МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой № 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| д-р физ.-мат. наук, проф. |  |  |  | В.Г.Фарафонов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

БАКАЛАВРСКАЯ РАБОТА

|  |  |
| --- | --- |
| на тему | Разработка файлового менеджера |
|  | |
|  | |

|  |  |
| --- | --- |
| выполнена | Любавским Даниилом Алексеевичем |
| фамилия, имя, отчество студента в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.03 |  | Прикладная информатика |
|  | код |  | наименование направления |
|  | | | |
| наименование направления | | | |
| направленности | 05 |  | Прикладная информатика |
|  | код |  | наименование направленности |
| в инновационной деятельности | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы № | М122 |  |  |  | Д. А. Любавский |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. экон. наук, доц. |  |  |  | С. В. Удахина |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой №\_\_\_

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

|  |  |  |  |
| --- | --- | --- | --- |
| студенту группы | М122 |  | Любавскому Даниилу Алексеевичу |
|  | номер |  | фамилия, имя, отчество |

|  |  |
| --- | --- |
| на тему | Разработка менеджера файлов |
|  | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| утвержденную приказом ГУАП от |  | № |  |

|  |  |
| --- | --- |
| Цель работы: | Разработать менеджер файлов с функцией поиска дубликатов файлов |
|  | |
|  | |

|  |  |
| --- | --- |
| Задачи, подлежащие решению: |  |
|  | |
|  | |

|  |  |
| --- | --- |
| Содержание работы (основные разделы): | Анализ и характеристика объекта и предмета |
| исследования, Разработка приложения, Тестирование программы | |
|  | |
|  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Срок сдачи работы « |  | » |  | 20 | 25 |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доц., канд. экон. наук, доц. |  |  |  | С. В. Удахина |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Задание принял(а) к исполнению

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| студент группы № | М122 |  |  |  | Д. А. Любавский |
|  |  |  | подпись, дата |  | инициалы, фамилия |

**СОДЕРЖАНИЕ**

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 3](#_Toc199598953)

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 5](#_Toc199598954)

[ВВЕДЕНИЕ 6](#_Toc199598955)

[1 Анализ и характеристика объекта и предмета исследования 10](#_Toc199598956)

[1.1 Анализ и характеристика файловых менеджеров 10](#_Toc199598957)

[1.2 Описание предмета исследования 16](#_Toc199598958)

[1.3 Структуры файловых систем компьютера 24](#_Toc199598959)

[2 Разработка приложения 33](#_Toc199598960)

[2.1 Постановка задачи разработки 33](#_Toc199598961)

[2.2 Выбор платформы для реализации 34](#_Toc199598962)

[2.3 Выбор языка разработки 36](#_Toc199598963)

[2.4 Выбор среды разработки 38](#_Toc199598964)

[2.5 Описание функциональной части программы 40](#_Toc199598965)

[2.6 Разработка программы 41](#_Toc199598966)

[2.7 Разработка интерфейса приложения 44](#_Toc199598967)

[3. Тестирование программы 50](#_Toc199598968)

[3.1 Тестирование взаимодействия с файлами и директориями 50](#_Toc199598969)

[3.2. Тестирование функции поиска дубликатов файлов 55](#_Toc199598970)

[ЗАКЛЮЧЕНИЕ 59](#_Toc199598971)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 60](#_Toc199598972)

[ПРИЛОЖЕНИЕ А. Код 62](#_Toc199598973)

# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В данной работе применяются следующие термины с соответствующими определениями:

Big-endian – порядок записи байтов в памяти от старшего к младшему;

Birthday-атака (Атака «дней рождения») – используемое в криптоанализе название для метода взлома шифров или поиска коллизий хеш-функций на основе парадокса дней рождения;

Аудит (в файловой системе) – ряд процессов для отслеживания изменений в файловой системе. Помогает предотвратить снижение уровня безопасности, программные сбои и неэффективное использование файловых ресурсов;

Графический интерфейс – интерфейс пользователя, представленный в виде графических элементов экрана;

Журналируемость – способность ведения журнала, в котором хранится список изменений. Данный журнал помогает сохранить целостность файловой системы при сбоях;

Интерфейс пользователя – совокупность средств, при помощи которых пользователь взаимодействует с различными программами и устройствами;

Квотирование – механизм ограничения доступного пользователям пространства в файловой системе;

Кластер – логическая единица хранения данных в таблице размещения файлов, объединяющая группу секторов;

Ортодоксальный файловый менеджер (он же файловый менеджер, управляемый командами) – файловый менеджер, интерфейс которого основан на текстовом меню. Обычно имеет три окна – две панели для отображения файлов и одно окно командной строки.

Приложение – программа, предназначенная для решения практической задачи и подразумевающая непосредственное взаимодействие с пользователем;

Файловая система – порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании;

Файловый менеджер – компьютерная программа, предоставляющая интерфейс пользователя для работы с файловой системой и файлами;

Хеш-сумма – последовательность символов фиксированной длины, полученная путём преобразования произвольных исходных данных при помощи специального математического алгоритма.

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В данной работе применяются следующие сокращения и обозначения:

ACL – Access Control List;

APFS – Apple File System;

ext4 – Fourth Extended File System;

FAT – File Allocation Table;

FAR – File and Archive manager;

GUI – Graphical User Interface;

IDEF – Integrated Defiition;

NTFS – New Technology File System;

MD5 – Message Digest 5;

SHA-1 – Secure Hash Algorithm 1;

SHA-256 – Secure Hash Algorithm 256;

TC – Total Commander;

АНБ – Агентство Национальной Безопасности;

ГОСТ – Межгосударственный стандарт;

ИТ – Информационные Технологии;

ОС – Операционная система;

ПК – Персональный компьютер;

ПО – Программное обеспечение;

ЯП – Язык программирования.

PDF – Portable Document Format;

# ВВЕДЕНИЕ

Файловый менеджер – это специализированное программное обеспечение, предназначенное для организации и управления файловой системой компьютера. Он предоставляет пользователю удобный интерфейс для выполнения различных операций с файлами и папками, таких как создание, копирование, перемещение, удаление, переименование, поиск и просмотр содержимого.

Файловые менеджеры значительно упрощают процесс навигации по файловой системе, позволяя быстро находить нужные файлы и папки, с помощью файлового менеджера можно создавать структурированную систему хранения информации, что облегчает поиск и упорядочивание файлов. Также многие файловые менеджеры обладают расширенным функционалом, включая поддержку архивов, синхронизацию с облачными хранилищами, встроенные текстовые редакторы и инструменты для просмотра изображений.

Файловый менеджер обладает некоторыми основными характеристиками, среди которых графический интерфейс пользователя (GUI) – наиболее распространенный тип интерфейса, обеспечивающий интуитивное взаимодействие с пользователем, операции для манипулирования файлами, которые уже были отмечены ранее. Желательно, чтобы файловый менеджер имел поддержку различных файловых систем, например NTFS, FAT32, FAT64 (используются операционной системой Windows), EXT2, EXT3, EXT4 (используются операционной системой Linux). Также в современных реалиях будет важным наличие инструментов для кастомизации внешнего вида файлового менеджера для повышения дружелюбности интерфейса к пользователю.

Большинство современных файловых менеджеров, несмотря на широкий спектр предоставляемых функций, не оснащены встроенным инструментом для обнаружения дубликатов файлов. Это связано с тем, что традиционно основным назначением файловых менеджеров считалось обеспечение удобного доступа к файловой системе и выполнения базовых операций с файлами и папками. Однако, отсутствие функции отслеживания дубликатов является существенным недостатком, который может привести к следующим проблемам:

* Затраты на хранение. Дубликаты файлов занимают дополнительное место на жестком диске, что может привести к его переполнению и снижению производительности системы;
* Затруднения в поиске файлов. Наличие множества копий одного и того же файла затрудняет поиск необходимой информации;
* Потеря данных. В некоторых случаях дубликаты могут привести к потере данных, если пользователь случайно удалит не тот файл.

Наличие функции отслеживания дубликатов файлов в файловом менеджере стало бы неотъемлемым преимуществом. Данная функция позволила бы сэкономить дисковое пространство компьютера благодаря автоматическому обнаружению и удалению дубликатов; позволила бы улучшить организацию данных, так как отсутствие дубликатов файлов способствует более упорядоченному хранению файлов и упрощает их поиск; повысило бы удобство использования, так как пользователю не придётся скачивать дополнительные программы. Но самым главным преимуществом данной функции является повышение производительности диска, так как меньшее количество файлов положительно сказывается на скорости работы системы, а также на её долговечность.

Таким образом, включение функции отслеживания дубликатов файлов в функционал файловых менеджеров является логичным и востребованным развитием. Это позволит пользователям более эффективно управлять своими данными и повысить производительность своих компьютеров.

Объектом исследования данной курсовой работы являются файловые менеджеры, предметом – процесс поиска дубликатов файлов в файловой системе компьютера. Целью данной работы является провести комплексный анализ файловых менеджеров, определить их роль в современной вычислительной среде, создать прототип файлового менеджера, который будет иметь в себе все перечисленные выше функции. Актуальность исследования обусловлена постоянным ростом объема информации, хранящейся на компьютерах, и необходимостью эффективных инструментов для ее организации и управления.

Практическая значимость работы будет заключатся в создании прототипа файлового менеджера, главным достоинством которого будет наличие функции отслеживания дубликатов файлов, преимущества которой были расписаны выше.

Для решения поставленной цели необходимо решить следующие задачи:

1. Характеристика и описание объекта и предмета исследования;
2. Анализ рынка информационных технологий, связанных с файловыми менеджерами;
3. Изучение структур популярных файловых систем компьютера,
4. Постановка задачи разработки;
5. Обоснование выбора ПО для разработки;
6. Разработка приложения;
7. Тестирование приложения;
8. Расчёт экономической эффективности.

Для решения поставленных задач используются такие методы исследования, как:

1. Изучение;
2. Анализ;
3. Индукция;
4. Синтез;
5. Описание;
6. Сравнение;
7. Практическое моделирование.

Внутреннее устройство файловой системы различных операционных систем разбирал вместе со своими коллегами профессор МГУ И. В. Машечкин в своей статье «СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ - файловые системы ОС UNIX и WINDOWS NT». О работе файловой системы можно прочитать в книге Эндрю Таненбаума «Операционные системы: современный подход», в которой подробно объяснено устройство операционной системы в целом, но также затрагивается и файловая система, без которой ОС существовать не смогла бы. Брайан Керниган и Роб Пайк в своей книге «UNIX: программирование» подробно осветили работу с UNIX-подобными операционными системами. Данная работа охватывает широкий список тем, включая работу с файловой системой, потоками, процессами, межпроцессорное взаимодействие, а также использование различных инструментов командной строки.

# Глава 1. Исследование предметной области

## 1.1 Описание объекта исследования

### 1.1.1 SAP2000

**SAP2000 (Structural Analysis Program 2000)** – программный комплекс для расчета и проектирования строительных конструкций, разработанный американской компанией **CSI (Computers and Structures, Inc.)**. Первая версия вышла в **1995 году**, и с тех пор программа претерпела множество изменений, став одним из самых популярных инструментов в области инженерной механики и строительного проектирования.

Основное предназначение SAP2000 – **статический и динамический анализ строительных конструкций**: от простых балок и рам до сложных пространственных сооружений, мостов, высотных зданий и промышленных объектов. Программа работает в **графическом интерфейсе**, где пользователь может создавать модель конструкции, задавать нагрузки, граничные условия и материалы, после чего выполнять анализ и получать результаты в визуальной форме.

SAP2000 использует **метод конечных элементов (МКЭ)**, поддерживает широкий спектр расчетов: линейные и нелинейные, модальные, спектральные, сейсмические, тепловые и другие. Предусмотрена возможность **автоматической генерации расчетных схем**, а также **интеграции с AutoCAD, Revit и Excel**, что делает работу инженера более гибкой и производительной.

Отдельное внимание разработчики уделили **визуализации и пользовательскому опыту**: присутствует трехмерное моделирование, анимация результатов, редактируемые графики, отчеты и таблицы. Интерфейс доступен на нескольких языках, включая русский, а документация охватывает как базовые примеры, так и сложные инженерные задачи.

SAP2000 может применяться как в учебных целях (имеется студенческая лицензия), так и в **проектных институтах и международных инженерных компаниях**, где требуется высокая точность и соответствие мировым стандартам (AISC, Eurocode, СНиП и др.).

Несмотря на свою мощность и универсальность, SAP2000 сохраняет **интуитивный интерфейс**, позволяющий быстро начать работу, а также **возможность автоматизации задач через API** (на языках VBScript, Python и C#).

Таким образом, SAP2000 остаётся востребованным и активно развивающимся инструментом, незаменимым в арсенале инженера-конструктора, проектировщика или исследователя в области строительной механики.

