдвиг влево на одну позицию. На этом цикл прерывается, чтобы не проверять весь оставшийся алфавит.

Если значение count после прохождения цикла оказалось отлично от нуля, то есть у нас остался неполный байт, мы сбрасываем его в файл. На этом упаковка считается законченной.

## **UML диаграмма активности**

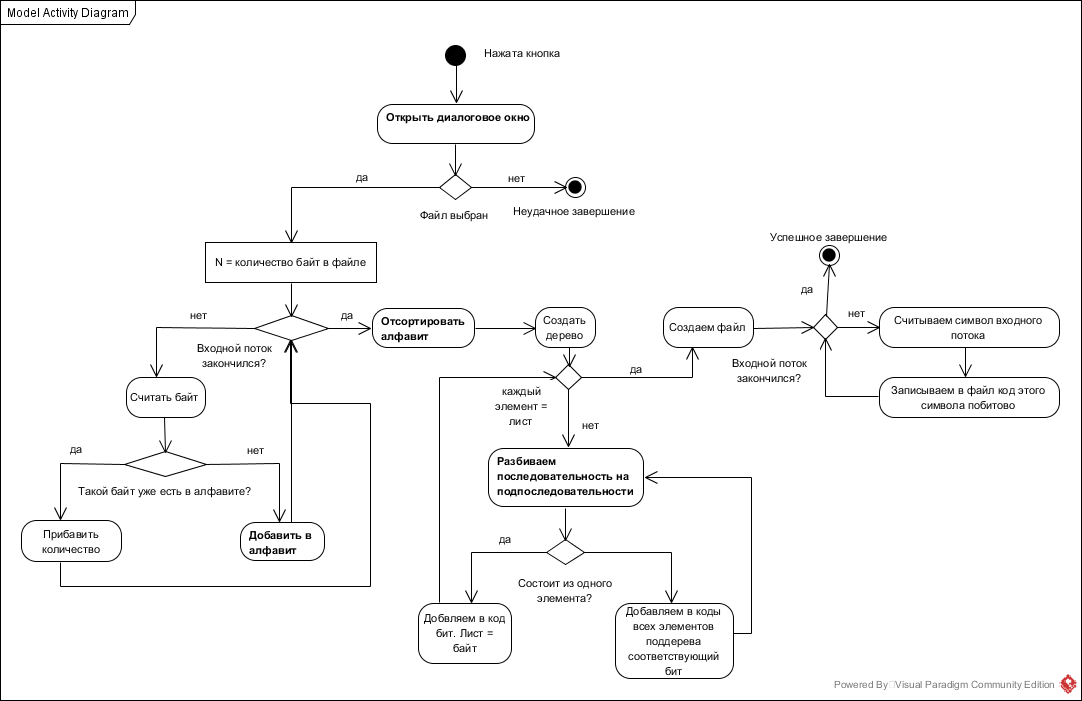


Рис. 1. Быстрый прототип кодера

## **Результаты испытаний**

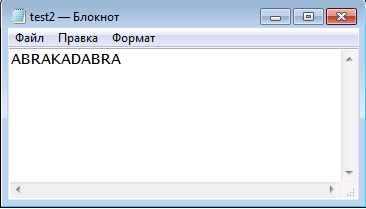


Рис. 2. Текст кодируемого файла

Открывая приложение мы видим окно, где нужно выбрать файл для кодирования и декодирования (рис.3).

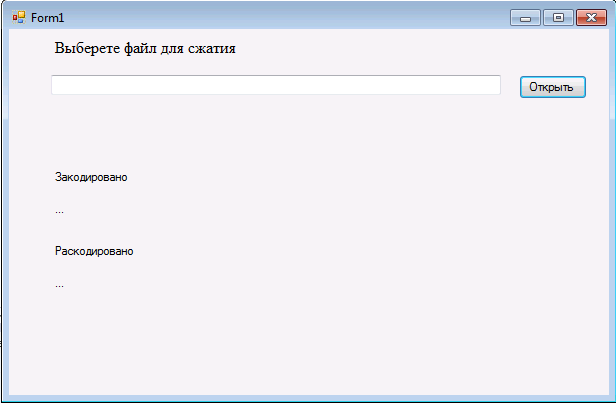


Рис.3. Стартовое окно.

Выбираем файл, и программа выводит то, что закодировано и раскодировано (рис.4).

