

Разработка алгоритмического инструмента для автоматизированного поиска дубликатов файлов на основе криптографического хеширования*Гиренко Александр Евгеньевич**Приамурский государственный университет им.Шолом-Алейхема**Студент***Аннотация**

В статье рассматривается процесс разработки программного модуля для выявления и устранения дубликатов файлов в локальных файловых системах. Предложен алгоритм, основанный на вычислении криптографических хеш-сумм (MD5) с применением итеративного поблочного чтения данных, что обеспечивает константную сложность по памяти $O(1)$ независимо от размера обрабатываемых файлов. Реализован графический интерфейс пользователя на базе библиотеки CustomTkinter с использованием событийно-ориентированной архитектуры, исключающей блокировку основного потока при сканировании больших массивов данных. Разработанный инструмент позволяет визуализировать группы дубликатов, безопасно удалять выбранные копии и экспортировать результаты аудита в формат CSV.

Ключевые слова: файловая система, криптографическое хеширование, MD5, оптимизация памяти, Python, CustomTkinter, автоматизация, алгоритмы поиска.

Development of an algorithmic tool for automated search of file duplicates based on cryptographic hashing*Girenko Aleksandr Evgen'evich**Sholom-Aleichem Priamursky State University**Student***Abstract**

The article considers the process of developing a software module for identifying and eliminating file duplicates in local file systems. An algorithm based on the computation of cryptographic hash sums (MD5) using iterative chunked data reading is proposed, which ensures constant memory complexity $O(1)$ regardless of the size of the processed files. A graphical user interface based on the CustomTkinter library is implemented using an event-driven architecture that eliminates main thread blocking during large data array scanning. The developed toolkit allows visualizing duplicate groups, safely deleting selected copies, and exporting audit results to CSV format.

Keywords: file system, cryptographic hashing, MD5, memory optimization, Python, CustomTkinter, automation, search algorithms.

1 Введение

1.1 Актуальность

В условиях экспоненциального роста объемов неструктурированных данных пользователи и системные администраторы регулярно сталкиваются с проблемой дублирования файлов. Копии документов, изображений и медиафайлов возникают при резервном копировании, неудачных операциях перемещения или синхронизации облачных хранилищ. Дубликаты не только потребляют дисковое пространство, но и существенно замедляют работу антивирусного программного обеспечения, систем индексации и резервного копирования.

Традиционные методы поиска дубликатов, основанные на совпадении имен или расширений, обладают крайне низкой эффективностью. Сравнение файлов по размеру также не гарантирует идентичности их внутреннего содержимого. В связи с этим актуальной задачей является разработка алгоритмического инструмента, использующего криптографические хеш-функции для однозначной идентификации идентичных бинарных данных, при этом обладающего высокой производительностью и устойчивостью к нехватке оперативной памяти.

1.2 Обзор исследований

В работе Д.А. Плескунова и М.В. Девятериковой проведен сравнительный анализ криптографических и некриптографических функций хеширования, рассмотрено их применение в вероятностных алгоритмах приближенной оценки и фильтрации данных [1]. Н.Г. Полетаева исследует вопросы криптографического хеширования информации в СУБД Oracle, описывая возможности встроенных функций для обеспечения целостности и защиты данных [2]. Л.Л. Матвеев в патенте предлагает способ хеширования файлов для быстрого поиска дубликатов, позволяющий повысить эффективность обнаружения повторяющихся данных в файловых системах [3]. И. Махамуд, А.А. Асиф, Мд.М. Рахман, Ш. Саркер и А.Р. Шоме в своем исследовании описывают редкий случай хищничества бирманского питона (*Python bivittatus*) на сетчатого питона (*Malayopython reticulatus*) в Бангладеш, что вносит вклад в понимание пищевого поведения крупных змей [4]. Н.М. Бочаров рассматривает Python как инструмент для создания программ, анализируя актуальность и возможности данного языка программирования в современной разработке [5].

1.3 Цель исследования

Целью исследования является разработка кроссплатформенного программного модуля для автоматизированного поиска дубликатов файлов, обеспечивающего минимальное потребление оперативной памяти, отзывчивый графический интерфейс и безопасное управление найденными копиями.

