МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА ПРОГРАММЫ ПОИСКА ПОВТОРОВ В ТЕКСТЕ

КУРСОВАЯ РАБОТА

Студентки 2 курса 251 группы
специальности 09.03.04 — Программная инженерия
факультета КНиИТ
Яфаровой Эльвины Эрнестовны

Научный руководитель
д. т. н., проф. _____ А. С. Богомолов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Постановка задачи и подход к ее решению 4
1.1 Описание задачи 4
1.2 Требования и ожидания пользователей 4
1.3 Подход к решению задачи 4
2 Алгоритмы и инструменты решения 6
2.1 Средства решения задачи 6
2.2 Алгоритмы для решения 9
3 Интерфейс программы 15
4 Руководство пользователя 18
5 Результаты использования программы 24
ЗАКЛЮЧЕНИЕ 26
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27
Приложение А Нумеруемые объекты в приложении 30
Приложение Б Листинг программы 43

ВВЕДЕНИЕ

В условиях стремительного развития информационных технологий обработка и анализ текстовых данных становятся важными инструментами для повышения эффективности работы в различных областях — от редактирования и подготовки аналитических отчётов до оптимизации контента и борьбы с дублированием информации [1]. Например, такая задача может быть актуальна при сборке большой монографии из текстов различных авторов по похожим темам. В этом случае может возникнуть необходимость анализа текста, в частности, на наличие повторяющихся фрагментов.

В настоящее время существуют приложения, декларирующие возможность решения подобных задач. Так, инструмент Duplicate Finder позволяет задавать минимальную длину фрагмента и процент совпадения между блоками, что эффективно используется для дедупликации файлов и оптимизации хранения данных. Однако, эта программа не позволяет находить повторы в тексте документа [2]. Duplicate Word Finder специализируется на поиске повторяющихся слов, однако не ищет повторяющиеся фрагменты текста. Инструмент Андубликатор предназначен для поиска и удаления конкретно заданных слов или строк. Такой подход не позволяет выявлять повторяющиеся фрагменты [3].

Существуют сервисы, ориентированные на стилистический анализ. Например, сервис «Свежий взгляд» для поиска тавтологий выявляет повторяющиеся фразы, но в основном фокусируется на улучшении стиля текста и не позволяет точно настроить параметры поиска на уровне фиксированной длины фрагмента [4]. Аналогично, инструмент Подсчет одинаковых слов (PLANETCALC) проводит базовый частотный анализ, не учитывая сложные критерии выделения повторяющихся блоков текста [5].

Таким образом, анализ существующих решений показывает, что на рынке отсутствует инструмент, обеспечивающий достаточно высокую автоматизацию поиска повторов в тексте. Также отметим, что передеча текстов различным сервисам по поиску плагиата, даже в случае, если они будут способны решить обозначенную задачу, может вызывать у авторов опасения по поводу риска утечки, что делает локальные инструменты более предпочтительными. Курсовая работа направлена на разработку инструмента, который позволит проводить поиск повторяющихся текстовых блоков с расширенными возможностями настройки параметров и упорядочением результатов.

1 Постановка задачи и подход к ее решению

1.1 Описание задачи

Основными задачами курсовой работы являются: изучение имеющихся решений; уточнение требований; подбор инструментов для реализации собственного решения; разработка и тестирование программы.

1.2 Требования и ожидания пользователей

— Импорт текстовых данных: обеспечить возможность загрузки текста из файлов различных форматов (TXT, DOCX, DOC, PDF) или ввода текста непосредственно через пользовательский интерфейс.
— Настройка параметров анализа: пользователь должен иметь возможность задавать длину искомого фрагмента в символах, а также использовать опцию фильтрации по начальному символу, позволяющую выбрать поиск фрагментов, начинающихся с заглавной русской буквы, заглавной английской буквы или любого символа.
— Сортировка результатов: необходимо реализовать функцию сортировки по количеству повторений и по алфавиту.
— Копирование и экспорт данных: пользователь должен иметь возможность копировать текст результатов непосредственно из текстового блока программы, а также при желании экспортировать полученные данные в виде файла формата TXT.

1.3 Подход к решению задачи

Этапы разработки программы:

1. **Исследование требований и ожиданий пользователей**
   — Определение потребностей пользователей.
   — Формирование ключевых функциональных требований к программе.
   — Установление критериев эффективности программы.
2. **Выбор технологий и инструментов**
   — Определение стека технологий.
   — Разработка структуры программы с учетом расширяемости и производительности.
3. **Разработка алгоритма поиска повторяющихся фрагментов**
   — Исследование и сравнение различных методов.

- Реализация оптимального алгоритма с возможностью настройки параметров.

4. **Создание пользовательского интерфейса**
   - Разработка удобного интерфейса для взаимодействия с программой.
   - Реализация функций загрузки файлов, просмотра и экспорта результатов.
5. **Тестирование и анализ результатов**
   - Проверка корректности работы алгоритма на различных текстах.
   - Анализ скорости и точности поиска повторяющихся фрагментов.
   - Оценка удобства использования интерфейса.

2 Алгоритмы и инструменты решения

2.1 Средства решения задачи

В этом проекте разработка программного продукта осуществляется на языке C# в рамках WPF-проекта. Выбор C# обусловлен не только его богатой стандартной библиотекой и высокой производительностью при работе с текстовыми данными, но и широкими возможностями по созданию надёжных, масштабируемых и легко поддерживаемых приложений благодаря объектноориентированному подходу. Этот язык позволяет быстро реализовывать сложные алгоритмы обработки строк и коллекций, что критически важно для нашего функционала поиска повторяющихся фрагментов текста [6].

Платформа WPF (Windows Presentation Foundation) была выбрана для создания пользовательского интерфейса благодаря своей гибкости и мощным средствам для разработки современных, интуитивно понятных и отзывчивых интерфейсов. WPF поддерживает декларативное описание интерфейса с помощью XAML, что значительно упрощает создание сложных визуальных компонентов, таких как динамические панели, списки и элементы управления с возможностью прокрутки. Кроме того, WPF предоставляет широкие возможности для настройки внешнего вида и поведения элементов, что позволяет точно адаптировать интерфейс под требования пользователя, обеспечивая удобство загрузки, анализа и отображения больших объёмов данных [7].

Для реализации операций открытия и сохранения файлов в различных форматах используются классы из пространства имен Microsoft.Win32, такие как OpenFileDialog и SaveFileDialog. Эти стандартные инструменты обеспечивают надёжное взаимодействие с файловой системой, позволяя пользователю легко загружать исходные документы и сохранять результаты анализа [8].

Чтение текстовых данных из файлов формата TXT в проекте осуществляется с помощью метода File.ReadAllText из пространства имён System.IO. Этот метод позволяет загружать весь текстовый контент файла в виде строки, обеспечивая простой и удобный доступ к данным [9].

При работе с документами формата DOCX используется библиотека DocumentFormat.OpenXml, которая позволяет извлекать текст непосредственно из файлов без необходимости установки офисного пакета. Однако для поддержки устаревшего формата DOC данное решение оказывается неэффективным. В связи с этим в проект интегрирована библиотека Aspose.Words [10].

В ходе разработки были также рассмотрены альтернативные решения для работы с документами формата DOC. Одним из них является NPOI — .NET-порт библиотеки Apache POI, включающий модуль HWPF, предназначенный для обработки старых форматов Word. Однако его функциональность ограничена: библиотека не поддерживает многие сложные элементы форматирования, такие как встроенные таблицы, графические объекты и сложные стили разметки, что может приводить к некорректному или частичному извлечению данных [11]. Другим вариантом является Microsoft.Office.Interop.Word — COM-интероп, предоставляющий широкий набор возможностей для работы с Word-документами, но требующий установленного офисного пакета. Это не только усложняет развертывание приложения, но и снижает производительность, особенно при обработке большого количества документов в многопоточной среде, создавая потенциальные проблемы совместимости между разными версиями Microsoft Office [12].

В сравнении с этими решениями, Aspose.Words демонстрирует наибольшую гибкость и удобство. Эта библиотека обеспечивает стабильную работу с документами без необходимости установки стороннего ПО, а также поддерживает широкий спектр форматов, включая как современные, так и устаревшие версии Word. Высокая точность извлечения данных, корректная обработка сложных элементов форматирования и удобный API делают Aspose.Words оптимальным выбором для реализации задач проекта [13].

Для обработки PDF-документов в проекте был выбран пакет iText7, который предоставляет мощные инструменты для работы с файлами данного формата. В его состав входят классы PdfReader, PdfDocument и PdfTextExtractor, обеспечивающие удобное и эффективное извлечение текстовой информации даже из сложных, многостраничных документов [14].

Основным преимуществом PdfReader является его способность корректно открывать PDF-файлы различной структуры, включая защищённые и зашифрованные документы. PdfDocument предоставляет гибкий интерфейс для работы с содержимым файла, позволяя не только считывать текст, но и анализировать структуру документа, включая обработку шрифтов, метаданных и закладок. Ключевую роль в извлечении текста играет PdfTextExtractor, который способен распознавать текст даже в случаях, когда PDF-документ содержит сложные элементы разметки, такие как многоуровневые таблицы, колонки или нестандартное

позиционирование текста [14].

Выбор iText7 обусловлен его высокой надёжностью и широкой функциональностью. В отличие от некоторых альтернатив, библиотека предоставляет детальный контроль над процессом обработки PDF, поддерживает разнообразные способы извлечения данных и адаптирована для работы в многопоточной среде. Кроме того, она не требует установки дополнительных компонентов или стороннего ПО, что делает её удобной для интеграции в различные приложения [14].

Для выполнения операций по фильтрации, сортировке и обработке коллекций повторяющихся фрагментов применяется LINQ (System.Linq), что позволяет легко группировать, упорядочивать и фильтровать данные для быстрого анализа и представления результатов [15].

Важную роль в корректном выделении текстовых фрагментов играют встроенные методы анализа символов, которые позволяют точно определять границы слов и структурировать текст. Метод Char.IsUpper используется для определения, начинается ли фрагмент с заглавной буквы, что актуально при отборе текстовых элементов, соответствующих заданным критериям (например, слов, начинающихся с заглавной буквы латинского или кириллического алфавита) [16]. В свою очередь, метод Char.IsWhiteSpace позволяет идентифицировать пробельные символы, что необходимо для точного разделения текста на логические единицы. При выборе алгоритмов для поиска повторяющихся фрагментов текста было рассмотрено несколько подходов [17].

Первый вариант — алгоритм полного перебора (Brute Force) — предполагает последовательное сравнение всех возможных подстрок заданной длины. Хотя этот метод отличается простотой реализации, его основное ограничение заключается в низкой производительности при работе с большими объемами данных, что делает его непрактичным для использования в реальных условиях [18].

Второй подход основан на использовании хеш-функций, таких как алгоритм Рабина-Карпа. Этот метод позволяет эффективно находить повторяющиеся фрагменты за счёт вычисления хешей для каждой подстроки, что значительно сокращает количество прямых сравнений. Однако, несмотря на свою эффективность, алгоритм Рабина-Карпа сталкивается со сложностями при обработке фрагментов произвольной длины, а также требует дополнительного механизма для обработки возможных коллизий хеш-функций.

Третий и наиболее эффективный с точки зрения производительности

подход основан на использовании суффиксных массивов в сочетании с вычислением наибольшего общего префикса (LCP). Суффиксный массив структурирует все суффиксы исходного текста, а LCP позволяет быстро определить длину общих префиксов между соседними элементами этого массива, что значительно ускоряет процесс поиска повторяющихся последовательностей. Несмотря на высокую эффективность этого метода, его реализация требует значительных вычислительных ресурсов и продуманного управления памятью [19].

В итоге был выбран гибридный подход, который объединяет преимущества методов на основе хеширования и принципов, используемых в суффиксных массивах. Такой выбор позволяет достичь оптимального баланса между производительностью и гибкостью настройки параметров поиска, что соответствует поставленным функциональным требованиям и позволяет эффективно обрабатывать большие объёмы текстовых данных с высокой точностью.

Таким образом, выбор инструментов, библиотек и алгоритмов для реализации проекта был основан на тщательном анализе требований, сравнении альтернативных решений и проведении тестирования. Итоговый технологический стек позволяет создать масштабируемое, надёжное и удобное в эксплуатации программное обеспечение, способное эффективно работать с разнообразными форматами текстовых данных и удовлетворять запросы пользователей по детальному анализу повторяющихся фрагментов.

2.2 Алгоритмы для решения

Алгоритм загрузки файла, представленный на рисунке 1, начинается с вызова диалогового окна, позволяющего пользователю выбрать файл F. Далее определяется его расширение Ext. Поддерживаются форматы TXT, DOCX, DOC и PDF; если расширение не входит в этот список, выводится ошибка. Затем вызывается функция загрузки файла (см. Рисунок 2), где текст извлекается в зависимости от формата: для TXT — считывается весь файл, для DOCX/DOC — открывается документ и извлекается текст, для PDF — вызывается алгоритм извлечения текста (см. Рисунок 3). Перед началом обработки текстовая переменная T и координаты базовой линии L инициализируются как пустые. Пороговое значение δ устанавливается равным 3, что используется для определения разрыва строки. Затем документ обрабатывается поэлементно: если текущий элемент является текстовым блоком, извлекается его содержимое и координаты. Далее проверяется расстояние до предыдущего блока, и если оно превышает уста-

новленное пороговое значение, перед добавлением текста вставляется разрыв строки. В конце собранный текст передаётся в интерфейс. Если загрузка прошла успешна, имя файла отображается в интерфейсе, иначе пользователю выводится сообщение об ошибке, после чего алгоритм завершается [19].