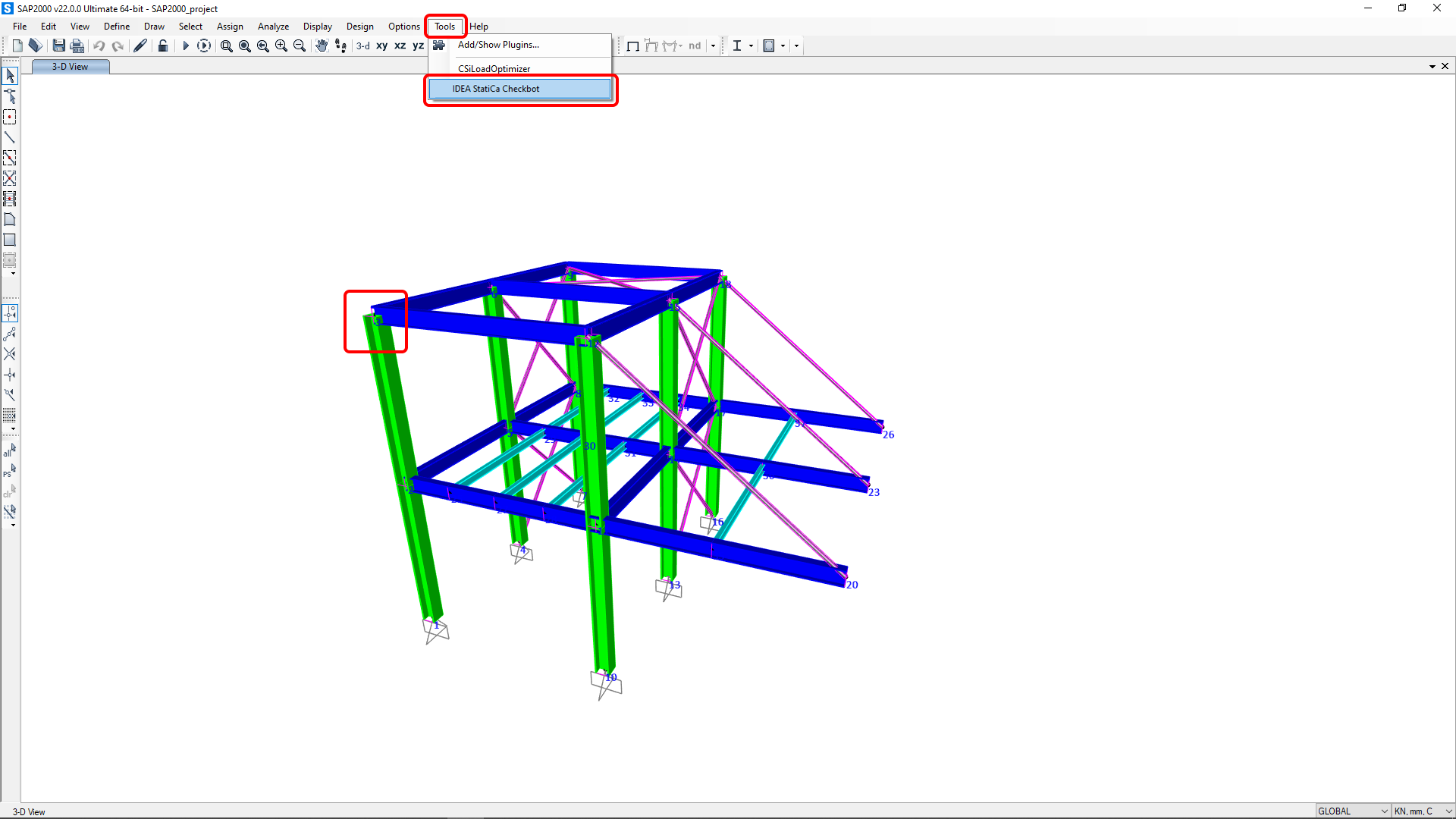


Рисунок 1 - Внешний вид Far Manager

### 1.1.2 ****LIRA-SAPR****

**LIRA-SAPR (ЛИРА-САПР)** – профессиональный программный комплекс для **расчета и проектирования строительных конструкций**, разработанный российской компанией **ЛИРА Софт**. Первая версия появилась в **1990-х годах**, и с тех пор программа стала одним из ведущих инструментов в области строительной механики, активно применяемым на территории России, СНГ и за их пределами.

Программа предназначена для выполнения **инженерных расчетов по методу конечных элементов (МКЭ)** и охватывает весь спектр задач, возникающих при проектировании зданий и сооружений – от простых схем до сложных многоуровневых конструкций, включая сейсмостойкие объекты, мосты, промышленные установки и высотные здания.

LIRA-SAPR поддерживает **линейный и нелинейный анализ**, статические и динамические расчёты, расчёт устойчивости, теплопередачи, прогрессирующего обрушения, а также **автоматическую проверку и подбор армирования** по нормативам (СП, СНиП, Eurocode и др.). Одним из ключевых преимуществ комплекса является **встроенный механизм генерации расчетной схемы на основе BIM-модели**, а также **интеграция с Revit, AutoCAD, IFC и другими CAD-системами**.

Интерфейс LIRA-SAPR сочетает **мощные аналитические инструменты и наглядную 3D-визуализацию**, позволяя пользователю быстро ориентироваться в структуре модели, редактировать параметры и отслеживать результаты в виде таблиц, графиков и цветных карт усилий и перемещений. Специальный модуль **"АРМ конструктора"** облегчает работу с результатами и оформлением проектной документации.

Программа реализует **модульный подход**, где пользователь может подключать только те компоненты, которые необходимы для конкретной задачи: от расчетов металлических и железобетонных конструкций до анализа грунтового основания и совместной работы системы "здание-основание".

**Преимуществами LIRA-SAPR** являются:

* Полная локализация на русском языке и поддержка нормативов СНиП/СП;
* Подробная документация и техническая поддержка;
* Возможность работы с крупными моделями и распределённые расчёты;
* Развитые инструменты автоматизации (скрипты, API, плагин для Grasshopper);
* Применяется в учебных заведениях и аккредитован в экспертизе проектной документации.

Несмотря на обилие функций, начальный порог освоения программы может быть выше, чем у некоторых аналогов, из-за высокой степени детализации настроек. Тем не менее, LIRA-SAPR остаётся одним из самых мощных и гибких решений для **профессионального проектирования и анализа строительных конструкций в русскоязычной инженерной среде**.

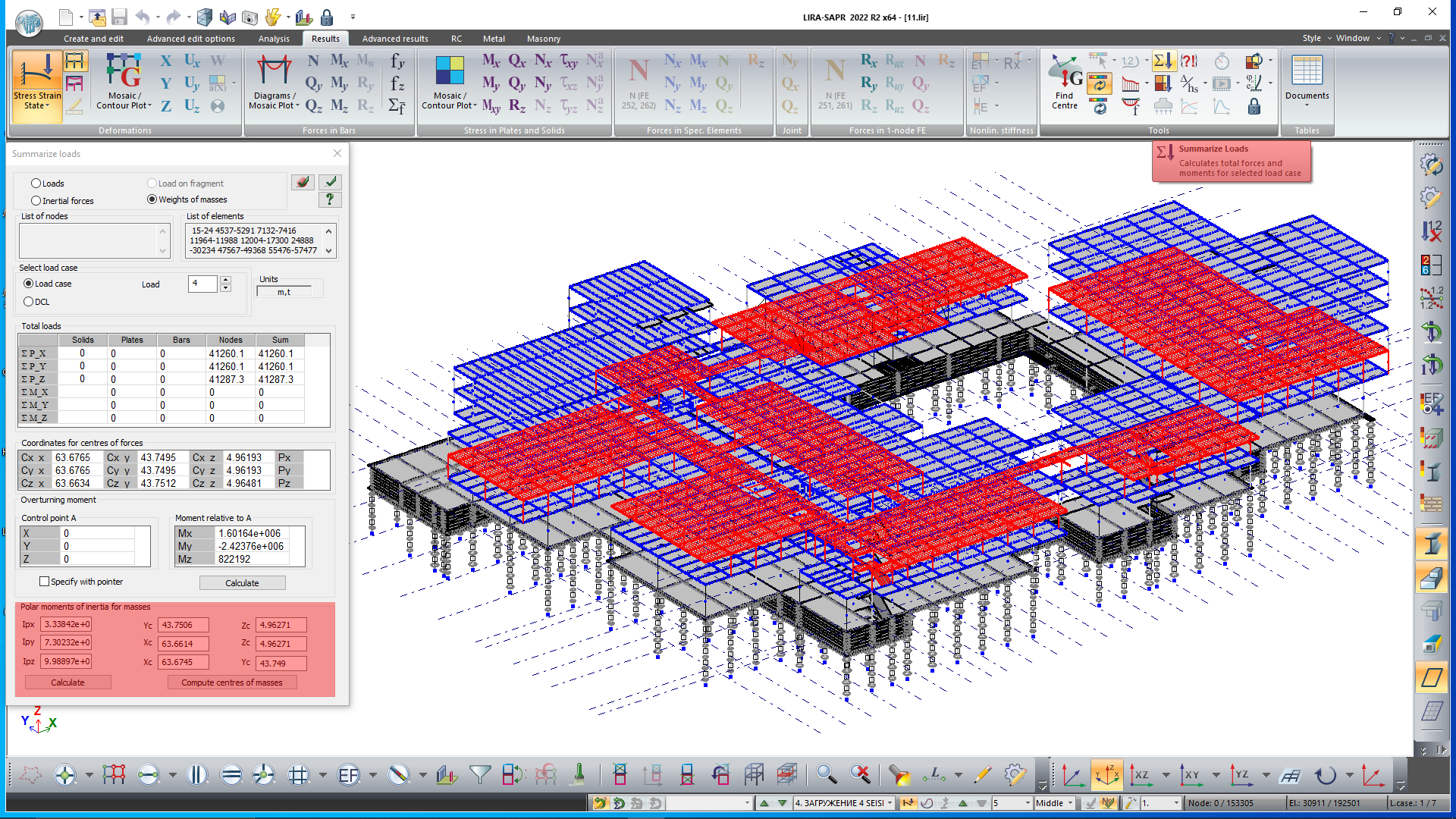


Рисунок 2 - Внешний вид Total Commander (на ОС Windows 7)

### 1.1.3 ****Frame3DD****

**Frame3DD** — это **кроссплатформенный программный пакет для анализа стержневых конструкций**, предназначенный для проведения **плоского и пространственного расчета рам, ферм и 3D-конструкций** методом конечных элементов. Проект был изначально разработан **автором Симоном Уэлсом (Simon Winther Wold)** и распространяется как **открытое программное обеспечение (open source)** под лицензией GPL. Благодаря своей простоте и гибкости, Frame3DD пользуется популярностью среди инженеров, студентов и исследователей, которым требуется легковесный, но точный расчетный инструмент.

Frame3DD работает через **командную строку**, что делает его особенно удобным для **автоматизации** и интеграции в расчетные пайплайны. Входные данные задаются в виде текстового файла с описанием узлов, элементов, нагрузок и граничных условий. После выполнения расчета программа создает выходные текстовые файлы с усилиями, перемещениями, реакциями и другими характеристиками.

Несмотря на отсутствие графического интерфейса, Frame3DD поддерживает экспорт данных в формат, совместимый с **gnuplot, MATLAB, Excel** и другими визуализаторами, а также может быть легко встроен в Python-скрипты для массовой генерации и анализа конструкций.

Программа поддерживает:

1. **Гравитационные и сосредоточенные нагрузки**, моменты, распределенные нагрузки;
2. **Геометрическую нелинейность** (в том числе эффект П–Δ);
3. **Связи (constraints), жесткие диафрагмы, начальные деформации, температурные эффекты**;
4. Расчеты на **устойчивость и модальные (собственные) частоты колебаний**;
5. Возможность анализа **многоузловых элементов с различными сечениями и свойствами**.

**Преимущества Frame3DD:**

* Полностью бесплатный и с открытым исходным кодом;
* Быстрый, легкий и не требующий установки (работает из консоли);
* Отлично подходит для **автоматизации** инженерных расчетов и исследований;
* Поддерживает **нелинейный расчет** и базовые динамические анализы;
* Простая и понятная структура входных файлов.

**Недостатки:**

1. Отсутствие встроенной графики: пользователю требуется сторонняя программа для визуализации;
2. Нет поддержки сложных материалов или бетонной арматуры;
3. Требуется подготовка текстовых файлов вручную или через скрипты;
4. Интерфейс и документация полностью на английском.

Несмотря на свою минималистичность, **Frame3DD является надежным и быстрым инструментом**, который может быть использован как в академических целях, так и для быстрого предварительного расчета реальных инженерных задач. Особенно популярен среди **инженеров-энтузиастов**, разрабатывающих собственные фреймворки анализа или интеграции с CAD/BIM.

## 1.2 Описание предмета исследования

Существует несколько способов определить, являются ли один файл дубликатом другого. Самым простым способом является сравнение имён и расширений файлов. Если два файла имеют одинаковое имя и одинаковое расширение, то, скорее всего, один из них является дубликатом другого. Это самый простой, но и самый не эффективный способ. Второй файл может не оказаться дубликатом первого, а быть его устаревшей или модифицированной версией или, что тоже вероятно, файлы могут быть не связаны между собой.

Данный способ можно усовершенствовать, если помимо имён и расширений сравнивать вес файлов. Если файлы весят одинаково, то и их содержимое, скорее всего, одинаково. Однако и так нельзя уверенно заявить, что перед нами находится дубликат файла. В расчёты можно также включить и время создания/модификации файла, что тоже не поможет усовершенствовать алгоритм поиска.

Единственным надёжным вариантом является хеш-сумма файла, вычисленная на основе его содержимого. Хеш-сумма (или контрольная сумма) файла — это уникальное строковое значение, получаемое в результате обработки содержимого файла хеш-функцией. Хеш-сумма используется для проверки целостности файла. Если содержимое файла изменяется, изменяется и его хеш-сумма. Это позволяет отслеживать изменения и сравнивать их с эталонным содержанием оригинала.

Обычно, метод вычисления хеш-суммы проверяют при передаче файла по сети, чтобы убедиться, что файл не был повреждён, но так как хеш-сумма зависит от содержимого файла, то данный метод можно применить и для поиска дубликатов файлов.

Для вычисления хеш-сумм существует множество алгоритмов. Рассмотрим основные их них.

*MD5 (англ. Message Digest 5)* — 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института в 1991 году. Предназначен для создания «отпечатков» или дайджестов сообщения произвольной длины и последующей проверки их подлинности. Широко применялся для проверки целостности информации и хранения хешей паролей.

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям. Ниже приведены 5 шагов алгоритма:

1. Выравнивание потока. Сначала к концу потока дописывают единичный бит. Затем добавляют некоторое число нулевых бит такое, чтобы новая длина потока L\* стала сравнима с 448 по модулю *512 (L\* = 512 \* N + 448)*. Выравнивание происходит в любом случае, даже если длина исходного потока уже сравнима с 448;
2. Добавление длины сообщения. В конец сообщения дописывают 64-битное представление длины данных (количество бит в сообщении) до выравнивания. Сначала записывают младшие 4 байта, затем старшие. Если длина превосходит , то дописывают только младшие биты (эквивалентно взятию по модулю ). После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит;
3. Инициализация буфера. Для вычислений инициализируются четыре переменные размером по 32 бита, начальные значения которых задаются шестнадцатеричными числами (порядок байтов little-endian):

А = 01 23 45 67; // 67452301h

В = 89 AB CD EF; // EFCDAB89h

С = FE DC BA 98; // 98BADCFEh

D = 76 54 32 10. // 10325476h

В этих переменных будут храниться результаты промежуточных вычислений. Начальное состояние ABCD называется *инициализирующим вектором.*

1. Вычисление в цикле. Определим функции и константы, которые понадобятся нам для вычислений. Для каждого раунда потребуется своя функция. Введём функции (обозначенные FunF, FunG, FunH и FunI) от трёх параметров — слов (обозначенные X, Y, Z), результатом также будет слово:

1-й этап:

2-й этап:

3-й этап:

4-й этап:

Определим таблицу констант T[1…64] – 64-элементная таблица данных, построенная следующим образом: . Каждый 512-битный блок проходит 4 этапа вычислений по 16 раундов. Для этого блок представляется в виде массива X из 16 слов по 32 бита. Все раунды однотипны и имеют вид: [abcd k s i], определяемый как , где k — номер 32-битного слова из текущего 512-битного блока сообщения, и << s — циклический сдвиг влево на s бит полученного 32-битного аргумента. Число s задается отдельно для каждого раунда.

Заносим в блок данных элемент n из массива 512-битных блоков. Сохраняются значения A, B, C и D, оставшиеся после операций над предыдущими блоками (или их начальные значения, если блок первый).

*Примечание*: при описании этапов вычисления между символами /\*\*/ размещено небольшое пояснение к вычислению.