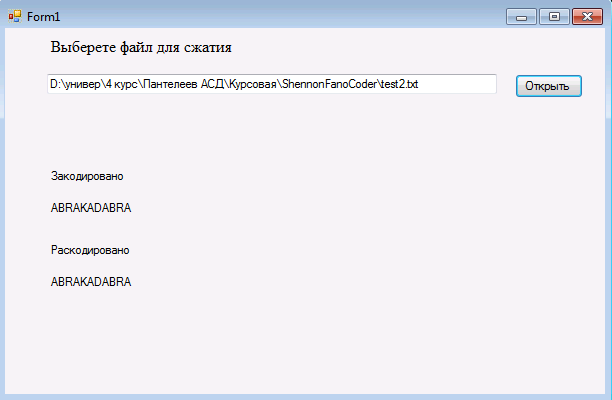


Рис.4. Результат работы программы.

Декодер помещает раскодированный текст в файл, мы можем так же открыть его (рис. 5).

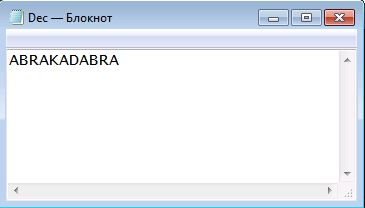


Рис.5. Текст декодированного файла.

Результат составления дерева представлен на рисунках 6-7.

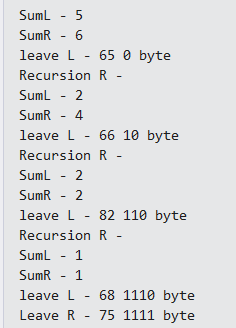


Рис.6.

A

B

R

K

D

Рис.7.

Закодированный файл, открытый через блокнот, будет иметь вид, представленный на рисунке 8.

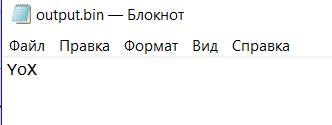


Рис.8

## **Вывод**

На этапе разработки быстрого прототипа мы изучили принципы работы алгоритма Шеннона-Фано. Построили дерево для кодирования байтов. Создали работающую версию программы, реализующую данный алгоритм.

На следующем этапе мы разработаем оптимизированную версию данного алгоритма, реализующей сжатие документов и их декодирование без допущений. Кодер и декодер будут работать согласованно, но независимо друг от друга.

# **Разработка оптимизированной версии**

## **План отмены допущений**

1. Передача от кодера декодеру длины входного потока в заголовке закодированного файла.
2. Передача от кодера декодеру модели кодирования в заголовке сжатого файла.