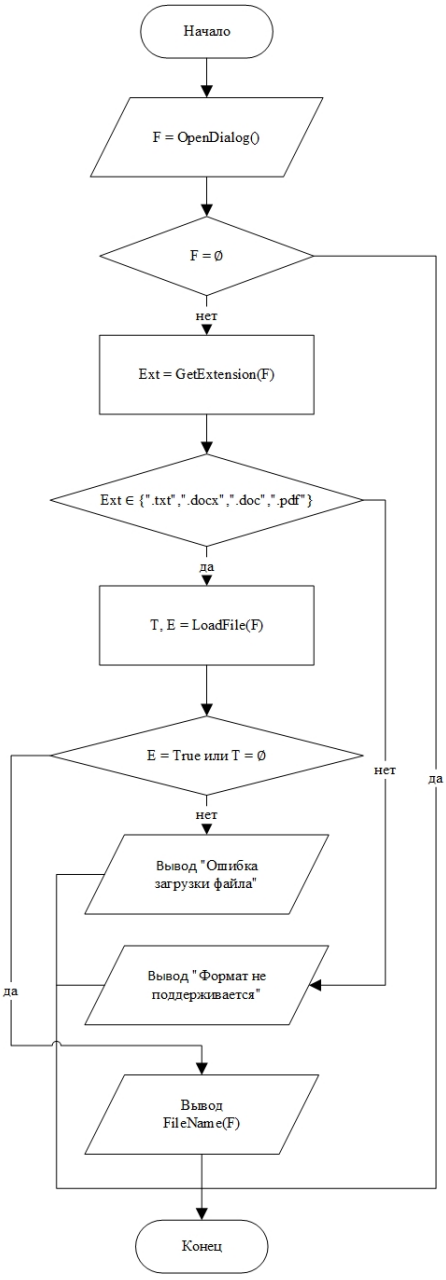

Рисунок 1 – Схема алгоритма загрузки файла

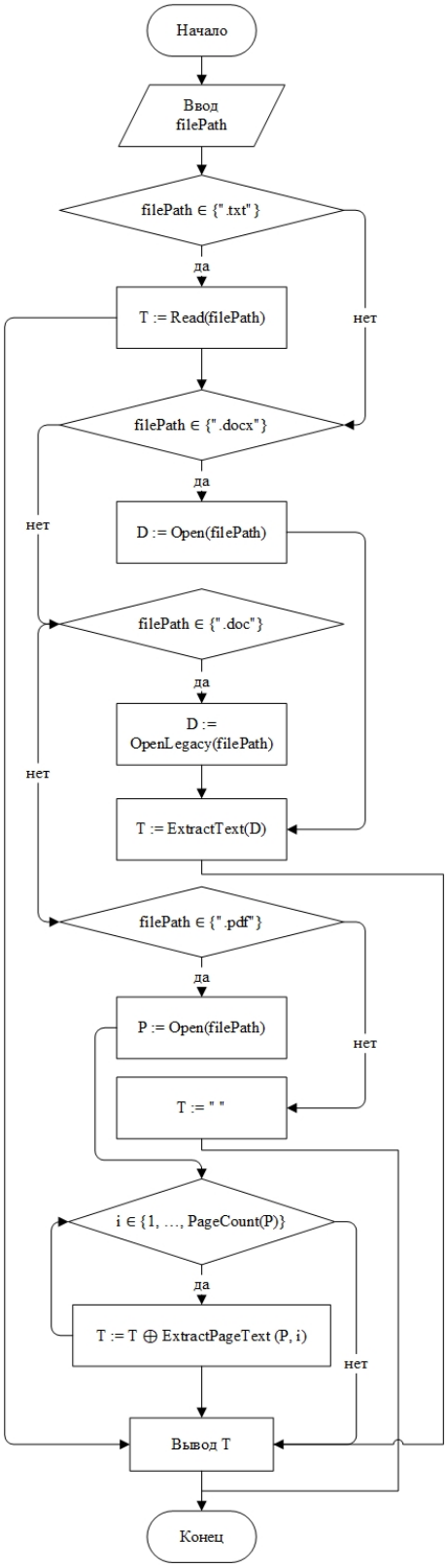

Рисунок 2 – Схема функции загрузки файла

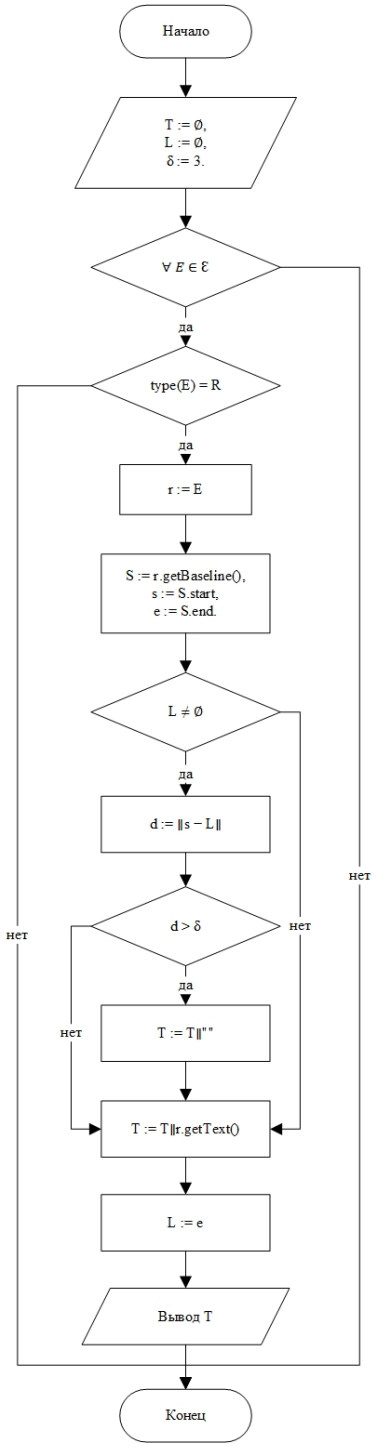

Рисунок 3 – Схема алгоритма извлечения текста из PDF

Алгоритм поиска повторяющихся фрагментов (см. Рисунок 4) начинается с проверки, был ли загружен текст (loadedText). Если текст отсутствует, пользователю выводится сообщение "Сначала загрузите файл". Далее выводится длина фрамгента n, и если она некорректна ($n \leq 0$), программа сообщает "Введите корректную длину фрагмента". После этого начинается перебор всех возможных фрагментов длины n в тексте s (общая длина L). Фрагмент f проверяется на корректность начала: он должен либо начинаться с начала строки, либо следовать

после пробела, а его первый символ фрагмента должен быть либо заглавной буквой кириллицы, либо заглавной буквой латиницы, либо любым буквенным символом. Если фрагмент соответствует условиям, он добавляется в словарь, где фиксируются повторы. После завершения перебора фрагменты, количество которых больше одного, сортируются сначала по убыванию частоты их появления, а затем по алфавиту, и выводятся в txtResult. [20].

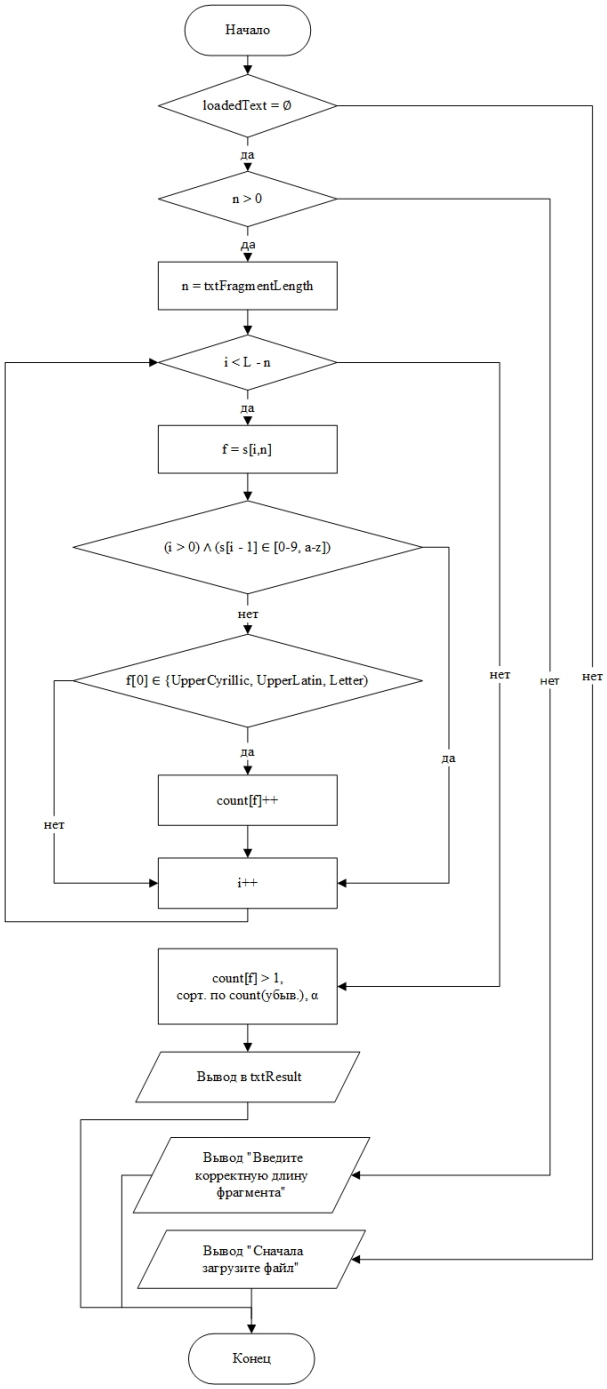

Рисунок 4 – Схема алгоритма поиска повторяющихся фрагментов

Алгоритм сохранения результатов (см. Рисунок 5) проверяет наличие текста T в поле ввода. Если $T \neq \emptyset$, вызывается диалоговое окно, и пользователь

выбирает путь F. Если данных нет, выводится сообщение об их отсутствии. После подтверждения выбора T записывается в файл F с расширением TXT. [20].

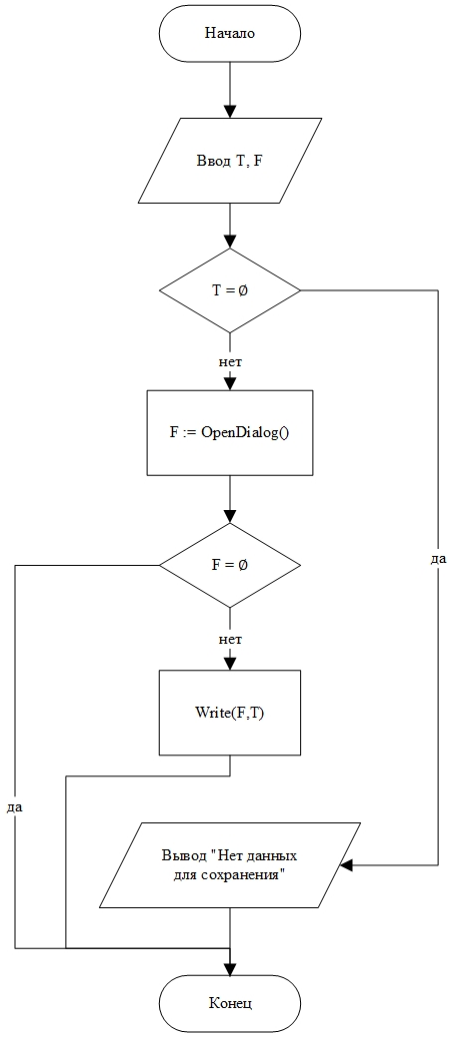

Рисунок 5 – Схема алгоритма сохранения результатов

3 Интерфейс программы

В разработанном оконном приложении размещены следующие элементы управления: три кнопки Button, три метки Label, четрые текстовых поля TextBox, одно из которых имеет ToolTip, и три переключателя RadioButton. Для удобства работы с текстом в окне отображения результата предусмотрены полосы прокрутки VerticalScrollBarVisibility и HorizontalScrollBarVisibility, обеспечивающие прокрутку содержимого при выходе за границы текстового поля. Внешний вид пользовательского интерфейса представлен на рисунке 6 [7].

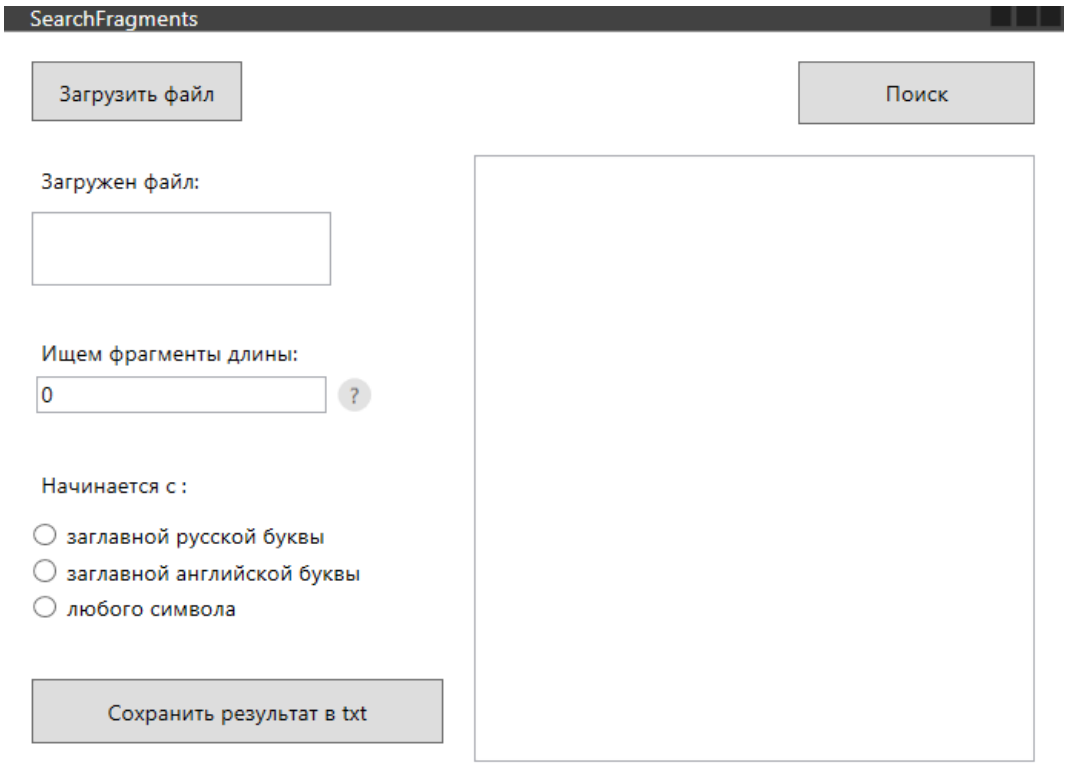

Рисунок 6 – Окно приложения «SearchFragments» открытое в конструкторе

У элементов изменены значения некоторых атрибутов. Значения измененных атрибутов представлены в таблице 1.

Таблица 1 – Значения атрибутов элементов в приложении «SearchFragments»

Наименование атрибута	Значение
Для формы	
Title	SearchFragments
Для первой кнопки	
(Name)	btnLoadFile
Content	Загружен файл
Для второй кнопки	
(Name)	btnSearch
Content	Поиск
Для третьей кнопки	
(Name)	btnSaveToFile
Content	Сохранить результат в txt
Для первой надписи	
(Name)	lblFileName
Content	Название файла
Для второй надписи	
(Name)	lblFragmentLength
Content	Длина искомого фрагмента
Для третьей надписи	
(Name)	lblSymbol
Content	Начинается с
Для первого текстового поля	
(Name)	txtFileName
Для второго текстового поля	
(Name)	txtFragmentLength
Text	0
Для третьего текстового поля	
(Name)	txtResult
Для четвертого текстового поля	
(Name)	?
(ToolTip)	Длина фрагмента учитывает пробелы
Для первой кнопки-переключателя	
(Name)	rbUpperCaseCyrillic
Content	заглавной русской буквы
Для второй кнопки-переключателя	
(Name)	rbUpperCaseLatin
Content	заглавной английской буквы
Для третьей кнопки-переключателя	
(Name)	rbAnySymbol
Content	любого символа

При нажатии кнопки «Загрузить файл» выполняется код (см. листинг, строки 96–136). В зависимости от формата загруженного файла его содержимое обрабатывается, после чего извлекается текст для дальнейшего поиска (см. листинг, строки 136–178).

Нажатие кнопки «Поиск» запускает выполнение кода, отвечающего за

поиск по извлечённому тексту (см. листинг, строки 182–295).