AA = A; BB = B; CC = C; DD = D

Этап 1:

/\* [abcd k s i] a = b + ((a + FunF(b,c,d) + X[k] + T[i]) << s). \*/

[ABCD 0 7 1] [DABC 1 12 2] [CDAB 2 17 3] [BCDA 3 22 4]

[ABCD 4 7 5] [DABC 5 12 6] [CDAB 6 17 7] [BCDA 7 22 8]

[ABCD 8 7 9] [DABC 9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]

[ABCD 12 7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16]

Этап 2:

/\* [abcd k s i] a = b + ((a + FunG(b,c,d) + X[k] + T[i]) << s). \*/

[ABCD 1 5 17] [DABC 6 9 18] [CDAB 11 14 19] [BCDA 0 20 20]

[ABCD 5 5 21] [DABC 10 9 22] [CDAB 15 14 23] [BCDA 4 20 24]

[ABCD 9 5 25] [DABC 14 9 26] [CDAB 3 14 27] [BCDA 8 20 28]

[ABCD 13 5 29] [DABC 2 9 30] [CDAB 7 14 31] [BCDA 12 20 32]

Этап 3:

/\* [abcd k s i] a = b + ((a + FunH(b,c,d) + X[k] + T[i]) << s). \*/

[ABCD 5 4 33] [DABC 8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]

[ABCD 1 4 37] [DABC 4 11 38] [CDAB 7 16 39] [BCDA 10 23 40]

[ABCD 13 4 41] [DABC 0 11 42] [CDAB 3 16 43] [BCDA 6 23 44]

[ABCD 9 4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA 2 23 48]

Этап 4:

/\* [abcd k s i] a = b + ((a + FunI(b,c,d) + X[k] + T[i]) << s). \*/

[ABCD 0 6 49] [DABC 7 10 50] [CDAB 14 15 51] [BCDA 5 21 52]

[ABCD 12 6 53] [DABC 3 10 54] [CDAB 10 15 55] [BCDA 1 21 56]

[ABCD 8 6 57] [DABC 15 10 58] [CDAB 6 15 59] [BCDA 13 21 60]

[ABCD 4 6 61] [DABC 11 10 62] [CDAB 2 15 63] [BCDA 9 21 64]

Суммируем с результатом предыдущего цикла:

*A = AA + A; B = BB + B; C = CC + C; D = DD + D*

1. Результат вычислений. Результат вычислений находится в буфере ABCD, это и есть хеш. Если выводить побайтово, начиная с младшего байта A и заканчивая старшим байтом D, то мы получим MD5-хеш.

Алгоритм, хоть и является достаточно простым в реализации, не считается безопасным. В 1993 году было установлено, что в алгоритме возможны псевдоколлизии, когда разным инициализирующим векторам соответствуют одинаковые дайджесты для входного сообщения.

В 1996 году была найдена коллизия в алгоритме, и уже в то время было предложено использовать другие алгоритмы хеширования, такие как Whirlpool, SHA-1 или RIPEMD-160.

Из-за небольшого размера хеша в 128 бит можно рассматривать birthday-атаки. В марте 2004 года был запущен проект MD5CRK с целью обнаружения уязвимостей алгоритма, при помощи birthday-атаки. Проект MD5CRK закончился 17 августа 2004 года, когда удалось обнаружить уязвимости в алгоритме.

*SHA-1 (Secure Hash Algorithm 1)* - алгоритм криптографического хеширования. Описан в RFC 3174. Для входного сообщения произвольной длины (максимум бит) алгоритм генерирует 160-битное (20 байт) хеш-значение, которое обычно отображается как шестнадцатеричное число длиной в 40 цифр. Используется во многих криптографических приложениях и протоколах. Также рекомендован в качестве основного для государственных учреждений в США. Принципы, положенные в основу SHA-1, аналогичны тем, которые использовались Рональдом Ривестом при проектировании MD4.

В 1993 году Агентство Национальной Безопасности США (АНБ) совместно с Национальным Институтом Стандартов и Технологий разработали алгоритм безопасного хеширования (сейчас известный как SHA-0) для стандарта безопасного хеширования. Однако вскоре АНБ отозвало данную версию, сославшись на обнаруженную ими ошибку, которая так и не была раскрыта, и заменило его исправленной версией, опубликованной в 1995 году в документе. Эта версия и считается тем, что называют SHA-1.

SHA-1 реализует хеш-функцию, построенную на идее функции сжатия. Входами функции сжатия являются блок сообщения длиной 512 бит и выход предыдущего блока сообщения. Выход представляет собой значение всех хеш-блоков до этого момента. Иными словами, хеш-блок *Mi* равен . Хеш-значением всего сообщения является выход последнего блока.

Исходное сообщение разбивается на блоки по 512 бит в каждом. Последний блок дополняется до длины, кратной 512 бит. Сначала добавляется 1 (бит), а потом — нули, чтобы длина блока стала равной 512 — 64 = 448 бит. В оставшиеся 64 бита записывается длина исходного сообщения в битах (в big-endian формате). Если последний блок имеет длину более 447, но менее 512 бит, то дополнение выполняется следующим образом: сначала добавляется 1 (бит), затем — нули вплоть до конца 512-битного блока; после этого создается ещё один 512-битный блок, который заполняется вплоть до 448 бит нулями, после чего в оставшиеся 64 бита записывается длина исходного сообщения в битах (в big-endian формате). Дополнение последнего блока осуществляется всегда, даже если сообщение уже имеет нужную длину.

Инициализируются пять 32-битовых переменных:

*A = 0x67452301*

*B = 0xEFCDAB89*

*C = 0x98BADCFE*

*D = 0x10325476*

*E = 0xC3D2E1F0*

Определяются четыре нелинейные операции в виде функций (обозначенные Ft, параметры для функции обозначены m, l, k) и четыре константы (обозначенные Kt). В зависимости от шага итерации (обозначенный t) используются разные константы и операции:

Главный цикл итеративно обрабатывает каждый 512-битный блок. В начале каждого цикла вводятся переменные a, b, c, d, e, которые инициализируются значениями A, B, C, D, E, соответственно. Блок сообщения преобразуется из 16 32-битовых слов Mi в 80 32-битовых слов Wj по следующему правилу:

Где t – шаг итерации, Mt – 16-битовое слово для t-ого шага итерации; Wt – 32-битовое слово для t-ого шага итерации; «<<» — циклический сдвиг влево.

Для t от 0 до 79:

*temp = (a << 5) + Ft(b, c, d) + e + Wt + Kt*

*e = d*

*d = c*

*c = b << 30*

*b = a*

*a = temp*

Где temp – переменная, используемая для временного хранения результатов вычислений; «+» — сложение беззнаковых 32-битных целых чисел с отбрасыванием избытка (33-го бита). После этого к A, B, C, D, E прибавляются значения a, b, c, d, e, соответственно. Начинается следующая итерация. Итоговым значением будет объединение пяти 32-битовых слов (A, B, C, D, E) в одно 160-битное хеш-значение.

SHA-1 является наиболее распространенным из всего семейства SHA и применяется в различных широко распространенных криптографических приложениях и алгоритмах. Например, данный алгоритм используется в системе контроля версий Git для идентификации каждого объекта по хешу от хранимой в объекте информации. Приложение BitTorrent использует данный алгоритм для проверки целостности загружаемых данных.

Несмотря на это, некоторые компании отказались от использования данного алгоритма, так как ещё в 2005 году вышли статьи о теоретическом взломе данного алгоритма. А 23 февраля 2017 года специалисты из Google и CWI и вовсе объявили о практическом взломе алгоритма, опубликовав 2 PDF-файла с одинаковой контрольной суммой SHA-1. Это потребовало перебора 9 \* 1018 вариантов, что заняло бы 110 лет на 1 GPU.

*SHA-256 (Secure Hash Algorithm 256-bit)* – криптографическая хеш-функция, которая превращает входные данные в строку размером 256 бит. Данный алгоритм гарантирует, что даже малейшее изменение во входных данных приведёт к созданию абсолютно другого по значению хеша, что делает данный алгоритм мощным инструментом для подтверждения целостности данных и защите информации. Алгоритм является частью семейства алгоритмов SHA-2, разработанного АНБ США.

Подробное описание работы алгоритмы достаточно громоздкое и сложное, потому обойдёмся лишь его общим описанием. Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хеш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя.

Алгоритм считается более стойким к коллизиям и более безопасным, чем MD-5. Алгоритм SHA-256 применяется во многих областях. Например, для хеширования паролей, создания цифровых подписей и даже в вычислениях в биткоине. Однако за высокую безопасность приходится жертвовать производительностью – согласно исследованиям, алгоритмы семейства SHA-2 работают в 2-3 раза медленнее других популярных алгоритмов, среди которых уже упомянутые MD-5 и SHA-1.

На основе описания перечисленных алгоритмов, выберем максимально подходящий для будущей разработки. Алгоритм MD-5 хотя и является достаточно быстрым в то же время является очень небезопасным в виду вероятности частых появлений коллизий. Алгоритм SHA-256 является самым безопасным из представленных, ведь для него ещё не была найдена коллизия, но в то же время является самым медленным, что точно не подойдёт для создания пользовательского приложения.

Таким образом, алгоритм SHA-1 является самым оптимальным вариантом из предложенных. Он считается таким же быстрым, как и алгоритм MD-5, но при этом гораздо более устойчивый к коллизиям. Конечно, он не является настолько же безопасным, как алгоритм SHA-256, однако стоит отметить, что для задачи нахождения дубликата файла не требуется такой уровень безопасности, и даже если произойдёт коллизия, что достаточно маловероятно для данного алгоритма, но вполне возможно, никаких критических последствий не произойдёт.

## 1.3 Структуры файловых систем компьютера

Современные операционные системы используют различные файловые системы, каждая из которых разработана с учётом специфических требований к производительности, надёжности и функциональности. Файловая система (ФС) определяет методы организации, хранения и управления данными на носителях информации, что напрямую влияет на работу всей вычислительной системы.

Самыми распространёнными файловыми системами на сегодняшний день являются NTFS, ext4 и APFS:

* NTFS (New Technology File System) — файловая система, разработанная Microsoft, которая обеспечивает высокую надёжность, поддержку шифрования, разграничения прав доступа и работу с большими файлами. Она широко применяется в современных версиях Windows.
* ext4 (Fourth Extended File System) — основная файловая система в Linux, отличающаяся стабильностью, высокой производительностью и поддержкой журналирования, что минимизирует риски потери данных.
* APFS (Apple File System) — файловая система Apple, оптимизированная для работы с SSD и флеш-накопителями. Она обеспечивает высокую скорость операций, поддержку снапшотов и встроенное шифрование.

Рассмотрим указанные файловые системы более подробно.

*NTFS (от англ. New Technology File System – файловая система новой технологии)* – стандартная файловая система на операционных систем семейства Windows. Впервые была представлена 27 июля 1993 года с выпуском операционной системы Windows NT 3.1 – первой крупной ОС семейства Windows NT для серверов и корпоративных рабочих станций на базе процессоров Intel x86, DEC Alpha и MIPS.

NTFS заменила собой файловую систему FAT (от англ. File Allocation Table – таблица размещения файлов), которая использовалась в ОС MS-DOS и ранних Windows. Разработана на основе файловой системы HPFS (от англ. high performance file system – высокопроизводительная файловая система), разработку которой компания Microsoft вела совместно с фирмой IBM при работе над операционной системой OS/2. Впоследствии разработчики из Microsoft добавили в HPFS квотирование, журналируемость, разграничение доступа и аудит и назвали получившееся NTFS. Данный функционал расширил возможности файловой системы, но также сильно снизил производительность, по сравнению с прародителем.

Для размещения файлов система использует кластеры. Размер кластера определяется при форматировании диска, и может быть выбран из следующих вариантов: 512 байт, 1 Кб, 2 Кб, 4 Кб, 8 Кб, 16 Кб, 32 Кб или 64 Кб. Начиная с Windows 10 также появились варианты кластеров размером 128 Кб, 256 Кб, 512 Кб, 1 Мб или 2 Мб. Размеры кластера стоит выбирать с умом. Кластеры маленького размера подходят для файлов маленького размера, таким образом будет лучше использоваться дисковое пространство, но увеличивается риск появления фрагментации. Для больших файлов подойдут кластеры более большого размера, это позволит увеличить скорость загрузки и снизит риск фрагментации, но для файлов меньшего размера дисковое пространство при больших размерах кластера будет использоваться нерационально, так как кластер, даже если будет заполнен не полностью, будет считаться целиком занятым.

В таблице 1 представлена наглядная разница между устаревшей системой FAT и более современной NTFS.

Таблица 1 – Сравнение файловых систем FAT16, FAT32, NTFS

|  |  |  |  |
| --- | --- | --- | --- |
| Ограничения / возможности | FAT16 | FAT32 | NTFS |
| Размеры диска | Приблизительно 8 ТиБ | | В теории 16 ЭиБ, на практике 255 ТБ на старых версиях и 8 ПБ на новых |
| Размер тома | До 127,53 ГБ | | Минимальный размер: 8 МиБ (рекомендованный: 10 МиБ) |
| Форматирование дискет | Поддерживается. За счёт отсутствия журналирования износ флэш-накопителей при этой операции уменьшался | | Не поддерживается |
| Поддержка ссылок разных типов | Не поддерживается | | Жёсткие и символьные ссылки, соединения для каталогов |
| Максимальный размер файла | 2 ГБ | 4 ГБ | В теории: 16 ЭиБ  На практике: 16 ТБ |
| Средства безопасности | Атрибуты файлов | | Атрибуты файлов, авторизация с использованием DACL, шифрование с использованием EFS |
| Аудит | Не поддерживается | | С использованием SACL |
| Поддержка сжатия | На уровне диска | Не поддерживается | На уровне файловой системы для файлов, каталогов и дисков (нельзя комбинировать с шифрованием). |
| Максимальное количество файлов | ~65 000 | 268 435 444 | 4 294 967 295 |

Спецификации файловой системы NTFS закрыты, из-за чего её полноценная поддержка имеется только в ОС семейства Windows.

*ext4 (от англ. fourth extended file system – четвёртая расширенная файловая система)* – журналируемая файловая система, основанная на файловой системе ext3. Первая экспериментальная реализация выпущена Эндрю Мортоном 10 октября 2006 года в виде обновления ядра Linux версии 2.6.19. Стабильная версия вышла 21 октября 2008 года.

Впервые файловая система ext была представлена в апреле 1992 года для ядра Linux версии 0.96c. Целью создания файловой системы являлось преодоление ограничений файловой системы Minix – в новой файловой системе размер раздела и файла увеличен до 2 Гб, а максимальная длина имени файла увеличен до 255 байт (в Minix размер раздела не мог превышать 64 Мб, а максимальная длина имени файла составляла 14 символов (байт)). ext стала первой файловой системой, которая расширила возможности файловой системы Minix.

