

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина – Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
АРХИВАТОР

БГУИР КП 1-40 02 01 207 ПЗ

Студент

А.А. Горох

Руководитель

Е.В. Богдан

МИНСК 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Б. В. Никульшин
(подпись)
_____ 2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Гороху Андрею Александровичу

Тема проекта Архиватор

2. Срок сдачи студентом законченного проекта 15 декабря 2023 г.

3. Исходные данные к проекту Файловая система NTFS (с установленной ОС Windows 10), набор инструментов разработки MinGW-w64, метаобъектный компилятор и компоненты фреймворка QT, среда разработки QT Creator.

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Задание.

2. Введение.

3. Обзор литературы.

3.1 Анализ существующих аналогов.

3.2 Обзор методов и алгоритмов решения поставленной задачи.

4. Функциональное проектирование.

4.1 Структура входных и выходных данных.

4.2 Разработка диаграммы классов.

4.3 Описание классов.

5. Разработка программных модулей.

5.1 Разработка схем алгоритмов.

5.2 Разработка алгоритмов.

6. Результаты работы.

6.1 Руководство пользователя.

7. Заключение.

8. Литература.

9. Приложения.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.

2. Схема алгоритма метода *Huffman_code::compress_file()*

3. Схема алгоритма метода *Huffman_code::compress_file()*

6. Консультант по проекту (с обозначением разделов проекта) Е.В. Богдан

7. Дата выдачи задания 15.09.2023г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала к 10.10.23 – 15 %;

разделы 2, 3 к 01.11.23 – 10 %;

разделы 4 к 01.11.23 – 20 %;

разделы 5 к 15.11.23 – 35 %;

раздел 6,7,8 к 01.12.23 – 5 %;

раздел 9 к 01.12.23 – 5%;

оформление пояснительной записки и графического материала к 15.12.23 – 10 %

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ Е.В. Богдан

(подпись)

Задание принял к исполнению _____

(дата и подпись студента)

СОДЕРЖАНИЕ

Введение	5
1 Постановка задачи	6
2 Обзор литературы	7
2.1 Анализ существующих аналогов	7
2.2 Обзор методов и алгоритмов решения поставленной задачи . .	10
3 Функциональное проектирование	14
3.1 Структура входных и выходных данных	14
3.2 Разработка диаграммы классов	14
3.3 Описание классов	14
4 Разработка программных модулей	21
4.1 Разработка схем алгоритмов	21
4.2 Разработка алгоритмов	22
5 Результаты работы	23
5.1 Руководство пользователя	23
Заключение	25
Список использованных источников	26
Приложение А (обязательное) Диаграмма классов	27
Приложение Б (обязательное) Блок-схемы алгоритмов методов	28
Приложение В (обязательное) Полный код программы	30
Приложение Г (обязательное) Ведомость документов	41

ВВЕДЕНИЕ

Семимильными шагами ступает прогресс по планете Земля. Каждый день мы можем наблюдать, как в мире происходит так называемая «цифровая революция», которая началась еще в последних десятилетиях прошлого века. Связана она с распространением информационных технологий и проникновением их во все сферы жизни общества.

В современном мире технологий, многие компьютерные программы и файлы занимают на электронных накопителях информации все больше и больше места. Общество со временем начало использовать специальные программы для сжатия информации и последующего извлечения – архиваторы. Со временем, такое программное обеспечение стало неотъемлемой частью нашей работы за компьютером. Но, казалось бы, простое сжатие информации тоже имеет свои загвоздки и нюансы. Поэтому, пользователям нужно понимать основные механизмы архивации данных.

Для изучения и создания программы архивации файлов в данном курсовом проекте будет использоваться компилируемый строго типизированный язык программирования C++, а также кроссплатформенный фреймворк QT. Чтобы увеличить эффективность программы и правильно структурировать исходный код, для реализации основного алгоритма используется объектно-ориентированная парадигма программирования.

С помощью абстракции, инкапсуляции и полиморфизма структурируется и упрощается основной алгоритм программы. Стандартная библиотека шаблонов (Standart Template Library, STL) предоставляет обширный набор обобщенных контейнеров и множество методов их обработки. Благодаря STL сохранность данных увеличивается, уменьшается вероятность утечек памяти – код становится более надежным. Наследование помогает правильно выстроить интерфейс и видоизменить в приложении объекты, предопределенные в фреймворке QT. QT в свою очередь дает возможность эффективно описывать файловую систему, создавать лаконичное визуальное оформление различных виджетов.

Данное оконное приложение позволяет визуализировать объекты файловой системы используя основные элементы оконных приложений, а также наглядно иллюстрирует работу алгоритма Хаффмана со всеми его недостатками.

1 ПОСТАНОВКА ЗАДАЧИ

Для реализации программы необходимо детально изучить алгоритмы считывания данных и построения деревьев поиска для дальнейшей компоновки кодов символов. При написании рабочего кода программы можно столкнуться с трудностями при чтении информации в файл, выводе таблицы кодировки и самих закодированных данных в архивный файл. Также не стоит забывать про ограниченные возможности персональных компьютеров и правильно организовывать запоминание информации в промежуточные сущности и контейнеры.

В данном приложении необходимо реализовать просмотр всех возможных файлов для сжатия и разархивации. Предусмотреть возможность выбора пользователем уже сжатого файла для дальнейшей архивации (программа может повести себя непредсказуемо). При любой операции программа должна предупреждать пользователя о результатах данной операции. Стоит предусмотреть возможность изменения названия файла при разархивации и конфликте имен.

Основная часть алгоритма для сжатия должна разделяться на следующие логические части:

- считывание исходного файла;
- построение дерева поиска;
- построение таблицы шифрования данных;
- шифрование данных;
- запись таблицы кодов в архивный файл;
- запись закодированных данных в архивный файл.

Для алгоритма разархивации необходимо иметь такие методы:

- считывание исходного файла;
- считывание таблицы кодирования данных;
- дешифрование данных;
- запись декодированных данных в новый файл.

2 ОБЗОР ЛИТЕРАТУРЫ

2.1 Анализ существующих аналогов

На данный момент существует огромное множество программ для архивации данных. Какие-то из них имеют свои преимущества, какие-то – свои недостатки. В данном разделе рассматриваются наиболее известные и удобные приложения для сжатия и менеджмента файлов.

2.1.1 WinRAR

WinRAR – это лицензионное программное обеспечение для сжатия данных различными способами. Программа хоть и лицензионная, но дает пробный период в 40 дней, по истечении которого пользователю снова предлагается приобрести лицензию. Также на официальном сайте доступен защищенный авторскими правами исходный код распаковщика UnRAR, который позволяет его использовать не только в самом WinRAR, но и в других продуктах.

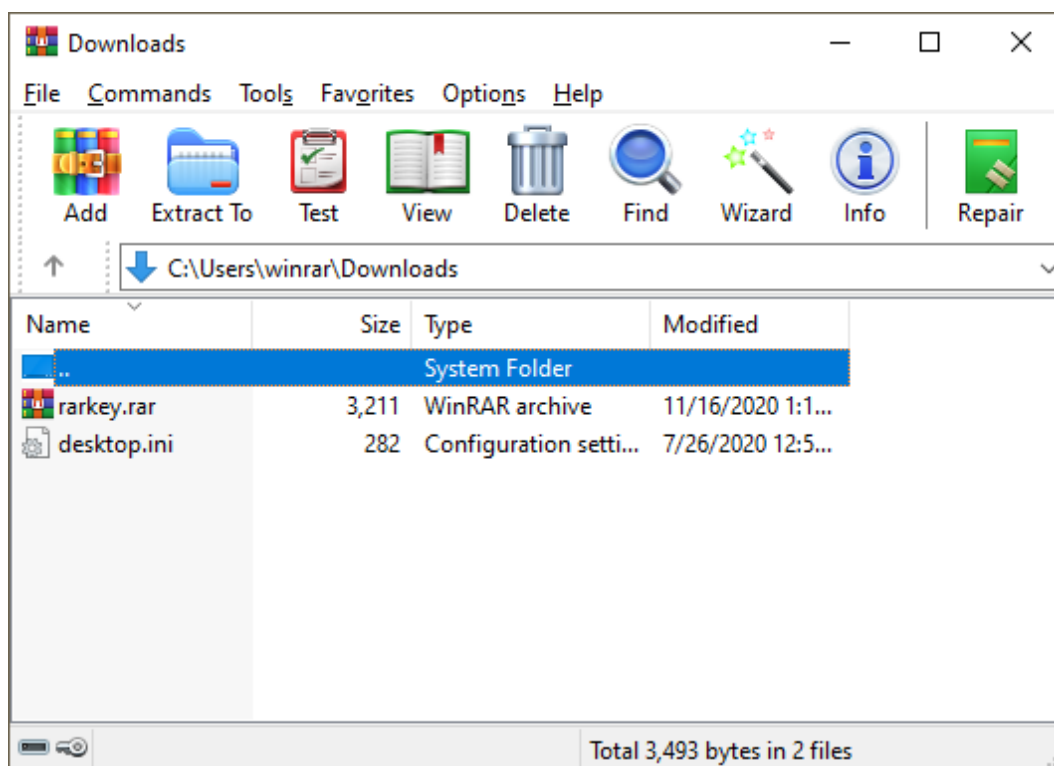


Рисунок 2.1 – Программа WinRAR

Сам алгоритм сжатия основан на двух других алгоритмах: PPM и LZSS. Главная задумка PPM – кодирование с помощью анализа данных и последующем предсказывании значения символов. Но основное сжатие в алгоритме PPM происходит с помощью усовершенствованного алгоритма Хаффмана. LZSS представляет собой словарный алгоритм основанный на

алгоритме Лемпеля и Зива (LZ). Именно этот алгоритм послужил основой большинства современных приложений для архивации файлов.