При нажатии кнопки «Сохранить результат в txt» предусмотрено выполнение соответствующего кода (см. листинг, строки 312–332).

4 Руководство пользователя

Перед установкой необходимо распаковать архив с дистрибутивом, содержащим исполняемые файлы, библиотеки и документацию. В распакованной директории «Программа» следует найти установочный файл setup.exe и запустить его двойным щелчком мыши. После запуска установщика необходимо следовать его пошаговым инструкциям. По завершении установки на рабочем столе и в меню «Пуск» появится ярлык установленного приложения [6].

Чтобы запустить программу, достаточно дважды кликнуть по соответствующему ярлыку или выбрать её в меню «Пуск». Сразу после этого на экране появляется главное окно приложения, в котором и будет происходить дальнейшая работа (см. Рисунок 7).

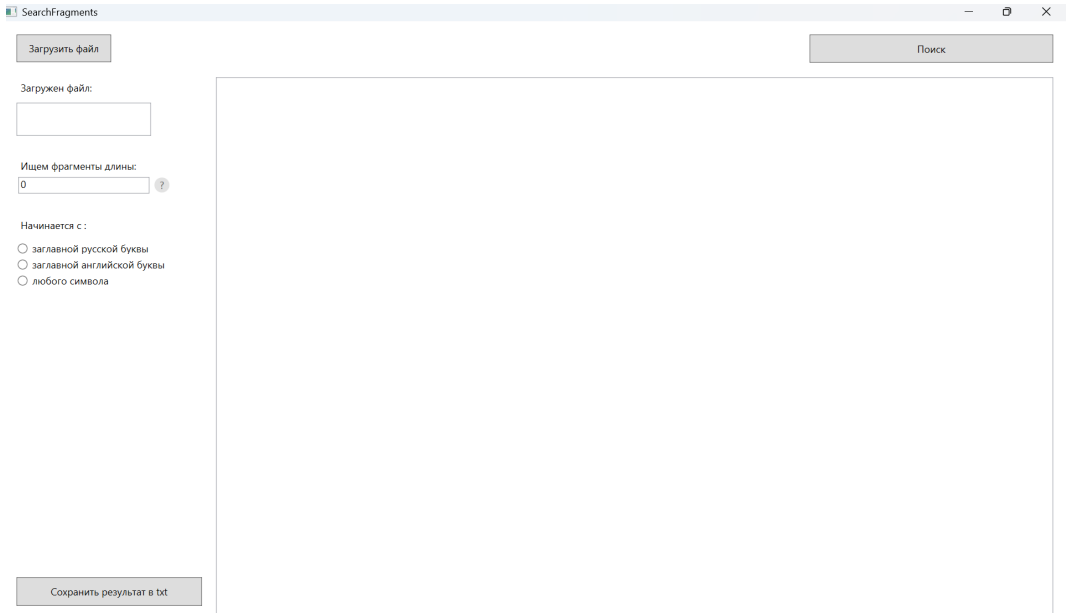

Рисунок 7 – Окно приложения «SearchFragments»: начальный запуск

Основное окно приложения содержит кнопку «Загрузить файл», при нажатии на которую открывается диалоговое окно выбора документа. Пользователь может выбрать файлы форматов TXT, DOCX, DOC или PDF. При корректной загрузке имя файла отобразится в соответствующем поле ввода (см. Рисунок 8) [21].

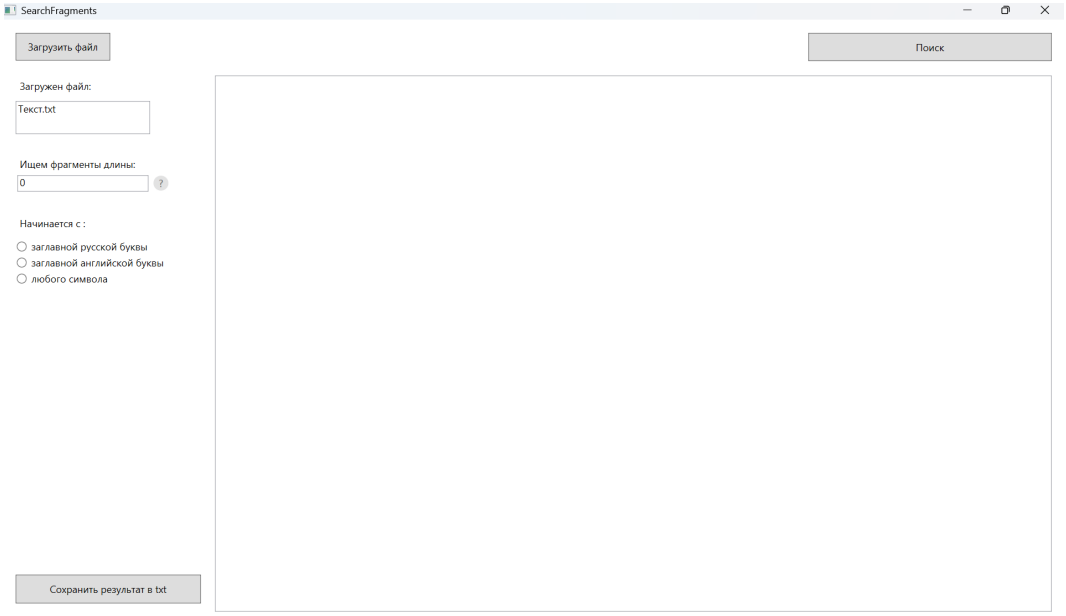

Рисунок 8 – Окно приложения «SearchFragments»: загрузка файла

Если же пользователь попытается открыть неподдерживаемый формат, программа отобразит сообщение об ошибке (см. Рисунок 9).

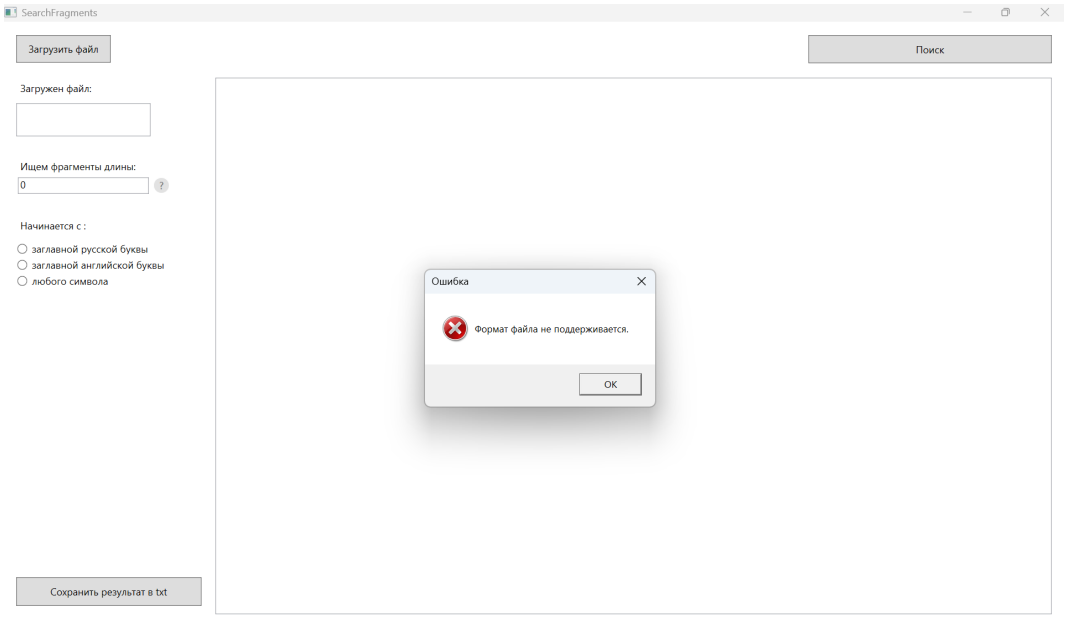

Рисунок 9 – Окно приложения «SearchFragments»: сообщение о неподдерживаемом формате загруженного файла

После успешной загрузки файла в поле «Длина фрагмента» указывается требуемое количество символов для анализа. В интерфейсе есть значок с вопросом, при наведении на который появляется пояснение: длина фрагмента учитывает пробелы (см. Рисунок 10).

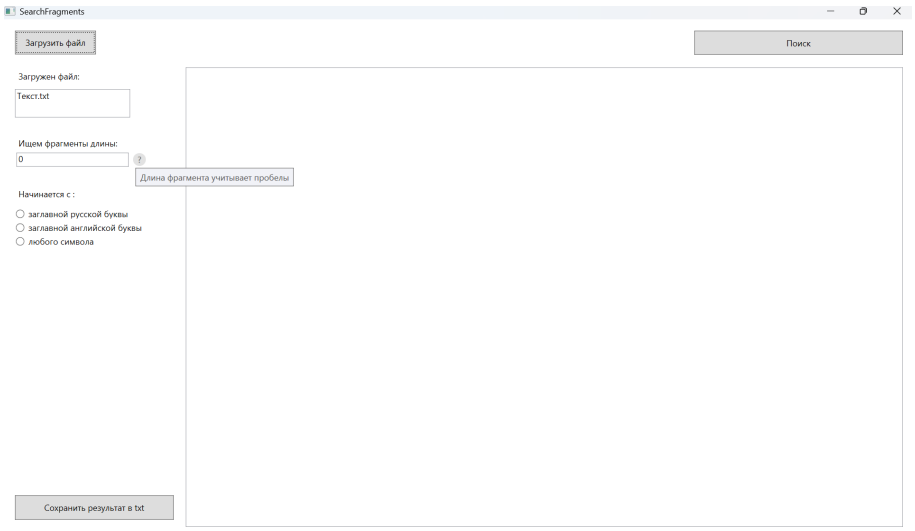

Рисунок 10 – Окно приложения «SearchFragments»: всплывающее пояснение о длине фрагмента

Далее выбирается, должен ли первый символ фрагмента быть заглавной русской буквой, заглавной латинской буквой или любым символом. Затем по нажатию кнопки «Поиск» приложение анализирует содержимое загруженного файла и отображает в поле «Результаты» перечень всех обнаруженных повторяющихся фрагментов, а также частоту их встречаемости.

Например, если указано, что первый символ должен быть заглавной русской буквой, результат анализа будет соответствующим образом отфильтрован (см. Рисунок 11).

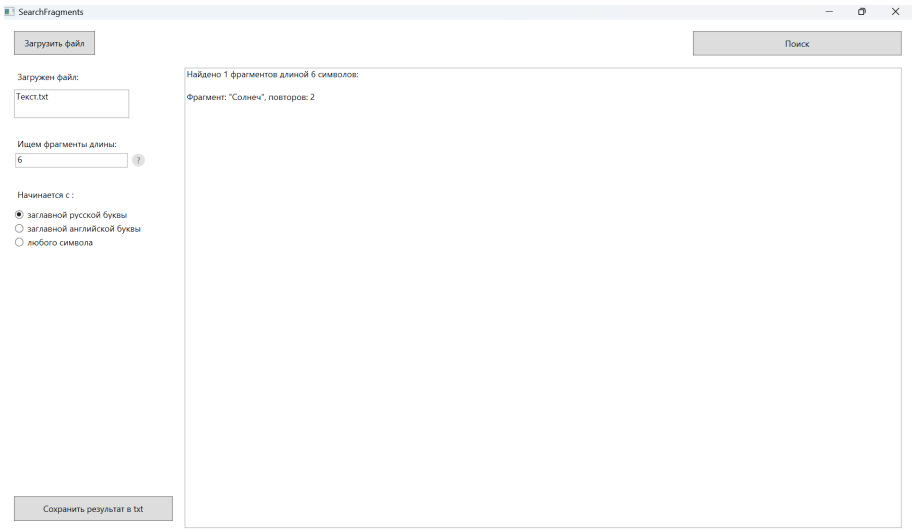

Рисунок 11 – Окно приложения «SearchFragments»: результат, если первый символ фрагмента начинается с заглавной русской буквы

Если же выбрана заглавная английская буква, то результат будет иным (см. Рисунок 12 [21].

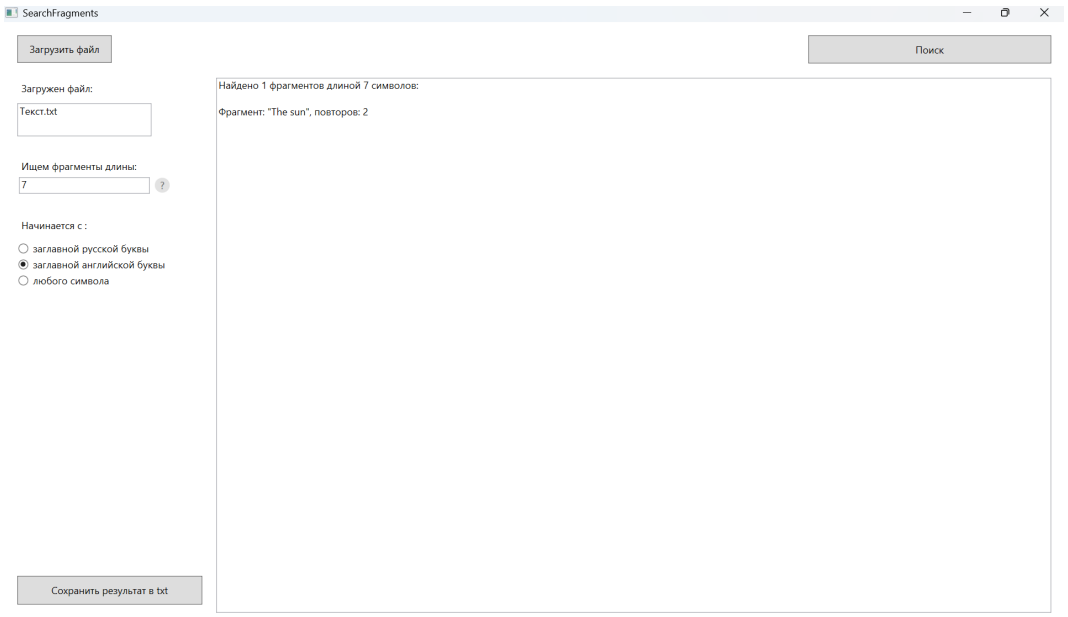

Рисунок 12 – Окно приложения «SearchFragments»: результат, если первый символ фрагмента начинается с заглавной английской буквы

В случае, когда поиск ведётся по любому первому символу, выводятся все возможные совпадения (см. Рисунок 13) [22].