## **UML диаграмма активности**

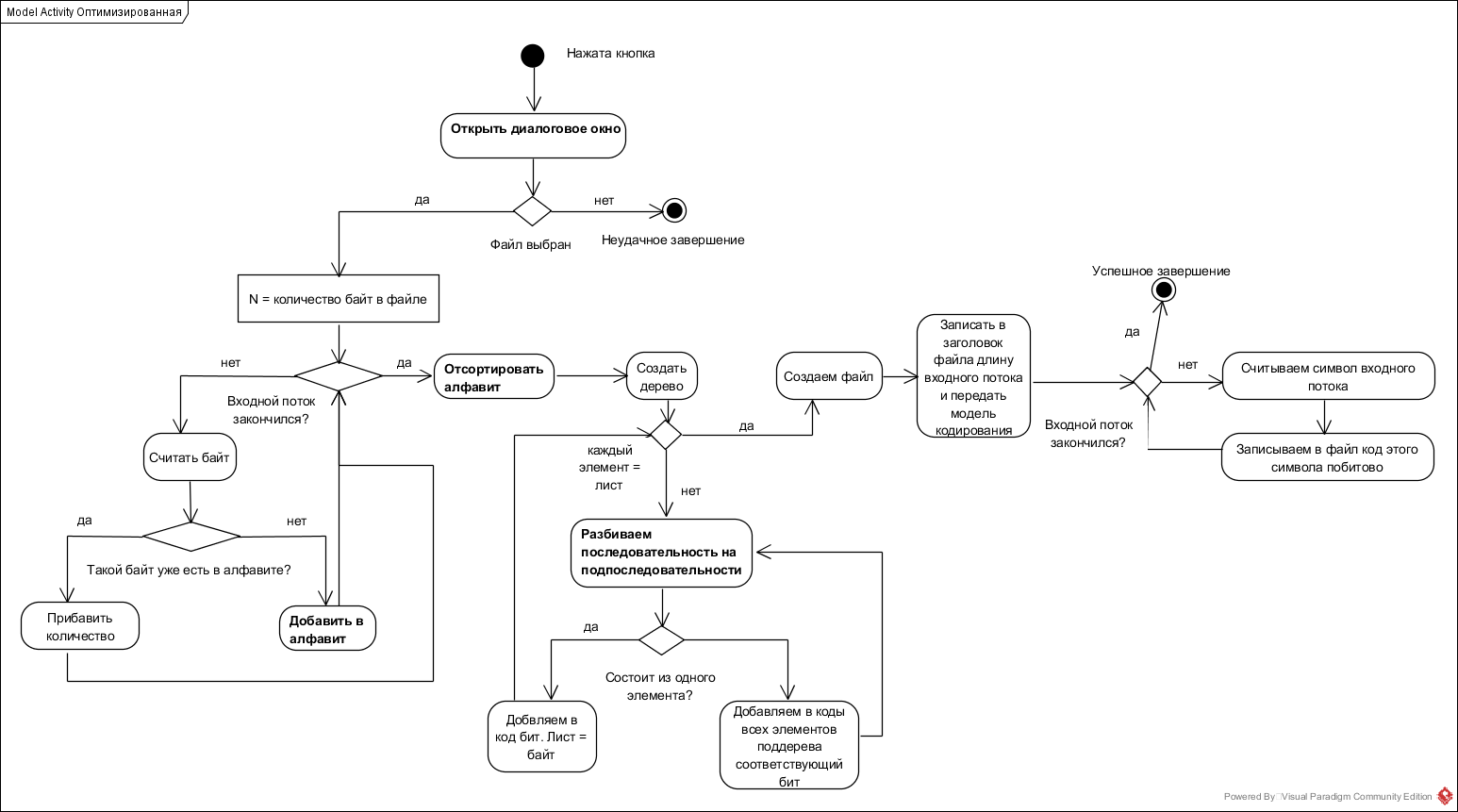


Рис. 9. Оптимизированная версия

## **Результаты испытаний**



Рис. 10. Кодируемый файл

Открывая приложение мы видим окно, где нужно выбрать файл для кодирования и декодирования (рис.11).

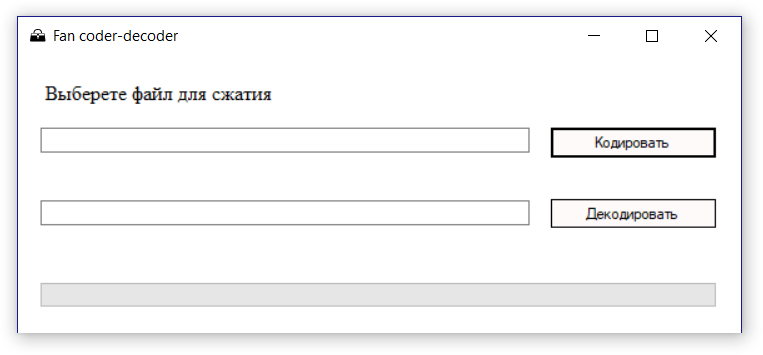


Рис.11. Стартовое окно.

Выбираем файл, и программа выводит то, что закодировано и раскодировано (рис.12).

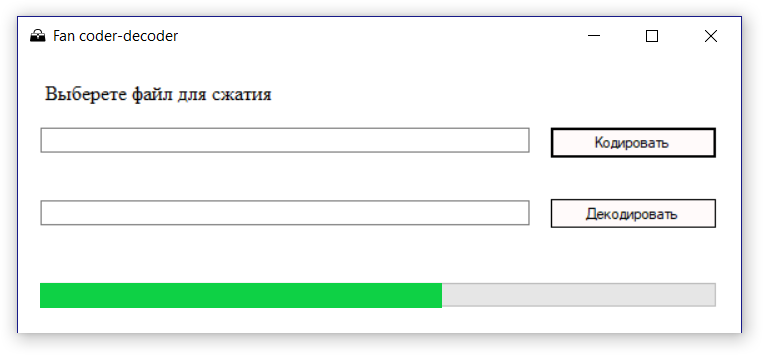


Рис.12. Процесс кодировки.

