МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

«ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ ЕВФРОСИНИИ ПОЛОЦКОЙ»

Факультет информационных технологий

Кафедра технологий программирования

**КУРСОВОЙ ПРОЕКТ**

**по дисциплине «Конструирование программного обеспечения»**

**Тема: «Архиватор»**

|  |  |
| --- | --- |
| Выполнил: | Катушёнок И.В.  студент группы 21ИТ-1 |
| Проверила: | Дьякова А.С.  должность кафедры ТП преподаватель-стажер кафедры ТП |

|  |
| --- |
| Отметка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата защиты: «\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.  Члены комиссии:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ А.А. Скуковская  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ А.С. Дьякова  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ М.В. Деканова |

Полоцк, 2022

# СОДЕРЖАНИЕ

Изм.

Лист

№ докум.

Подпись

Дата

КИВ.502900 ПЗ

Разраб.

Катушёнок И.В.

Провер.

Дьякова А.С.

Т. Контр.

Н. Контр.

Утверд.

Лист

Лит.

Учреждение образования <<Полоцкий государственный университет имени Евфросинии Полоцкой>>, гр. 21-ИТ-1

Листов

Отчет по учебной ознакомительной практике

5

21

**ВВЕДЕНИЕ**

Существующие алгоритмы сжатия данных можно разделить на два больших класса - с потерями, и без. Алгоритмы с потерями обычно применяются для сжатия изображений и аудио. Эти алгоритмы позволяют достичь больших степеней сжатия благодаря избирательной потере качества. Однако, по определению, восстановить первоначальные данные из сжатого результата невозможно. [1]

Алгоритм Хаффмана - жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных. [2]

В рамках курсового проекта создано простейшее windows-приложение, «Архиватор».

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900.ПЗ

Для разработки курсового проекта была выбрана среда разработки Microsoft Visual Studio 2022, на объекто-ориентированном языке программирования общего назначения C#. Так же в разработке задействован интерфейс программирования приложений, отвечающий за графический интерфейс пользователя Windows Forms.

# 1 ПОСТАНОВКА ЗАДАЧ РЕАЛИЗАЦИИ

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

Архиватор - программа, предназначенная для упаковки без потерь одного и более файлов в единый файл-архив или в серию архивов для удобства переноса и/или хранения данных. Распаковка архивов выполняется с помощью того же архиватора либо посредством сторонних совместимых утилит. Большинство современных архиваторов также выполняет сжатие упаковываемых в архив данных. [3]

Алгоритмы сжатия без потерь применяются для уменьшения размера данных, и работают таким образом, что возможно восстановить данные в точности такими, какие они были до сжатия. Они применяются в коммуникациях, архиваторах и некоторых алгоритмах сжатии аудио и графической информации.

Идея алгоритма состоит в следующем: зная вероятности появления символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частотностей символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево). [2]

- Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.

-Выбираются два свободных узла дерева с наименьшими весами.

-Создается их родитель с весом, равным их суммарному весу.

-Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.

-Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой - бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.

-Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева. [2]

# 2 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Проектирование программного обеспечения - процесс создания проекта программного обеспечения (ПО), а также дисциплина, изучающая методы проектирования. Проектирование ПО является частным случаем проектирования продуктов и процессов. [5]

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

Перед разработкой программы необходимо определится с архитектурой компонентов, интерфейсов. Необходимо спроектировать модули включающие в себя функции программы.

Модульное программирование — это организация программы как совокупности небольших независимых блоков, называемых модулями, структура и поведение которых подчиняются определённым правилам. Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. [6]

Основные программные модули :

-Huffman.cs

-Node.cs

-PriorityQueue.cs

-Form1.cs

Программа состоит из окна разрешением 440x320, кнопки отвечающей за выбор директории файла, кнопки отвечающей за архивацию файла, кнопки отвечающей за разархивацию файла и трех ярлыков для вывода текущего пути и для отображения произведенной программой операцией.

# 3 РЕАЛИЗАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

# 3.1 Модуль Huffman

Модуль Huffman содержит основные функции архивации и разархивации.

CompressFile – Основная функция архивации файлов. Считывает байты файла. Сжимает файл и записывает сжатый файл.