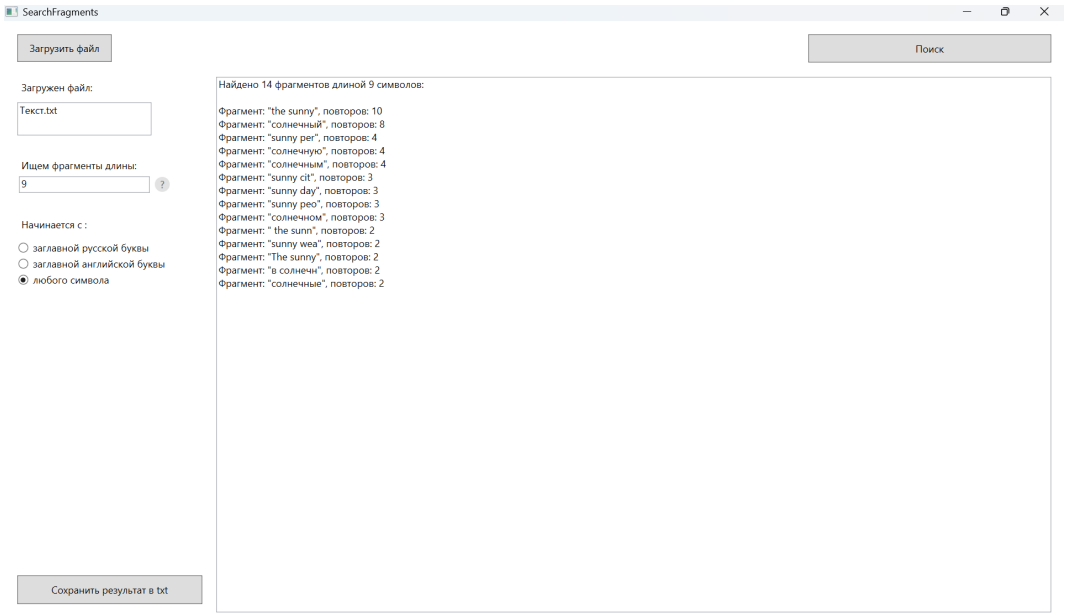

Рисунок 13 – Окно приложения «SearchFragments»: результат, если первый символ фрагмента является любым

Если в тексте нет повторений, появляется уведомление «Найдено 0 фрагментов N-ого количества символов» (см. Рисунок 14).

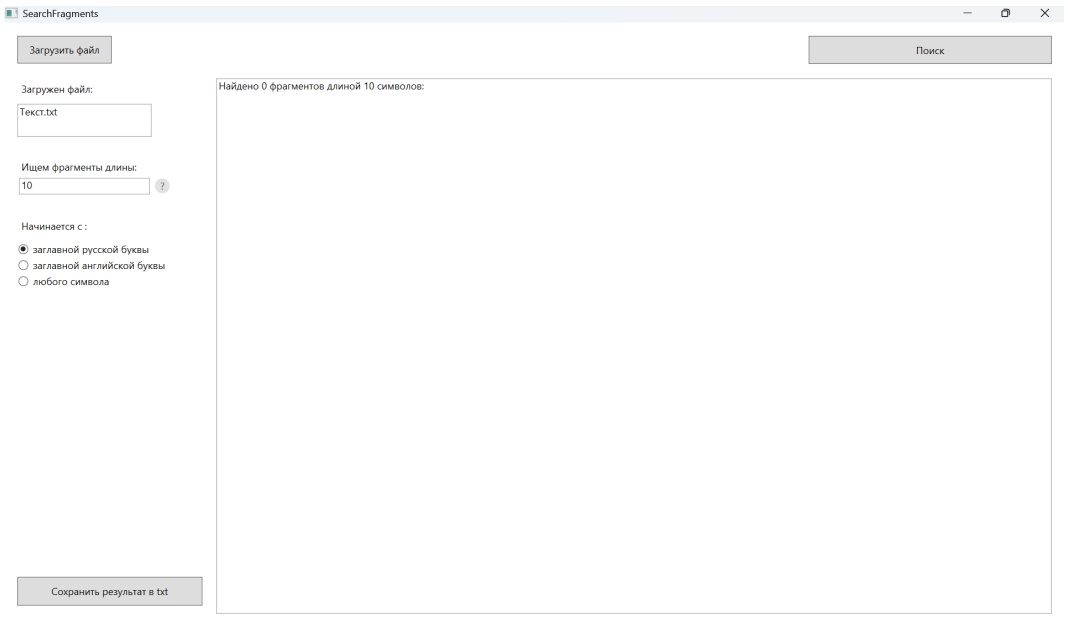

Рисунок 14 – Окно приложения «SearchFragments»: результат, когда повторяющихся фрагментов не найдено

Если пользователь нажмёт кнопку «Поиск», не загрузив при этом никакого файла, программа отобразит уведомление о необходимости предварительно выбрать документ (см. Рисунок 15).

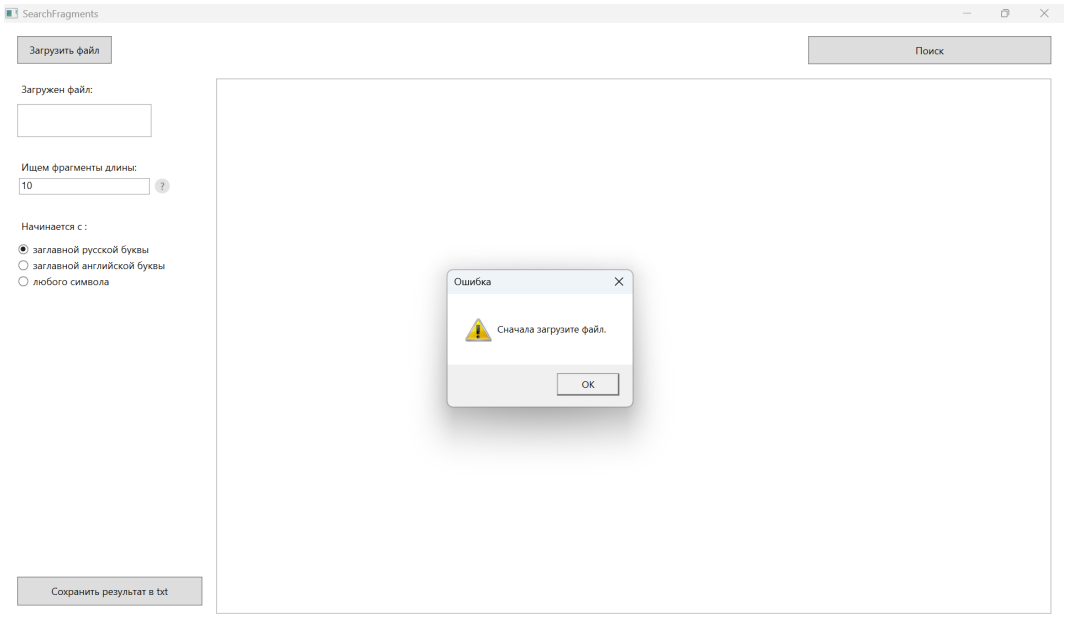

Рисунок 15 – Окно приложения «SearchFragments»: сообщение о просьбе пользователя перед работой загрузить файл

Аналогично, если длина фрагмента не задана или указано некорректное значение, отобразится предупреждение о необходимости ввести число, большее нуля (см. Рисунок 16) [22].

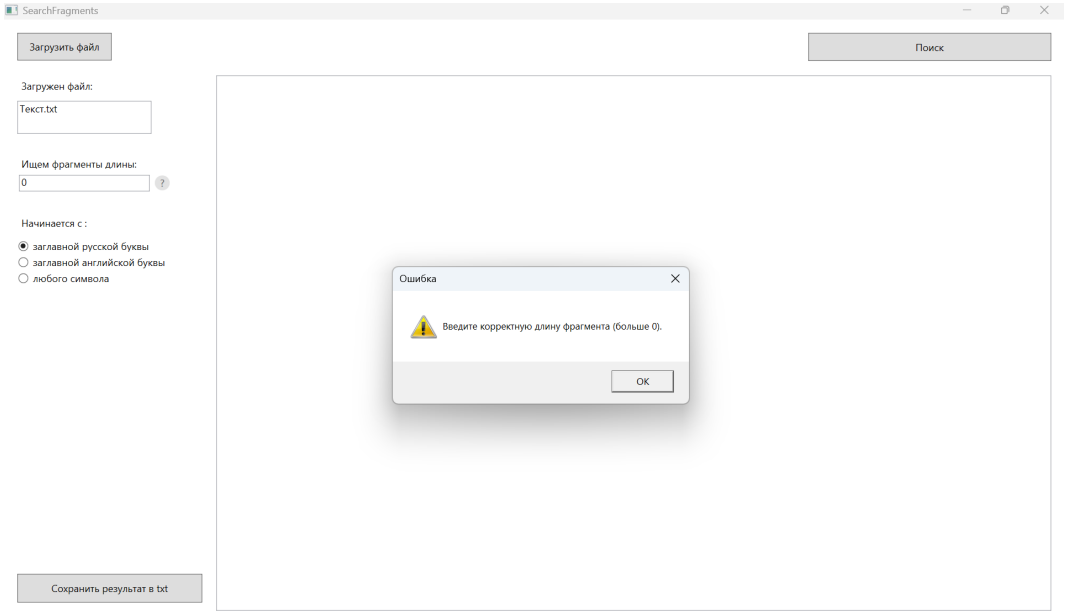

Рисунок 16 – Окно приложения «SearchFragments»: сообщение о просьбе пользователя перед работой ввести длину фрагмента больше нуля

В ситуациях, когда результат поиска оказывается слишком объёмным и не умещается в видимой области окна, предусмотрена полоса прокрутки ScrollViewer для просмотра всего списка (см. Рисунок 17) [22].

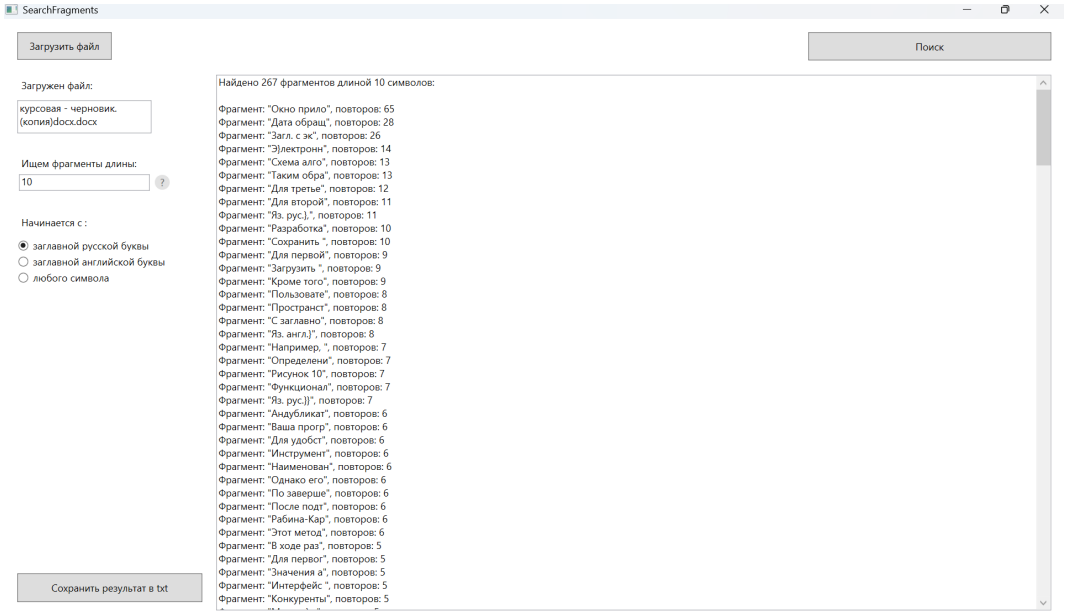

Рисунок 17 – Окно приложения «SearchFragments»: появление полосы прокрутки для просмотра всего списка

Завершив анализ, пользователь может сохранить полученные данные, нажав кнопку «Сохранить результаты». Появится диалоговое окно, где следует указать имя выходного файла (по умолчанию предлагается SearchResults.txt) и выбрать место сохранения.

5 Результаты использования программы

В ходе разработки программного обеспечения были проведены тесты на различных наборах входных данных. Это позволило выявить и устранить ошибки, а также определить зависимость времени выполнения от объёма текста. Для наглядного представления результатов был построен график (см. Рисунок 18), демонстрирующий зависимость времени выполнения поиска от объёма текста. По оси X указано количество страниц, по оси Y — время выполнения (в секундах) [20].

Данные, полученные в ходе тестирования, представлены в таблице 2.

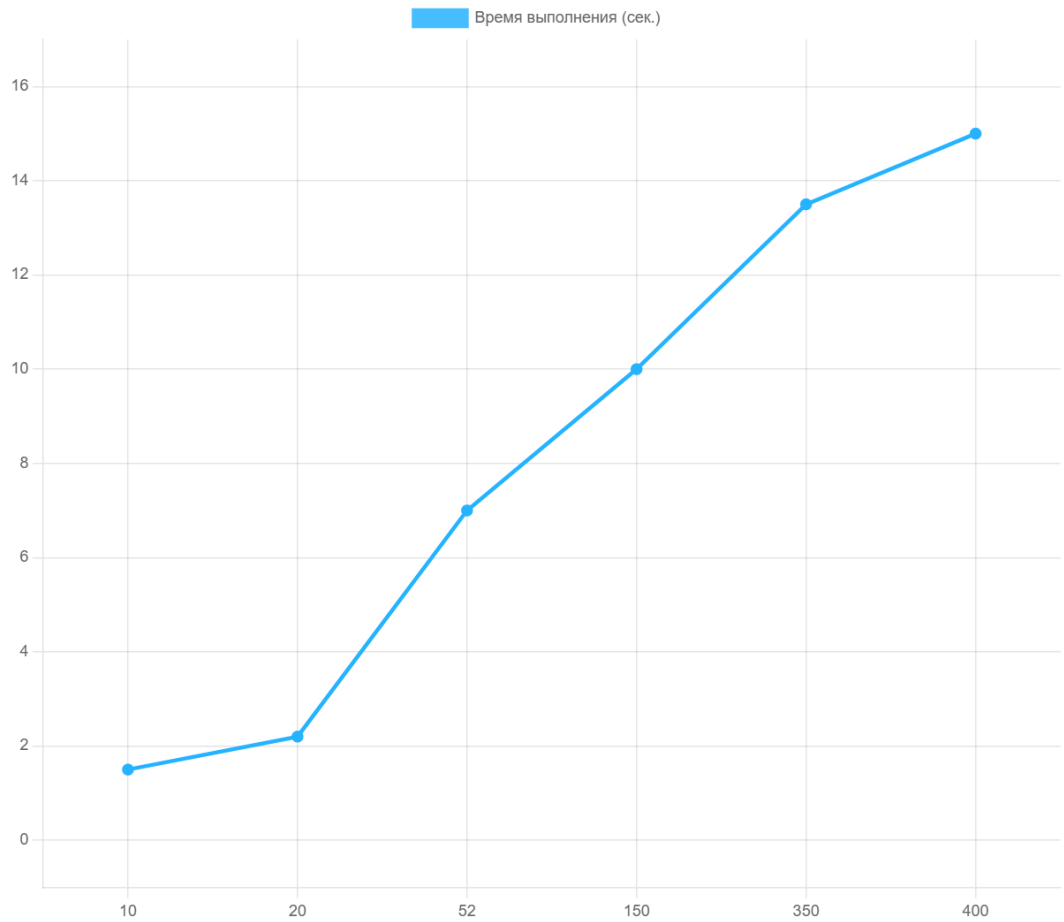

Рисунок 18 – График характеристик тестирования ПО

Таблица 2 – Таблица экспериментальных данных

Количество страниц	Время обработки (сек.)
10-20 страниц	2.2
40-70 страниц	7
350-400 страниц	13.5

Анализ результатов показывает, что время обработки зависит от объёма

текста: для больших (более 350 страниц) оно составляет 13,5 секунды, для средних (40–70 страниц) — 7 секунд, а для малых (10–20 страниц) — 2,2 секунды. При этом анализ масштабируемости алгоритма подтверждает его стабильную производительность при увеличении объёма текста, что свидетельствует о высокой эффективности и возможности обработки больших данных с минимальными затратами ресурсов.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был создан инструмент поиска повторяющихся фрагментов текста с заданными параметрами. Реализована загрузка файлов различных форматов, применение фильтров, сужающих поиск, и выгрузка результатов в удобном виде. Время обработки данных не превышает 14 секунд для книг обычных размеров (до 500 стр.) с распознаваемым текстовым слоем.