Помимо архивов RAR программа поддерживает следующие файлы: ARJ, bz2, CAB, GZ, ISO, JAR, LZH, TAR, UAE, XZ, Z, ZIP, ZIPX, 7z. Данный продукт обладает достаточно большим функционалом для работы с архивами и файлами, из-за чего в свою очередь он стал самым распространенным приложением-архиватором в мире.

2.1.2 7-Zip

Программа 7-Zip является бесплатной и имеет открытый исходный код. Автором программы является Игорь Павлов – российский разработчик, выпускник Уфимского государственного авиационного технического университета. Основной формат архивации можно угадать из названия приложения – 7z. Он имеет очень высокую степень сжатия благодаря использованию алгоритма LZMA (все тот же усовершенствованный Лемпеля и Зива – LZ). Но при этом у пользователя нет возможности управлять файлом внутри архива и сжатые файлы данного формата не защищены от повреждений.

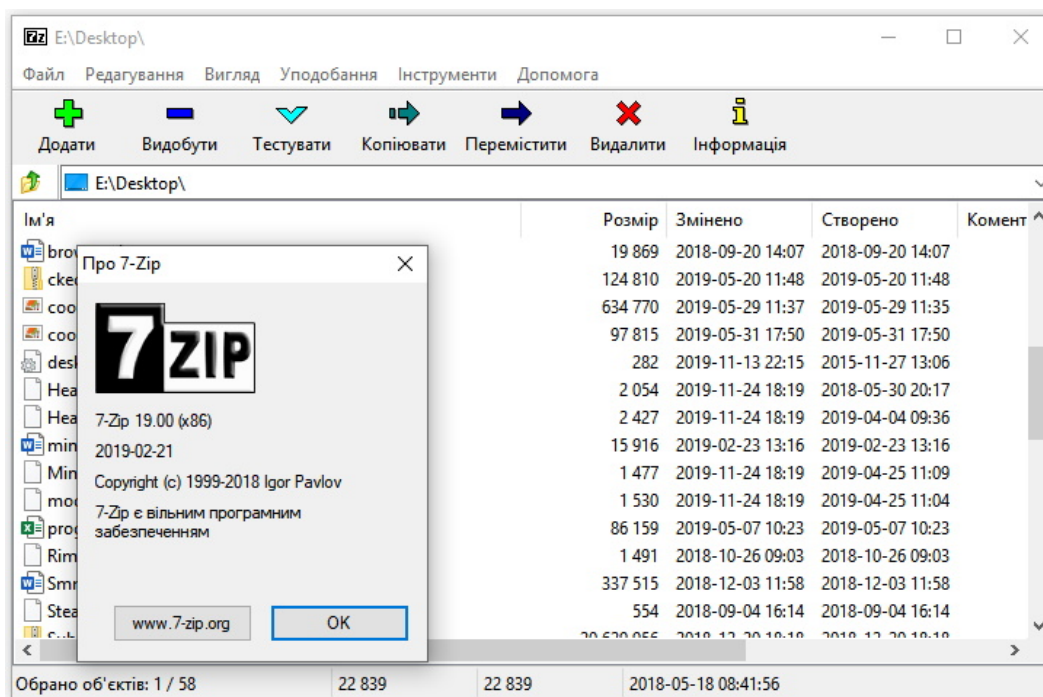


Рисунок 2.2 – Программа 7-Zip

Кроме типа 7z архиватор имеет возможность упаковки и распаковки файлов: BZIP2 (BZ2, TB2, TBZ, TBZ2), GZIP (GZ, TGZ), TAR, ZIP, XZ, WIM. А также только распаковка расширений типов: ARJ, CAB, CHM, CPIO, CramFS, DEB, DMG, FAT, HFS, MBR, ISO, LZH, LZMA, MSI, NSIS, NTFS, RAR, RPM, SquashFS, UDF, VHD, XAR, Z. Приложение 7-Zip из-за

своей открытой лицензии позволяет устанавливать внешние плагины для работы с другими, непредусмотренными расширениями.

2.1.3 WinZip

Одним из первых архиваторов с графическим интерфейсом стал WinZip. Он был создан в начале 90-х и используется по сей день как основная программа-архиватор для Windows, macOS, IOS и Android. WinZip как и WinRAR имеет платную лицензию.

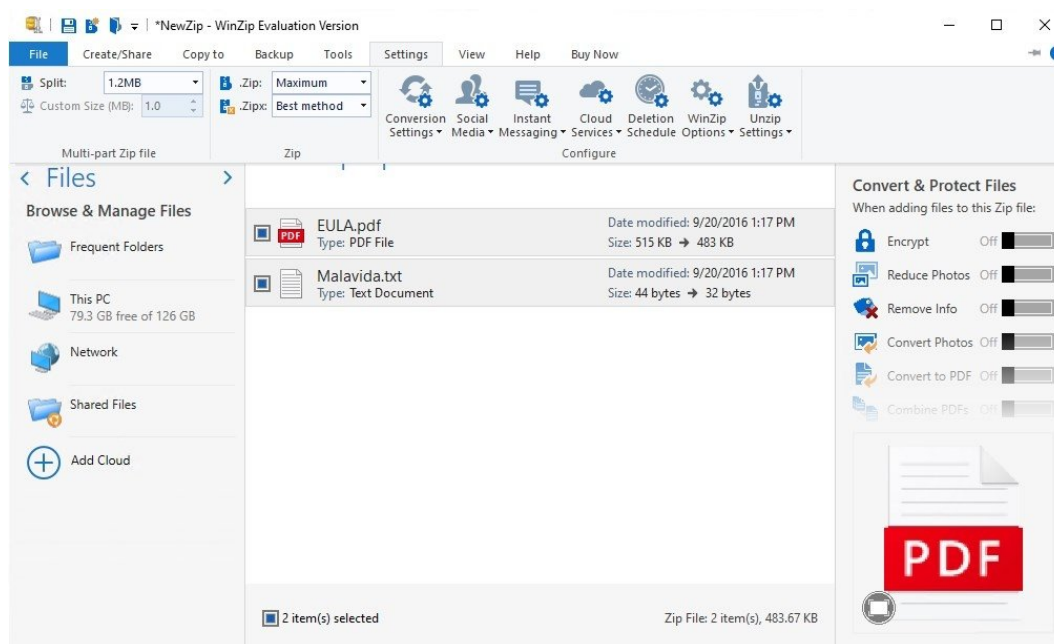


Рисунок 2.3 – Программа WinZip

Данная программа может работать со следующими файлами: ZIP, tar, gzip, Z, Cabinet, RAR, bzip2, LHA, 7z, XZ, VHD и VMDK. На данный момент WinZip поддерживает работу с облачными хранилищами, такими как Google Диск, SkyDrive, DropBox и другие.

Все вышеперечисленные программы обладают схожими функциями и очень широкими возможностями: от управления файлами в операционной системе до интеграций с облачными сервисами. Но даже такие разные программы по сути внутри имеют все ту же реализацию кода Хаффмана и алгоритма Лемпеля и Зива. Поэтому, при сравнении конкурентов по скорости и степени сжатия данных, программы будут примерно равны по своим возможностям. Главные отличия всех вышеперечисленных программ: дополнительные встроенные возможности приложений и их графический интерфейс. Здесь уже создатели приложений не ограничиваются фантазией и вставляют в приложения любой всевозможный функционал.

В этом курсовом проекте главная задача – понять принцип действия предка современных способов сжатия данных – алгоритма Хаффмана,

осуществить на практике и спроектировать его в виде полноценного оконного приложения с графическим интерфейсом.

2.2 Обзор методов и алгоритмов решения поставленной задачи

В курсовом проекте используется алгоритм шифрования и дешифрования, названный в честь ученого Дэвида Хаффмана, разработавшего его в ходе выполнения курсовой работы в Массачусетском технологическом институте в 1952 году. Данный способ архивации относится к группе "жадных" алгоритмов, где принимаются локально оптимальные решения на каждом шаге, допуская конечное решение также оптимальным. Он представляет собой процесс кодирования файла по словарю префиксных значений заданного выражения.

Идея, лежащая в основе кода Хаффмана, достаточно проста. Вместо того чтобы кодировать все символы одинаковым числом бит (например, в ASCII кодировке на каждый символ отводится ровно по 8 бит), символам с большей частотой ставятся в соответствие более короткие коды. Соответственно редким символам будет соответствовать более длинный код. В данном случае под кодом подразумеваются биты, используемые для представления конкретного символа или другой единицы объема информации.