В январе 1993 года началась разработка файловой системы ext2 на базе ext. Хоть ext и решала проблемы файловой системы Minix, она всё ещё имела ряд существенных недостатков, среди которых отсутствие поддержки раздельного доступа и временных меток модификации данных. Эти проблемы были решены в файловой системе ext2, в которой помимо перечисленных проблем также были реализованы соответствующие стандарту POSIX списки контроля доступа ACL и расширенные атрибуты файлов. Файловая система ext2 является самой быстрой файловой системой семейства ext, уступая лишь по скорости чтения системе ext4.

Файловая система ext3 была представлена в ноябре 2001 года с обновлением ядра Linux версии 2.4.15. Главным отличием от предыдущей версии является поддержка журнала, записывающего некоторые данные, что позволяет восстанавливать файловую систему при сбоях. Новая система получила поддержку файлов размером до 1 Тб, поддержку блочных устройств до 4 Тб, максимальное число блоков – 232.

Как уже было сказано ранее, файловая система ext4 создана на основе файловой системы ext3, то есть ext4 также является журналируемой файловой системой. Основные изменения файловой системы по сравнению с предыдущей версией следующие:

* увеличен максимальный объём одного раздела диска до 1 эксбибайта при размере блока 4 килобайта;
* увеличен размер одного файла до 16 тебибайт (244 байт);
* введён механизм протяжённой (extent) записи файлов, уменьшающий фрагментацию и повышающий производительность (новая информация добавляется в конец области диска, выделенной заранее по соседству с областью, занятой файлом);
* поднято ограничение на число вложенных каталогов с 32 000 подкаталогов до 65 535 (при этом в некоторых случаях требуется изменить константы ядра).

ext4 поддерживается всеми Linux дистрибутивами, а в некоторых даже идёт, как система по умолчанию. Также существуют драйверы для работы с ext4 на Windows.

*Apple File System (от англ. файловая система «Apple»)* – файловая система, разработанная компанией Apple для замены ранее использовавшейся системе HFS+ (от англ. hierarchical file system+ - иерархическая файловая система+) для использования в операционных системах iOS, macOS, tvOS, watchOS. Впервые была представлена 27 марта 2017 года. Файловая система оптимизирована для работы с твердотельными накопителями, поддерживает шифрование, копирование при записи, сжатие, клонирование файлов и каталогов, создание мгновенных снимков, контрольные суммы для проверки целостности метаданных (но не для пользовательских данных).

Является файловой системой по умолчанию в iOS версии 10.3. Возможность выбрать APFS в качестве основной на этапе установки появилась в macOS High Sierra (версия 10.13.4); тома, отформатированные в macOS Sierra в APFS, могут быть несовместимы с будущими версиями macOS, так же, как и с окончательной версией APFS.

Файловая система HFS+ была представлена 19 января 1998 года с выходом macOS версии 8.1. С тех пор разработчики всячески улучшали файловую систему, однако с бурным ростом технологий переделывать старые технологии становилось всё труднее, из-за чего было принято решение о создании новой файловой системы с нуля. В частности, у HFS+ отсутствовали наносекундные метки времени (т.е. точность хранения дат в системе равнялась секунде, хотя даже в NTFS, созданной на 5 лет раньше это значение равно 100 наносекунд). Одной из важных проблем HFS+ являлось то, что к файловой системе могли одновременно обращаться несколько процессов, что могло предоставить угрозу конфиденциальности пользователей.

Основные нововведения APFS:

* Контейнеры и тома. Контейнер – основной объект хранения данных в новой файловой системе. Контейнеры имеют собственную систему защиты от сбоев и распределения дискового пространства. Каждый контейнер содержит один или больше томов или файловых систем, в каждой из которых имеется собственное пространство имён;
* Наносекундные метки времени;
* Защита от сбоев. В файловой системе реализована схема метаданных copy-on-write (копирование при записи), которая гарантирует, что изменения в файловой системе и записи в журнале остаются в синхронизированном виде, если возникает внезапная ошибка во время записи;
* Возможность создавать разреженные файлы;
* Шифрование. Для каждого тома в контейнере устанавливается одна из нескольких моделей шифрования: без шифрования, шифрование с одним ключом, шифрование с несколькими ключами;
* Клонирование файлов и директорий. Клонирование – практически мгновенное копирование файла, при котором не требуется дополнительное место для хранения данных. При изменении клона файловая система записывает только изменённые данные, что способствует хранению множества версий больших файлов, занимая меньше дискового пространства;
* Снапшоты. Являются открытыми только для чтения «снимками» файловой системы в томе. Используется для более эффективной процедуры резервного копирования.

Пожалуй, главным недостатком файловой системы является то, что в первую очередь она создана для твердотельных накопителей и не очень хорошо работает с жёсткими дисками.

В таблице 2 представлено сравнение упомянутых выше файловых систем между собой.

Таблица 2 – сравнение файловых систем

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | | NTFS | ext4 | APFS |
| Метка тома | MBR | 0x07 | 0x83 | - |
| GPT | EBD0A0A2-B9E5-4433-87C0-68B6B72699C7 | | 7C3457EF-0000-11AA-AA11-00306543ECAC |
| Содержимое папок | | B+-дерево | Связный список, B-дерево | B-дерево |
| Размещение файлов | | Битовая карта | | B-дерево |
| Сбойные сектора | | Сбойные кластеры ($badclus) | Таблица (Table) | B-дерево |
| Максимальный размер файла | | 16 ЭиБ – 1 КиБ | 16 ТиБ | 8 ЭиБ |
| Максимум файлов | | 232 - 1 | 4 миллиарда | 263 |
| Максимальная длина имени файла | | 255 16-битовых слов в кодировке UTF-16 | 255 байт | Неизвестно |
| Максимальный размер тома | | 264 – 1 кластер | 1 ЭиБ | Неизвестно |
| Допустимые символы в названиях | | В пространстве имён Posix: все, кроме NUL, «/».  В пространстве имён Win32: все, кроме NUL, «/», «\», «:», «\*», «?», «”», «<», «>», «|» | Все, кроме NUL и «/» | Все, кроме NUL и «:» |
| Чувствительность к регистру | | В пространстве имён Posix: да  В пространстве имён Win32: нет | Да | Неизвестно |
| Диапазон дат | | 01.01.1601 – 28.05.60056 | 14.12.1901 – 25.04.2514 | 01.01.1970 – 21.07.2554 |
| Точность хранения дат | | 100 наносекунд | 1 наносекунда | 1 наносекунда |
| Потоки метаданных | | Да | Нет | Неизвестно |
| Атрибуты | | Только чтение; скрытый; системный; требует архивирования; не проиндексирован; недоступен; временный; сжатый; зашифрованный | (no)extends; (no)mballoc; (no)delalloc; data=journal; data=ordered; data=writeback; commit=nrsec; orlov; oldalloc; user\_xattr; nouser\_xattr; (no)acl; bsddf; minixdf; (no)bh; journal\_dev | access; attributes; modified; contents modified; created |
| Права доступа | | ACL | POSIX | Unix permissions, NFS, ACLs |
| Фоновая компрессия | | Пофайлово, LZ77 | Нет | decmpfs |
| Фоновое шифрование | | Пофайлово, DESX, Triple DES, AES | Нет | Есть |

# 2 Разработка приложения

## 2.1 Постановка задачи разработки

Программное решение рассчитано на пользователей персонального компьютера, осуществляющих работу с большим объёмом файлов различных форматов, среди которых могут появляться дубликаты. Целями создания файлового менеджера являются:

* Предоставление приятного и современного пользовательского интерфейса для взаимодействия с файловой системой компьютера;
* Увеличение скорости работы с файловой системой;
* Внедрение функции поиска дубликатов файлов;
* Повышение уровня безопасности.

Назначением приложения является взаимодействие с файловой системой компьютера.

В качестве требований в разработке предъявляется:

* Наличие двухпанельного графического интерфейса;
* Осуществление всех основных операций над файлами в системе (перемещение, копирование, переименование, создание, открытие и тому подобное);
* Наличие настраиваемой функции поиска дубликатов файлов (наличие настроек области поиска, выбора действий над найденными дубликатами);
* Наличие возможности кастомизации пользовательский интерфейс (настроить размер и внешний вид иконок, цвета интерфейса);
* Открытый исходный код, позволяющий создавать собственные расширения для файлового менеджера.

В качестве входных данных для работы файлового менеджера используется:

* Структура файловой системы используемой операционной системы;
* Метаданные файлов;
* Действия пользователя, принимаемые графическим интерфейсом

Выходными данными для файлового менеджера являются:

* Результаты поиска дубликатов;
* Результаты файловых операций;
* Графический интерфейс файлового менеджера.

## 2.2 Выбор платформы для реализации

Согласно целям разработки, файловый менеджер должен быть кроссплатформенным, то есть запускаться на различных операционных системах. Речь идёт о операционных системах персонального компьютера, то есть мобильные платформы пока в задачи разработки не входят. Самыми популярными ОС на персональные компьютеры являются Windows, Linux и macOS, график с соответствующим исследованием приведён на рисунке 4. За неимением устройства под управлением последней из перечисленных ОС, разработка будет вестись только для первых двух, то есть для Windows и Linux.

По данным исследования, доля ОС Windows на март 2025 года составляет 71,72% по всему миру, доля устройств на операционной системе macOS (OS X) составляет 13,66%, доля устройств на Linux – 3,99%

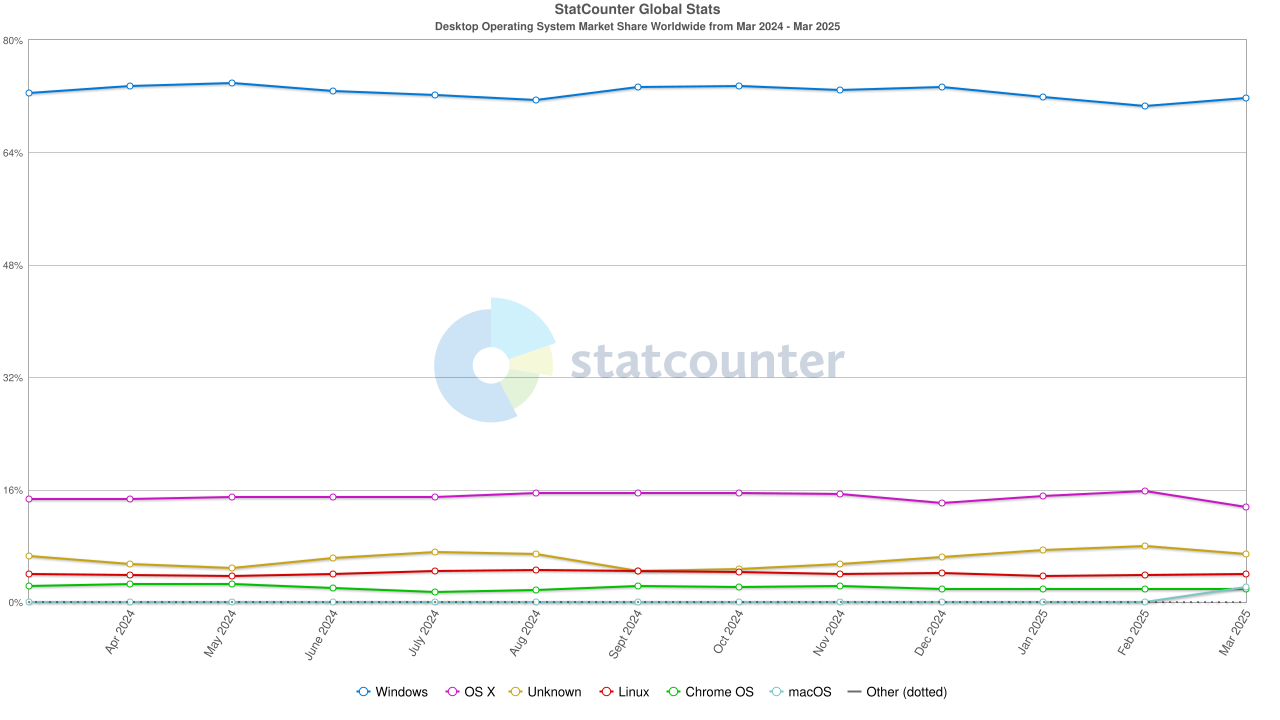


Рисунок 4 – Данные о популярности ОС по всему миру

*Примечание*: операционная система macOS до 2016 года именовалась OS X.

Таким образом, приложение будет адаптировано для ОС Windows и Linux.

Файловый менеджер предназначен для запуска на персональных компьютерах или ноутбуках. Для взаимодействия пользователя с ним понадобятся периферийные устройства, в числе которых клавиатура, мышь или тачпад.

Главным критерием для использования файлового менеджера является достаточно большой объём оперативно памяти, так как операция поиска дубликатов требует большого числа операций чтения и временные данные хранятся как раз в ОЗУ.

К устройству пользователя предъявляются следующие системные требования:

* **Процессор**: 4-ядерный (или больше) с частотой 2 ГГц (или больше);
* **Оперативная память**: 4 ГБ (рекомендуется 8 ГБ)
* **Видеокарта**: с поддержкой OpenGL 2.0
* **Место на диске**: 500 МБ

## 2.3 Выбор языка разработки

Выбор языков программирования для создания программного решения зависит от требуемых от продукта функций. Для создания кроссплатформенного файлового менеджера необходимо наличие для языка кроссплатформенных библиотек. Необходимы библиотеки, которые предоставляют возможность работать с графическими интерфейсом, то есть создавать окна приложения и различные виджеты для взаимодействия с этим самым приложением, поддержку событий, таких как перетаскивание (так называемый Drag-and-Drop), нажатие клавиш на клавиатуре и компьютерной мыши. Также требуется библиотека, которая предоставит функционал для работы с различными алгоритмами вычисления хешей, в частности библиотека должна поддерживать алгоритм SHA-1. Сам же язык должен обладать хотя бы минимальными инструментами для работы с файловой системой, как-то чтение и запись данных в файл. В соответствии с данными критериями, выбор пал на следующие языки программирования: Python, Java и C++.

Python – это интерпретируемый мультипарадигменный язык программирования с динамической типизацией, разработанный Гвидо ван Россумом. Код на Python может запуститься на любой машине, на которой установлен интерпретатор этого языка, что делает его очень удобным для кроссплатформенной разработки. Python знаменит количеством разнообразных библиотек, созданных пользователями языка. Например, сообществом создано большое количество библиотек для создания графического интерфейса. Рассмотрим некоторые из них.

Tkinter – кроссплатформенная событийно-ориентированная графическая библиотека, которая является свободно распространяемым ПО. Входит в набор стандартных библиотек, поставляемых вместе с языком Python. Библиотека не отличается большим количеством виджетов, что, с одной стороны, упрощает процесс изучения и процесс создания приложения, но с другой стороны, усложняем процесс создания сложных виджетов и красивых интерфейсов. Поэтому данная библиотека больше подходит для обучения или создания совсем простеньких программ.