Практическая значимость разработанного программного обеспечения подтверждена его использованием при работе над совместной монографией сотрудников СГУ и ИПУ им. В. А. Трапезникова РАН объемом более 350 страниц.

Дальнейшее развитие программы может включать введение новых критериев поиска, поддержку семантического анализа текстов и реализацию локальной проверки новых документов на повторы текста.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Маликов, А. В.* Алгоритм обнаружения фактов дублирования информации в документированных результатах самостоятельной учебной деятельности студентов, устойчивый к незначительным изменениям текста / А. В. Маликов, А. С. Целиковский // *Известия вузов. Северо-Кавказский регион. Серия: Технические науки*. — 2011. — № 4. — С. 40–42.

2 Официальный сайт Duplicate Finder [Электронный ресурс]. — URL: https://flounder.dev/ru/duplicate-finder/ (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

3 Официальный сайт Duplicate Word Finder [Электронный ресурс]. — URL: https://duplicatewordfinder.com/ (Дата обращения 23.01.2025). Загл. с экр. Яз. англ.

4 Официальный сайт Свежий взгляд [Электронный ресурс]. — URL: https://petr-panda.ru/servis-tavtalogij/ (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

5 Официальный сайт PLANETCALC [Электронный ресурс]. — URL: https://planetcalc.ru/3205/ (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

6 *Рихтер, Д.* CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C# / Д. Рихтер. — 4-е изд. изд. — СПб.: Питер, 2013. — С. 896.

7 *Мак-Дональд, М.* WPF: Windows Presentation Foundation в .NET 4.5 с примерами на C# 5.0 для профессионалов / М. Мак-Дональд. — 4-е издание изд. — Москва: Вильямс, 2013. — С. 1024.

8 Microsoft.Win32 Пространство имен [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/dotnet/api/microsoft.win32?view=net-8.0 (Дата обращения 23.01.2025). Загл. с экр. Яз. англ.

9 System.IO Пространство имен [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.io?view=net-8.0 (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

10 Пользовательские функции пакета SDK [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/office/open-xml/general/features (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

11 POI-HWPF - A Quick Guide [Электронный ресурс]. — URL: https://poi.apache.org/components/document/quick-guide (Дата обращения 23.01.2025). Загл. с экр. Яз. англ.

12 Как получить доступ к объектам взаимодействия Office [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/advanced-topics/interop/how-to-access-office-interop-objects (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

13 Supported Features [Электронный ресурс]. — URL: https://docs.aspose.com/words/net/features/ (Дата обращения 23.01.2025). Загл. с экр. Яз. англ.

14 iText: Jump-Start Tutorial for .NET [Электронный ресурс]. — URL: https://kb.itextpdf.com/itext/itext-jump-start-tutorial-for-net (Дата обращения 23.01.2025). Загл. с экр. Яз. англ.

15 System.Linq Пространство имен [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.linq?view=net-9.0 (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

16 Char.IsUpper Метод [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.char.isupper?view=net-7.0 (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

17 Char.IsWhiteSpace Метод [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/dotnet/api/system.char.iswhitespace?view=net-7.0 (Дата обращения 23.01.2025). Загл. с экр. Яз. рус.

18 *Игнатов, Д. И.* Разработка и апробация системы поиска дубликатов в текстах проектной документации / Д. И. Игнатов, С. О. Кузнецов, В. Б. Лопатникова, И. А. Селицкий // *Бизнес-информатика*. — 2008. — № 4. — С. 21–28.

19 *Смит, Б.* Методы и алгоритмы вычислений на строках / Б. Смит. — Москва: ООО “И.Д. Вильямс”, 2006. — С. 496.

20 *Кудрина, Е. В.* Основы алгоритмизации и программирования на языке C# / Е. В. Кудрина, М. В. Огнева. — Профессиональное образование. Москва: Издательство Юрайт, 2025. — С. 322.

21 *Скит, Д.* C# in Depth / Д. Скит, Э. Липперт. — 4-е издание edition. — Shelter Island: Manning Publications Co., 2019. — P. 528.

22 *Павловская, Т. А.* C#. Программирование на языке высокого уровня. Учебник для вузов / Т. А. Павловская. — СПб.: Питер Пресс, 2009. — С. 432.