Для однозначного шифрования символов (значение должно быть единственным) коды Хаффмана должны обладать свойством префиксности – каждое кодовое слово не должно быть префиксом другого. Таким образом можно использовать для кодирования меньшее число бит, чем максимально возможное, а значит и эффективность сжатия возрастает. Главное учитывать, что при такой архивации усложняется декодирование сжатой информации.

Стоит отметить, что такой алгоритм Хаффмана в первоначальном виде будет однозначно работать только с файлами, представляемыми в текстовом виде: .txt, .tex, .html, .cpp и другие форматы файлов преобразуемые в текст. Если же попробовать закодировать нашим алгоритмом, например, изображение, результат может быть неопределенным, так как будет видоизменяться структура самого файла.

Алгоритм Хаффмана двухпроходный. На первом проходе строится частотный словарь и генерируются коды. На втором проходе происходит непосредственное кодирование всех символов.

При построении частотного словаря создается контейнер для записи значений (можно использовать последовательные или ассоциативные – в данном случае реализация зависит только от предпочтений разработчика) и производится проход по файлу. Во время чтения файла количество вхождений каждого символа исходного алфавита записывается в

соответствующую ячейку контейнера для построения бинарного дерева поиска (Н-дерево).

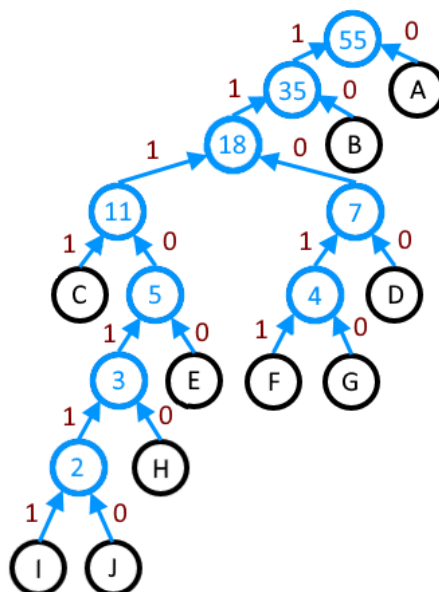


Рисунок 2.4 – Пример построенного дерева Хаффмана

Оптимальный код файла всегда будет представлен полным бинарным деревом (см. рис. 2.4), каждый узел (кроме листьев) которого имеет по два дочерних узла. При отсутствии префиксности и использовании кодов одинаковой длины дерево становится неполным, а значит и кодирование файла становится менее эффективным.

Дерево кодирования символов формируется из созданного ранее частотного словаря по следующему алгоритму:

- 1 Составляется список кодируемых символов (при этом каждый символ является одноэлементным бинарным деревом, вес которого равен весу символа и количеству вхождений этого символа в кодируемое сообщение).

- 2 Выбираются два узла из списка с наименьшими весами.

- 3 Формируется новый узел и присоединяется к нему, в качестве дочерних, два узла выбранных из списка. При этом вес сформированного узла равен сумме весов дочерних узлов.

- 4 Сформированный узел добавляется к списку.

- 5 Если в списке больше одного узла, то повторить пункты 2-5.

Таким образом при движении по дереву от корня к листьям у нас образуется кодировка символа, находящегося в листе дерева. По этому дереву строится контейнер, содержащий кодировки всех символов алфавита. Используя данные из контейнера кодируется вся исходная последовательность.

Чтобы правильно декодировать уже сжатый файл, в его начало записывается таблица частот алфавита и уже по ней расшифровывается закодированная последовательность, записанная после таблицы.

Именно из-за записи частотной таблицы архивация становится менее эффективной. Если же не учитывать длину этой таблицы, то длина сжатого файла всегда будет меньше либо равной первоначальной. Обычно в таких случаях, только при записи закодированных данных, сжатие позволяет сэкономить от 20% до 90% объема информации.

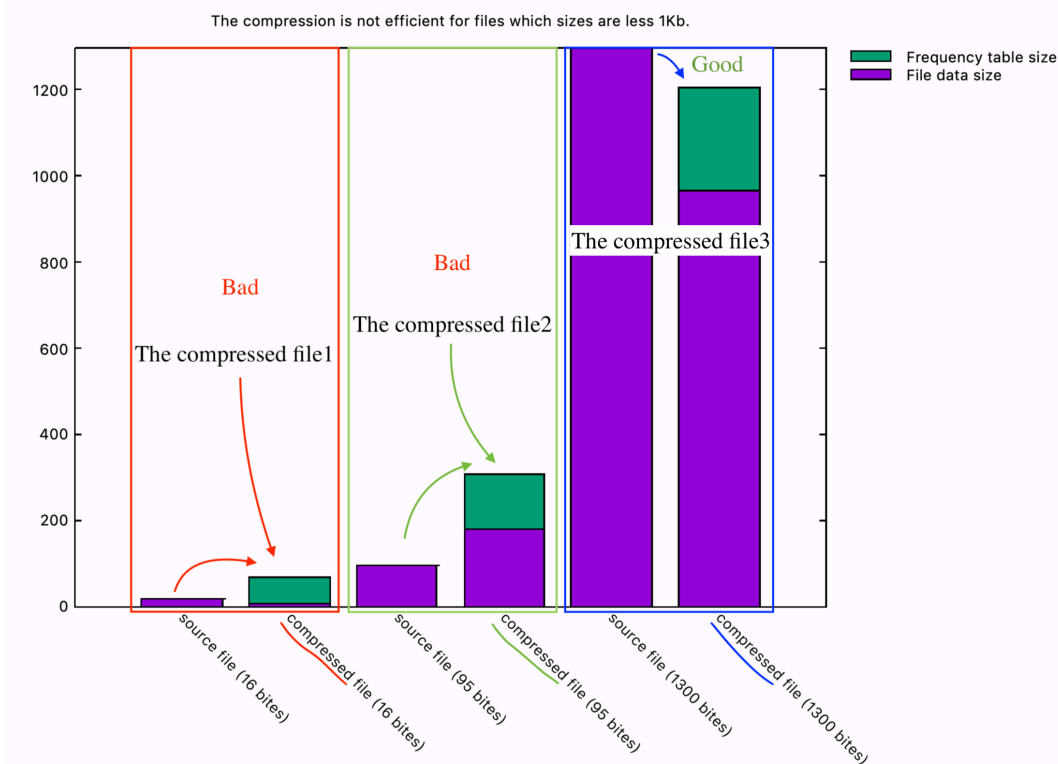


Рисунок 2.5 – Эффективность сжатия небольших файлов

Опытным путем для нашей программы было установлено, что коэффициент сжатия становится меньше единицы (отношение занимаемой памяти архивированного и исходного файлов) при размере исходного файла больше 1 Кб. Если размер файла меньше 1 Кб, то с почти стопроцентной вероятностью размер архива будет больше, чем первоначальный размер файла. Эффективность архивации маленьких файлов наиболее наглядно представлена на рисунке 2.5.

Стоит понимать, что длина таблицы частот также зависит от количества символов в алфавите данных. Например, если вся последовательность будет состоять только из букв "a" и "b", то соответственно и сама таблица будет иметь только две записи, а это значит, что и эффективность сжатия такой последовательности будет при небольшом размере исходных данных более высокой.

Рассмотрим пример: представим, что у нас есть некоторый текст, который состоит только из символов "а", "b", "с", "d" и "е", а частоты их появления равны 15, 7, 6, 6 и 5 соответственно. Используя описанный выше алгоритм построения дерева, создадим сам граф и рассмотрим формирование кодировок символов.

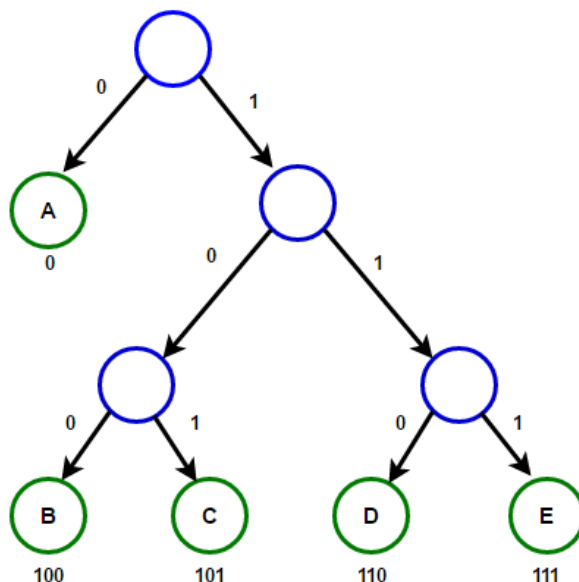


Рисунок 2.6 – Дерево Хаффмана для выбранного алфавита

На изображении 2.6 показаны веса соответствующих узлов и указаны соответствующие этим узлам листья со значениями символов. Таким образом символ "а" кодируется значением 0, "b" – 100, "с" – 101 и так далее. По рисунку наглядно видно как образуется кодировка разных листьев. При обходе дерева в глубину, проходя по каждому узлу, кодировка символа, лежащего в нижнем листе увеличивается на один бит. В итоге, при проходе по дереву, начиная с его корня, заканчивая листьями, мы получаем кодировки всех символов, нужных нам для создания таблицы и последующей записи их в архивный файл. В заключение стоит отметить, что для алфавита произвольной длины дерево должно иметь ровно столько же листьев, сколько и символов в данном алфавите.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

3.1 Структура входных и выходных данных

В качестве входных данных могут использоваться любые файлы приводимые к текстовому типу: .txt, .tex, .html, .cpp, .odt и так далее. Программа работает с файловой системой NTFS и операционной системой Windows 10.

