Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Информационных технологий и управления

Кафедра Интеллектуальных информационных технологий

**ОТЧЁТ**

по дисциплине «Естественно-языковой интерфейс интеллектуальных систем»

Лабораторная работа №2

Тема: Построение и использование корпусов текстов естественного языка

Выполнили: Демидовец Д.В.

Козырев Д.А.

гр. 221703

Проверил: Крапивин Ю.Б.

Минск 2025

**Цели работы:**

1. изучить принципы построения корпусов текстов, виды разметки и способы

аннотирования, инструменты работы с корпусами текстов,

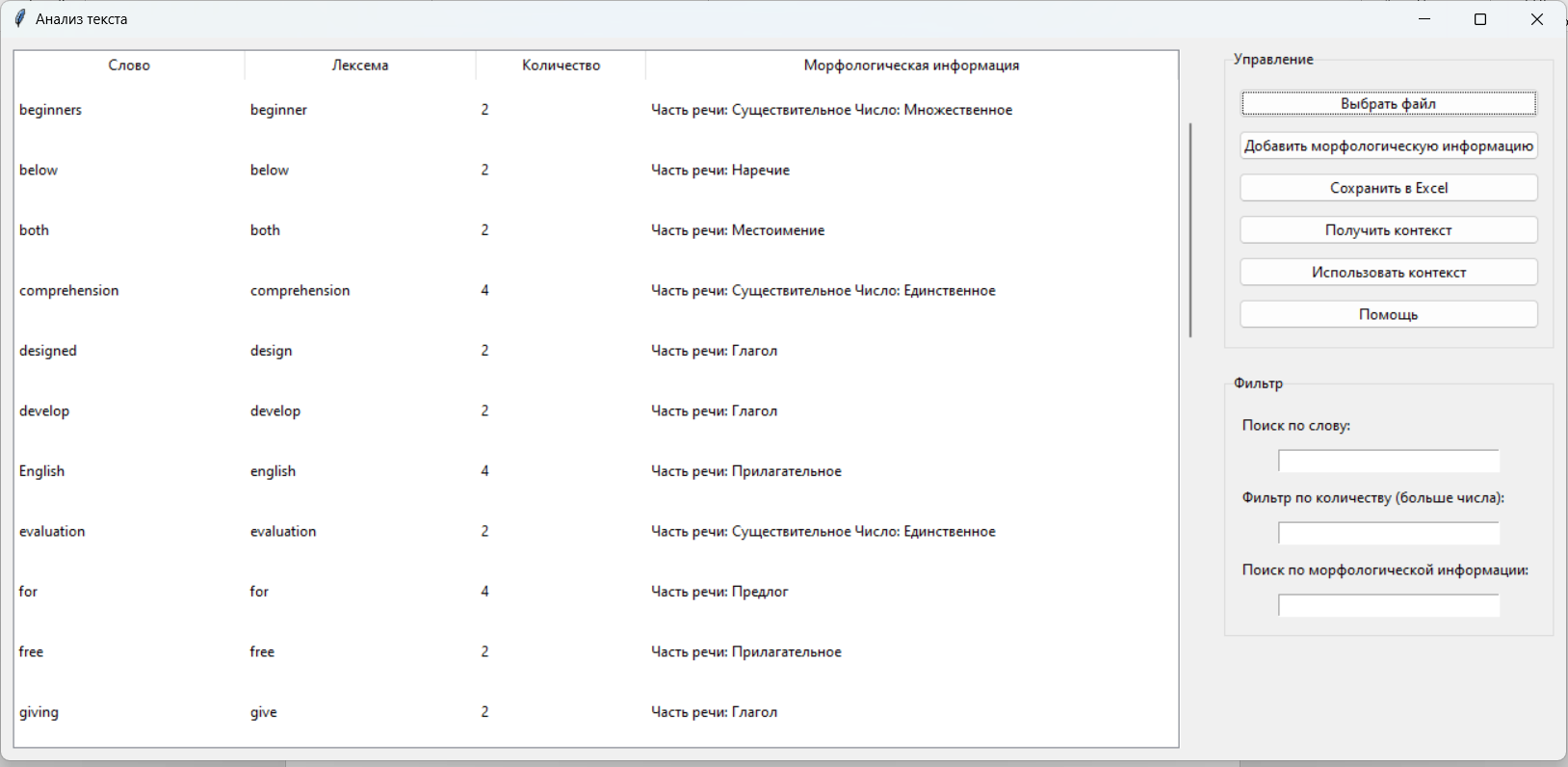
2. построить корпус текстов и разработать корпусный менеджер.