Более подходящим вариантом для создания интерфейса является библиотека wxPython – это кроссплатформенная обёртка библиотеки wxWidgets, написанной на C++, для языка Python. Данная библиотека является open-source и распространяется свободно. Первая версия этой библиотеки вышла в 1998 году и до сих пор активно разрабатывается и обновляется. В библиотеке имеется большое количество разнообразных виджетов и инструментов, облегчающих процесс написания графического интерфейса. На wxPython написано множество известных программ, к примеру BitTorrent, Dropbox, настольный клиент для Google Диск.

Данная библиотека является отличным вариантом для разработки. Как уже было сказано ранее, библиотека основана на библиотеке wxWidgets, написанной на C++, поэтому wxPython совмещает в себе лаконичность языка Python и производительность языка C++, что несколько нивелирует недостаток Python в плане скорости работы.

Язык программирования Python, как уже было отмечено, имеет один серьёзный недостаток – он работает несколько медленнее, чем, например, компилируемые языки, такие как C и уже упомянутый C++. Поэтому, вычисления контрольных сумм лучше перенести на другие, более быстрые языки, например, Java.

Java – строго типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems. Особенностью этого языка является то, что язык транслируется в специальный байт-код, который затем обрабатывается виртуальной машиной Java. Реализация этой виртуальной машины есть для всех операционных систем, поэтому можно считать Java кроссплатформенным языком программирования в том плане, что не требуется вносить изменения в написанные программы, чтобы перенести их с одного устройства на другое. Так как этот язык будет использоваться для вычисления контрольных сумм необходимо, чтобы для этого языка существовал соответствующий инструмент. В Java имеются встроенные инструменты для работы с алгоритмами. Одним из таких инструментов является MessageDigest.

Ещё одним вариантом для описания алгоритма может послужить язык программирования C++. Это компилируемый, статически типизированный язык программирования общего назначения, разработанный Бьёрном Страуструпом. Данный язык программирования знаменит своей системой управления памяти через указатели – специальные типы данных, предназначенные для общения с ячейками оперативной памяти. Язык хоть и является достаточно быстрым и эффективным, сложен для применения в разработке, особенно кроссплатформенных приложений, так как некоторые типы данных языка являются платформозависимыми. Язык C++ критикуется многими разработчиками за свою избыточную сложность, небезопасность и ненадёжность, поэтому многие разработчики предпочитают использовать языки C или Rust.

Таким образом, для разработки файлового менеджера выбраны языки Python (для создания пользовательского интерфейса) и Java (для работы над хеш-суммами).

## 2.4 Выбор среды разработки

Исходя из выбранного стека технологий, определим IDE, на которой будет производиться разработка программного продукта. В таблице 3 приведена сравнительная характеристика некоторых IDE и редакторов кода.

Таблица 3 – Сравнение сред разработки

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| IDE | Поддержка Python | Поддержка Java | Интеграция с wxPython | Интеграция с Java-библиотеками | Отладка |
| IntelliJ IDEA | Имеется | Полная | Отсутствует | Полная | Полная отладка Java кода |
| PyCharm | Полная | Имеется | Имеется | Имеется (при наличии нужных библиотек) | Полная отладка Python кода |
| Eclipse | Имеется | Имеется | Отсутствует | Полная | Полная отладка Java кода |
| VS Code | Полная | Имеется | Имеется | Имеется | Полная отладка Java и Python кода |

На основании данных таблицы были выбраны PyCharm и Intellij IDEA.

Intellij IDEA Community Edition – IDE для разработки на языках программирования Java и Kotlin, созданная компанией JetBrains. IDE имеет полную интеграцию с системами сборки Maven и Gradle, что, при наличии установленного пакета разработчика Java, не требует более никаких дополнительных действий. IDE предлагает удобные инструменты для написания кода, например, в ней есть технологии умного завершения кода и цепочного завершения, которые предлагают релевантные варианты для написания кода, учитывая контекст программы. IDE имеет удобные инструменты для рефакторинга, глубокую интеграцию с фреймворками и поддерживает множество плагинов.

PyCharm Community Edition – IDE для разработки на языке программирования Python от тех же разработчиков, что и Intellij IDEA. Как и в IDE для Java, разработчики реализовали здесь все удачные инструменты для помощи в разработке кода, тут также имеются технологии умного завершения кода и цепочное завершение. Кроме того, IDE предоставляет удобный графический интерфейс для взаимодействия с пакетным менеджером pip, что избавляет от необходимости писать в терминал команды для скачивания библиотек, всё делается легко и быстро при помощи удобного поиска. Кроме того, JetBrains реализовала в своих IDE удобную интеграцию с Git. Прямо внутри IDE можно просмотреть все ветви проекта, создавать коммиты и даже отправлять их в удалённый репозиторий.

## 2.5 Описание функциональной части программы

Рассмотрим, как использование нового файлового менеджера упростит работу пользователя персонального компьютера, а конкретно процесс поиска дубликата файлов в системе. На рисунках 5.1 – 5.4 представлена схема процесса поиска дубликатов файлов в виде диаграммы нотации IDEF0.

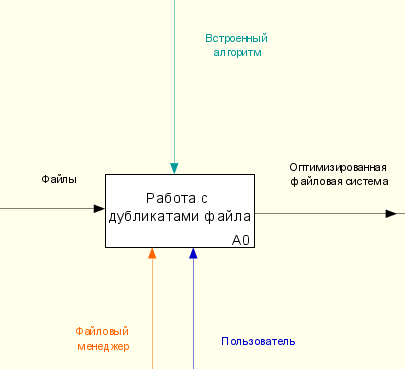


Рисунок 5.1 – Контекстная диаграмма

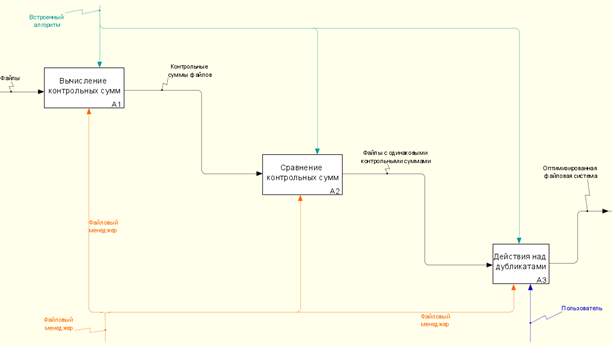


Рисунок 5.2 – Детализация концептуальной модели

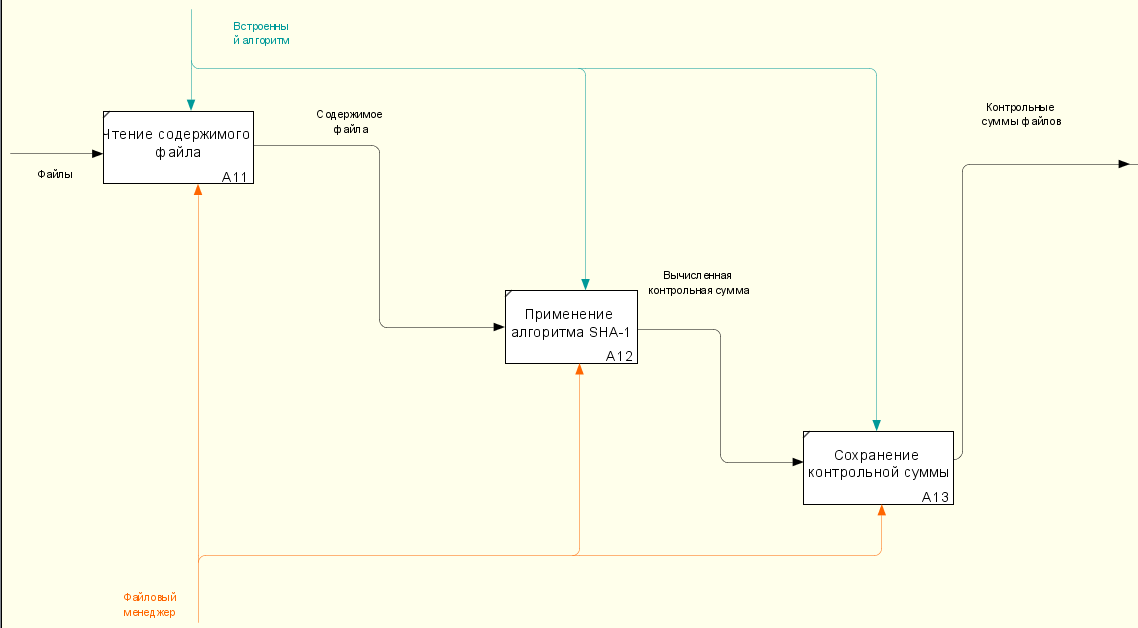


Рисунок 5.3 – Детализация процесса «Вычисление контрольных сумм»

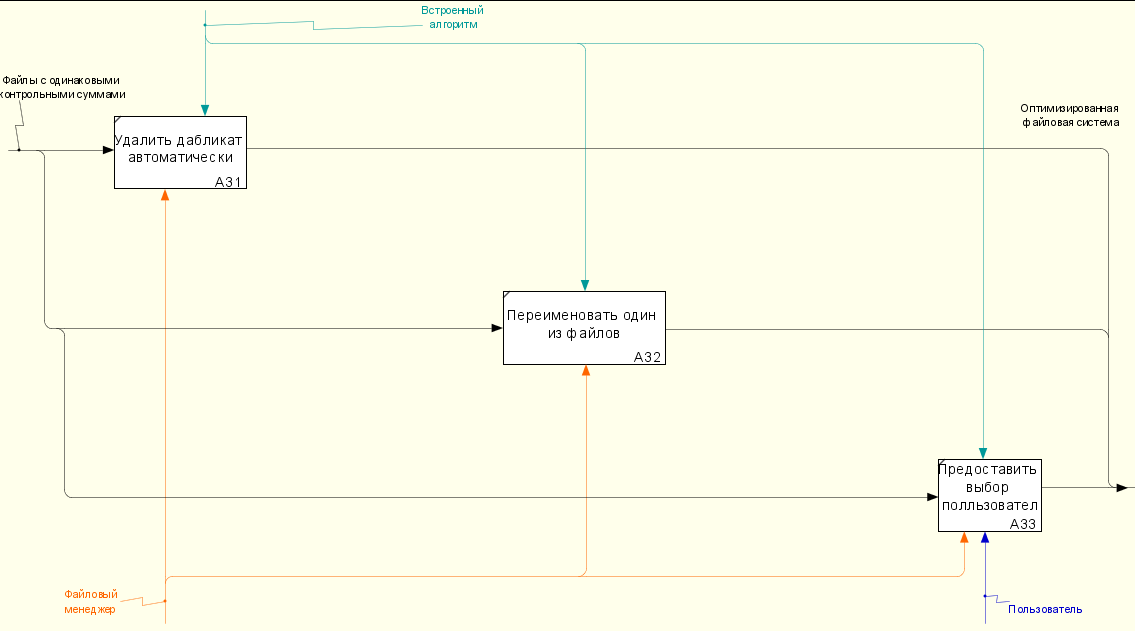


Рисунок 5.4 – Детализация процесса «Действия над дубликатами»

## 2.6 Разработка программы

Разберём структуру проекта файлового менеджера. На рисунке 6.1 представлена структура проекта по разработке файлового менеджера. Разберём поподробнее представленные модули:

* .venv – виртуальная среда проекта, содержащая свой независимый набор Python-библиотек для текущего проекта;
* framework – модуль с общими для проекта файлами – утилитами, помогающими работе других файлов; абстрактные классы, обобщающие логику других классов и прочее;
* icons – иконки, используемые в приложении;
* models – класс с моделями. Модели используются для более удобной работы с json-документами;
* settings – модуль с настройками проекта. Здесь хранятся конфигурации для окон (их стили, размеры), используемые размеры иконок и прочие настройки;
* widgets – модуль с реализованными виджетами, используемыми в проекте. Каждый файл в данном модуле – отдельный виджет со своей логикой работы;
* windows – модуль с реализованными окнами, используемыми в проекте. Каждый файл – отдельное окно со своим интерфейсом.
* app.py – файл с описанием самого приложения. Главный файл программы;
* main.py – входная точка программы. Здесь происходит создание самого приложения и его запуск.
* README.md – файл с кратким описание проекта;
* requirements.txt – файл со списком используемых в проекта библиотек с указанием их версий.

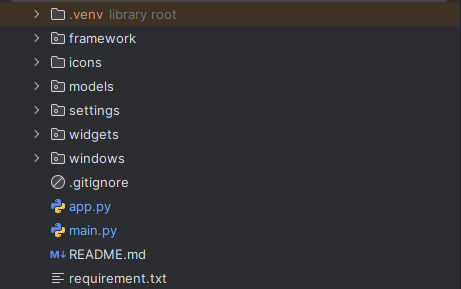


Рисунок 6.1 – Структура проекта

Рассмотрим содержимое модулей программы. На рисунке 6.1.1 изображено содержимое модуля framework. Внутри располагаются модули с общими для проекта инструментами. Рассмотрим, что располагается внутри:

* base\_windows – модуль с описанием некоторых базовых окон приложения. В таких файлах описана некоторая общая логика для работы окна, а конкретная реализация выполняется при помощи механизма наследования от этих файлов;
* events – некоторые пользовательские события, нужные для работы приложения. Как уже было отмечено ранее, библиотека wxPython является событийно-ориентированной, т.е. различные модули программы порождают события, и другие модули на эти события реагируют;
* singleton – модуль для описания паттерны singleton. Singleton – паттерн проектирования, гарантирующий, что в программе будет существовать только один объект заданного класса;
* utils – различные вспомогательные утилиты для проекта.

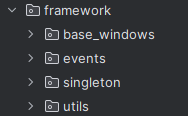


Рисунок 6.1.1 – Структура модуля framework

На примере директории utils в папке framework разберём, как выглядит структура Python пакета. Просмотреть содержимое директории можно на рисунке 6.1.2. Python пакет – это набор взаимосвязанных модулей, предназначенных для решения задач определённого класса некоторой предметной области. Например, данный пакет utils содержит вспомогательные классы для нашего приложения, например, класс для работы с файловой системой, класс для чтения файлов и прочее. Главным файлом Python пакета является файл \_\_init\_\_.py. Именно этот файл и сообщает интерпретатору Python, что директорию следует воспринимать, как пакет. Благодаря этому файлу интерпретатор понимает, что директория состоит из модулей, и будет здесь их искать. Сам файл \_\_init\_\_.py может быть пустым, а может содержать какой-либо код для инициализации пакета. Например, в директории framework имеется пакет events, который состоит целиком из файла \_\_init\_\_.py, а в нём прописаны все события, не входящие в изначально в библиотеку wxPython.