Листинг 3.1 – Функция CompressFile

public void CompressFile(string dataFilename, string archiveFilename)

{

byte[] data = File.ReadAllBytes(dataFilename);

byte[] arch = CompressBytes(data);

File.WriteAllBytes(archiveFilename, arch);

}

DecompressFile – Основная функция разархивации. Считывает байты архива, разархивирует его и записывает.

Листинг 3.2 – Функция DecompressFile

public void DecompressFile(string archiveFilename, string dataFilename)

{

byte[] archive = File.ReadAllBytes(archiveFilename);

byte[] data = DecompressBytes(archive);

File.WriteAllBytes(dataFilename, data);

}

DecompressBytes – На основе функций Decompress и построенного дерева Хаффмана с частотами полученными из функции CalculateFreq сжимает байты.

Листинг 3.3 – Функция DecompressBytes

public byte[] DecompressBytes(byte[] archive)

{

ParseHeader(archive, out int dataLength, out int startIndex, out int[] freqs);

node root = CreateHuffmanTree(freqs);

byte[] data = Decompress(archive, startIndex, dataLength, root);

return data;

}

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

Decompress – Основная функция разархивации байтов.

Листинг 3.4 – Функция Decompress

private byte[] Decompress(byte[] archive, int startIndex, int dataLength, node root)

{

int size = 0;

node curr = root;

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

for (int j = startIndex; j < archive.Length; j++)

for (int bit = 1; bit <= 128; bit <<= 1)

{

bool zero = (archive[j] & bit) == 0;

if (zero)

curr = curr.bit0;

else curr = curr.bit1;

if (curr.bit0 != null)

continue;

if (size++ < dataLength)

data.Add(curr.symbol);

curr = root;

}

return data.ToArray();

}

ParseHeader – Функция считывающая заголовок кодов Хаффмана.

Листинг 3.5 – Функция ParseHeader

private void ParseHeader(byte[] archive,

out int dataLength,

out int startIndex,

out int[] freqs)

{

dataLength = archive[0] |

(archive[1] << 8) |

(archive[1] << 16) |

(archive[1] << 24);

freqs = new int[256];

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

freqs[j] = archive[4 + j];

startIndex = 4 + 256;

}

CompressBytes – функция содержащая все функции для сжатия файлов.

Листинг 3.6 – Функция CompressBytes

private byte[] CompressBytes(byte[] data)

{

int[] freqs = CalculateFreq(data);

byte[] head = CreatHeadder(data.Length, freqs);

node root = CreateHuffmanTree(freqs);

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

string[] codes = CreateHuffmanCode(root);

byte[] bits = Compress(data, codes);

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

return head.Concat(bits).ToArray();

}

CreatHeadder – Функция осуществляет запись кодов Хаффмана в заголовок.

Листинг 3.7 – Функция CreatHeadder

private byte[] CreatHeadder(int dataLength, int[] freqs)

{

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

head.Add((byte)(dataLength & 255));

head.Add((byte)(dataLength >> 8 & 255));

head.Add((byte)(dataLength >> 16 & 255));

head.Add((byte)(dataLength >> 24 & 255));

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

head.Add((byte)freqs[j]);

return head.ToArray();

}

Compress – Основная функция сжатия байт на основе созданных кодов Хаффмана.

Листинг 3.8 – Функция Compress

private byte[] Compress(byte[] data, string[] codes)

{

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

byte sum = 0;

byte bit = 1;

foreach (byte symbol in data)

foreach (char c in codes[symbol])

{

if (c == '1')

sum |= bit;

if (bit < 128)

bit <<= 1;

else

{

bits.Add(sum);

sum = 0;

bit = 1;

}

}

if (bit > 1)

bits.Add(sum);

return bits.ToArray();

}

CreateHuffmanCode – Функция отвечающая за создание кода Хаффмана для байтов.

Листинг 3.9 – Функция CreateHuffmanCode

private string[] CreateHuffmanCode(node root)

{

string[] codes = new string[256];

Next(root, "");

return codes;

void Next(node Node, string code)

{

if (Node.bit0 == null)

codes[Node.symbol] = code;

else

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

{

Next(Node.bit0, code + "0");

Next(Node.bit1, code + "1");

}

}

}

CalculateFreq – Функция высчитывающая частоты байт файла.