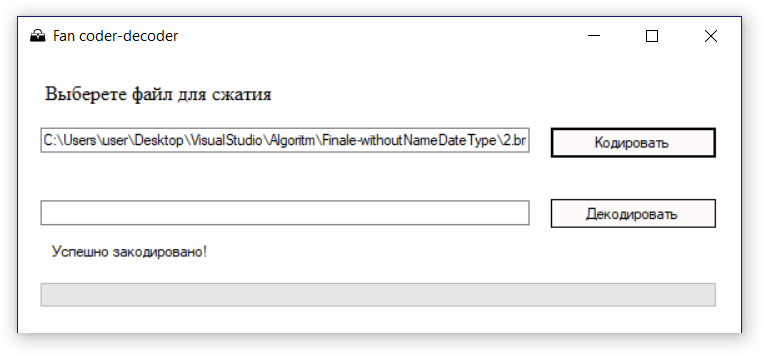


Рис.12. Результат работы кодера.

Кодер помещает закодированные данные в файл с расширением .fan, если открыть его через блокнот, файл будет иметь вид (рис. 13).



Рис.13. Текст закодированного файла.

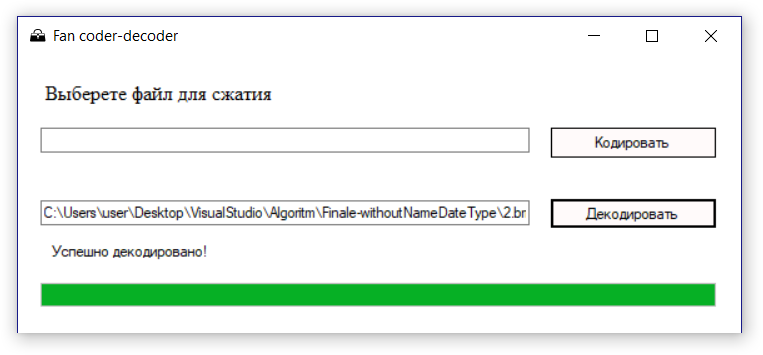


Рис.13. Результат декодирования файла.

Декодированный файл, будет иметь то же название файла и то же расширение, что и исходный файл. Декодированная картинка представлена на рисунке 14.



Рис.14. Результат работы декодера

**Эффективность**

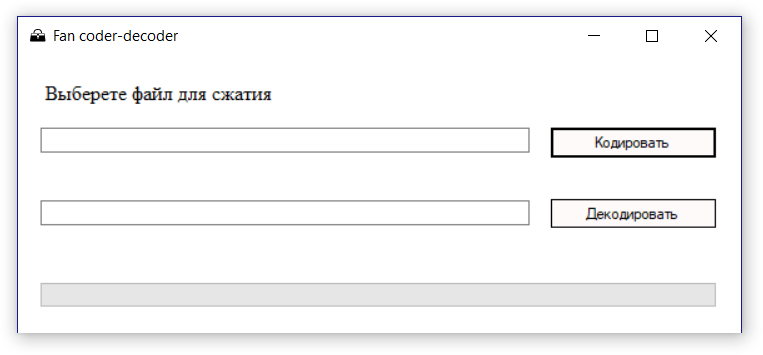
Для того, чтобы проанализировать эффективность программы сравним степень сжатия различных файлов.

|  |  |  |
| --- | --- | --- |
| До сжатия | После сжатия | Тип файла |
| 6076 КБ | 4629 КБ | bmp |
| 7,86 МБ | 8,02 МБ | pdf |

**Вывод**

о практическом испытании алгоритма

# **Разработка оконного интерфейса**



# **Приложение 1. Листинг быстрого прототипа.**

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ShennonFanoCoder

{

public partial class Form1 : Form

{

private OpenFileDialog openFileDialog1; //opening file

int N; //amount of symbols

public List<symbol> Cfile; //symbols of file to code in array = alphabet

char[] messfile; //the whole message from the file to code

// TreeView<symbol> Tree; //TreeNode -- node

public class Tree<T>

{

public int weight = 0;//sums of parts of table

public Tree<T> Left, Right;

public T leave = default(T);//null

public Tree<T> parent;

}

Tree<symbol> tree;

/// <summary>

/// Класс, содержащий байт, вероятность его встречи и его код

/// </summary>

public class symbol

{

public char ch; // the symbol itself

public int prob = 0; // number of symbol and their probability

public string code = ""; //the code they are going to have

public List<Byte> byteList = new List<Byte>();