**Задание:**

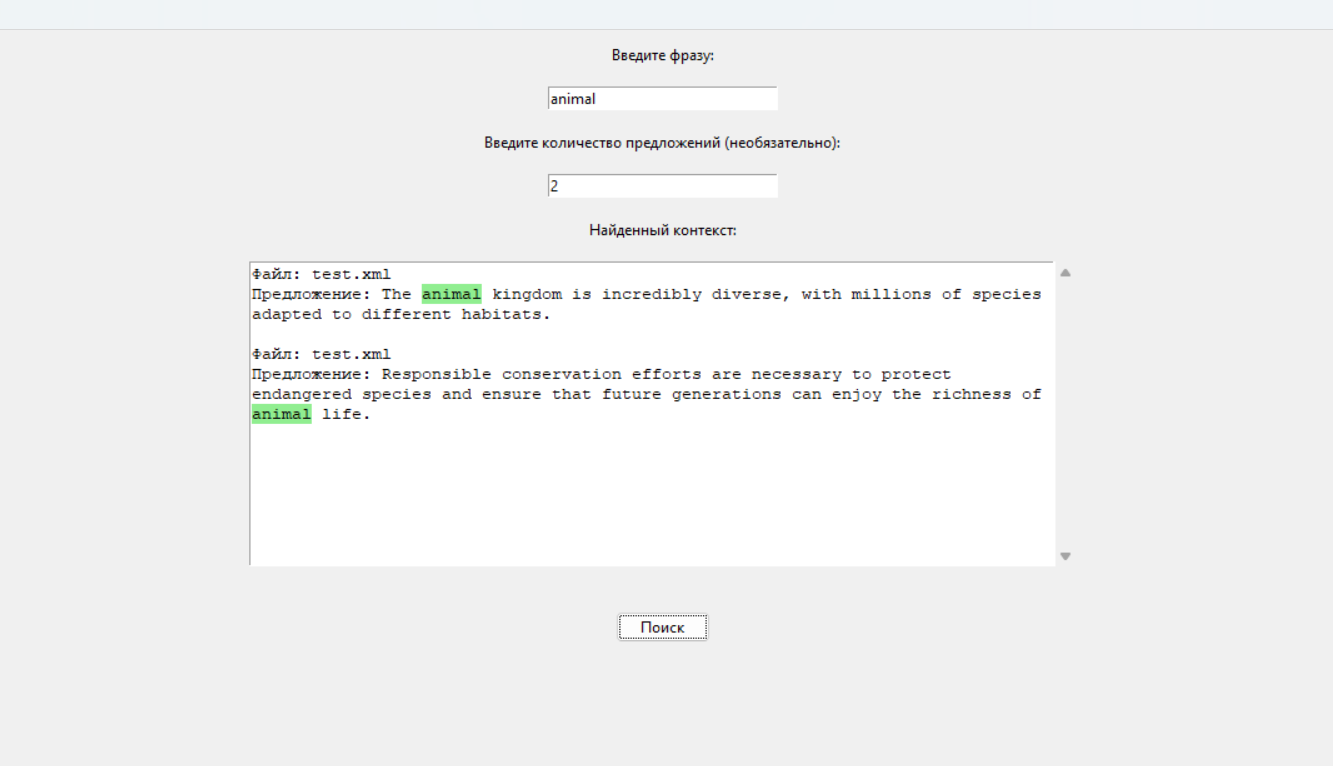
1. Сформировать электронный корпус текстов по выбранной предметной области.

2. Используя результаты лабораторной работы №1 (возможность получения лингвистических сведений для произвольной лексемы естественного языка) разработать корпусный менеджер, обеспечивающий базовую функциональность работы с созданным корпусом текстов.