При сжатии создается новый файл, к названию которого после расширения добавляется постфикс ".arch". Чтобы файл не занимал много места в памяти, расширение не записывается в сам архивный файл, а остается в его названии. Таким образом файл "index.html" архивируется в файл "index.html.arch".

При разархивации файла постфикс ".arch" убирается. Если файл с таким же названием уже существует (без постфикса), то перед расширением вставляется подстрока "_1".

При проведении каждой операции выводится уведомление об ошибке или ее успешном завершении.

3.2 Разработка диаграммы классов

Диаграмма классов для курсового проекта приведена в приложении А. Суммарно на диаграмме изображено 8 классов, используемых в работе архиватора.

3.3 Описание классов

3.3.1 Класс узла дерева поиска Node

Класс Node является простейшим узлом дерева Хаффмана. Сначала из этих узлов составляется очередь, потом по этой очереди строится само дерево. Рассмотрим основные поля и методы данного класса.

Ptr left, right и parent. Для предотвращения утечек памяти в данном классе используются умные указатели на узлы листьев (левый и правый потомок) и родителя. Так как для построения дерева нам понадобится изменять указатели или использовать несколько указателей на один объект, то для таких случаев подходит тип `std::shared_ptr<Node>`. Данный тип указателя отслеживает свое время жизни и количество ссылок указывающих на него. При выходе из всех

блоков кода или удалении всех ссылок, для объекта `std::shared_ptr` вызывается деструктор и память, выделенная ранее для объекта класса, освобождается. Для чистоты и упрощения кода тип данных этих трех полей является пользовательским: `typedef std::shared_ptr<Node> ptr`.

Эти три поля публичные, для возможности прохода по дереву. Все остальные поля класса `Node` являются приватными для защиты данных от внешних изменений из других функций и классов.

`Unsigned char ch`. Данное поле хранит в себе байт данных (символ в случае текстовых файлов).

`Int frequency`. В этой переменной хранится количество повторений переменной `ch` в заданном объеме данных.

`Std::string code_string`. Здесь хранится уже окончательная кодировка символа, которая будет записана в сжатый файл.

`Std::string name` не является обязательным атрибутом класса, но его удобно использовать при логировании работы программы чтобы проверить правильность выполнения алгоритма. Имеет то же значение что и `ch`, но в другом типе данных.

`Node()`, `Node(unsigned char uch, int f)` и `Node(const std::string& n, int f)`. Для всех возможных входных данных существует несколько конструкторов: нулевой (поля класса не заполняются), с входным символом и его частотой, с входной строкой (состоящей из одного символа) и количеством ее вхождений.

`Bool operator < (const Node& oth)`. Этот метод предназначен для сравнения частот двух узлов и последующего построения бинарного дерева.

`Std::string get_name()` и `std::ostream& operator < (std::ostream &out, Node node)`. Данная функция и дружественная перегрузка оператора ввода предназначены как и поле `std::string name` для логирования алгоритма. Они позволяют просмотреть какие значения хранит дерево и сверить итоговую таблицу кодирования.

`Std::string code()`, `unsigned char get_byte()` и `int get_frequency()`. Эти методы предназначены для возвращения значений соответствующих атрибутов узла – `std::string code_string`, `ch` и `frequency`.

`Void set_frequency(int f)` и `void code(const std::string& c)`. Данные методы нужны для присваивания значения одноименным переменным (полю `ch` значения присваивается только в конструкторе). Функция изменения атрибута `frequency` обязательна из-за правила построения дерева Хаффмана, суть которого состоит в присвоении частоте родителя суммы частот его листьев.

Из-за использования умных указателей, необходимость в использовании деструктора отсутствует.

3.3.2 Класс шаблонной очереди `Queue_t` и компаратора `LowestPriority`

Класс пользовательской очереди `Queue_t` предназначен для начального хранения данных. В контексте данной курсовой работы этот класс используется при первом проходе по сжимаемому файлу. Данная структура данных создана на основе шаблона `std::priority_queue`. Для шаблона установлен хранимый тип `Node::ptr`, используемый в основе контейнер `std::vector<Node::ptr>` и класс компаратор `LowestPriority`. Все эти параметры позволяют создать очередь из узлов `Node`, в которой все новые значения записываются в приоритетном порядке с наименьшим элементом в верши не очереди.

`Std::priority_queue<Node::ptr, std::vector<Node::ptr>, LowestPriority> q_data` Это главное и единственное поле содержит саму очередь с данными.

`Queue_t(std::vector<int>& frequency)`. Конструктор создает очередь из вектора содержащего частоты символов. Проходя по вектору, в конструкторе создается новый узел и записывается в очередь пока вектор не закончится.

`Void build_tree()` Этот метод на основе уже заполненной очереди строит бинарное дерево по алгоритму, описанному в предыдущем разделе.

`Node::ptr top()` Функция возвращает верхний узел в очереди. Именно он будет вершиной дерева Хаффмана.

3.3.3 Класс кода Хаффмана для файла `Huffman_code`

Класс `Huffman_code` хранит в себе имя файла, данные для записи в другой файл и функции обрабатывающие эти данные. Рассмотрим его поля и методы.

`Std::string filename`. Поле содержит имя файла, выбранного для архивации или разархивирования.

Для части нижеперечисленных методов установлен спецификатор доступа `private` с целью предотвратить использование данных функций вне публичных методов.

`Std::string message`. Содержит в себе последовательность данных. В случае сжатия файла представляет собой конечные закодированные данные. Если архив преобразуется в первоначальный файл, то `message` хранит исходные раскодированные данные.

`Int read_file(std::vector<int>& frequency)`. Данный метод считывает исходный файл и заполняет частотную таблицу в виде вектора. В векторе под индексом, равным ASCII коду символа хранится

количество повторений данного символа в кодируемой последовательности данных.

`Void make_code(Node::ptr& node, std::string str, std::vector<std::string>& codes).` Этот метод рекурсивно формирует кодировку каждого символа для исходного алфавита последовательности из бинарного дерева поиска. По своей сути он напоминает обход графа в глубину. В ходе прохода по дереву кодировка формируется в переменной `str` и при достижении листа записывается в вектор `codes`.

`Int message_to_code(const std::vector<std::string>& codes).` Функция преобразует первоначальный набор данных в закодированную последовательность, которая сохраняется в переменную `message`.

`Int write_file(std::vector<int>& frequency).` Метод записывает таблицу частот и всю закодированную информацию в архивный файл.

`Int read_decoding_file(std::vector<int>& frequency).` Данная функция считывает из файла частотную таблицу и всю закодированную информацию после нее.

`Void make_char(const Node::ptr& root, std::string& text).` Этот метод по дереву Хаффмана преобразует закодированные данные в последовательность данных до сжатия.

`Int write_decoding_file(std::string& text).` Метод создает файл и записывает в него всю информацию, которая была сжата в архиве.

Остальные методы класса `Huffman_code` являются публичными и будут использоваться напрямую в самом оконном приложении.

`Huffman_code(std::string file)` и `~Huffman_code()`. Конструктор и деструктор предназначен для установки поля `filename` и очистки полей `filename` и `message` соответственно. Прямого влияния на алгоритм сжатия они не оказывают.

`Bool is_compressed()`. Метод возвращает `true`, если выбранный файл является уже сжатым, и `false`, если файл не является архивом.

`Bool can_compress()`. Метод возвращает `true`, если выбранный файл можно сжать, и `false`, если файл уже сжат.

`Std::string compress_file()`. Данная функция собственно сжимает файл. Она использует некоторые вышеперечисленные приватные методы, а также методы класса `Queue_t`. В качестве возвращаемого параметра, функция передает сообщение об успешности или ошибки проведения операции сжатия файла.

`Std::string decompress_file()`. В отличие от предыдущего метода, этот разархивирует файл, используя оставшиеся защищенные методы. Функция возвращает сообщение об ошибках или успешном выполнении.

3.3.4 Класс окна уведомлений `MyBox`

Класс `MyBox` предназначен для вывода на экран сообщения об успешности проведения операций архивирования и разархивирования файлов. Он наследуется от встроенного в QT виджета `QMessageBox`.

Для создания графического интерфейса необходимо лишь изменить конструктор. В качестве аргумента `MyBox(QString message)` принимает строку, которая будет выводиться на виджете `QMessageBox`.