ПРИЛОЖЕНИЕ А
Нумеруемые объекты в приложении

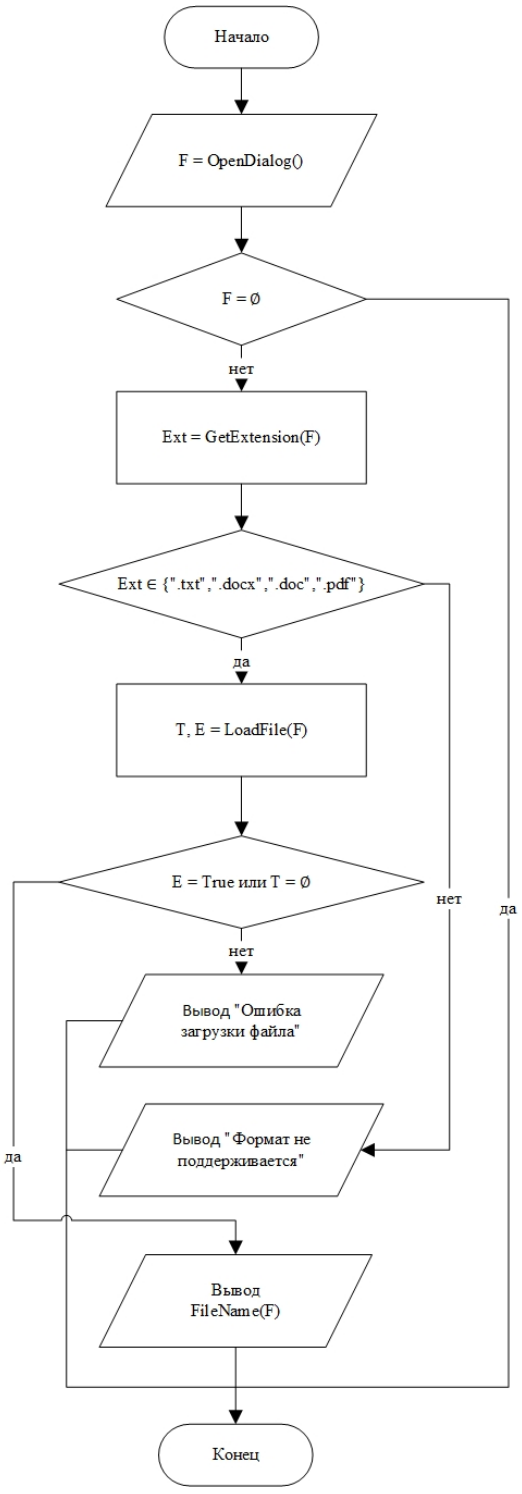

Рисунок 1 – Схема алгоритма загрузки файла

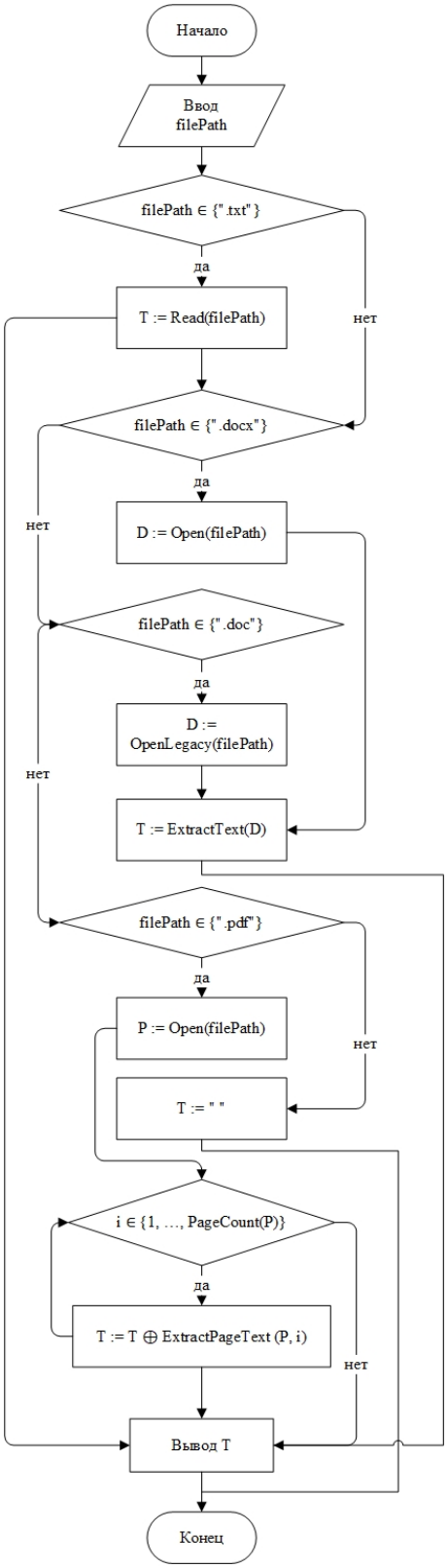

Рисунок 2 – Схема функции загрузки файла

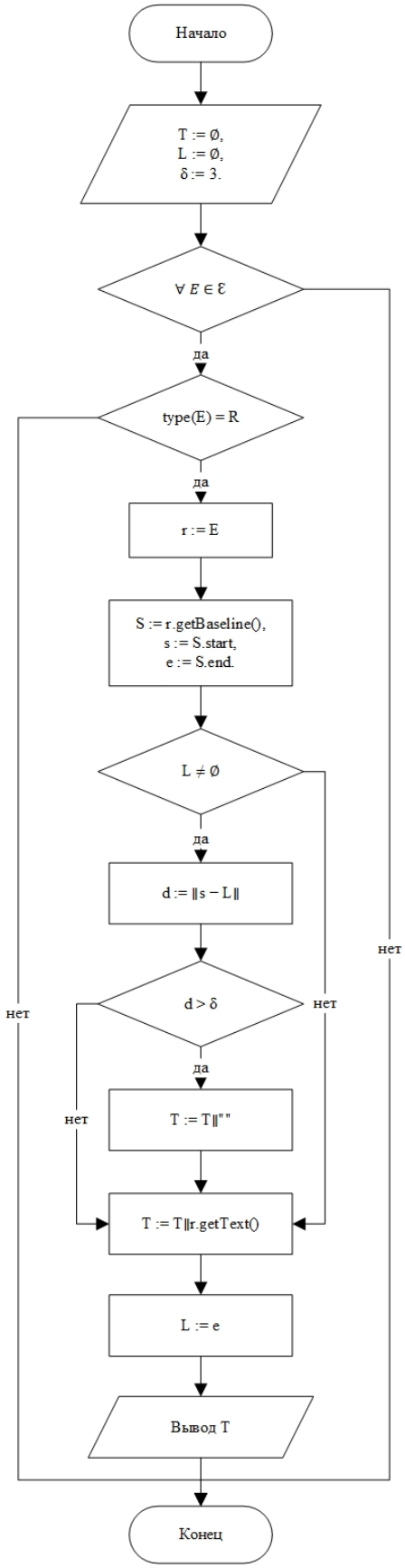

Рисунок 3 – Схема алгоритма извлечения текста из PDF

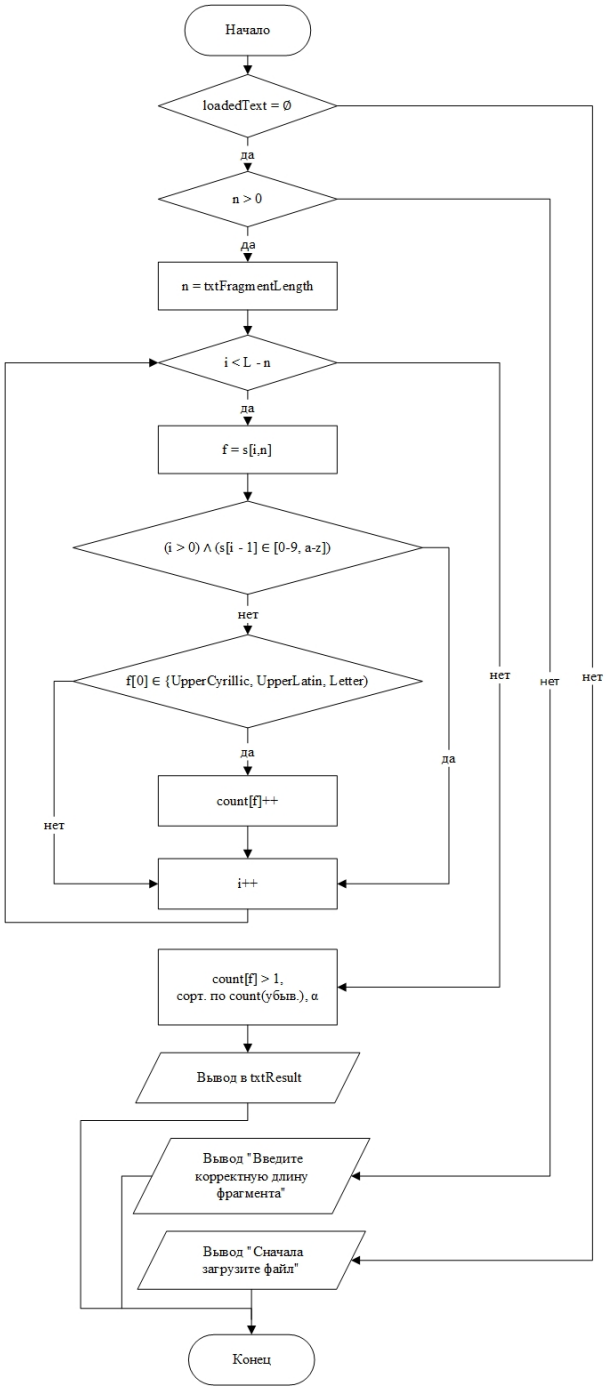

Рисунок 4 – Схема алгоритма поиска повторяющихся фрагментов

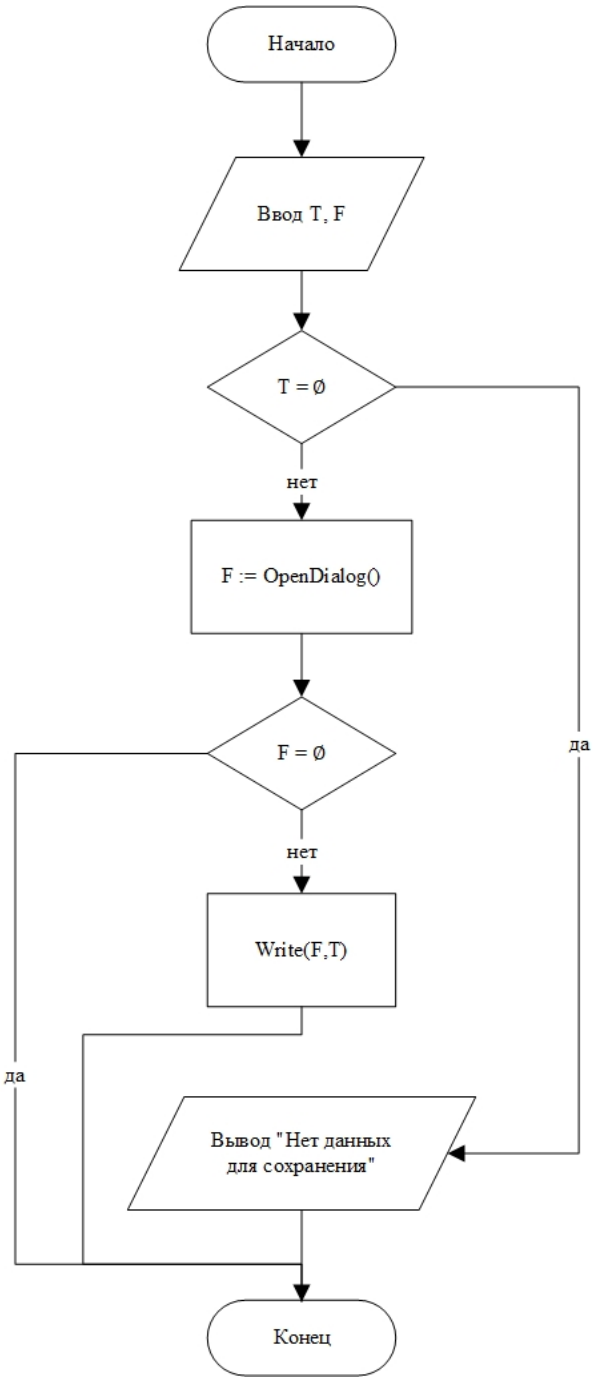

Рисунок 5 – Схема алгоритма сохранения результатов

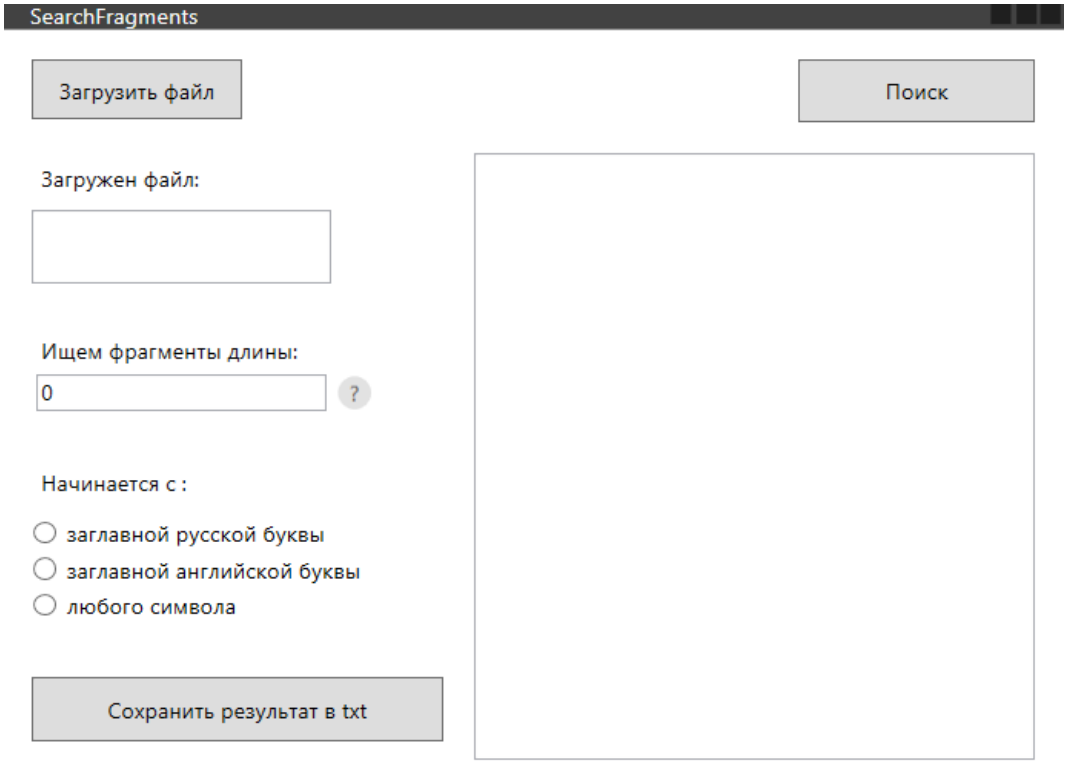

Рисунок 6 – Окно приложения «SearchFragments» открытое в конструкторе

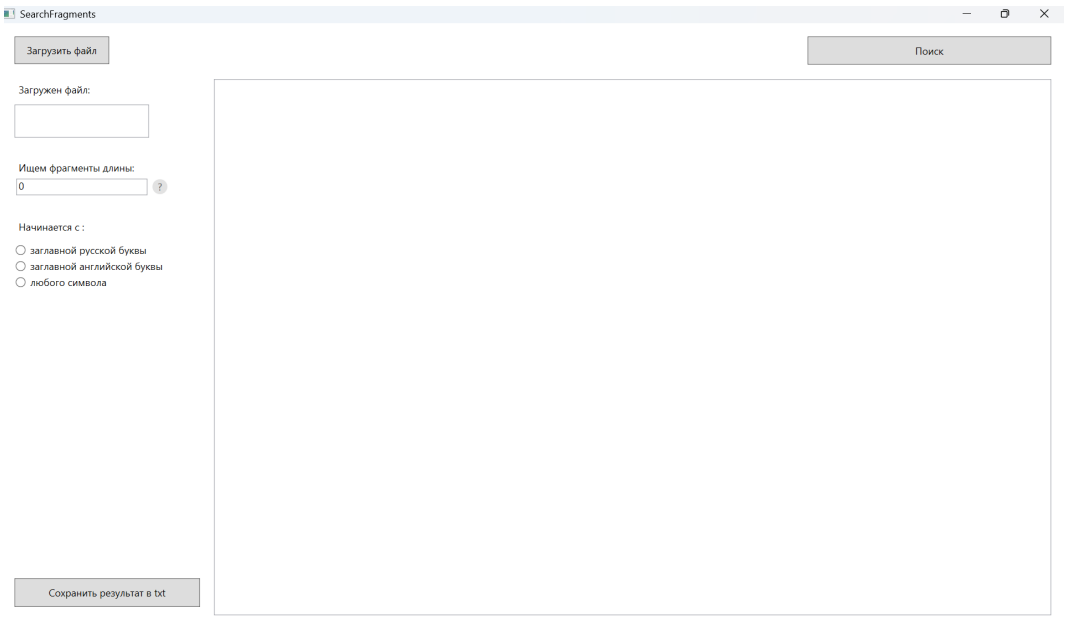

Рисунок 7 – Окно приложения «SearchFragments»: начальный запуск

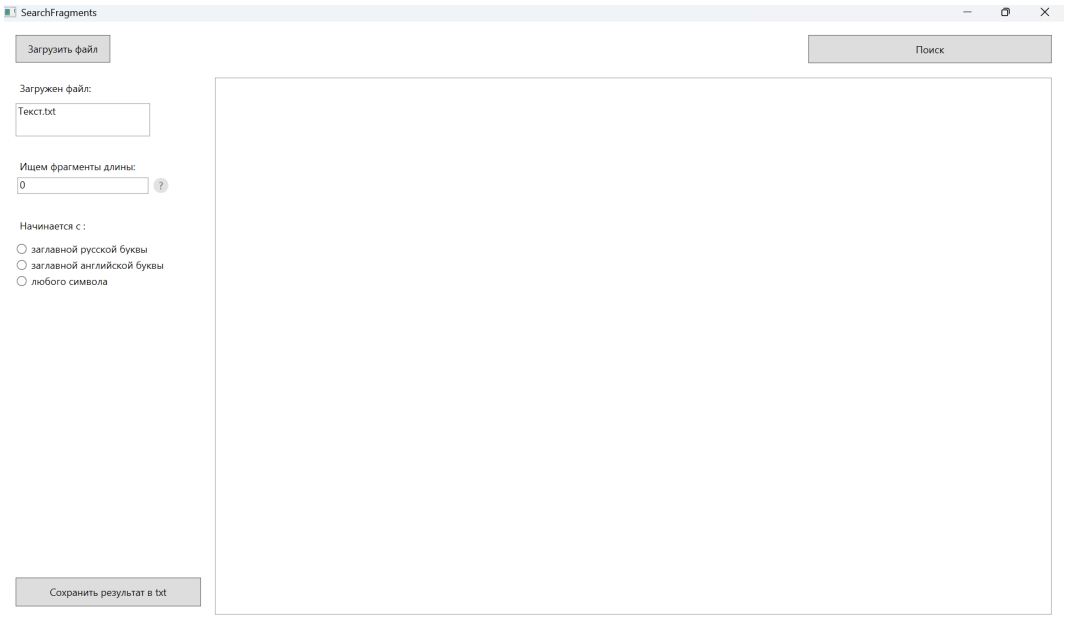

Рисунок 8 – Окно приложения «SearchFragments»: загрузка файла

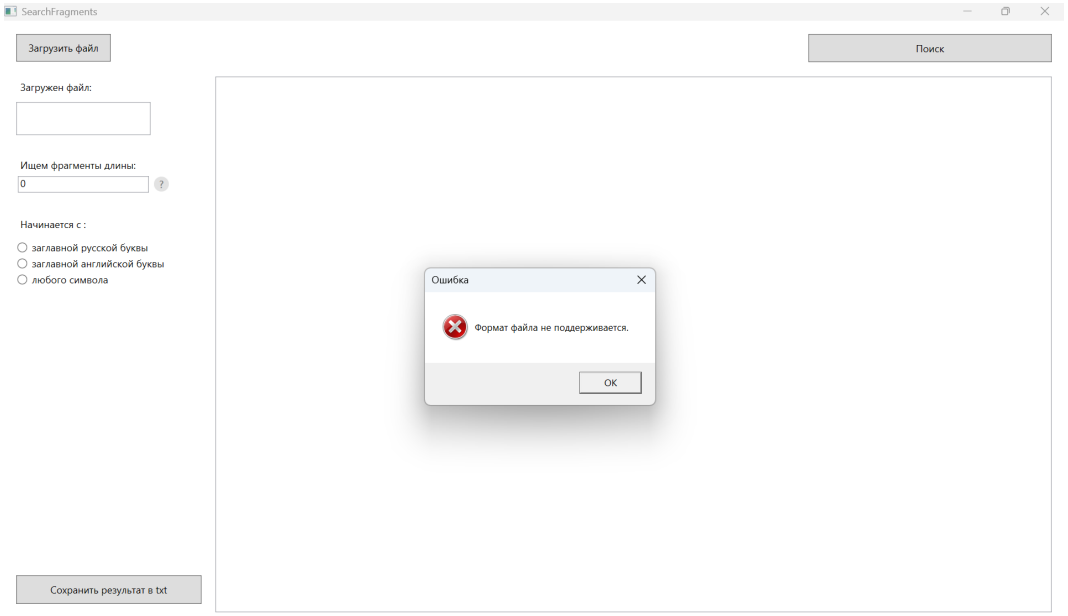

Рисунок 9 – Окно приложения «SearchFragments»: сообщение о неподдерживаемом формате загруженного файла

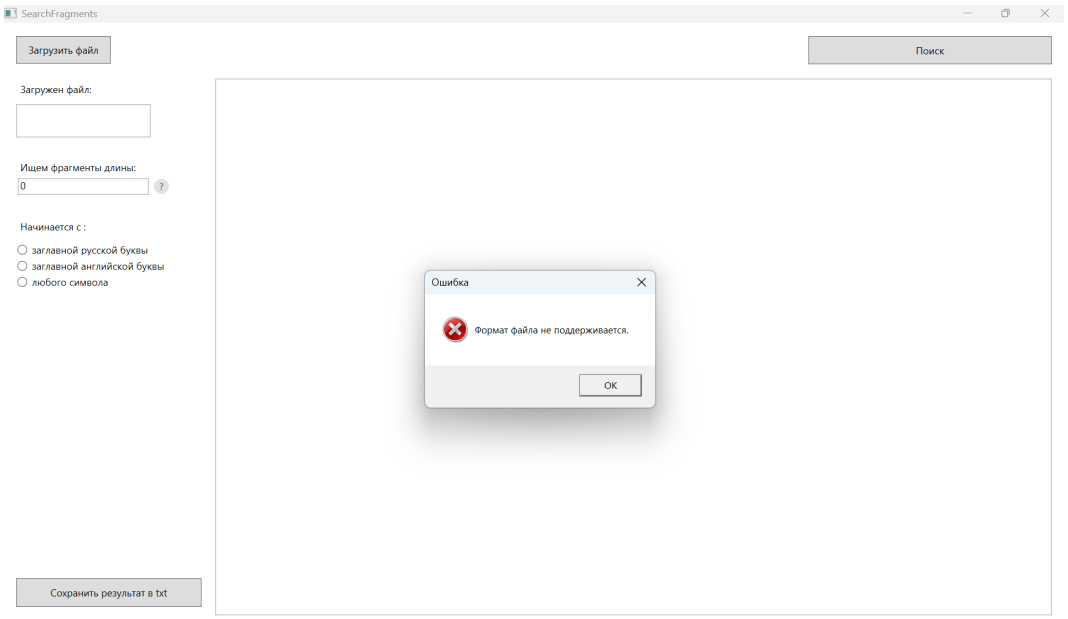

Рисунок 10 – Окно приложения «SearchFragments»: всплывающее пояснение о длине фрагмента