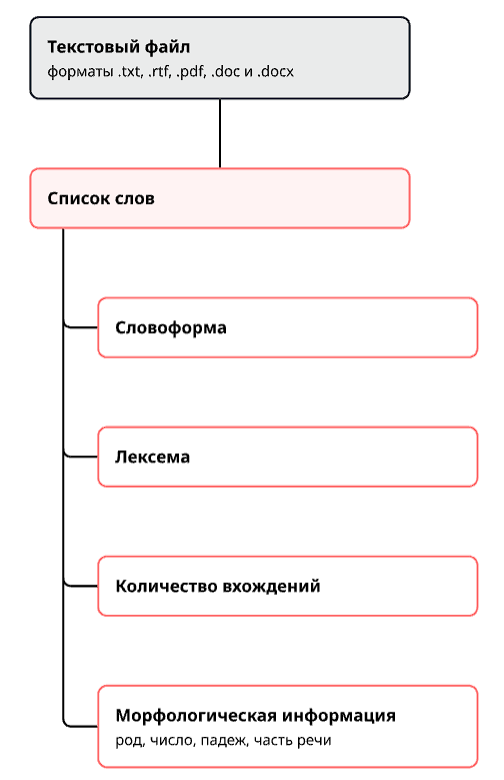
**Интерфейс:**

****

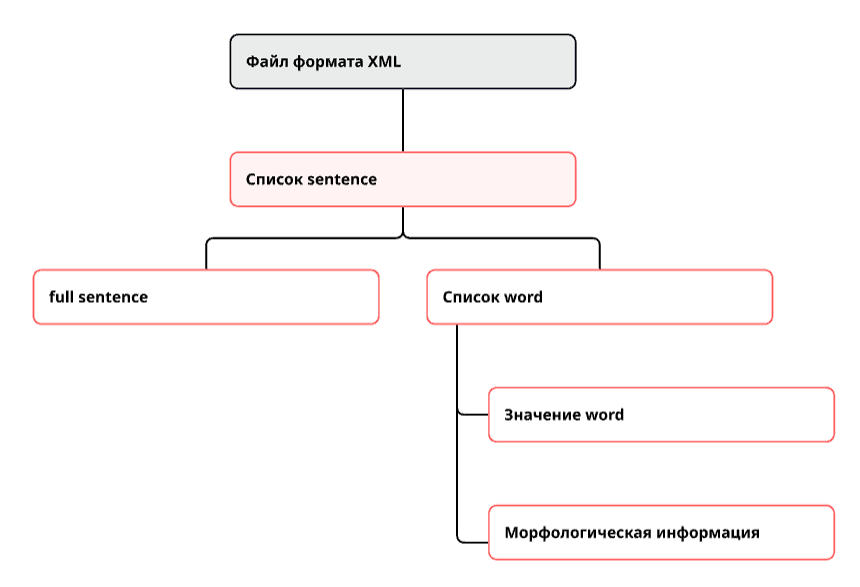
**Поиск по контексту:**

****

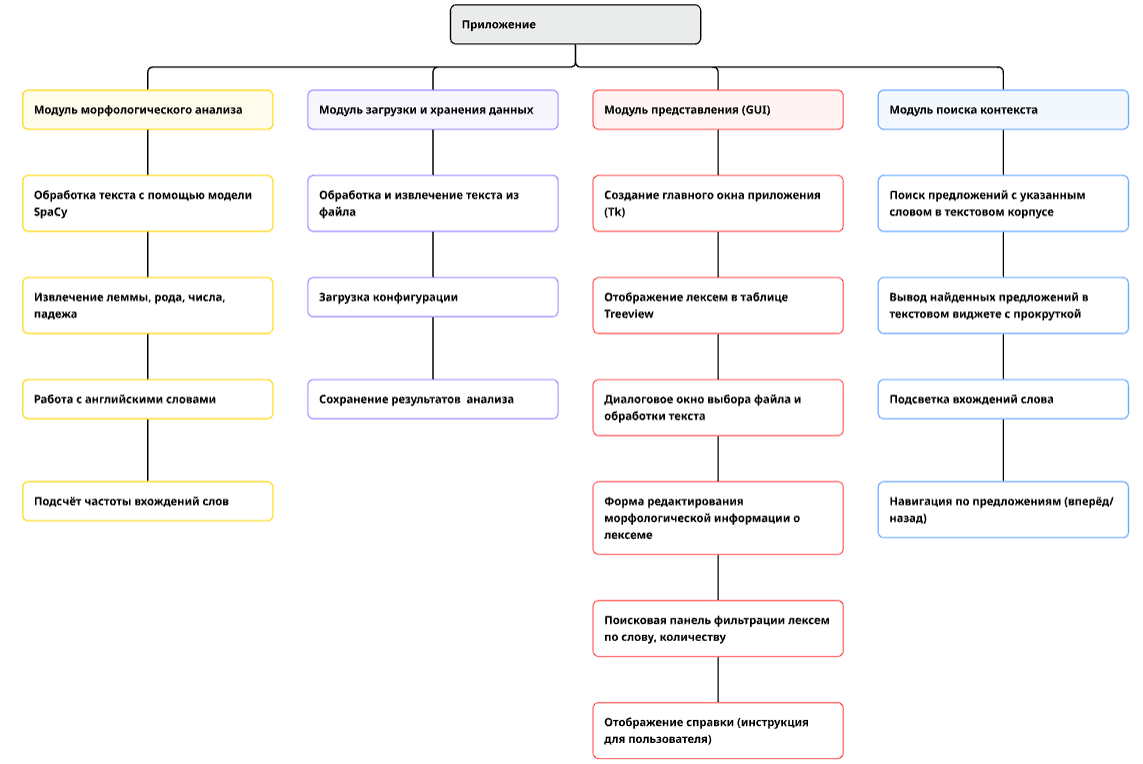
**Структура хранения данных:**

****

**Структурная схема для хранения корпуса текста для одного из файлов:**

****

**Структурная схема приложения:**

****

**Использованные библиотеки:**

* Библиотека **spaCy** применяется для обработки текста на естественном языке. С её помощью выполняется разбиение текста на токены, лемматизация (приведение слова к начальной форме), определение частей речи, морфологических признаков (род, число, падеж), а также получение синтаксической информации.
* Библиотека **tkinter** используется для создания графического интерфейса пользователя. С её помощью реализованы основное окно приложения, кнопки, таблица с лексемами (через Treeview), диалоговые окна для открытия файлов, добавления и редактирования лексем, окно справки и окно контекста слова.
* Библиотека **fitz** (модуль PyMuPDF) применяется для чтения PDF-документов. Используется при загрузке файлов в формате .pdf, позволяет извлекать текст для дальнейшего анализа.
* Библиотека **docx** (из модуля python-docx) используется для чтения документов Microsoft Word в формате .docx. Применяется при загрузке таких файлов пользователем.
* Библиотека **openpyxl** применяется для создания и сохранения файлов в формате Excel (.xlsx). Используется при экспорте таблицы лексем в Excel-документ.
* Библиотека **striprtf** (модуль striprtf.striprtf) используется для извлечения текста из RTF-документов. Применяется при загрузке файлов .rtf, преобразуя их в обычный текст.
* Библиотека **json** применяется для сохранения и загрузки информации о лексемах и настройках приложения в формате JSON.
* Библиотека **os** используется для работы с файловой системой: построения путей, проверки наличия директорий, обхода папок с файлами корпуса.
* Библиотека **re** используется для работы с регулярными выражениями. Применяется при очистке текста от лишних символов, извлечении слов и предложений из текста.