3.3.5 Класс виджета вывода директорий `MyTreeView`

Элемент `MyTreeView` выводит на экран файловую систему `QFileSystemModel` определенную в фреймворке QT. Он представляет все файлы и директории в виде иерархического дерева. При таком выводе файловой модели на экран внутренние файлы и дочерние папки выпадают списком из родительской директории.

Данный класс наследуется от уже предопределенного в QT класса `QTreeView`. Рассмотрим поля класса `MyTreeView`.

`QFileSystemModel *dirmodel` представляет сам объект файловой системы.

`QString sPath`. Данная переменная содержит путь к рабочей папке, где программа будет архивировать и разархивировать файлы.

`int nameWidth`. Это поле содержит длину столбца "Name" в виджете `MyTreeView`. По умолчанию, чтобы виджет показывал полностью название большинства папок и директорий, его значение установлено как 240.

Так же как и в предыдущем классе, для функционирования интерфейса необходим только конструктор `MyTreeView(QString path)`. В качестве аргумента в него поступает строка, которая содержит начальный путь к папке, которую будет показывать виджет. По умолчанию этот путь нулевой, а значит `MyTreeView` выведет на экран дисковое пространство компьютера. Также в конструкторе устанавливаются остальные параметры виджета: ширина столбцов, параметры вывода элементов `QFileSystemModel` и другое.

3.3.6 Класс виджета вывода файлов **MyListView**

MyListView очень похож на класс **MyTreeView**. Но все же есть и различия. В отличие от **MyTreeView**, данный класс выводит только файлы, но не в виде дерева, а в виде простейшего списка.

QString sPath тоже содержит в себе начальный путь к папке, которая откроется при открытии оконного приложения.

QFileSystemModel *filemodel представляет объект файловой системы, которая содержит в себе только файлы.

Конструктор **MyListView(QString path)** точно так же как и в предыдущем классе устанавливает в качестве объектной модели **filemodel** и начальный путь к папке. В отличие от **MyTreeView** в данном классе нет определенных столбцов с информацией (выводятся только названия файлов).

3.3.7 Класс главного окна приложения **MainWindow**

Данный класс как и несколько предыдущих наследуется от уже определенного в фреймворке QT класса **QMainWindow**.

Так как с этим классом кроме функции **int main()** ничего не взаимодействует, то соответственно и все методы и поля (кроме конструктора и деструктора) имеют приватный спецификатор доступа. Детально рассмотрим поля и методы данного класса.

Ui::MainWindow *u. Представляет переменную описывающую файл **mainwindow.ui** с помощью которого описывается весь интерфейс и взаимодействие его элементов друг с другом.

QWidget *widget. Виджет предназначен для компоновки других графических элементов: кнопок, виджетов **MyTreeView** и **MyListView**.

QPushButton *compress и **QPushButton *decompress**. Поля кнопок, отвечающих за архивацию и разархивацию выбранных файлов.

MyTreeView *treeview и **MyListView *listview**. Поля для вывода файловой информации (см. описание соответствующих классов).

QHBoxLayout *views, **QHBoxLayout *buttons** и **QVBoxLayout *vlayout**. Эти объекты автоматически выравнивают все виджеты, добавленные в них. Для кнопок и файловых виджетов предназначены два разных горизонтальных объекта выравнивания. Все вместе эти два объекта состоят в едином вертикальном классе выравнивания.

QTimer *timer. Специальный таймер, предопределенный в фреймворке QT. С его помощью регулируется время стостояния нажатых кнопок.

`void working_file(const QModelIndex &index).` Данный метод предназначен для непосредственной работы с выбранным файлом: его можно открыть, заархивировать или разархивировать.

`void on_treeView_clicked(const QModelIndex &index)`

Эта функция вызывается при нажатии элемент на виджет вывода директорий `MyTreeView`. Она меняет директорию виджета вывода файлов `MyListView` на выбранную и выводит все файлы, находящиеся в ней.

`void on_listView_clicked(const QModelIndex &index)`

Здесь обрабатывается двойное нажатие на элемент виджета вывода файлов. Когда на определенный файл два раза кликают мышью, он открывается в соответствующем редакторе, установленном по умолчанию для открытия данного типа файлов.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Разработка схем алгоритмов

4.1.1 Разработка схемы функции `compress_file()`

Функция `compress_file()` сжимает выбранный файл и возвращает сообщение о ошибке или об успешной работе функции.

Алгоритм по шагам:

- 1 Начало.
- 2 Если файл можно сжать (он не является архивом), то переход к пункту 2, иначе – к пункту 9.
- 3 Прочитать файл, из прочитанных данных создать вектор с частотами символов.
- 4 Создать очередь из узлов для будущего бинарного дерева.
- 5 Построить из преоритетной очереди дерево Хаффмана.
- 6 Создать таблицу кодировки символов используя обход графа в глубину.
- 7 Заново пройдясь по файлу и используя таблицу кодировок, преобразовать начальные данные в закодированную последовательность.
- 8 Вывести сжатые данные вместе с таблицей частот в новый архивный файл.
- 9 Конец.

4.1.2 Разработка схемы функции `decompress_file()`

Функция `decompress_file()` разархивирует выбранный файл и так же возвращает строку с ошибкой или уведомлением об успешной работе функции.

Алгоритм по шагам:

- 1 Начало.
- 2 Если файл является архивом, то переход к пункту 2, иначе – к пункту 8.
- 3 Прочитать файл, извлечь таблицу частот символов и сжатые данные.
- 4 Создать очередь из узлов для будущего бинарного дерева.
- 5 Построить из преоритетной очереди дерево Хаффмана.
- 6 Раскодировать сжатые данные в первоначальные.
- 7 Вывести исходный набор данных в новый файл.
- 8 Конец.

4.2 Разработка алгоритмов

Схемы алгоритмов функций `compress_file()` и `decompress_file()` приведена в приложении Б.

По схеме видно что эти два метода состоят из множества других подфункций. В каждой из них может произойти ошибка: от невозможности открытия файла, до выхода за пределы вектора или очереди. Поэтому в таких случаях функция возвращает строку, содержащую информацию об ошибке. Любое значение, которое возвращают эти две функции выводится на экран в специальном отдельном окне `MyBox`.

При разработке алгоритма были использованы такие стандартные библиотеки языка C++ как `fstream`, `vector`, `bitset`, `queue`, `memory`, `algorithm`.

С помощью заголовочного файла `fstream` производится чтение и запись информации в файл. Функции `std::fstream::read` и `std::fstream::write` дают возможность читать и записывать данные из (в) файл побайтово, регулируя количество выводимых байт. Считывая и записывая таким образом символы и таблицу частот, у нас уменьшается вероятность получить ошибку при чтении или неправильно записать таблицу в файл. Чтобы воспользоваться данными функциями чтения и записи нужно в качестве аргументов передавать указатель типа `char*` и количество байт, используемых в одной операции. Для преобразования данных в нужный тип в программе задействован оператор `reinterpret_cast<char*> (&value)`, где `&value` это адрес данных в памяти компьютера.

В большинстве случаев вся длина конечной закодированной последовательности не будет кратна размеру одного байта. Поэтому, при записи данных в файл после таблицы частот записывается длина конечной последовательности в байтах и остаток из бит не образующих один байт.

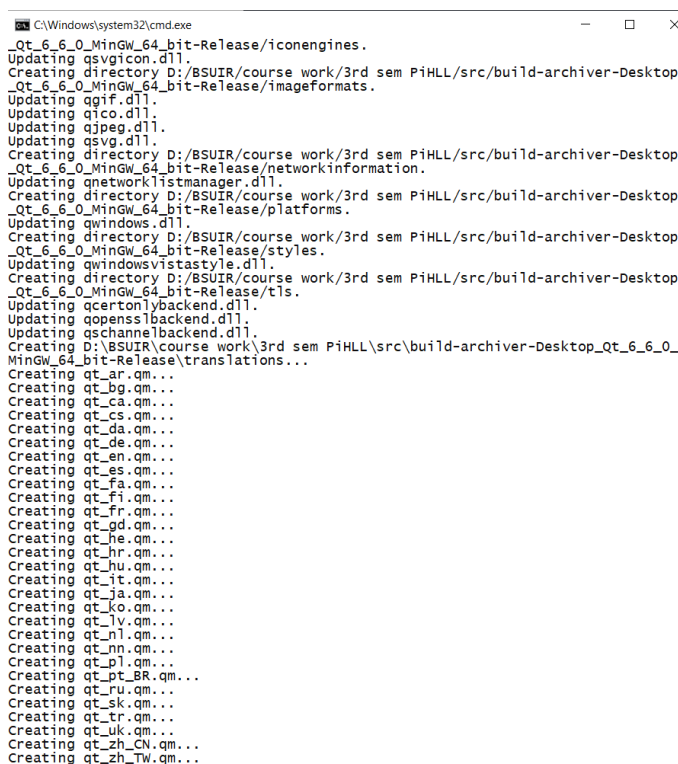
Для записи в файл исходной последовательности используется структура `std::bitset<CHAR_BIT>`. В ее конструктор передается код символа и с его помощью эта кодировка преобразуется в набор бит (каждый символ '1' или '0' превращается в соответствующий бит). Потом структура содержащая 8 бит приводится к типу `unsigned long` и с помощью оператора `static_cast<unsigned char>` преобразуется в беззнаковый символный тип. Далее такая операция проводится в цикле до тех пор пока мы не запишем всю последовательность вместе с битовым остатком.