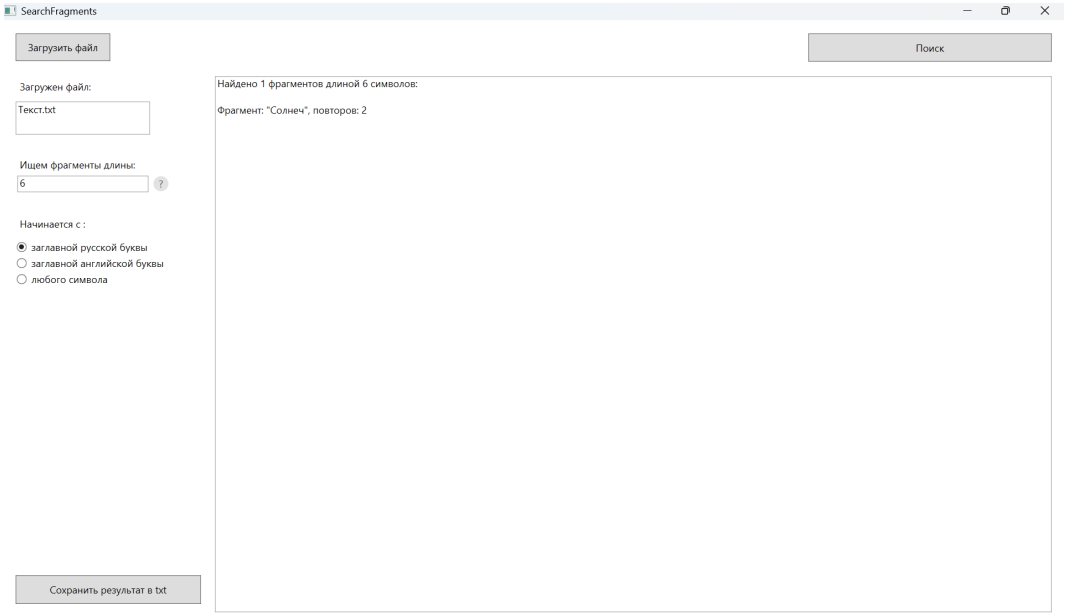

Рисунок 11 – Окно приложения «SearchFragments»: результат, если первый символ фрагмента начинается с заглавной русской буквы

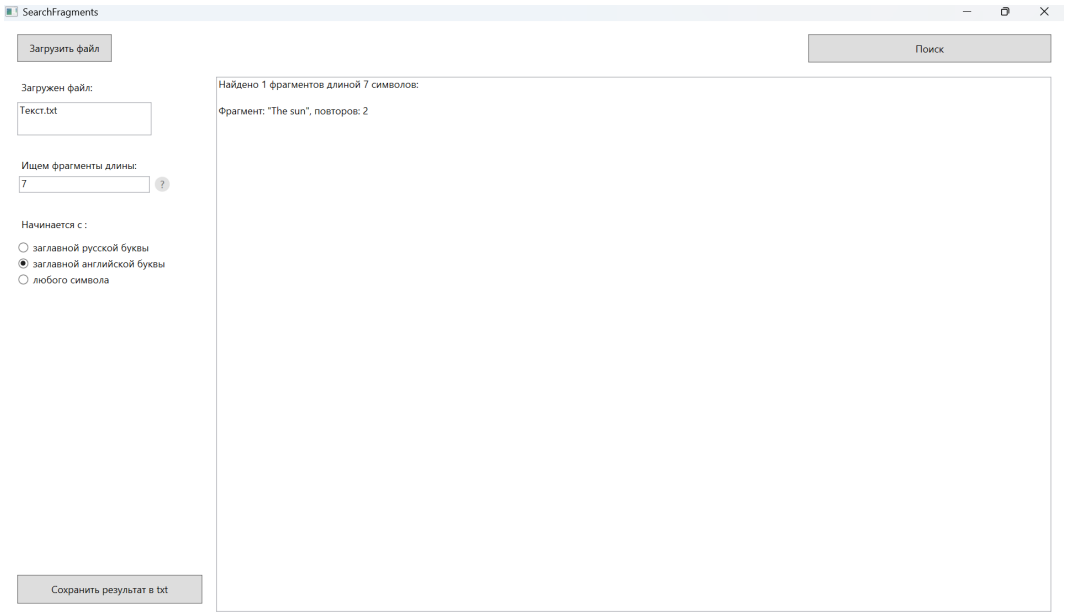

Рисунок 12 – Окно приложения «SearchFragments»: результат, если первый символ фрагмента начинается с заглавной английской буквы

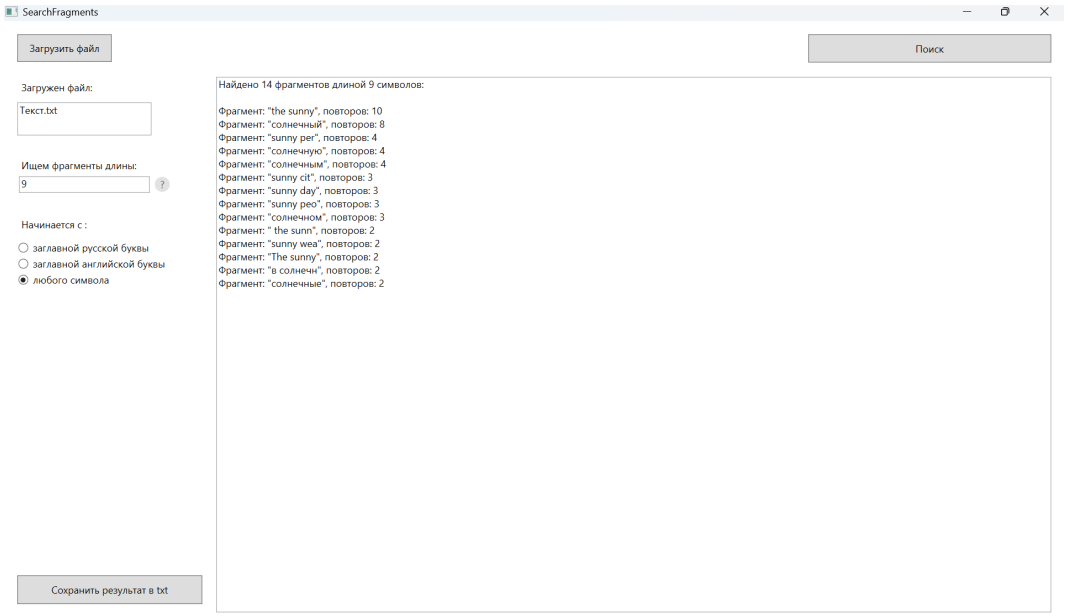

Рисунок 13 – Рисунок 10 – Окно приложения «SearchFragments»: результат, если первый символ фрагмента является любым

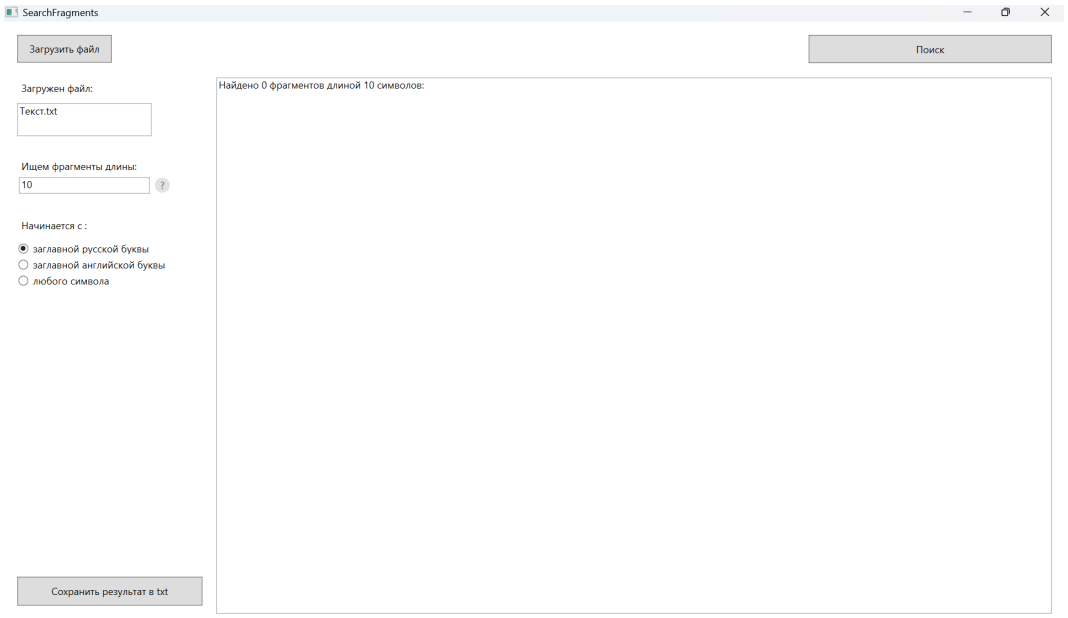

Рисунок 14 – Окно приложения «SearchFragments»: результат, когда повторяющихся фрагментов не найдено

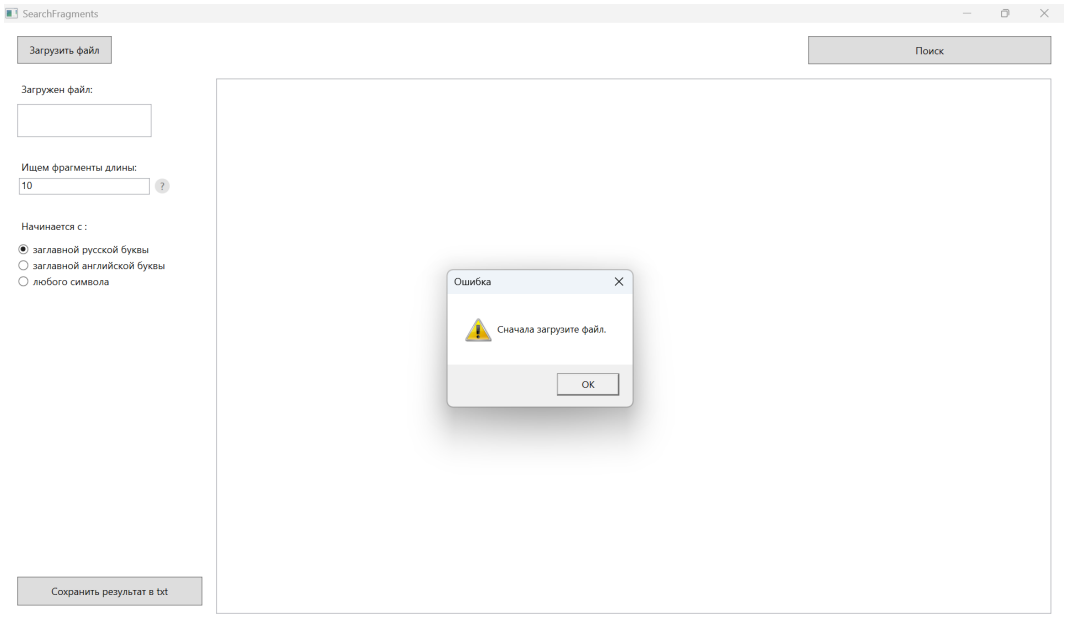

Рисунок 15 – Окно приложения «SearchFragments»: сообщение о просьбе пользователя перед работой загрузить файл

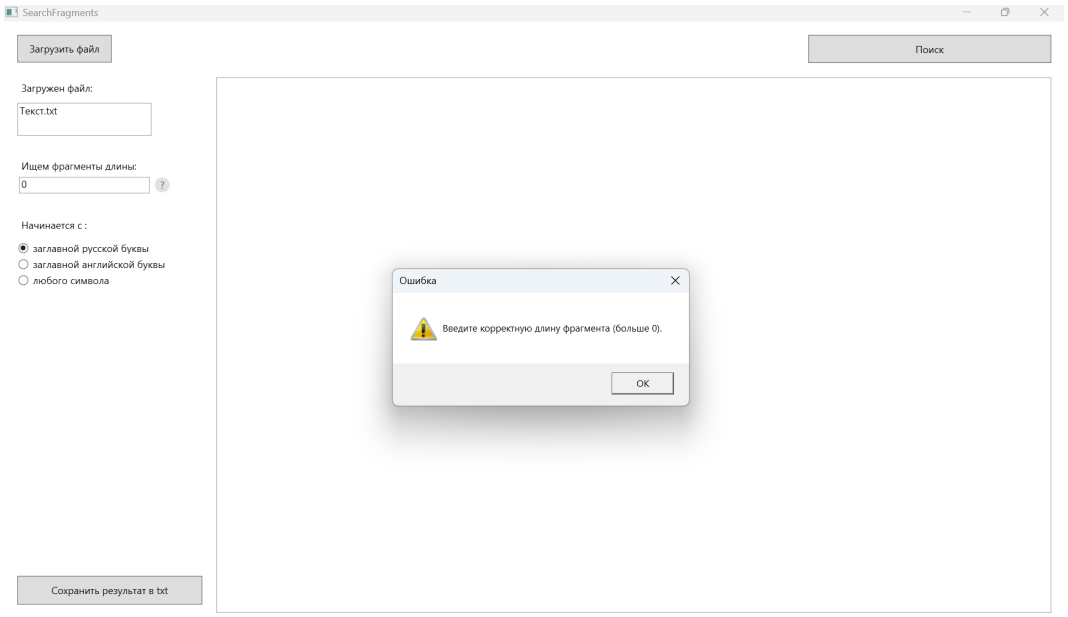

Рисунок 16 – Окно приложения «SearchFragments»: сообщение о просьбе пользователя перед работой ввести длину фрагмента больше нуля

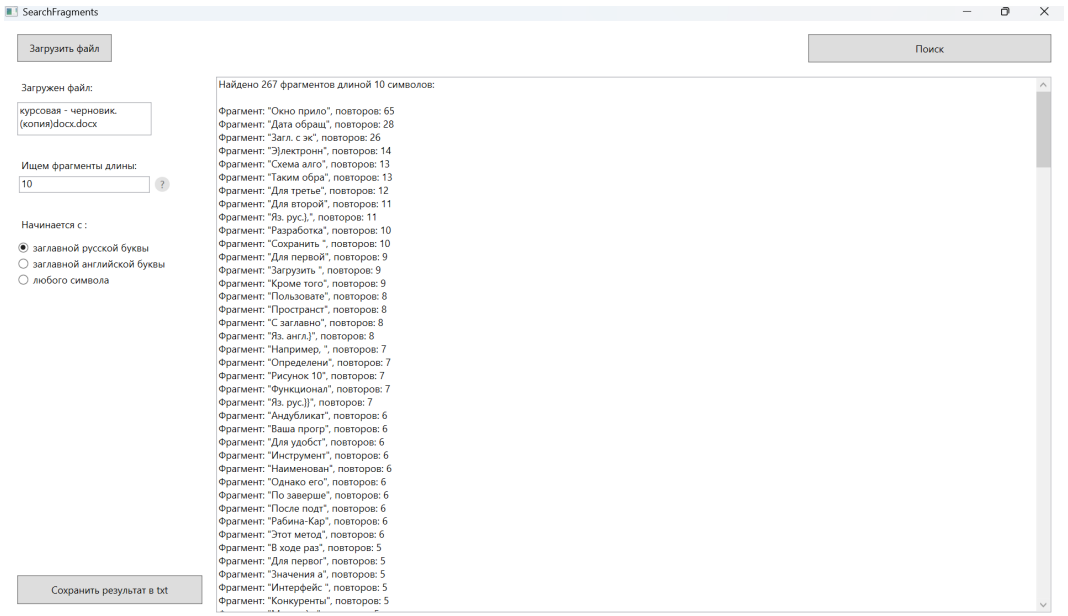

Рисунок 17 – Окно приложения «SearchFragments»: появление полосы прокрутки для просмотра всего списка