* Библиотека **textwrap** применяется для форматирования длинных текстов с переносом строк. Используется при отображении контекста слова в отдельном окне, чтобы текст был читаемым.
* Библиотека **collections.Counter** используется для подсчета количества повторений лексем. Применяется для вычисления частотности слов.
* Библиотека **glob** используется для получения списка файлов в папке по заданному шаблону (например, \*.txt). Применяется при обработке корпуса текстов.

**Описание алгоритмов:**

**1. Загрузка и предварительная обработка текста из файла:**  
Основной алгоритм реализован в функции process\_files, вызываемой при открытии файла пользователем:

1. Начало.
2. Через графический интерфейс пользователь выбирает текстовый файл формата .txt, .pdf, .docx или .rtf.
3. С помощью метода os.path.splitext извлекаем расширение файла, и на основе этого принимаем решение о способе обработки.
4. В зависимости от расширения вызываем соответствующую функцию:
   * Для .txt файлов — содержимое читаем напрямую в строку и переходим к пункту 4.
   * Для .pdf — используем библиотеку fitz (PyMuPDF), извлекаем текст из всех страниц и переходим к пункту 4.
   * Для .docx — применяем библиотеку python-docx, текст извлекаем из всех абзацев документа и переходим к пункту 4.
   * Для .rtf — используем функцию rtf\_to\_text из библиотеки striprtf, преобразующую RTF в обычный текст, и переходим к пункту 4.
5. После извлечения текст проходит предварительную очистку: удаляем лишние пробелы, табуляции, управляющие символы (\n, \r и др.), применяем регулярные выражения для устранения лишних спецсимволов и объединения строк.
6. Используем регулярное выражение для разбиения текста на предложения по знакам препинания (., !, ?) с учётом пробелов. Полученные предложения сохраняем для дальнейшего анализа контекста.
7. Возвращаем пару из строки очищенного текста и списка предложений.
8. Полученные данные затем передаются в модуль лингвистического анализа (extract\_token\_info), где начинается морфологическая обработка.
9. Конец.

**2. Морфологическая обработка текста и извлечение лексем:**

Основной алгоритм реализован в функции extract\_token\_info.