Листинг 3.10 – Функция CalculateFreq

private int[] CalculateFreq(byte[] data)

{

int[] freqs = new int[256];

foreach (byte d in data)

freqs[d]++;

NormalizeFreqs(freqs);

return freqs;

void NormalizeFreqs(int[] freqs)

{

int max = freqs.Max();

if (max <= 255) return;

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

if (freqs[j] > 0)

freqs[j] = 1 + freqs[j] \* 255 / (max + 1);

}

}

CreateHuffmanTree – Функция создающая дерево Хаффмана

Листинг 3.11 – Функция CreateHuffmanTree

private node CreateHuffmanTree(int[] freqs)

{

PriorityQueue<node> pq = new PriorityQueue<node>();

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

{

if (freqs[j] > 0)

pq.Enqueue(freqs[j], new node((byte)j, freqs[j]));

}

while (pq.Size() > 1)

{

node bit0 = pq.Dequeue();

node bit1 = pq.Dequeue();

int freq = bit0.freq + bit1.freq;

node next = new node(freq, bit0, bit1);

pq.Enqueue(freq, next);

}

return pq.Dequeue();

}

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

# 3.2 Модуль Node

Модуль Node содержит функции необходимые для реализации структур данных бинарного дерева и очереди с приоритетом.

Node – функция заполняющая узел очереди с приоритетом

Листинг 3.12 – Функция Node

public node(byte symbol, int freq)

{

this.symbol = symbol;

this.freq = freq;

}

Node – функция заполняющая узел бинарного дерева

Листинг 3.13 – Функция Node

public node(int freq, node bit0, node bit1)

{

this.freq = freq;

this.bit0 = bit0;

this.bit1 = bit1;

}

# 3.3 Модуль PriorityQueue

Модуль PriorityQueue содержит функции PriorityQueue, Size, Enqueue, Dequeue необходимые для работы с очередью с приоритетом.

PriorityQueue – Функция являющаяся конструктором для очереди с приоритетом.

Листинг 3.14 – Функция PriorityQueue

public PriorityQueue()

{

storage = new SortedDictionary<int, Queue<T>>();

size = 0;

}

Size – Функция возвращающая размер очереди

Листинг 3.15 – Функция PriorityQueue

public int Size() => size;

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

Enqueue – Функция добавляющая элемент в очередь с приоритетом

Листинг 3.16 – Функция Enqueue

public void Enqueue(int priority, T item)

{

if (!storage.ContainsKey(priority))

storage.Add(priority, new Queue<T>());

storage[priority].Enqueue(item);

size++;

}

Dequeue – Функция удаляющая элементы из очереди с приоритетом

Листинг 3.17 – Функция Dequeue

public T Dequeue()

{

if (size == 0)

throw new System.Exception("Queue is empty");

size--;

foreach (Queue<T> q in storage.Values)

if (q.Count > 0)

return q.Dequeue();

throw new System.Exception("Queue error");

}

# 3.4 Модуль Form1

Модуль Form1 содержит функции: SelectFilePath, ComressButton\_Click, DecompressButton\_Click, SPButton\_Click, FileInf. Вышеперечисленные функции отвечают за взаимодействие активных элементов графического интерфейса Windows Forms.

SelectFilePath – Функция отвечающая за вызов окна выбора пути к файлу.

Листинг 3.18 – Функция SelectFilePath

public string SelectFilePath()

{

var fileContent = string.Empty;

var filePath = string.Empty;

using (OpenFileDialog openFileDialog = new OpenFileDialog())

{

openFileDialog.InitialDirectory = "c:\\";

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

openFileDialog.FilterIndex = 2;

openFileDialog.RestoreDirectory = true;

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

//Get the path of specified file

filePath = openFileDialog.FileName;

}

}

return filePath;

}

CompressButton\_Click – Функция отвечающая за сжатия выбранного файла при нажатии соответствующей кнопки.