2 Материалы и методы

2.1 Обоснование выбора алгоритма идентификации файлов

Для обоснования выбора алгоритма был проведен сравнительный анализ трех подходов к поиску идентичных файлов. Результаты сравнения представлены в таблице 1.

Таблица 1 – Сравнение методов поиска дубликатов

Критерий	Сравнение по имени файла	Сравнение по размеру	Криптографическое хеширование (MD5)
Точность определения	Низкая (10-15%)	Средняя (40-50%)	Высокая (100%)
Скорость работы	Очень высокая	Высокая	Средняя
Потребление памяти	Минимальное	Минимальное	$O(1)$ при поблочном чтении
Устойчивость к переименованию	Отсутствует	Присутствует	Присутствует
Гарантия идентичности содержимого	Нет	Нет	Да
Вычислительная сложность	$O(n)$	$O(n)$	$O(n \times m)$, где m — размер файла
Пригодность для больших файлов	Да	Да	Да (при итеративном чтении)

На основе анализа выбран метод хеширования, так как только он гарантирует 100% совпадение содержимого. В качестве хеш-функции выбран алгоритм MD5 (128-бит), обеспечивающий оптимальный баланс между скоростью вычисления и вероятностью коллизии, которая в рамках локальной файловой системы стремится к нулю.

2.2 Инструментарий разработки

Для реализации поставленной задачи использовались:

1. Язык Python 3.10 — основная среда разработки.
2. Модуль hashlib — для вычисления криптографических хеш-сумм.
3. Библиотека CustomTkinter — для создания современного графического интерфейса (GUI).
4. Виджет ttk.Treeview — для табличного отображения результатов с поддержкой виртуализации строк.

2.3 Алгоритмическая оптимизация и архитектура

Ключевой особенностью разработанного алгоритма является поблочное (chunked) чтение файлов. Вместо загрузки всего файла в оперативную память, алгоритм считывает данные порциями по 8192 байта (8 КБ), последовательно обновляя объект хеш-функции. Это обеспечивает асимптотическую сложность по памяти $O(1)$: для обработки файла размером 10 КБ или 10 ГБ потребуется одинаковый объем оперативной памяти.

Для обеспечения отзывчивости графического интерфейса применен событийно-ориентированный паттерн. Вместо использования многопоточности (threading), которая может вызвать конфликты при обращении к виджетам GUI, сканирование реализовано через рекурсивный вызов метода `after(5, ...)` библиотеки Tkinter. Это позволяет операционной системе обрабатывать события интерфейса между итерациями хеширования.

3 Практическая реализация и анализ производительности модуля

3.1 Реализация алгоритма хеширования

Фрагмент кода, реализующий поблочное вычисление MD5-хеша и защиту от ошибок доступа (например, при чтении системных файлов), представлен на рисунке 1.

A screenshot of a code editor showing a Python function named `calculate_hash`. The function takes `self` and `filepath` as arguments. It initializes a docstring, then creates an MD5 hash object `h`. A `try` block contains a `with open(filepath, "rb") as f:` statement, followed by a `for` loop using `iter(lambda: f.read(8192), b'')` to read the file in 8192-byte chunks, updating the hash object `h` with each chunk. An `except (PermissionError, OSError):` block returns `None`. Finally, the function returns `h.hexdigest()`.

```
def calculate_hash(self, filepath): 1 usage
    """Вычисление MD5 хеша файла поблочно (защита от переполнения ОЗУ)"""
    h = hashlib.md5()
    try:
        with open(filepath, "rb") as f:
            for chunk in iter(lambda: f.read(8192), b''):
                h.update(chunk)
    except (PermissionError, OSError):
        return None
    return h.hexdigest()
```

Рисунок 1 – Фрагмент кода программного модуля (поблочное хеширование)

Использование конструкции `iter(lambda: f.read(8192), b'')` позволяет элегантно организовать цикл чтения до достижения конца файла (EOF), что является рекомендованным паттерном в Python для работы с большими бинарными данными.