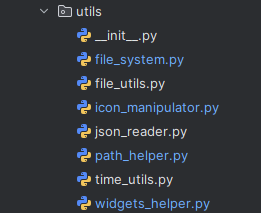


Рисунок 6.1.2 – Пример структура Python пакета

## 2.7 Разработка интерфейса приложения

К итоговому интерфейсу программы, согласно ГОСТР ИСО 9241-161-2016 выдвигаются следующие требования:

* интерфейс должен быть доступным. То есть ПО, обеспечивающее элементы интерфейса пользователя, должно использовать общедоступные услуги, предоставляемые платформой, совместно со вспомогательными технологиями;
* в интерфейсе должна быть реализована взаимосвязь методов ввода и графических элементов интерфейса пользователя;
* в интерфейсе должны отображаться состояния графических элементов интерфейса пользователя. Элементы интерфейса пользователя в зависимости от состояния системы и действия пользователя могут иметь различные состояния. Необходимо, чтобы каждое состояние было чётко визуально отличимо от другого;

Также, интерфейс программы должен соблюдать принципы, перечисленные в ГОСТ Р ИСО 14915-1-2016:

* информативность;
* управляемость;
* соответствие ожиданиям пользователя;
* устойчивость к ошибкам;
* пригодность для изучения;
* пригодность для восприятия и понимания;
* привлекательность.

Исходя из всех перечисленных требований и принципов, к интерфейсу файлового менеджера выдвигаются следующие требования:

* В файловом менеджере должен быть реализован двухпанельный интерфейс. Данное решение зарекомендовало себя, как удобное и практичное решение для файлового менеджера, поэтому в итоговом продукте оно также должно присутствовать;
* Обе панели файлового менеджера отображаются одновременно, позволяя в один момент времени просмотреть содержимое двух директорий;
* Интерфейс должен быть современным и минималистичным, без лишних элементов;
* Файловый менеджер должен поддерживать так называемые «горячие» клавиши – команды, посылаемые с клавиатуры для быстрого выполнения некоторых действий над файлами. Например, перемещение файла из одной директории в другую посредством нажатия клавиши F5;
* Файловый менеджер должен поддерживать Drag & Drop операции – копирование (или перемещение, в зависимости от выставленных настроек) файлов (директорий) из одной директории в другую посредством перетаскивания;
* Интерфейс должен быть настраиваемым. Пользователь должен иметь возможность выбрать размер иконок, цвет элементов управления, качество и количество отображаемой информации о файле и подобное;
* Файловый менеджер должен поддерживать открытие файлов (с использованием посторонних программ, если потребуется);
* При нажатии ПКМ по объекту файла должно вызываться контекстное меню, предлагающее совершить какую-либо операцию над файлом (удалить, переместить, копировать и подобное);
* Интерфейс должен давать достаточное количество информации для понимания его работы. Элементы интерфейса, не имеющие текста, должны иметь всплывающие подсказки, помогающие понять их назначение.

Итоговый интерфейс программы можно увидеть на рисунках 7.1-7.6.

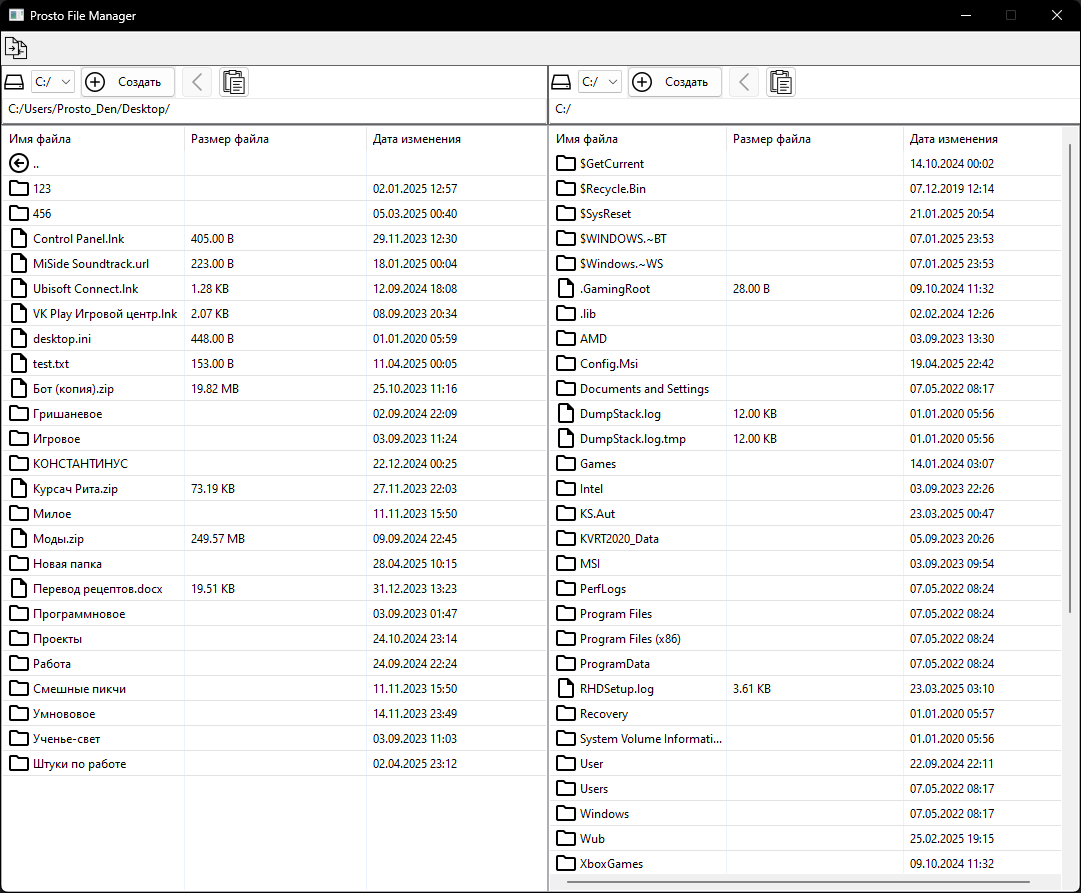


Рисунок 7.1 – Интерфейс главного окна программы

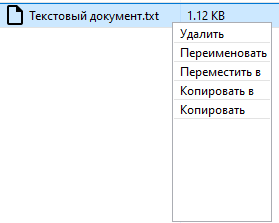


Рисунок 7.2 – Интерфейс контекстного меню

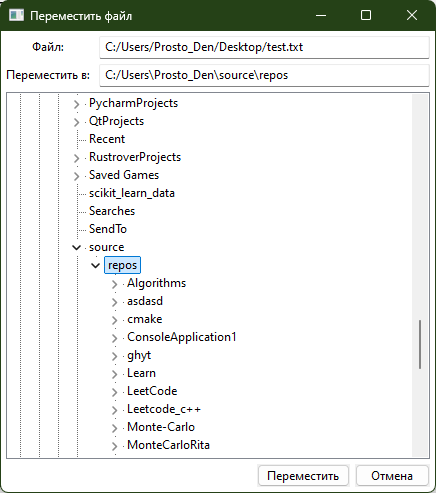


Рисунок 7.3 – Окно перемещения файла

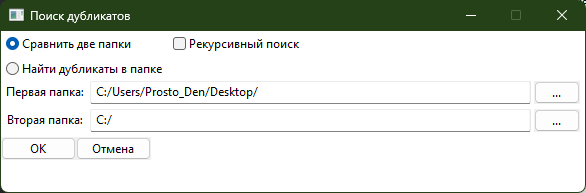


Рисунок 7.4 – Окно настройки поиска дубликатов

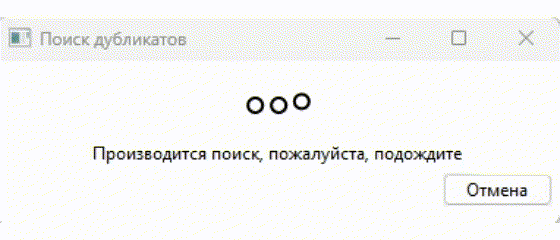


Рисунок 7.5 – Окно загрузки



Рисунок 7.6 – Результаты поиска дубликатов

# 3. Тестирование программы

## 3.1 Тестирование взаимодействия с файлами и директориями

Протестируем возможности созданного файлового менеджера. Откроем какую-нибудь пустую директорию, чтобы на ней лучше продемонстрировать создание новых файлов и директорий. На рисунке 8.1 представлен изначальный вид директории.

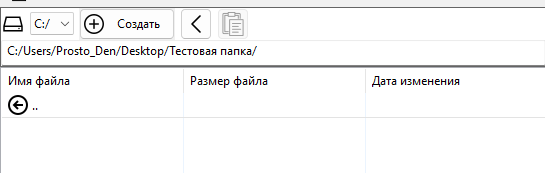


Рисунок 8.1 – Изначальный вид директории

Нажмём на кнопку «Создать» и из контекстного меню, изображённого на рисунке 8.2 выберем по очереди каждую из опций. На рисунке 8.3 видно, что директория больше не пуста, теперь там имеются папка с названием «Новая папка», текстовый файл с названием «Документ.txt» и файл для Microsoft Word с названием «Документ.docx». Эти названия выдаются по умолчанию. При этом, если мы попробуем создать ещё один, например, текстовый документ, его название будет немного отличаться, туда добавиться порядковый номер документа, так как в папке уже есть документ с именем «Документ.txt». Данное поведение видно на рисунке 8.4.

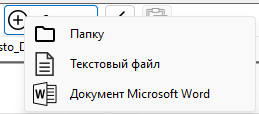


Рисунок 8.2 – Контекстное меню создания файлов

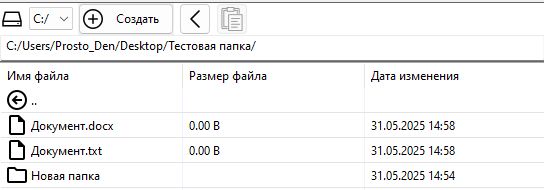


Рисунок 8.3 – Содержимое директории после создания файлов

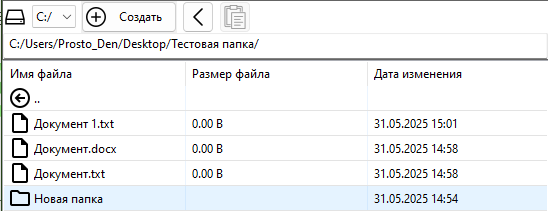


Рисунок 8.4 – Тестирование создания документа для непустой папки

При двойном нажатии левой кнопкой мыши по одному из документов он откроется в соответствующей программе. На рисунке 8.5 видно, что после двойного нажатия по файлу «Документ.txt» его содержимое открылось в программе «Блокнот». Это встроенный текстовый редактор в ОС Windows и является программой по умолчанию для открытия текстовых документов.

Теперь попробуем перейти в созданную нами папку. На рисунке 8.6 видно, что путь поменялся и теперь мы просматриваем содержимое нашей папки. Пока что там пусто.

Теперь воспользуемся второй панелью. На ней отобразим директорию, в которой мы были изначально. На рисунке 8.7 видно, что на левой панели изображено содержимое директории, которую мы только что создали и куда перешли на предыдущем шаге, а справа изображено содержимое изначальной директории.