// public List<Bit> byteList = new List<Byte>();

public BitArray[] bitArray = new BitArray[8];

}

public Form1()

{

InitializeComponent();

}

/// <summary>

/// Выбор файла

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void textBox1\_MouseDoubleClick(object sender, MouseEventArgs e) //open file

{

openFileDialog1 = new OpenFileDialog();

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

textBox1.Text = openFileDialog1.FileName;

Opening();

}

}

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

{

openFileDialog1 = new OpenFileDialog();

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

textBox1.Text = openFileDialog1.FileName;

Opening();

}

} //open file

void Opening()

{

Cfile = new List<symbol>();

System.IO.StreamReader sr = new

System.IO.StreamReader(openFileDialog1.FileName); // got file to code

N = Convert.ToInt32(sr.BaseStream.Length); //got amount of symbols in file to code

sr.BaseStream.Position = 0;

messfile = new char[N];

label3.Text = "";

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

{

messfile[i] = Convert.ToChar(sr.Read());

label3.Text += messfile[i]; //getting the message

if (Cfile.Count == 0) //if first symbol

{

symbol c = new symbol();

c.ch = messfile[i];

c.code = null;

c.prob = 1;

Cfile.Add(c);

}

else // if not

{

bool done = false; // if the alphabet was renewed

for (int j = 0; j < Cfile.Count; j++) //getting the alphabet

{

if (!done)

{

if (Cfile[j].ch == messfile[i]) // if exists

{

Cfile[j].prob++;

done = true;

}

else

if (j == Cfile.Count - 1)

{

symbol c = new symbol();

c.ch = messfile[i];

c.code = null;

c.prob = 1;

Cfile.Add(c);

done = true;

}

}

}

}

}

sr.Close();

Sort();

}

public class NameComparer : IComparer<symbol>

{

public int Compare(symbol k1, symbol k2)

{

if (k1.prob > k2.prob)

{

return -1;

}

else if (k1.prob < k2.prob)

{

return 1;

}

return 0;

}

}

void Sort()

{

NameComparer cn = new NameComparer();

Cfile.Sort(cn);

CreateTree();

Packing();

Decoder();

}

void CreateTree()

{

tree = new Tree<symbol>();

int sum = 0;

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

{

sum += Cfile[i].prob;

}

tree.weight = sum;

tree.parent = null;

BuildingTree(0, Cfile.Count - 1, sum, tree);

}

void BuildingTree(int Li, int Ri, int sum, Tree<symbol> node)

//index Left, Right, Sum of all elements, parent

{

int Lb = Li, Rb = Ri;

int sumL = Cfile[Li].prob, sumR = sum - Cfile[Li].prob; //sums of the parts

for (int LeftIndex = Li + 1; (LeftIndex < Cfile.Count) && (sumL + Cfile[LeftIndex].prob <= sumR - Cfile[LeftIndex].prob); LeftIndex++) //while sums are not close

{

sumL += Cfile[LeftIndex].prob;

sumR -= Cfile[LeftIndex].prob;

Li++;

}

Tree<symbol> node1 = new Tree<symbol>();

node1.weight = sumL;

Console.WriteLine("SumL - " + sumL);

Tree<symbol> node2 = new Tree<symbol>();

node2.weight = sumR;

Console.WriteLine("SumR - " + sumR);

node.Left = node1;

node.Right = node2;

node1.parent = node;

node2.parent = node;

if (Li < Cfile.Count && Rb < Cfile.Count)

{

if (sumL > 0)

if (Li - Lb == 0) //if consists of one element

{

node1.leave = Cfile[Lb];

node1.leave.code += "0";

node1.leave.byteList.Add(0);

// node1.leave.byteList.Add(Convert.ToByte(0, 2));

Console.WriteLine("leave L - " + node1.leave.ch + " " + node1.leave.code + " byte " + Encoding.ASCII.GetString(node1.leave.byteList.ToArray()));

}

else

{

Console.WriteLine("Recursion L -");

for (int i = Lb; i <= Li; i++)

{

Cfile[i].code += "0";

Cfile[i].byteList.Add(0);

}

BuildingTree(Lb, Li, sumL, node1);