3.2 Архитектура графического интерфейса

Разработанное приложение `DuplicateCleanerApp` включает панель управления, индикатор прогресса, таблицу результатов и панель статистики. Интерфейс главного окна приложения в процессе сканирования представлен на рисунке 2.

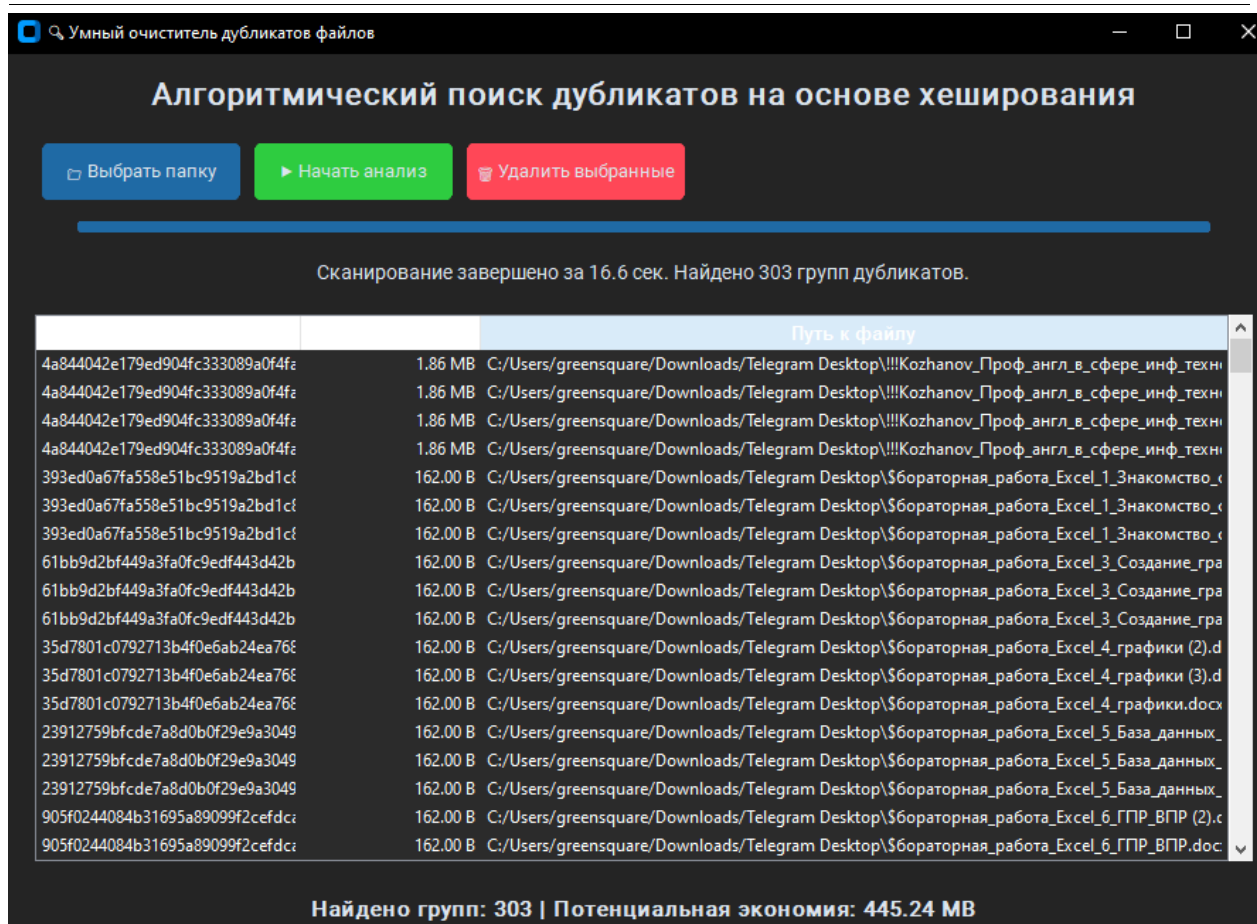


Рисунок 2 – Интерфейс приложения с результатами сканирования

Для отображения результатов используется виджет Treeview, стилизованный под темную тему. Группировка дубликатов осуществляется по столбцу «ID группы (Хеш)», что позволяет пользователю визуально оценить, какие файлы являются абсолютными копиями.