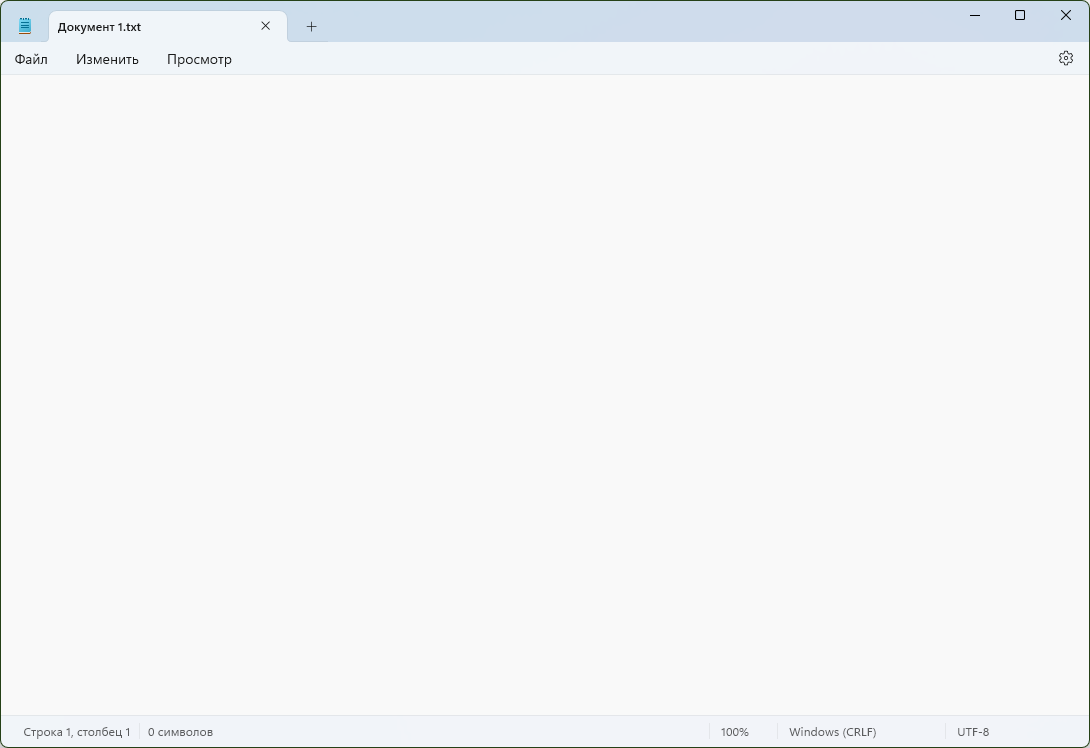


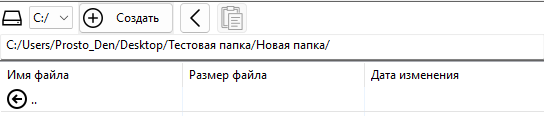
Рисунок 8.5 – Открытие документа

Рисунок 8.6 – Тестирование перехода по директориям

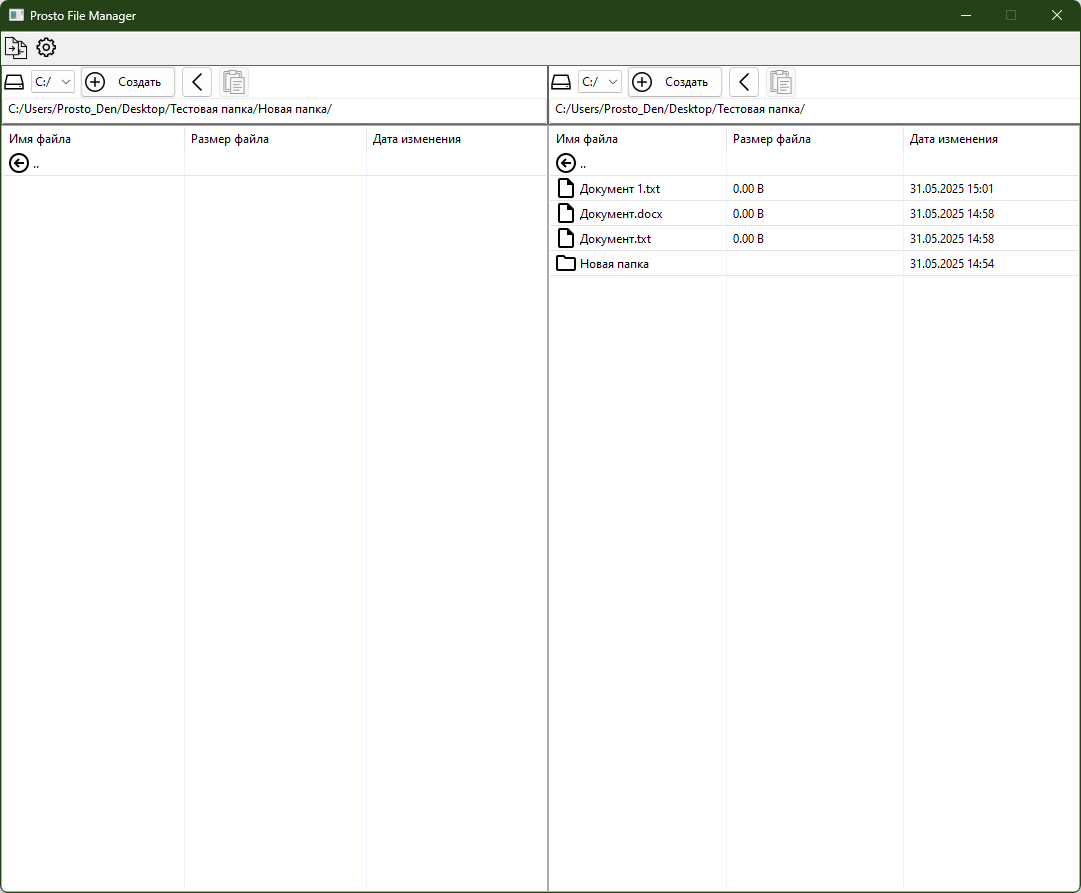


Рисунок 8.7 – Демонстрация двух директорий

Запишем какую-либо информацию в файл, изображённый на рисунке 8.5. Например, напишем туда предложение «Hello World!». Содержимое документа изображено на рисунке 8.8, а на рисунке 8.9 видно, что размер документа, изображаемый в файловом менеджере, а также даты последнего изменения поменялся. Теперь файл занимает не 0 байт, а 14, а датой последнего изменения теперь значится 31 мая, 15:09.

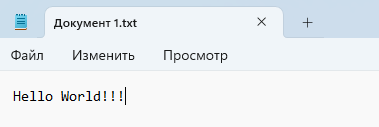


Рисунок 8.8 – Запись содержимого в документ

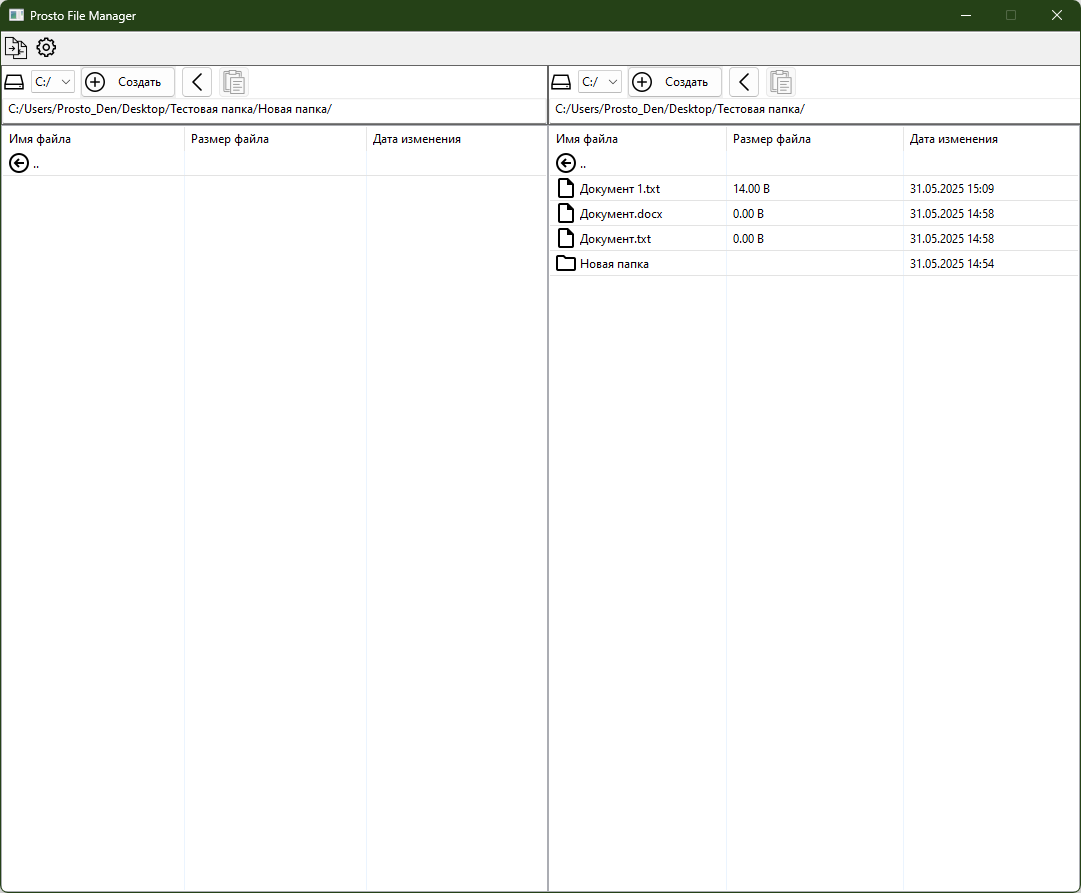


Рисунок 8.9 – Изменение размера файла (в правой панели)

Протестируем переименование файла. Кликнем правой кнопкой мыши по документу, в который мы только что записали сообщение. Появится контекстное меню, которое можно было увидеть на рисунке 7.2. Выберем в этом меню пункт «Переименовать». Имя документа преобразится, на его месте появится текстовое поля для ввода, в котором будет записано текущее название документа и всё, помимо расширения файла, будет выделено. Пронаблюдать это можно на рисунке 8.10.

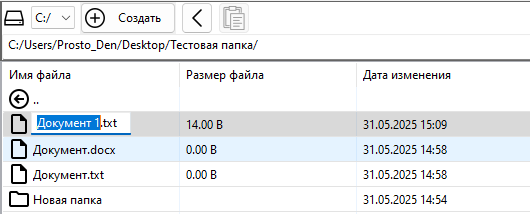


Рисунок 8.10 – Переименование документа

Введём новое название, например, назовём файл world.txt. На рисунке 8.11 видно, что название файла поменялось. Убедимся, что это тот же файл – откроем его, и увидим, что там находится то предложение, которое мы и оставляли (рисунок 8.12).

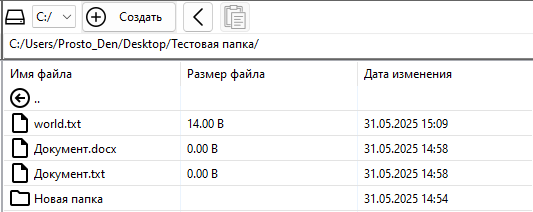


Рисунок 8.11 – Документ переименован

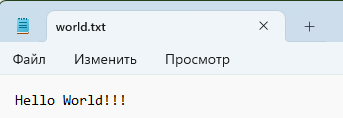


Рисунок 8.12 – Проверка содержимого документа

Попробуем перенести документ из этой папки в созданную ранее папку. Нажимаем правую кнопку мыши по документу и в контекстном меню выбираем опцию «Переместить в». Откроется окно, изображённое на рисунке 7.3. В верхнем поле будет указан путь документа, который мы перемещаем. Снизу указывается директория, в которую файл будет перемещён. В это поле разрешено вводить текст. Если пользователь сразу знает путь, в который нужно переместить файл, он может ввести его в поле и нажать на кнопку «Переместить». Или пользователь может построить путь с помощью дерева, расположенным под полем ввода. На рисунке 8.13 продемонстрирован процесс построения пути перемещения с помощью дерева. После нажатии на кнопку «Переместить» файл переместиться из изначальной директории в указанную. На рисунке 8.14 видно, что файла больше нет в директории правой панели, но теперь он появился в директории левой панели. Процесс копирования файла в определённую директорию осуществляется аналогично, за тем лишь исключением, что файл останется на своём месте, а в указанной директории появится его копия.

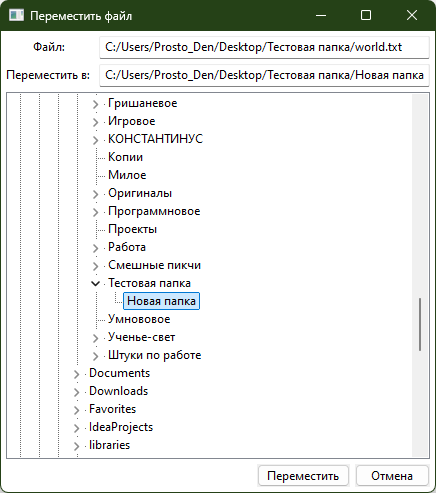


Рисунок 8.13 – Построение пути к директории при помощи дерева

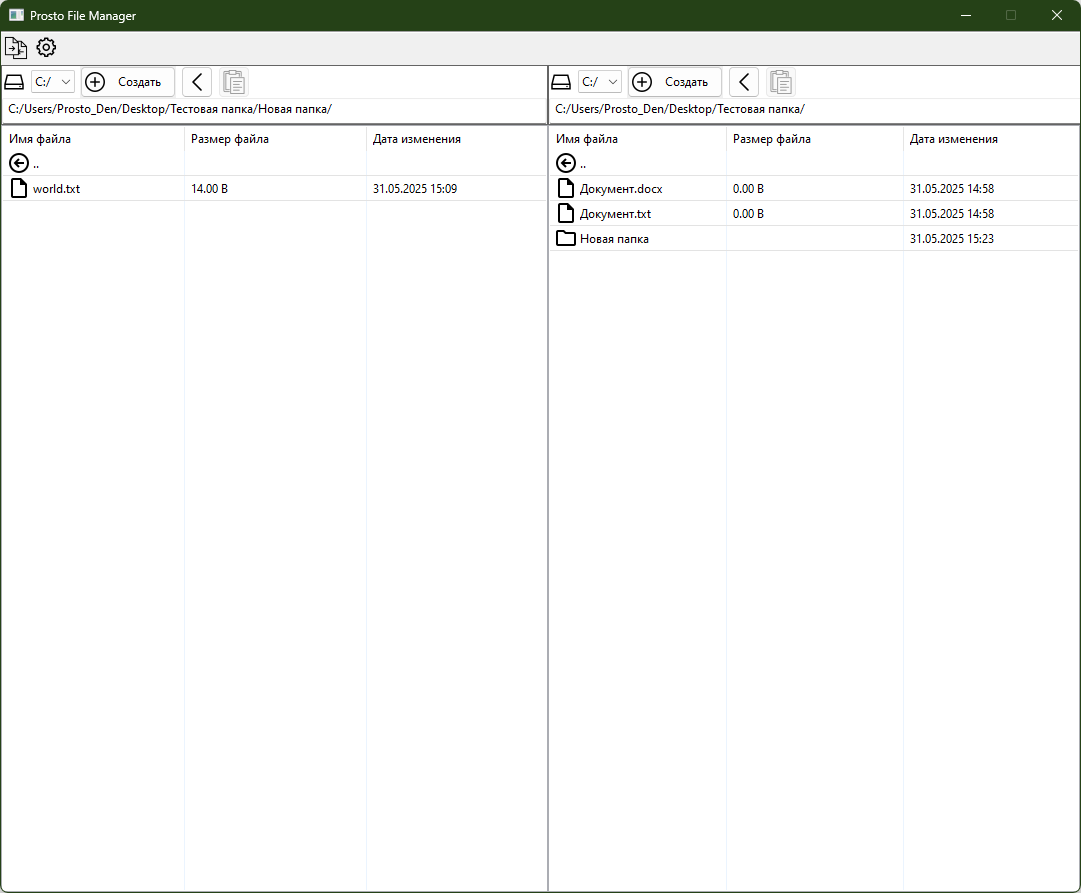


Рисунок 8.14 – Тестирование перемещения файла

## 3.2. Тестирование функции поиска дубликатов файлов

Перейдём к тестированию функции поиска дубликатов файлов. Создадим две директории, условно назовём их «Оригиналы» и «Копии» соответственно. В процессе тестирования будут сравниваться содержимые этих каталогов. В директории «Оригиналы» создадим вложенную папку, в которой так же будут располагаться файлы. На рисунках 9.1-9.3 изображены содержимые всех каталогов.