Листинг 3.19 – Функция CompressButton\_Click

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

{

try

{

string formatPath = path;

formatPath += ".huf";

huffman.CompressFile(path, formatPath);

Output.Text = "Compress Complete";

}

catch

{

Output.Text = "Compress Failed";

}

}

DecompressButton\_Click – Функция отвечающая за разархивацию выбранного файла при нажатии соответствующей кнопки.

Листинг 3.20 – Функция DecompressButton\_Click

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

{

try

{

string formatPath = path;

if (formatPath.IndexOf(".huf") == -1) throw new System.Exception("Incorrect path");

formatPath = formatPath.Remove(formatPath.Length - 4, 4);

huffman.DecompressFile(path, formatPath);

Output.Text = "Decompress Complete";

}

catch

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

{

Output.Text = "Decompress Failed";

}

}

SPButton\_Click – Функция отвечающая за выполнение функции SelectFilePath выбирающий путь к файлу при нажатии соответствующей кнопки.

Листинг 3.21 – Функция SPButton\_Click

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

{

try

{

path = SelectFilePath();

string TextPath = "";

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

{

TextPath += path[i];

if (i % 29 == 0 && i != 0) TextPath += System.Environment.NewLine;

}

PathLabel.Text = TextPath;

FileInfo();

Output.Text = "Path Selected";

}

catch

{

Output.Text = "Path Secelt Failed";

}

}

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

FileInfo – Функция отвечающая

Листинг 3.22 – Функция FileInfo

private void FileInfo()

{

string OutputInfo = "";

InfoBox.Text = "";

//NAME

OutputInfo += "Name : ";

string temp = "";

for (int i = path.Length - 1; path[i] != '\\'; i--)

temp += path[i];

string name = "";

for (int i = temp.Length - 1; i >= 0; i--)

name += temp[i];

OutputInfo += name;

OutputInfo += Environment.NewLine;

//TYPE

OutputInfo += "Type : ";

if (path.IndexOf(".txt") != -1)

{

OutputInfo += "Text Document";

}

else if (path.IndexOf(".png") != -1 || path.IndexOf(".jpg") != -1 || path.IndexOf(".bmp") != -1)

{

OutputInfo += "Picture";

}

else if (path.IndexOf(".mp3") != -1)

{

OutputInfo += "Sound";

}

else if (path.IndexOf(".mp4") != -1)

{

OutputInfo += "Video";

}

else

{

string tempType = "";

for (int i = path.Length - 1; path[i] != '.'; i--)

{

tempType += path[i];

}

string Type = "";

for (int i = tempType.Length - 1; i >= 0; i--)

{

Type += tempType[i];

}

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

OutputInfo += Type;

}

OutputInfo += Environment.NewLine;

//SIZE

OutputInfo += "Size : ";

byte[] data = File.ReadAllBytes(path);

int dataSize = 0;

dataSize = data.Length;

if (dataSize > 1048576)

{

dataSize /= 1048576;

OutputInfo += Convert.ToString(dataSize) + " MB";

}

else if (dataSize > 1024)

{

dataSize /= 1024;

Изм.

Лист

№ докум.

Подпись

Дата

Лист

6

КИВ.502900 ПЗ

OutputInfo += Convert.ToString(dataSize) + " KB";

}

else

{

OutputInfo += Convert.ToString(data.Length) + " Bytes";

}

OutputInfo += Environment.NewLine;

//MODIFIED

OutputInfo += "Modified : ";

var modified = File.GetLastWriteTime(path);

OutputInfo += Convert.ToString(modified);

InfoBox.Text = OutputInfo;

}

# 4 АНАЛИЗ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Анализ и тестирование являются важными частями при разработке программного обеспечения.

Тестиирование програаммного обеспеечения – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом. [7]

Существует несколько различных уровней тестирования :

Тестирование компонентов – тестируется минимально возможный для тестирования компонент, например, отдельный класс или функция. Часто тестирование компонентов осуществляется разработчиками программного обеспечения.

Интеграционное тестирование – тестируются интерфейсы между компонентами, подсистемами или системами. При наличии резерва времени на данной стадии тестирование ведётся итерационно, с постепенным подключением последующих подсистем.

Системное тестирование – тестируется интегрированная система на её соответствие требованиям.