1. Начало.
2. Ранее очищенный текст поступает на вход функции extract\_token\_info, где он передаётся в NLP-анализатор spaCy.
3. Обрабатываем текст через морфоанализатор spaCy: текст разбивается на токены, для каждого токена извлекаются следующие характеристики:
   * Слово
   * Лемма
   * Количество
   * Морфологическая информация (число, род, падеж и др.)
4. Создаём словарь для хранения информации о лексемах.
5. Перебираем токены и наполняем словарь:
   * Если токен состоит только из букв — приводим его к нижнему регистру.
   * Если лемма ещё не встречалась — создаём новую запись с её признаками и счётчиком вхождений.
   * Если лексема уже есть — увеличиваем счётчик на единицу.
6. После анализа преобразуем словарь лексем в список словарей (один элемент — одна лексема).
7. Итоговый список лексем сортируется по алфавиту без учёта регистра для удобства отображения.
8. Возвращаем список лексем в основной интерфейс.
9. Конец алгоритма.

### ****3. Поиск предложений(контекста) с заданным словом в XML-корпусе:****

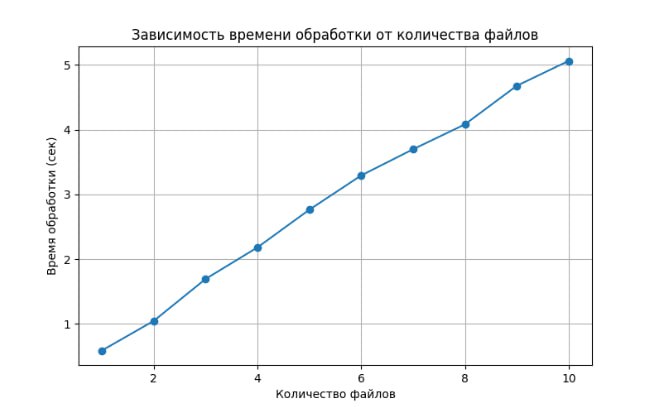
Реализуется с помощью функции find\_sentences\_with\_word из модуля context.py.

1. **Начало**
2. **Пользователь выбирает лексему.**
3. Функция получает выбранную лексему и список предложений, ранее извлечённых при загрузке текста.
4. Используем регулярное выражение re.search для нахождения всех предложений, содержащих заданную лексему (с учётом границ слов).
5. Для удобства чтения длинные предложения сокращаем с помощью textwrap.shorten.
6. Найденные предложения передаем для отображения в отдельном диалоговом окне.
7. Конец алгоритма.

**4. Отображение данных в графическом интерфейсе**

1. Начало.
2. При запуске программы создается главное окно и задаются его свойства, после чего размещаются основные элементы: меню, текстовые поля, таблицы и кнопки.
3. Происходит связывание действий пользователя с обработчиками событий.
4. Таблица отображается при загрузке и обновляется после фильтрации и редактирования данных.
5. Для редактирования, просмотра текста и вывода сообщений открываются отдельные диалоговые окна.
6. Процесс ожидания новых действий пользователя продолжается, пока запущен цикл событий.
7. Цикл событий прерывается, как только пользователь закрывает окно программы.
8. Конец алгоритма.

**Тест производительности:**

****

В ходе эксперимента было проведено измерение времени выполнения программы при создании корпуса текста из n файлов. Полученный график демонстрирует прямую зависимость между количеством слов и временем обработки чем больше слов содержит документ, тем больше времени требуется на его обработку.

Такой результат объясняется тем, что для каждого слова выполняется последовательный анализ с использованием морфологического разборщика, что линейно увеличивает общее время выполнения по мере роста количества входных данных. Этот результат подтверждает предсказуемость и стабильность работы программы при увеличении количества файлов.

**Вывод:**

В ходе выполнения лабораторной работы было разработано программное приложение с графическим интерфейсом для лингвистического анализа текстов на английском языке. Программа позволяет загружать документы различных форматов (.txt, .pdf, .docx, .rtf), извлекать из них текстовую информацию и автоматически выполнять морфологический разбор с использованием библиотеки spaCy. В процессе анализа определяется лемма, часть речи, грамматические признаки, а также количество упоминаний каждого слова. Реализованная система также предоставляет возможность просматривать контекст слова в исходных текстах. Разработка обеспечивает удобную и гибкую работу с корпусами текстов, а её модульная архитектура позволяет легко расширять функциональность в будущем, включая поддержку новых языков и интеграцию с внешними лингвистическими инструментами.