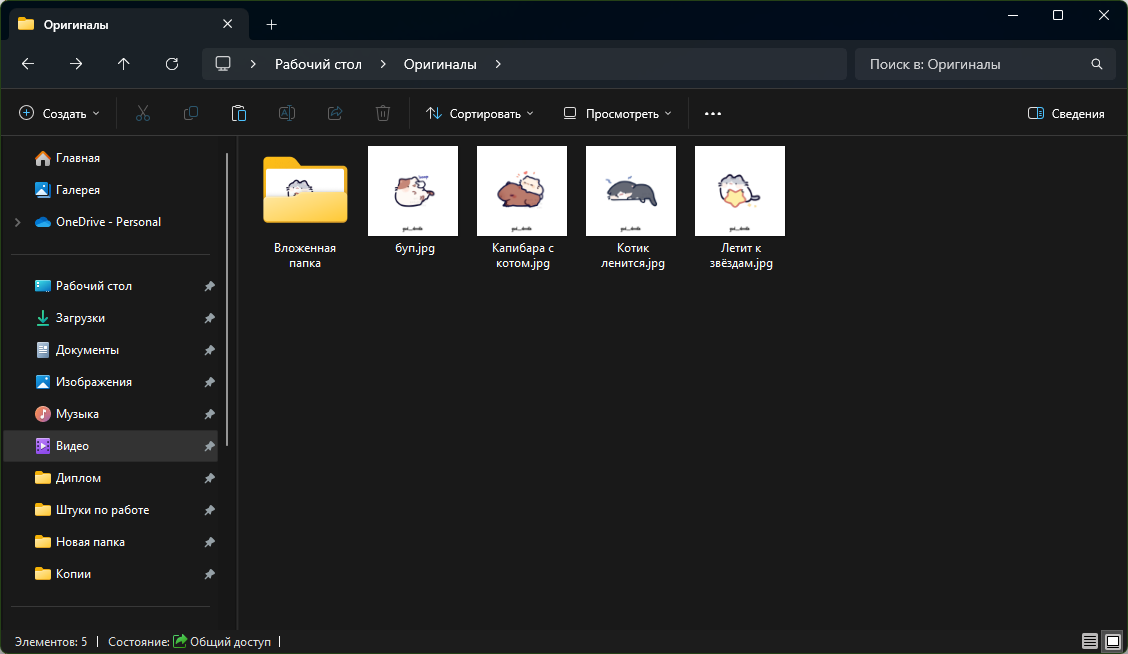


Рисунок 9.1 – Содержимое папки «Оригиналы»

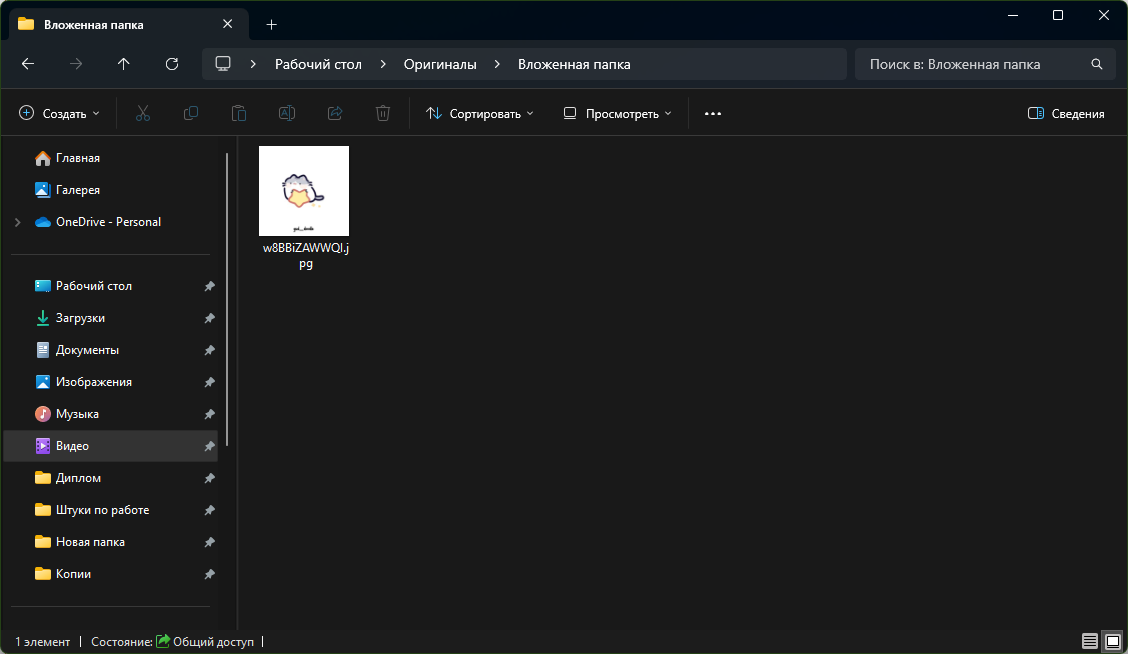


Рисунок 9.2 – Содержимое каталога «Вложенная папка», находящегося в папке «Оригиналы»

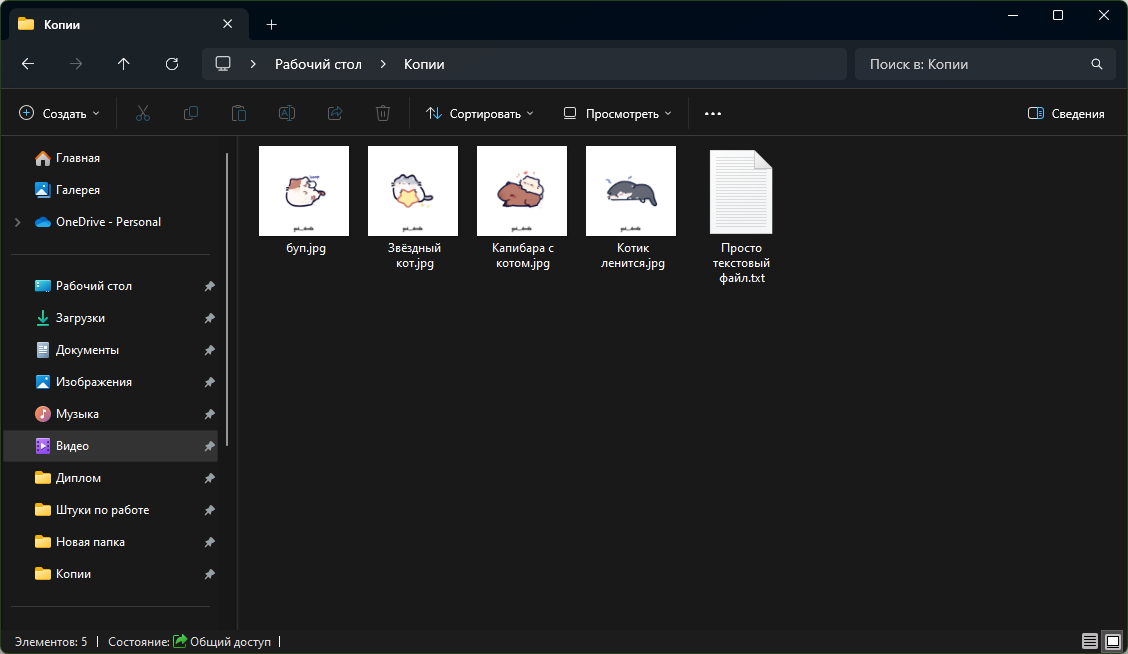


Рисунок 9.3 – Содержимое каталога «Копии»

Итого, исходя из содержимых директорий, получаем следующее:

* Файл «буп.jpg» из папки «Оригиналы» имеет один дубликат в папке «Копии» с таким же названием. Аналогично с файлами «Капибара с котом.jpg» и «Котик лениться.jpg»;
* Файл «Летит к звёздам.jpg» имеет два дубликата и все они имеют разные названия. Один дубликат расположен во «Вложенной папке» и имеет название «w8BBiZAWWQI.jpg», второй – в папке «Копии» и имеет название «Звёздный кот.jpg»;
* Файл «Просто текстовый файл.txt» не имеет дубликатов, соответственно, программа не должна его определить.

Откроем в файловом менеджере эти две директории так, как показано на рисунке 9.4. На этом же рисунке красной областью отмечена функция поиска дубликатов файлов, на которую необходимо нажать для начала настройки процесса поиска.

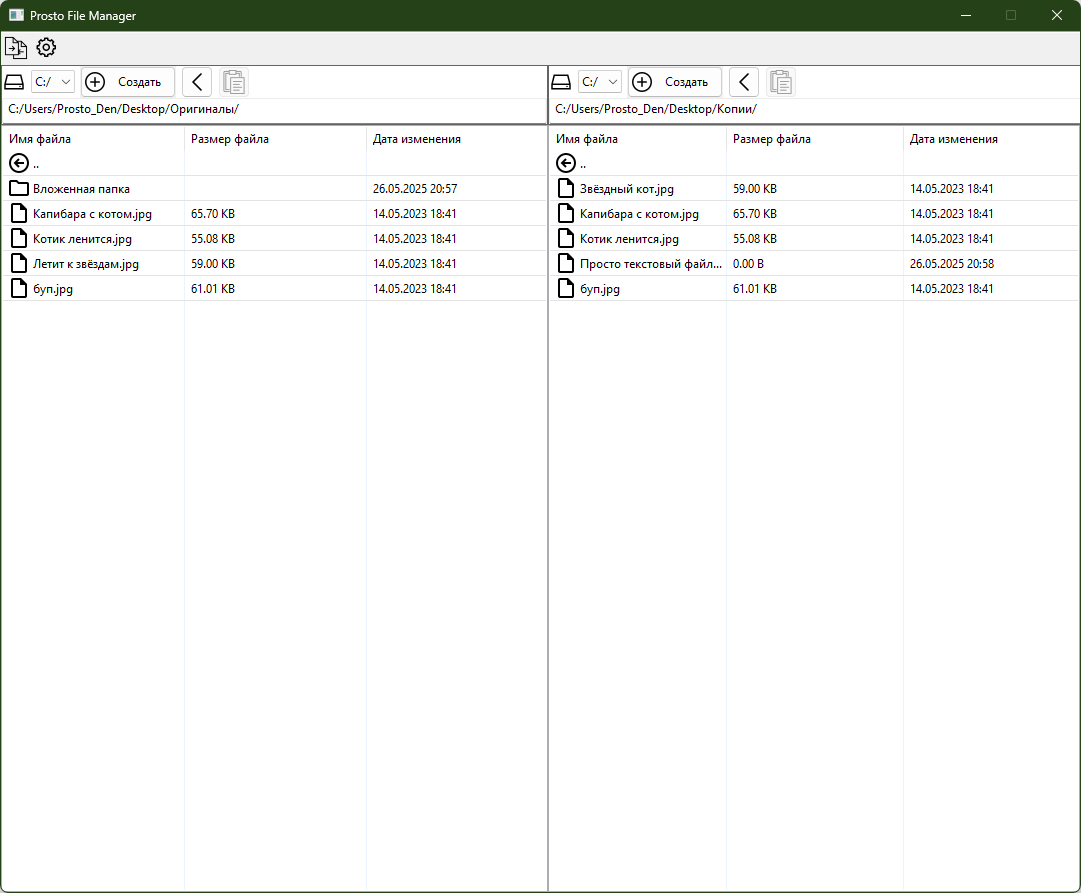


Рисунок 9.4 – Приготовление к поиску дубликатов

При нажатии на кнопку, обозначенной на рисунке 9.4 откроется окно с настройкой. Содержимое окна представлено на рисунке 9.5. Обратим внимание, что в текстовых полях автоматически были выставлены пути к открытым директориям, так что пользователю не придётся заново прописывать пути. Чтобы учесть содержимое вложенной папки, необходимо поставить галочку в поле «Рекурсивный поиск». После чего можно нажимать на кнопку «ОК».

Файлов в директориях немного и весят они мало, поэтому окно с результатом поиска (рисунок 9.6) появляется на экране почти мгновенно. Однако, если бы поиск затянулся, сначала бы пользователь увидел окно с просьбой подождать, изображённое на рисунке 7.5.

Как видно из рисунка 9.6, программа правильно определила все дубликаты. Для файла «Летит к звёздам.jpg» было найдено два дубликата, текстовый файл затронут не был, так как дубликатов он не имел, а остальные файлы имеют по одному дубликату.

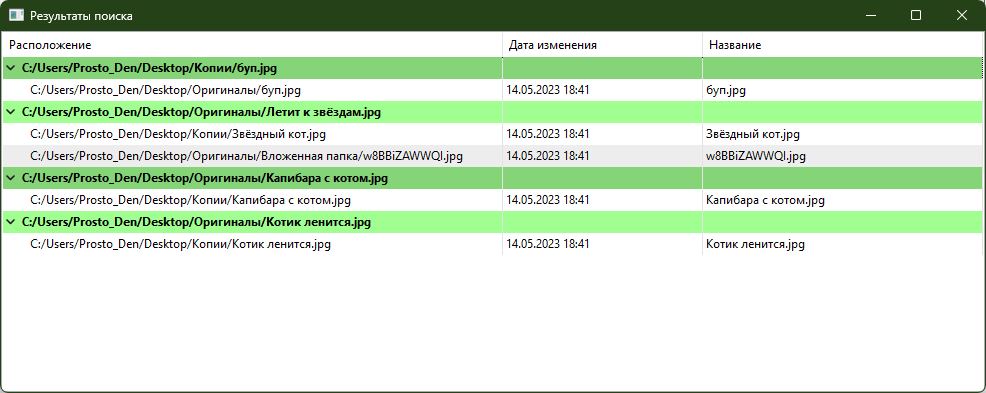


Рисунок 9.6 – Результат поиска дубликатов

# ЗАКЛЮЧЕНИЕ

В ходе выпускной квалификационной работы был разработан файловый менеджер с функцией поиска дубликатов файлов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Машечкин И.В., Петровский М.И., Скулачев П.Д., Терехин А.Н. СИСТЕМНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ - файловые системы ОС UNIX и WINDOWS NT. URL: https://iit.cs.msu.ru/media /media/educational\_materials/Системное\_программное\_обеспечение\_-\_файловые\_системы\_ОС\_Unix\_zmAwX5u.pdf (дата обращения: 03.11.2024)
2. Таненбаум Эндрю С. Современные операционные системы. 4-е изд. / С. Э. Таненбаум, Бос Херберт; пер. с англ. Н. Вильчинский, А. Леонтьева, М. Малышева. – СПб.: Питер, 2022. – 1120 с.
3. Уорд Брайан. Внутреннее устройство Linux / Брайан Уорд. – СПб.: Питер, 2020. – 448 с.
4. Официальный сайт Far Manager. URL: https://www.farmanager.com (дата обращения: 02.11.2024)
5. Официальная документация файловой системы ext4. URL: https://docs.kernel.org/filesystems/ext4/ (дата обращения 04.11.2024)
6. Официальный сайт Total Comander. URL: https://www.ghisler.com/download.htm (дата обращения: 02.11.2024)
7. Официальное описание работы алгоритма SHA-1. URL: https://datatracker.ietf.org/doc/html/rfc3174 (дата обращения 02.11.2024)
8. Семёнов Ю. А. Telecommunication & information technologies - телекоммуникационные и информационные технологии. URL: http://book.itep.ru/6/sha1.htm (дата обращения: 02.11.2024)
9. Удахина С. В. Проектирование информационных систем: учебно-методическое пособие / С. В. Удахина; С.-Петерб. гос. ун-т аэрокосм. приборостр-я. - СПб: Изд-во ГУАП, 2023. - 105 с.
10. ГОСТ 2.105-2019 «ЕСКД. Общие требования к текстовым документам». – М.: Стандартинформ, 2019. – 36 с.
11. ГОСТ 7.32-2017 «СИБИД. Отчет о научно-исследовательской работе. Структура и правила оформления». – М.: Стандартинформ, 2017. – 32 с.
12. ГОСТ 7.0.100-2018 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления» – М.: Стандартинформ, 2018. – 130 с.
13. ГОСТ Р ИСО 9241-161-2016 «Эргономика взаимодействия человек-система» - М.: Стандартинформ, 2016 – 54 с.
14. ГОСТ Р ИСО 14915-1-2016 «Эргономика мультимедийных пользовательских интерфейсов» - М.: Стандартинформ, 2016 – 14 с.

# ПРИЛОЖЕНИЕ А. Код