}

if (sumR > 0)

if (Rb - (Li + 1) == 0) //if consists of one element

{

node2.leave = Cfile[Rb];

node2.leave.code += "1";

node2.leave.byteList.Add(1);

Console.WriteLine("Leave R - " + node2.leave.ch + " " + node2.leave.code + " byte " + Encoding.ASCII.GetString(node2.leave.byteList.ToArray()));

}

else

{

Console.WriteLine("Recursion R - ");

for (int i = Li + 1; i <= Rb; i++)

{

Cfile[i].code += "1";

Cfile[i].byteList.Add(1);

}

BuildingTree(Li + 1, Rb, sumR, node2);

}

}

}

void Packing()

{

string path = System.IO.Path.Combine(Environment.CurrentDirectory, "output.bin");

try

{// Delete the file if it exists.

if (File.Exists(path))

{

// Note that no lock is put on the

// file and the possibility exists

// that another process could do

// something with it between

// the calls to Exists and Delete.

File.Delete(path);

}

// Create the file.

using (FileStream fs = File.Create(path))

{

int b=0;

int count=0;

foreach (char c in messfile)

{

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

{

if (c == Cfile[i].ch)

{

// Byte[] info = new UTF8Encoding(true).GetBytes(Cfile[i].code);

// fs.WriteByte(Convert.ToByte(Cfile[i].code));

//fs.Write(Cfile[i].byteList.ToArray(), 0, Cfile[i].byteList.Count);

for (int j=0;j<Cfile[i].code.Length;j++)

{

b += Convert.ToInt32(Cfile[i].code[j])-48;

count++;

if (count==8)

{

byte k = Convert.ToByte(b % 256);

fs.WriteByte(k);

count = 0;

b = 0;

}

else

{

b = Convert.ToInt32(b) << 1;

}

}

break;

}

}

}

if (count != 0)

{

byte k = Convert.ToByte(b % 256);

fs.WriteByte(k);

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

void Decoder()

{

FileStream fs = new FileStream("output.bin", FileMode.Open,FileAccess.Read);

BinaryReader r = new BinaryReader(fs);

// CreateTree();

byte b=0;

string str="";

int k;

int count = 0;

Tree<symbol> current = tree ;

while (str.Length<N)

{

if(count==0)

{

b = r.ReadByte();

count = 8;

}

// for (int i=0; str.Length<N; i++)

count--;

k = ((Convert.ToInt32(b)) >> count)&1;

if (k == 0)//в левое поддерево

{

current = current.Left;

}

else current = current.Right;

if(current.leave != null)

{

str = str + Convert.ToString(current.leave.ch);

current = tree;

}

}

StreamWriter SW = new StreamWriter(new FileStream("Dec.txt", FileMode.Create, FileAccess.Write));

SW.Write(str);

SW.Close();

}

}

}

# **Приложение 2. Листинг оптимизированной версии**

using System;

using System.Collections;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ShennonFanoCoder

{

public partial class Form1 : Form

{

private OpenFileDialog openFileDialog1; //opening file

int N; //amount of symbols

public List<symbol> Cfile; //symbols of file to code in array = alphabet

// byte[] messfile; //the whole message from the file to code

// TreeView<symbol> Tree; //TreeNode -- node

int[] alphabet = new int[256];

public class Tree<T>

{

public int weight = 0;//sums of parts of table

public Tree<T> Left, Right;

public T leave = default(T);//null

public Tree<T> parent;

}

Tree<symbol> tree;

/// <summary>

/// Класс, содержащий байт, вероятность его встречи и его код

/// </summary>

public class symbol

{

public byte ch; // the symbol itself

public int prob = 0; // number of symbol and their probability

public string code = ""; //the code they are going to have

}

public Form1()

{

InitializeComponent();

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

{

alphabet[i] = 0;

}

}

/// <summary>

/// Выбор файла

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void textBox1\_MouseDoubleClick(object sender, MouseEventArgs e) //open file

{

openFileDialog1 = new OpenFileDialog();

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

textBox1.Text = openFileDialog1.FileName;

Opening();

}

}

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

{

openFileDialog1 = new OpenFileDialog();

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

textBox1.Text = openFileDialog1.FileName;

Opening();

}

} //open file

/// <summary>

/// Считывание файла и заполнение алфавита

/// </summary>

void Opening()

{

Cfile = new List<symbol>();

System.IO.BinaryReader sr = new

System.IO.BinaryReader(new FileStream(openFileDialog1.FileName, FileMode.Open)); // got file to code