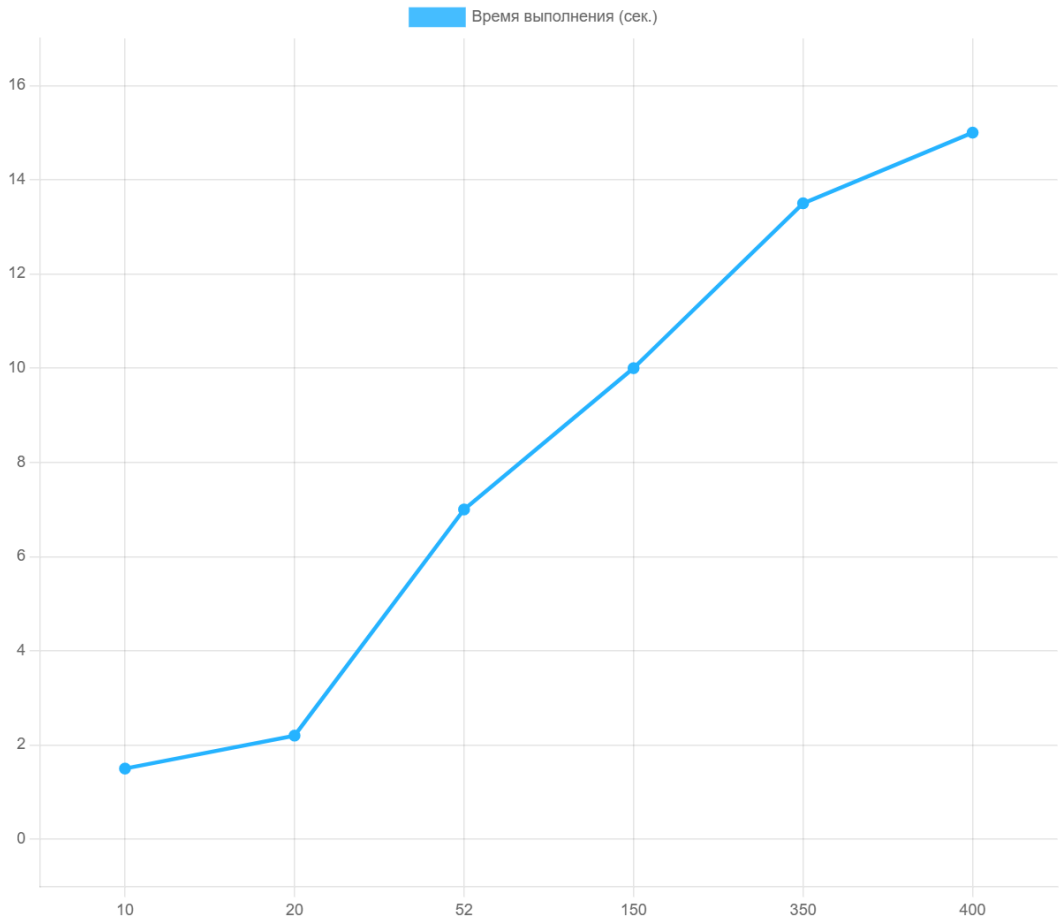

Рисунок 18 – График характеристик тестирования ПО

Таблица 1 – Значения атрибутов элементов в приложении «SearchFragments»

Наименование атрибута	Значение
Для формы	
Title	SearchFragments
Для первой кнопки	
(Name)	btnLoadFile
Content	Загружен файл
Для второй кнопки	
(Name)	btnSearch
Content	Поиск
Для третьей кнопки	
(Name)	btnSaveToFile
Content	Сохранить результат в txt
Для первой надписи	
(Name)	lblFileName
Content	Название файла
Для второй надписи	
(Name)	lblFragmentLength
Content	Длина искомого фрагмента
Для третьей надписи	
(Name)	lblSymbol
Content	Начинается с
Для первого текстового поля	
(Name)	txtFileName
Для второго текстового поля	
(Name)	txtFragmentLength
Text	0
Для третьего текстового поля	
(Name)	txtResult
Для четвертого текстового поля	
(Name)	?
(ToolTip)	Длина фрагмента учитывает пробелы
Для первой кнопки-переключателя	
(Name)	rbUpperCaseCyrillic
Content	заглавной русской буквы
Для второй кнопки-переключателя	
(Name)	rbUpperCaseLatin
Content	заглавной английской буквы
Для третьей кнопки-переключателя	
(Name)	rbAnySymbol
Content	любого символа

Таблица 2 – Таблица экспериментальных данных

Количество страниц	Время обработки (сек.)
10-20 страниц	2.2
40-70 страниц	7
350-400 страниц	13.5

ПРИЛОЖЕНИЕ Б

Листинг программы

Код приложения task.pl.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Windows;
using Microsoft.Win32;
using iText.Kernel.Pdf;
using iText.Kernel.Pdf.Canvas.Parser;
using System.Windows.Controls;
using System.Text;
using DocumentFormat.OpenXml.Packaging;
using Aspose.Words;
using iText.Kernel.Pdf.Canvas.Parser.Listener;
using iText.Kernel.Geom;
using iText.Kernel.Pdf.Canvas.Parser.Data;


public class SpacingAwareTextExtractionStrategy : ITextExtractionStrategy
{
    private readonly StringBuilder sb = new StringBuilder();
    private iText.Kernel.Geom.Vector lastEnd;
    private readonly float spaceThreshold;

    public SpacingAwareTextExtractionStrategy(float spaceThreshold = 3f)
    {
        this.spaceThreshold = spaceThreshold;
    }

    public void EventOccurred(IEventData data, EventType type)
    {
        if (type == EventType.RENDER_TEXT)
        {
            var renderInfo = (TextRenderInfo)data;
            iText.Kernel.Geom.LineSegment segment = renderInfo.GetBaseline();
            iText.Kernel.Geom.Vector start = segment.GetStartPoint();
            iText.Kernel.Geom.Vector end = segment.GetEndPoint();

```

```
38              if (lastEnd != null)
39              {
40                  float distance = start.Subtract(lastEnd).Length();
41                  if (distance > spaceThreshold)
42                  {
43                      sb.Append(' ');
44                  }
45              }
46
47              sb.Append(renderInfo.GetText());
48              lastEnd = end;
49          }
50      }
51
52      public string GetResultantText() => sb.ToString();
53
54      public ICollection<EventType> GetSupportedEvents() => null;
55
56      public void BeginTextBlock() { }
57
58      public void EndTextBlock() { }
59
60      public void RenderText(TextRenderInfo renderInfo) { }
61
62      public void RenderImage(ImageRenderInfo renderInfo) { }
63  }
64
65  namespace SearchFragments
66  {
67      /// <summary>
68      /// Логика взаимодействия для MainWindow.xaml
69      /// </summary>
70      public partial class MainWindow : IMainWindow
71      {
72          private string loadedText; // Объявление переменной класса для
73  хранения текста
74
75          public MainWindow()
76          {
77              InitializeComponent();
78          }
```

```
79
80          private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
81          {
82
83          }
84
85          private void RadioButton_Checked(object sender, RoutedEventArgs e)
86          {
87
88          }
89
90          private void RadioButton_Checked_1(object sender, RoutedEventArgs e)
91          {
92
93          }
94
95
96          private void BtnLoadFile_Click(object sender, RoutedEventArgs e)
97          {
98              OpenFileDialog openFileDialog = new OpenFileDialog
99              {
100                 Filter = "All files (*.*)|*.*|Text files (*.txt)|
101 *.txt|Word documents (*.docx;*.doc)|*.docx;*.doc|PDF files (*.pdf)|*.pdf",
102                 FilterIndex = 1, // Устанавливаем начальный фильтр (все файлы)
103                 Title = "Выберите файл"
104             };
105
106             if (openFileDialog.ShowDialog() == true)
107             {
108                 string filePath = openFileDialog.FileName;
109
110                 try
111                 {
112                     string fileExtension = System.IO.Path.GetExtension(filePath)
113 .ToLower();
114                     if (fileExtension == ".txt" || fileExtension == ".docx" ||
115 fileExtension == ".doc" || fileExtension == ".pdf")
116                     {
117                         loadedText = LoadFile(filePath); // Загружаем содержимое
118 файла
119                         txtFileName.Text = System.IO.Path.GetFileName(filePath);
```

```
120 // Выводим название файла в TextBox
121                 }
122                 else
123                 {
124                     MessageBox.Show("Формат файла не поддерживается.",
125 "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
126                 }
127             }
128             catch (Exception ex)
129             {
130                 MessageBox.Show("Ошибка загрузки файла: " + ex.Message,
131 "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
132             }
133         }
134     }
135 
136     private string LoadFile(string filePath)
137     {
138 
139         if (filePath.EndsWith(".txt"))
140         {
141             return File.ReadAllText(filePath); // Читает текст из .txt файла
142         }
143         else if (filePath.EndsWith(".docx"))
144         {
145             using (WordprocessingDocument doc = WordprocessingDocument.
146 Open(filePath, false))
147             {
148                 return doc.MainDocumentPart.Document.Body.InnerText;
149 // Получает текст из .docx файла
150             }
151         }
152         else if (filePath.EndsWith(".pdf"))
153         {
154             using (PdfReader pdfReader = new PdfReader(filePath))
155             using (PdfDocument pdf = new PdfDocument(pdfReader))
156             {
157                 StringBuilder sb = new StringBuilder();
158                 for (int page = 1; page <= pdf.GetNumberOfPages(); page++)
159                 {
160                     var strategy = new SpacingAwareTextExtractionStrategy(2.5f);
```

```
// указанный порог, можно изменять
                    var parser = new PdfCanvasProcessor(strategy);
                    parser.ProcessPageContent(pdf.GetPage(page));
                    sb.Append(strategy.GetResultantText());
                }
                return sb.ToString();
            }
        }
        else if (filePath.EndsWith(".doc"))
        {
            // Загружаем документ через Aspose.Words
            Document doc = new Document(filePath);
            return doc.GetText(); // Возвращаем текст документа
        }


        return ""; // Возвращает пустую строку, если файл не поддерживается
    }



    private void BtnSearch_Click(object sender, RoutedEventArgs e)
    {
        if (string.IsNullOrWhiteSpace(loadedText))
        {
            MessageBox.Show("Сначала загрузите файл.", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

        string processedText = loadedText;


        if (!int.TryParse(txtFragmentLength.Text, out int fragmentLength)
|| fragmentLength < 1)
        {
            MessageBox.Show("Введите корректную длину фрагмента (больше 0).
", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);
            return;
        }

```

```
202             // Словарь для хранения фрагментов и их количества
203             Dictionary<string, int> fragmentCounts = new Dictionary<string, int>();
204 
205             // Определяем, какой вариант выбран
206             bool checkUpperCaseCyrillicn = rbUpperCaseCyrillic.IsChecked == true;
207             bool checkUpperCaseLatin = rbUpperCaseLatin.IsChecked == true;
208             bool checkAnySymbol = rbAnySymbol.IsChecked == true;
209 
210             // Перебираем текст посимвольно для поиска фрагментов
211             for (int i = 0; i <= loadedText.Length - fragmentLength; i++)
212             {
213                 // Проверяем, что фрагмент находится в допустимых пределах
214                 if (i + fragmentLength > processedText.Length)
215                 {
216                     break; // Прерываем цикл, если конец фрагмента выходит
217 за пределы строки
218                 }
219 
220                 // Извлекаем фрагмент текста указанной длины
221                 string fragment = loadedText.Substring(i, fragmentLength);
222 
223                 // Проверяем, что фрагмент начинается с пробела или с начала строки
224                 if (i > 0 && char.IsLetterOrDigit(processedText[i - 1]))
225 // Если перед фрагментом есть буква или цифра (это не начало слова)
226                 {
227                     continue; // Пропускаем фрагмент, если он не начинается
228 с пробела или начала строки
229                 }
230 
231                 // Проверяем, что фрагмент начинается с нужной буквы
232                 if (checkUpperCaseCyrillicn && !(char.IsUpper(fragment[0])
233 && fragment[0] >= 'А' && fragment[0] <= 'Я'))
234                 {
235                     continue;
236                 }
237                 else if (checkUpperCaseLatin && !(char.IsUpper(fragment[0])
238 && fragment[0] >= 'A' && fragment[0] <= 'Z'))
239                 {
240                     continue;
241                 }
242                 else if (!checkAnySymbol && !char.IsLetter(fragment[0]))
```

```
243             {
244                 // Проверяем, что фрагмент начинается с буквы (русской
245 или английской), без знаков препинания
246                 if (!char.IsLetter(fragment[0]))
247                 {
248                     continue; // Пропускаем, если фрагмент не начинается
249 с буквы
250                 }
251 
252                 // Проверяем, что весь фрагмент состоит из букв
253                 if (!fragment.All(c => char.IsLetter(c)))
254                 {
255                     continue; // Пропускаем фрагмент, если в нем есть символы,
256 которые не являются буквами
257                 }
258 
259 
260             }
261 
262             // Подсчитываем фрагмент
263             if (fragmentCounts.ContainsKey(fragment))
264             {
265                 fragmentCounts[fragment]++;
266             }
267             else
268             {
269                 fragmentCounts[fragment] = 1;
270             }
271 
272         }
273 
274         var sortedFragments = fragmentCounts
275         .Where(kv => kv.Value > 1)
276         .OrderByDescending(kv => kv.Value) // Сортировка по количеству
277 повторений (по убыванию)
278         .ThenBy(kv => kv.Key, StringComparer.CurrentCulture) // Корректная
279 сортировка по алфавиту для разных языков
280         .ToList();
281 
282 
283 
```

```
284             StringBuilder resultBuilder = new StringBuilder();
285             resultBuilder.AppendLine($"Найдено {sortedFragments.Count}
286 фрагментов длиной {fragmentLength} символов:\n");
287 
288             foreach (var kv in sortedFragments)
289             {
290                 resultBuilder.AppendLine($"Фрагмент: \"{kv.Key}\",
291 повторов: {kv.Value}");
292             }
293 
294             txtResult.Text = resultBuilder.ToString();
295         }
296 
297 
298 
299 
300         private void TxtFragmentLength_TextChanged(object sender,
301 TextChangedEventArgs e)
302         {
303 
304         }
305 
306         private void TxtFileName_TextChanged(object sender,
307 TextChangedEventArgs e)
308         {
309 
310         }
311 
312         private void Button_Click(object sender, RoutedEventArgs e)
313         {
314             if (string.IsNullOrWhiteSpace(txtResult.Text))
315             {
316                 MessageBox.Show("Нет данных для сохранения.",
317 "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);
318                 return;
319             }
320 
321             SaveFileDialog saveFileDialog = new SaveFileDialog
322             {
323                 Filter = "Text files (*.txt)|*.txt",
324                 Title = "Сохранить результаты",
```

```
325                 FileName = "SearchResults.txt"
326             };
327 
328             if (saveFileDialog.ShowDialog() == true)
329             {
330                 File.WriteAllText(saveFileDialog.FileName, txtResult.Text);
331             }
332         }
333     }
334 }
```