Альфа-тестирование – имитация реальной работы с системой штатными разработчиками, либо реальная работа с системой потенциальными пользователями/заказчиком. Чаще всего альфа-тестирование проводится на ранней стадии разработки продукта, но в некоторых случаях может применяться для законченного продукта в качестве внутреннего приёмочного тестирования. Иногда альфа-тестирование выполняется под отладчиком или с использованием окружения, которое помогает быстро выявлять найденные ошибки. Обнаруженные ошибки могут быть переданы тестировщикам для дополнительного исследования в окружении, подобном тому, в котором будет использоваться программа.

Бета-тестирование – в некоторых случаях выполняется распространение предварительной версии (в случае проприетарного программного обеспечения иногда с ограничениями по функциональности или времени работы) для некоторой большей группы лиц с тем, чтобы убедиться, что продукт содержит достаточно мало ошибок. Иногда бета-тестирование выполняется для того, чтобы получить обратную связь о продукте от его будущих пользователей.[7]

При запуске программы можно наблюдать её интуитивно понятный пользовательский интерфейс (Рисунок 4.1)

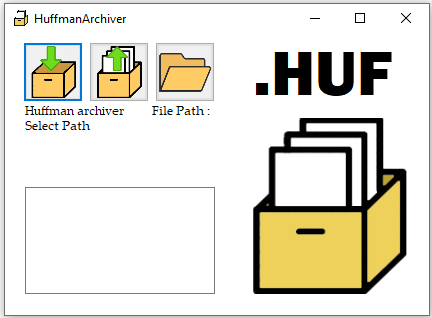


Рисунок 4.1 – Пользовательский интерфейс программы

Пользовательский интерфейс программы представляет собой три кнопки для архивации выбранного файла, кнопки для его разархивации и кнопки для выбора пути к файлу. Также можно увидеть поле для отображения информации о выбранном файле.

Для тестирования работы программы создадим текстовый файл и заполним его текстом. Так же стоит обратить внимание на Размер, исходного файла, который составил 1161 байт. (Рисунок 4.2)

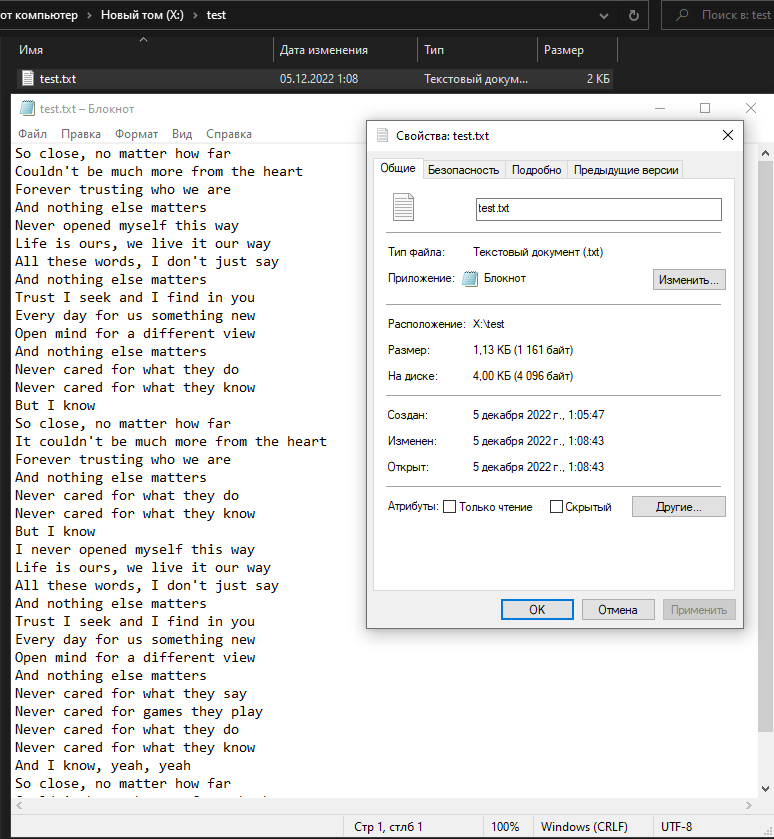


Рисунок 4.2 – Текстовый файл

# ЗАКЛЮЧЕНИЕ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