N = Convert.ToInt32(sr.BaseStream.Length); //got amount of symbols in file to code

sr.BaseStream.Position = 0;

// messfile = new byte[N];

// label3.Text = "";

byte buf = default(byte);

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

{

// messfile[i] = (sr.ReadByte());//Convert.ToByte

// label3.Text += messfile[i]; //getting the message

buf = sr.ReadByte();

alphabet[(int)buf]++;

if (Cfile.Count == 0) //if first symbol

{

symbol c = new symbol();

c.ch = buf;

c.code = "";

c.prob = 1;

Cfile.Add(c);

}

else // if not

{

bool done = false; // if the alphabet was renewed

for (int j = 0; j < Cfile.Count; j++) //getting the alphabet

{

if (!done)

{

if (Cfile[j].ch == buf) // if exists

{

Cfile[j].prob++;

done = true;

}

else

if (j == Cfile.Count - 1)

{

symbol c = new symbol();

c.ch = buf;

c.code = null;

c.prob = 1;

Cfile.Add(c);

done = true;

}

}

}

}

}

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

// Console.Write(alphabet[i]);

// Console.WriteLine("");

sr.Close();

Sort();

Packing();

}

/// <summary>

/// функция сравнения алфавита по частоте

/// </summary>

public class NameComparer : IComparer<symbol>

{

public int Compare(symbol k1, symbol k2)

{

if (k1.prob > k2.prob)

{

// Console.WriteLine(k1.prob + " > " + k2.prob);

return -1; //меньше

}

else if (k1.prob < k2.prob)

{

// Console.WriteLine(k1.prob + " < " + k2.prob + " or " + k1.ch + " < " + k2.ch);

return 1; //больше

}

else return 0; // равно

}

}

/// <summary>

/// функция сравнения алфавита по байту

/// </summary>

public class NameComparer1 : IComparer<symbol>

{

public int Compare(symbol k1, symbol k2)

{

if (k1.ch > k2.ch)

{

// Console.WriteLine(k1.ch + " > " + k2.ch);

return 1; //больше

}

else if (k1.ch < k2.ch)

{

// Console.WriteLine(k1.ch + " < " + k2.ch);

return -1; //меньше

}

else return 0; // равно

}

}

/// <summary>

/// сортировка алфавита по частоте

/// </summary>

void Sort()

{

NameComparer1 cn1 = new NameComparer1();

Cfile.Sort(cn1);

NameComparer cn = new NameComparer();

Cfile.Sort(cn);

CreateTree();

}

/// <summary>

/// создание дерева и его корня

/// </summary>

void CreateTree()

{

tree = new Tree<symbol>();

int sum = 0;

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

{

sum += Cfile[i].prob;

}

tree.weight = sum;

tree.parent = null;

BuildingTree(0, Cfile.Count - 1, sum, tree);

}

/// <summary>

/// построение дерева рекурсивно

/// </summary>

void BuildingTree(int Li, int Ri, int sum, Tree<symbol> node)

//index Left, Right, Sum of all elements, parent

{

int Lb = Li, Rb = Ri;

int sumL = Cfile[Li].prob, sumR = sum - Cfile[Li].prob; //sums of the parts

for (int LeftIndex = Li + 1; (LeftIndex < Cfile.Count) && (sumL + Cfile[LeftIndex].prob <= sumR - Cfile[LeftIndex].prob); LeftIndex++) //while sums are not close

{

sumL += Cfile[LeftIndex].prob;

sumR -= Cfile[LeftIndex].prob;

Li++;

}

Tree<symbol> node1 = new Tree<symbol>();

node1.weight = sumL;

// Console.WriteLine("SumL - " + sumL);

Tree<symbol> node2 = new Tree<symbol>();

node2.weight = sumR;

// Console.WriteLine("SumR - " + sumR);

node.Left = node1;

node.Right = node2;

node1.parent = node;

node2.parent = node;

if (Li < Cfile.Count && Rb < Cfile.Count)

{

if (sumL > 0)

if (Li - Lb == 0) //if consists of one element

{

node1.leave = Cfile[Lb];

node1.leave.code += "0";

// node1.leave.byteList.Add(0);

// node1.leave.byteList.Add(Convert.ToByte(0, 2));

// Console.WriteLine("leave L - " + node1.leave.ch + " " + node1.leave.code + " byte ");