При чтении сжатых данных производится похожая манипуляция. Главное отличие в том, что при чтении сначала `unsigned char` образует структуру битового набора. Потом с помощью функции `std::bitset::to_string()` структура приводится к типу `std::string`. Далее разархивация происходит по алгоритму описанному ранее.

5 РЕЗУЛЬТАТЫ РАБОТЫ

5.1 Руководство пользователя

Чтобы запустить приложение вне среды разработки QT Creator необходимо его развернуть. Под разверткой подразумевается, что в директорию самого приложения и его ".exe" файла будут перенесены все нужные файлы библиотек и подключенных зависимостей. Когда проект запускается в среде разработки она сама находит и подключает все заголовочные файлы. Когда же исполняемый файл запускается вне QT Creator, он не знает где находятся нужные ему библиотеки. Для того чтобы это исправить, в QT есть специальная утилита для развертки приложений – windeployqt.exe.



```
C:\Windows\system32\cmd.exe
Qt_6_6_0_MinGW_64_bit-Release/iconengines.
Updating qsvgicon.dll.
Creating directory D:\BSUIR\course work\3rd sem PiHLL\src\build-archiver-Desktop
Qt_6_6_0_MinGW_64_bit-Release/imageformats.
Updating qgif.dll.
Updating qico.dll.
Updating qjpeg.dll.
Updating qsvg.dll.
Creating directory D:\BSUIR\course work\3rd sem PiHLL\src\build-archiver-Desktop
Qt_6_6_0_MinGW_64_bit-Release/networkinformation.
Updating qnetworklistmanager.dll.
Creating directory D:\BSUIR\course work\3rd sem PiHLL\src\build-archiver-Desktop
Qt_6_6_0_MinGW_64_bit-Release/platforms.
Updating qwindows.dll.
Creating directory D:\BSUIR\course work\3rd sem PiHLL\src\build-archiver-Desktop
Qt_6_6_0_MinGW_64_bit-Release/styles.
Updating qwindowsvistastyle.dll.
Creating directory D:\BSUIR\course work\3rd sem PiHLL\src\build-archiver-Desktop
Qt_6_6_0_MinGW_64_bit-Release/tls.
Updating qcertonlybackend.dll.
Updating qopensslbackend.dll.
Updating qchannelbackend.dll.
Creating D:\BSUIR\course work\3rd sem PiHLL\src\build-archiver-Desktop_Qt_6_6_0_
MinGW_64_bit-Release\translations...
Creating qt_ar.qm...
Creating qt_bg.qm...
Creating qt_ca.qm...
Creating qt_cs.qm...
Creating qt_da.qm...
Creating qt_de.qm...
Creating qt_en.qm...
Creating qt_es.qm...
Creating qt_fa.qm...
Creating qt_fi.qm...
Creating qt_fr.qm...
Creating qt_gd.qm...
Creating qt_he.qm...
Creating qt_hr.qm...
Creating qt_hu.qm...
Creating qt_it.qm...
Creating qt_ja.qm...
Creating qt_ko.qm...
Creating qt_lv.qm...
Creating qt_nl.qm...
Creating qt_nn.qm...
Creating qt_pl.qm...
Creating qt_pt_BR.qm...
Creating qt_ru.qm...
Creating qt_sk.qm...
Creating qt_tr.qm...
Creating qt_uk.qm...
Creating qt_zh_CN.qm...
Creating qt_zh_TW.qm...
```

Рисунок 5.1 – Результат работы утилиты windeployqt.exe

Чтобы "отдеплоить" приложения с помощью встроенной утилиты необходимо воспользоваться командной строкой Windows (cmd). После открытия командной строки используется команда "cd", которая устанавливает текущую директорию содержащую нашу утилиту. Если директория находится не на диске C, то необходимо перейти в нужный диск. Например для перехода в диск D необходимо прописать команду "D:". Потом необходимо прописать название утилиты и через пробел указать полный путь к ".exe" файлу нашего приложения. Результат работы утилиты можно увидеть на рисунке 5.1.

После того как приложение было развернуто, им можно полноценно пользоваться.

При открытии приложения по умолчанию пользователю предлагается выбрать рабочий диск. Далее пользователь выбирает рабочую папку, где он будет взаимодействовать с файлами. Когда папка выбрана, на левом виджете показывается ее содержимое.

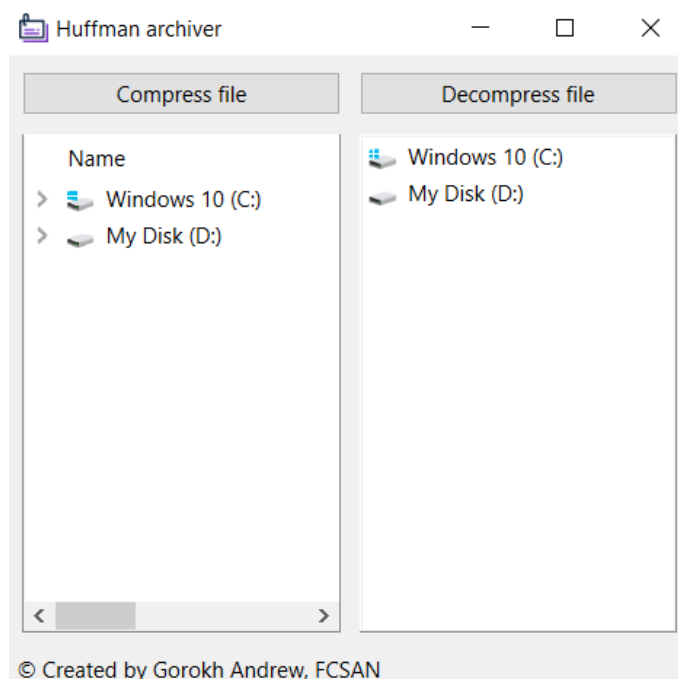


Рисунок 5.2 – Оконное приложение Архиватор

Чтобы сжать файл надо нажать на кнопку "Compress file" и выбрать в виджете сжимаемый файл (см. рис. 5.2). После завершения операции на экране высветится сообщение об успешности ее проведения.

Чтобы разархивировать файл необходимо кликнуть по кнопке "Decompress file" и выбрать нужный файл в правом виджете. Если файл не является архивом, на экран выводится сообщение о невозможности проведения операции разархивирования. При успешной распаковке создается новый файл с исходными данными и выводится сообщение об успешном разархивировании.

Чтобы открыть файл надо нажать дважды на элемент правого виджета и файл откроется в соответствующем редакторе, предустановленном в ОС для открытия данного типа файлов.

ЗАКЛЮЧЕНИЕ

В ходе курсового проектирования были рассмотрены и изучены основные алгоритмы сжатия файлов. Наиболее детально был изучен алгоритм кодирования Хаффмана. Для реализации данного алгоритма использованы встроенные функции и структуры данных языка C++. В ходе работы закрепились и использованы на практике знания полученные при изучении дисциплины "Программирование на языках высокого уровня".

Для визуализации и создания полноценного оконного приложения использованы библиотеки и основные виджеты фреймворка QT. Были изучены основные правила взаимодействия графического интерфейса между собой и реализованы в курсовом проекте.

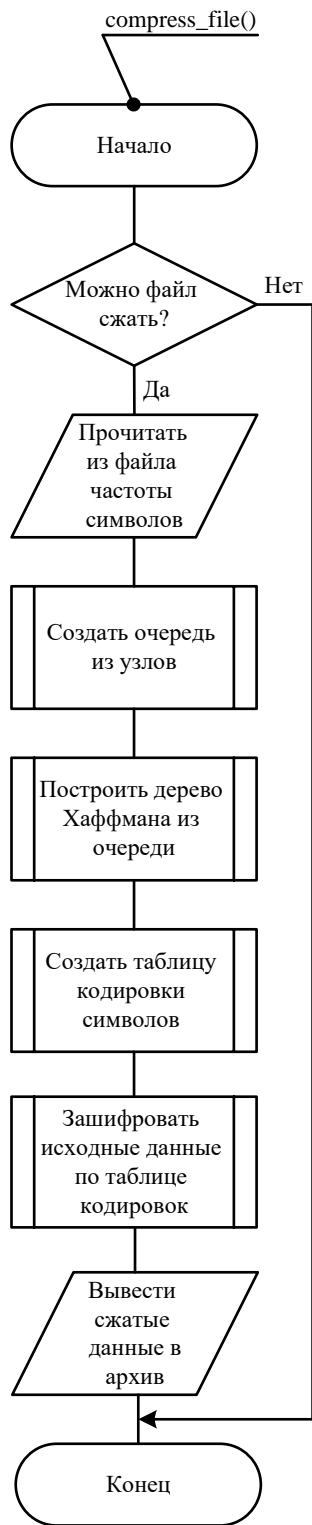
На основе полученных знаний можно углубиться в проектирование приложений похожей тематики. Для улучшения качества следующих проектов необходимо более досконально изучить математический аппарат описывающий сжатие информации и возможности графического интерфейса библиотеки QT.