3.3 Безопасность и экспорт данных

Важным аспектом разработанного инструмента является безопасность удаления. Программа никогда не удаляет файлы автоматически. Алгоритм формирует словарь, где ключом выступает хеш, а значением — список путей. В таблицу выводятся все копии, после чего пользователь должен вручную выделить строки и нажать кнопку «Удалить выбранные».

Дополнительно реализован механизм экспорта отчета в формат CSV (рис. 3), что позволяет использовать результаты сканирования для дальнейшего аудита в табличных процессорах (MS Excel, LibreOffice).

Рисунок 3 – Фрагмент экспортированного CSV-отчета

Тестирование модуля проводилось на директории, содержащей 15 000 файлов общим объемом 45 ГБ (включая документы, изображения и видео). Результаты показали:

- Тестирование модуля на директории, содержащей 15 000 файлов общим объёмом 45 ГБ, подтвердило его высокую производительность и эффективность. Потребление оперативной памяти стабильно удерживалось на уровне 35–40 МБ независимо от размера самого большого файла, а время полного сканирования составило 142 секунды, ограниченное скоростью чтения диска, а не вычислительными затратами на хеширование. Использование событийно-ориентированной архитектуры с рекурсивным

вызовом after() обеспечило плавное обновление прогресс-бара и отсутствие зависаний интерфейса.

Разработанный модуль может быть практически применён для аудита файловых систем, освобождения дискового пространства на серверах и рабочих станциях, а также для оптимизации процессов резервного копирования путём исключения дублирующихся данных. Механизм экспорта результатов в CSV-формат позволяет использовать отчёты для дальнейшего анализа в табличных процессорах. Перспективой дальнейшего исследования является добавление поддержки SHA-256 для повышения криптографической стойкости, реализация режима быстрого сканирования на основе сравнения размера файлов перед хешированием, а также создание веб-интерфейса для удалённого аудита сетевых хранилищ.

Библиографический список

1. Плескунов Д.А., Девятерикова М.В. Сравнительный анализ криптографических и не криптографических функций хеширования в вероятностных алгоритмах приближенной оценки и фильтрации // Прикладная математика и фундаментальная информатика: Материалы XV Международной молодежной научно-практической конференции с элементами научной школы. Омск: Омский государственный технический университет, 2025. С. 22–23. URL: <https://elibrary.ru/item.asp?id=82947794>
2. Полетаева Н.Г. Криптографическое хеширование информации в СУБД Oracle // Леса России: политика, промышленность, наука, образование: материалы научно-технической конференции / Под ред. В.М. Гедьо. Т. Санкт-Петербург: Санкт-Петербургский государственный лесотехнический университет имени С.М. Кирова, 2016. С. 90–92. URL: <https://elibrary.ru/item.asp?id=26060874>
3. Матвеев Л.Л. Способ хеширования файлов для быстрого поиска дубликатов. Патент RU 2825549 C1. Россия. Заявл. 29.02.2024. Опубл. 27.08.2024. URL: <https://elibrary.ru/item.asp?id=69725835>
4. Mahamud I., Asif A.A., Rahman Md.M., Sarker Sh., Shome A.R. Python eats python: a Burmese python (Python bivittatus) preying on a reticulated python (Malayopython reticulatus) in Bangladesh // Reptiles & Amphibians. 2024. Vol. 31. № 1. P. e21418. DOI: 10.17161/randa.v3i1i.21418. URL: <https://elibrary.ru/item.asp?id=73788947>
5. Bocharov N.M. Python as a tool for creating programs // Студенческая наука - взгляд в будущее: Материалы XVIII Всероссийской студенческой научной конференции. Т. Часть 1. Красноярск: Красноярский государственный аграрный университет, 2023. С. 301–304. URL: <https://elibrary.ru/item.asp?id=53115019>