}

else

{

// Console.WriteLine("Recursion L -");

for (int i = Lb; i <= Li; i++)

{

Cfile[i].code += "0";

// Cfile[i].byteList.Add(0);

}

BuildingTree(Lb, Li, sumL, node1);

}

if (sumR > 0)

if (Rb - (Li + 1) == 0) //if consists of one element

{

node2.leave = Cfile[Rb];

node2.leave.code += "1";

// node2.leave.byteList.Add(1);

// Console.WriteLine("Leave R - " + node2.leave.ch + " " + node2.leave.code + " byte ");

}

else

{

// Console.WriteLine("Recursion R - ");

for (int i = Li + 1; i <= Rb; i++)

{

Cfile[i].code += "1";

// Cfile[i].byteList.Add(1);

}

BuildingTree(Li + 1, Rb, sumR, node2);

}

}

}

void Packing()

{

string path = System.IO.Path.Combine(Environment.CurrentDirectory, "output.bin");

try

{// Delete the file if it exists.

if (File.Exists(path))

{

// Note that no lock is put on the

// file and the possibility exists

// that another process could do

// something with it between

// the calls to Exists and Delete.

File.Delete(path);

}

System.IO.BinaryReader sr = new

System.IO.BinaryReader(new FileStream(openFileDialog1.FileName, FileMode.Open)); // got file to code

N = Convert.ToInt32(sr.BaseStream.Length); //got amount of symbols in file to code

sr.BaseStream.Position = 0;

// Create the file.

using (FileStream fs = File.Create(path))

{

// длина входного потока

byte[] bytes = BitConverter.GetBytes(N);

fs.Write(bytes, 0, bytes.Length);

//таблица кодирования

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

{

bytes = BitConverter.GetBytes(alphabet[i]);

fs.Write(bytes, 0, bytes.Length);

}

//сообщение

int b = 0;

int count = 0;

for (int t = 0; t < N; t++)

{

byte buf = sr.ReadByte();

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

{

if (buf == Cfile[i].ch)

{

for (int j = 0; j < Cfile[i].code.Length; j++)

{

b += Convert.ToInt32(Cfile[i].code[j]) - 48;

count++;

if (count == 8)

{

byte k = Convert.ToByte(b % 256);

fs.WriteByte(k);

count = 0;

b = 0;

// Console.WriteLine("Pushed symbol " + k);

}

else

{

b = Convert.ToInt32(b) << 1;

}

}

// Console.WriteLine("Read symbol " + buf);

// Console.WriteLine("Code symbol " + Cfile[i].code);

break;

}

}

}

if (count != 0)

{

byte k = Convert.ToByte(b % 256);

fs.WriteByte(k);

// Console.WriteLine("Pushed symbol " + k);

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

void Decoder(string name)

{

FileStream fs = new FileStream(name, FileMode.Open, FileAccess.Read);

BinaryReader r = new BinaryReader(fs);

byte b = 0;

int k;

int count = 0;

N = 0;

N = r.ReadInt32();

Cfile = null;

int per = 0;

Cfile = new List<symbol>();

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

{

per = r.ReadInt32();

if (per != 0)

{

symbol c = new symbol();

c.ch = (byte)i;

c.prob = per;

Cfile.Add(c);

}

}

tree = null;

Sort();

Tree<symbol> current = tree;

string path = System.IO.Path.Combine(Environment.CurrentDirectory, name);

try

{// Delete the file if it exists.

if (File.Exists(path))

{

File.Delete(path);

}

int g=0;

using (FileStream stream = File.Create(path))

{

while (g < N)

{

if (count == 0)

{

b = r.ReadByte();

// Console.WriteLine("Read " + b);

count = 8;

}

count--;

k = ((Convert.ToInt32(b)) >> count) & 1;

if (k == 0)//в левое поддерево

{

current = current.Left;

// Console.WriteLine("to left");

}

else {

current = current.Right;

// Console.WriteLine("to right");

}

if (current.leave != null)

{

stream.WriteByte(current.leave.ch);

g++;

// Console.WriteLine("Write " + current.leave.ch);

current = tree;

}

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

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

{

openFileDialog1 = new OpenFileDialog();

if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)

{

textBox2.Text = openFileDialog1.FileName;

}

Decoder(openFileDialog1.FileName);

Application.Exit();

}

}

}