Данная курсовая работа может послужить хорошей основой для углубленного изучения языка и фреймворков C++. Знания, полученные в процессе написания работы пригодятся в ходе дальнейшего обучения на специальности и помогут реализовать себя в будущей карьере инженера-системотехника.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А
(обязательное)
Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)
Блок-схемы алгоритмов методов



Изм.	Лист	№ докум.	Подп.	Дата
Разраб.	Горох			
Пров.	Богдан			
Т. контр.				
Реценз.				
Н. контр.				
Утв.				

ГУИР.400201.207 ПД1

Архиватор
Блок-схемы функций
compress_file() и
decompress_file()

Лит.	Масса	Масштаб
у		
Лист	1	Листов 1

ЭВМ, зр. 250502

ПРИЛОЖЕНИЕ В

(обязательное)

Полный код программы

Файл structs.h:

```
#include <iostream>
#include <memory>
#include <algorithm>

class Node {
public:
    typedef std::shared_ptr<Node> ptr;
    ptr left{nullptr};
    ptr right{nullptr};
    ptr parent{nullptr};
    Node() = default;
    Node(unsigned char uch, int f) : ch(uch), frequency(f) {};
    Node(const std::string& n, int f) : name(n), frequency(f) {};
    bool operator < (const Node& oth) const;
    std::string get_name() const ;
    unsigned char get_byte();
    int get_frequency() const;
    void set_frequency(int f);
    std::string code();
    void code(const std::string& c);
    friend std::ostream& operator << (std::ostream &out, Node node);
private:
    std::string name{""};
    unsigned char ch{0};
    int frequency{0};
    std::string code_string{""};
};
```

Файл structs.cpp:

```
#include "structs.h"

bool Node::operator < (const Node& oth) const {
    return frequency < oth.frequency;
}

std::ostream& operator << (std::ostream &out, Node node) {
    out << node.get_name() << " - " << node.frequency << std::endl;
    return out;
}

std::string Node::get_name() const {
    if (ch == 0) { return name; }
    else {
        if(ch == 10) { return "\\n"; }
        return std::string(1, static_cast<char>(ch));
    }
}

unsigned char Node::get_byte() {
    return ch;
}

int Node::get_frequency() const {
    return frequency;
}
```

```

}
void Node::set_frequency(int f) {
    frequency = f;
}
std::string Node::code() {
    return code_string;
}
void Node::code(const std::string& c) {
    code_string = c;
}

```

Файл queue_t.h:

```

#include <queue>
#include "structs.h"

class LowestPriority {
public:
    bool operator () (const Node::ptr& left, const Node::ptr& right) const;
};

class Queue_t {
private:
    std::priority_queue<Node::ptr, std::vector<Node::ptr>, LowestPriority> q_data;
public:
    Queue_t(std::vector<int>& frequency);
    void build_tree();
    Node::ptr top();
};

```

Файл queue_t.cpp:

```

#include "queue_t.h"

bool LowestPriority::operator () (const Node::ptr& left, const Node::ptr& right) const {
    return left->get_frequency() > right->get_frequency();
};

Queue_t::Queue_t(std::vector<int>& frequency) {
    unsigned char byte = 0;
    for_each(frequency.begin(), frequency.end(), [this, &byte] (const auto &value) {
        if (value != 0) {
            Node::ptr node = std::make_shared<Node>(byte, value);
            q_data.push(node);
        }
        byte++;
    });
}

void Queue_t::build_tree() {
    while (q_data.size() > 1) {
        Node::ptr x = q_data.top();
        q_data.pop();
        Node::ptr y = q_data.top();
        q_data.pop();
        Node::ptr z = std::make_shared<Node>(0, x->get_frequency() + y->get_frequency());

        z->left = x;
        z->right = y;

        x->parent = z;
    }
}

```

```

        x->parent = z;

        q_data.push(z);
    }
}
Node::ptr Queue_t::top() {
    return q_data.top();
}

```

Файл huffman.h:

```

#include "structs.h"
#include "queue_t.h"
#include <fstream>
#include <vector>
#include <bitset>

class Huffman_code {
private:
    std::string filename;
    std::string message = "";
    int read_file(std::vector<int>& frequency);
    void make_code(Node::ptr& node, std::string str, std::vector<std::string>& codes);
    int message_to_code(const std::vector<std::string>& codes);
    int write_file(std::vector<int>& frequency);
    int read_decoding_file(std::vector<int>& frequency);
    void make_char(const Node::ptr& root, std::string& text);
    int write_decoding_file(std::string& text);
public:
    Huffman_code(std::string file) : filename(file) {}
    ~Huffman_code();
    bool is_compressed();
    bool can_compress();
    std::string compress_file();
    std::string decompress_file();
};

```

Файл huffman.cpp:

```

#include "huffman.h"

int Huffman_code::read_file( std::vector<int>& frequency) {
    std::ifstream ifs(filename, std::ifstream::binary);
    if(!ifs.is_open()) {
        return 1;
    }
    char ch;
    while(ifs.read(&ch, 1)) {
        frequency[static_cast<unsigned char>(ch)]++; //to unsigned char
    }
    ifs.close();
    return 0;
}

void Huffman_code::make_code(Node::ptr& node, std::string str, std::vector<std::string>& codes) {
    if(node->left != nullptr) {
        make_code(node->left, str + '0', codes);
    }
    if(node->right != nullptr) {
        make_code(node->right, str + '1', codes);
    }
}

```



```

    }
    if(node->left == nullptr && node->right == nullptr) {
        node->code(str);
        codes[node->get_byte()] = str;
    }
}

int Huffman_code::message_to_code(const std::vector<std::string>& codes) {
    std::ifstream ifs(filename, std::ifstream::binary);
    if(!ifs.is_open()) {
        return 1;
    }
    char ch;
    while(ifs.read(&ch, 1)) {
        message += codes[static_cast<unsigned char>(ch)];
    }
    return 0;
}

int Huffman_code::write_file(std::vector<int>& frequency) {
    std::string new_filename = filename + ".arch";
    std::ofstream ofs(new_filename, std::ofstream::binary);
    if(!ofs.is_open()) {
        return 1;
    }
    unsigned char count = count_if(frequency.begin(), frequency.end(), [](const int& value) {
        ofs.write(reinterpret_cast<char*> (&count), sizeof(count));
        unsigned char index = 0;
        for_each(frequency.begin(), frequency.end(), [&index, &ofs] (int &value) mutable {
            if(value != 0) {
                ofs.write(reinterpret_cast<char*> (&index), sizeof(index));
                ofs.write(reinterpret_cast<char*> (&value), sizeof(value));
            }
            index++;
        });
        int byte_count = message.size() / CHAR_BIT;
        unsigned char modulo = message.size() % CHAR_BIT;
        ofs.write(reinterpret_cast<char*> (&byte_count), sizeof(byte_count));
        ofs.write(reinterpret_cast<char*> (&modulo), sizeof(modulo));
        int i = 0;
        for(; i < byte_count; i++) {
            std::bitset<CHAR_BIT> b(message.substr(i * CHAR_BIT, CHAR_BIT));
            unsigned char value = static_cast<unsigned char>(b.to_ulong());
            ofs.write(reinterpret_cast<char*> (&value), sizeof(value));
        }
        if(modulo > 0) {
            std::bitset<CHAR_BIT> b(message.substr(i * CHAR_BIT, modulo));
            unsigned char value = static_cast<unsigned char>(b.to_ulong());
            ofs.write(reinterpret_cast<char*> (&value), sizeof(value));
        }
        ofs.close();
        return 0;
    }

int Huffman_code::read_decoding_file(std::vector<int>& frequency) {
    std::ifstream ifs(filename, std::ifstream::binary);
    if(!ifs.is_open()) {
        return 1;
    }
    unsigned char count = 0;
    ifs.read(reinterpret_cast<char*> (&count), sizeof(count));
    int i = 0;

```

```

while(i < count) {
    unsigned char ch;
    ifs.read(reinterpret_cast<char*> (&ch), sizeof(ch));

    int f = 0;
    ifs.read(reinterpret_cast<char*> (&f), sizeof(f));
    frequency[ch] = f;
    i++;
}
int byte_count = 0;
unsigned char modulo = 0;
ifs.read(reinterpret_cast<char*> (&byte_count), sizeof(byte_count));
ifs.read(reinterpret_cast<char*> (&modulo), sizeof(modulo));
i = 0;
for(; i < byte_count; i++) {
    unsigned char byte;
    ifs.read(reinterpret_cast<char*> (&byte), sizeof(byte));
    std::bitset<CHAR_BIT> b(byte);
    message += b.to_string();
}
if(modulo > 0) {
    unsigned char byte;
    ifs.read(reinterpret_cast<char*> (&byte), sizeof(byte));
    std::bitset<CHAR_BIT> b(byte);
    message += b.to_string().substr(CHAR_BIT - modulo, CHAR_BIT);
}
return 0;
}

void Huffman_code::make_char(const Node::ptr& root, std::string&text) {
    Node::ptr node = root;
    for(size_t i = 0; i < message.size(); i++) {
        char ch = message[i];
        if(ch == '0') {
            if (node->left != nullptr) {
                node = node->left;
                if (node->left == nullptr && node->right == nullptr) {
                    text += node->get_byte();
                    node = root;
                }
            }
        }
        else if(ch == '1') {
            if (node->right != nullptr) {
                node = node->right;
                if (node->left == nullptr && node->right == nullptr) {
                    text += node->get_byte();
                    node = root;
                }
            }
        }
    }
}

int Huffman_code::write_decoding_file(std::string& text) {
    std::string old_filename = filename.substr(0, filename.rfind(".arch")) ;
    std::string new_filename;
    std::ifstream exist(old_filename);
    if(exist.is_open()) {
        new_filename = old_filename.substr(0, old_filename.rfind('.'))
            + "_1" + old_filename.substr(old_filename.rfind('.'), old_filename.size

```

```

        exist.close();
    }
    else
        new_filename = old_filename;
    std::ofstream ofs(new_filename, std::ofstream::binary);
    if(!ofs.is_open()) {
        return 1;
    }
    ofs << text;
    ofs.close();
    return 0;
}

Huffman_code::~Huffman_code() {
    message.clear();
    filename.clear();
}

bool Huffman_code::is_compressed() {
    if(filename.rfind(".arch") == std::string::npos)
        return false;
    return true;
}

bool Huffman_code::can_compress() {
    if(filename.rfind(".arch") == std::string::npos && filename.rfind(".png") == std::str
        return true;
    return false;
}

std::string Huffman_code::compress_file() {
    if(!can_compress())
        return std::string("Can't compress this file");
    std::vector<int> frequency(0x100, 0);
    if(read_file(frequency) == 1)
        return std::string("Error file reading");
    Queue_t queue(frequency);
    queue.build_tree();
    std::vector<std::string> codes(0x100, "");
    Node::ptr root = queue.top();
    make_code(root, "", codes);
    if(message_to_code(codes) == 1)
        return std::string("Error making code");
    if(write_file(frequency) == 1)
        return std::string("Error writing data to file");
    return std::string("Compressing was successfull");
}

std::string Huffman_code::decompress_file() {
    if(!is_compressed())
        return std::string("Cannot decompress this file");
    std::vector<int> frequency(0x100, 0);
    if(read_decoding_file(frequency) == 1)
        return std::string("Error file reading");
    Queue_t queue2(frequency);
    queue2.build_tree();
    Node::ptr root = queue2.top();
    std::string text = "";
    make_char(root, text);
    if(write_decoding_file(text) == 1)
        return std::string("Error writing data to file");
    return std::string("Decompressing was succesfull");
}

```

Файл box.h:

```
#include <QtGui>
#include <QtCore>
#include "QMessageBox"

class MyBox : public QMessageBox {
    Q_OBJECT

public:
    MyBox(QString message);
};
```

Файл box.cpp:

```
#include "box.h"

MyBox::MyBox(QString message) {
    setIcon(QMessageBox::Information);
    setText(message);
    setStyleSheet("QLabel { text-align: center; }");
    addButton(QMessageBox::Ok);
}
```

Файл listview.h:

```
#include <QtGui>
#include <QtCore>
#include <QListView>
#include <QStandardItemModel>

class MyListView : public QListView {
    Q_OBJECT
public:
    QFileSystemModel *filemodel;
    MyListView(QString path);
private:
    QString sPath;
};
```

Файл listview.cpp:

```
#include "listview.h"

MyListView::MyListView(QString path) {
    filemodel = new QFileSystemModel(this);
    filemodel->setFilter(QDir::NoDotAndDotDot | QDir::Files);
    filemodel->setRootPath(path);
    setModel(filemodel);
}
```

Файл treeview.h:

```
#include <QtGui>
#include <QtCore>
#include <QTreeView>
```

```

class MyTreeView : public QTreeView {
    Q_OBJECT
public:
    QFileSystemModel *dirmodel;
    MyTreeView(QString path);
private:
    QString sPath;
    int nameWidth = 240;
};

```

Файл treeview.cpp:

```

#include "treeview.h"

```

```

MyTreeView::MyTreeView(QString path) {
    dirmodel = new QFileSystemModel(this);
    dirmodel->setFilter(QDir::NoDotAndDotDot | QDir::AllDirs | QDir::Files);
    dirmodel->setRootPath(path);
    setModel(dirmodel);
    setColumnWidth(0, nameWidth);
}

```

Файл mainwindow.h:

```

#include <QMainWindow>
#include <QtGui>
#include <QtCore>
#include <QTreeView>
#include <QBoxLayout>
#include <QPushButton>
#include <QAbstractButton>
#include <QUrl>
#include <QDesktopServices>
#include <QTimer>
#include <QIcon>
#include "box.h"
#include "listview.h"
#include "treeview.h"
#include "../algorythm/huffman.h"

```

```

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

```

```

class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private:
    Ui::MainWindow *ui;
    QWidget *widget;
    QPushButton *compress;
    QPushButton *decompress;
    MyTreeView *treeview;
    MyListView *listview;
    QHBoxLayout *views;
}

```

```

    QHBoxLayout *buttons;
    QVBoxLayout *vlayout;
    QTimer *timer;
    void working_file(const QModelIndex &index);
    void on_treeView_clicked(const QModelIndex &index);
    void on_listView_clicked(const QModelIndex &index);
};

```

Файл mainwindow.cpp:

```

#include "mainwindow.h"
#include "../ui_mainwindow.h"

```

```

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowIcon(QIcon("D:/BSUIR/course work/3rd sem PiHLL/src/archiver/icons/icon.ico"));
    setWindowTitle("Huffman archiver");
    widget = new QWidget;
    setCentralWidget(widget);
    QString sPath = "";
    treeview = new MyTreeView(sPath);
    listview = new MyListView(sPath);
    compress = new QPushButton("Compress file", this);
    compress->setCheckable(true);
    decompress = new QPushButton("Decompress file", this);
    decompress->setCheckable(true);
    views = new QHBoxLayout;
    views->setSpacing(10);
    views->addWidget(treeview);
    views->addWidget(listview);
    buttons = new QHBoxLayout;
    buttons->setSpacing(10);
    buttons->addWidget(compress);
    buttons->addWidget(decompress);
    statusBar()->showMessage("© Created by Gorokh Andrew, FCSAN");
    vlayout = new QVBoxLayout;
    vlayout->setSpacing(10);
    vlayout->addLayout(buttons);
    vlayout->addLayout(views);
    widget->setLayout(vlayout);
    timer = new QTimer(this);
    connect(treeview, &QTreeView::clicked, this, &MainWindow::on_treeView_clicked);
    connect(listview, &QListView::doubleClicked, this, &MainWindow::on_listView_clicked);
    connect(listview, &QListView::clicked, this, &MainWindow::working_file);
    connect(compress, &QPushButton::clicked, this, [this] {
        compress->setChecked(true);
        connect(timer, QTimer::timeout, this,
            [this] { compress->setChecked(false); timer->stop(); } );
        timer->start(3000); } );
    connect(decompress, &QPushButton::clicked, this, [this] {
        decompress->setChecked(true);
        connect(timer, QTimer::timeout, this,
            [this] { decompress->setChecked(false); timer->stop(); } );
        timer->start(3000); } );
}
MainWindow::~MainWindow()

```

```

{
    delete treeview;
    delete listview;
    delete compress;
    delete decompress;
    delete ui;
}
void MainWindow::working_file(const QModelIndex &index) {
    if(compress->isChecked() && decompress->isChecked()) {
        compress->setChecked(false);
        decompress->setChecked(false);
    }
    else {
        QString sPath = listview->filemodel->fileInfo(index).absoluteFilePath();
        std::string path = sPath.toStdString();
        if(compress->isChecked()) {
            Huffman_code file(path);
            QString res = QString::fromStdString(file.compress_file());
            compress->setChecked(false);
            MyBox *compressBox = new MyBox(res);
            compressBox->exec();
            delete compressBox;
        }
        if(decompress->isChecked()) {
            Huffman_code file(path);
            QString res = QString::fromStdString(file.decompress_file());
            decompress->setChecked(false);
            MyBox *decompressBox = new MyBox(res);
            decompressBox->exec();
            delete decompressBox;
        }
    }
}
void MainWindow::on_treeView_clicked(const QModelIndex &index) {
    QString sPath = treeview->dirmodel->fileInfo(index).absoluteFilePath();
    listview->setRootIndex(listview->filemodel->setRootPath(sPath));
}
void MainWindow::on_listView_clicked(const QModelIndex &index)
{
    QString sPath = listview->filemodel->fileInfo(index).absoluteFilePath();
    QDesktopServices::openUrl(QUrl::fromLocalFile(sPath));
}

```

Файл main.cpp:

```

#include "ui/mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```

ПРИЛОЖЕНИЕ Г

(обязательное)

Ведомость документов

includepdf[pages=-, fitpaper]annexes/statement.pdf