Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине: «Естественно-языковой интерфейс ИС»

# Тема: «Построение и использование корпусов текстов естественного языка»

Выполнил:

Студент 3 курса

Группы ИИ-23

Романюк А. П.

Проверил:

Булей Е. В.

Брест 2025

**Ход работы**

**Задание:** Разработка корпусного менеджера

Разработать приложение, которое предоставляет пользователю возможность обрабатывать фрагменты текста на естественном языке, запрашивая различные частотные и морфологические характеристики словоформ.

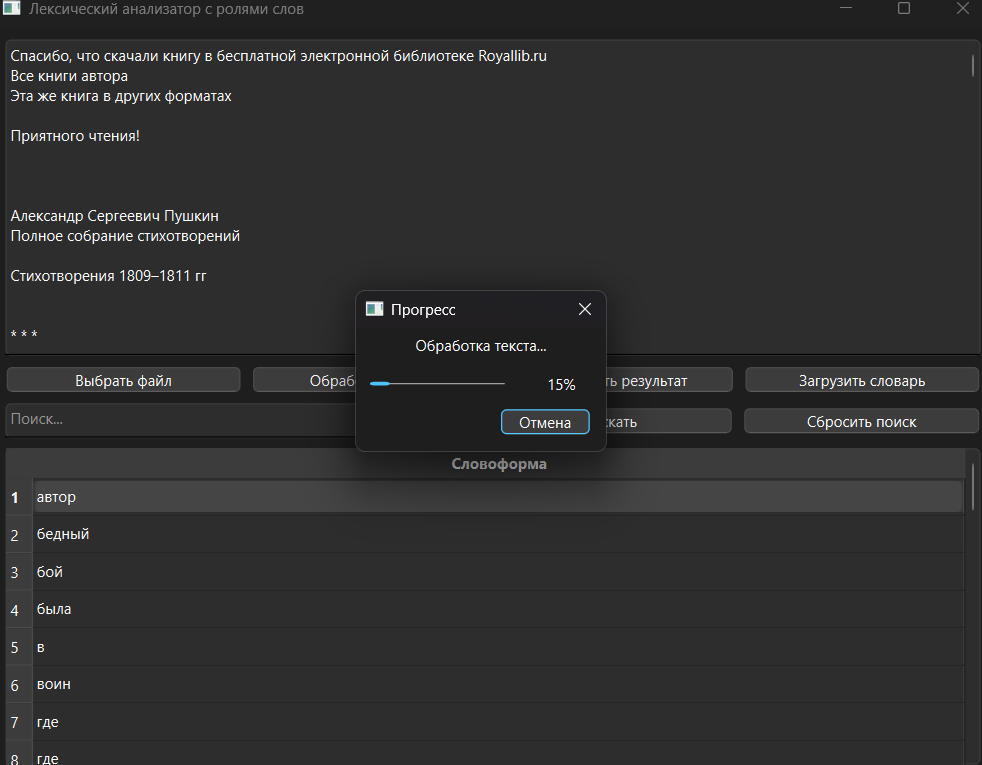
**Требования:**

***Входные данные:*** Пользователь вводит фрагмент текста (фразу или слово) на естественном языке в качестве запроса к корпусному менеджеру.

***Выходные данные:*** Приложение предоставляет следующие выходные данные:

* Частотные характеристики словоформ и лексем.
* Грамматические категории.
* Леммы слов.
* Морфологические характеристики словоформ.

***Взаимодействие с пользователем:*** Пользователь взаимодействует с приложением через графический интерфейс, который должен быть интуитивно-понятным и дружественным для пользователя.



*Процесс обработки корпуса текста*

*Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.*

*Приложение с загруженным корпусом текста*

Изображение выглядит как текст, программное обеспечение, Мультимедийное программное обеспечение, Графическое программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

*Просмотр характеристик слова*

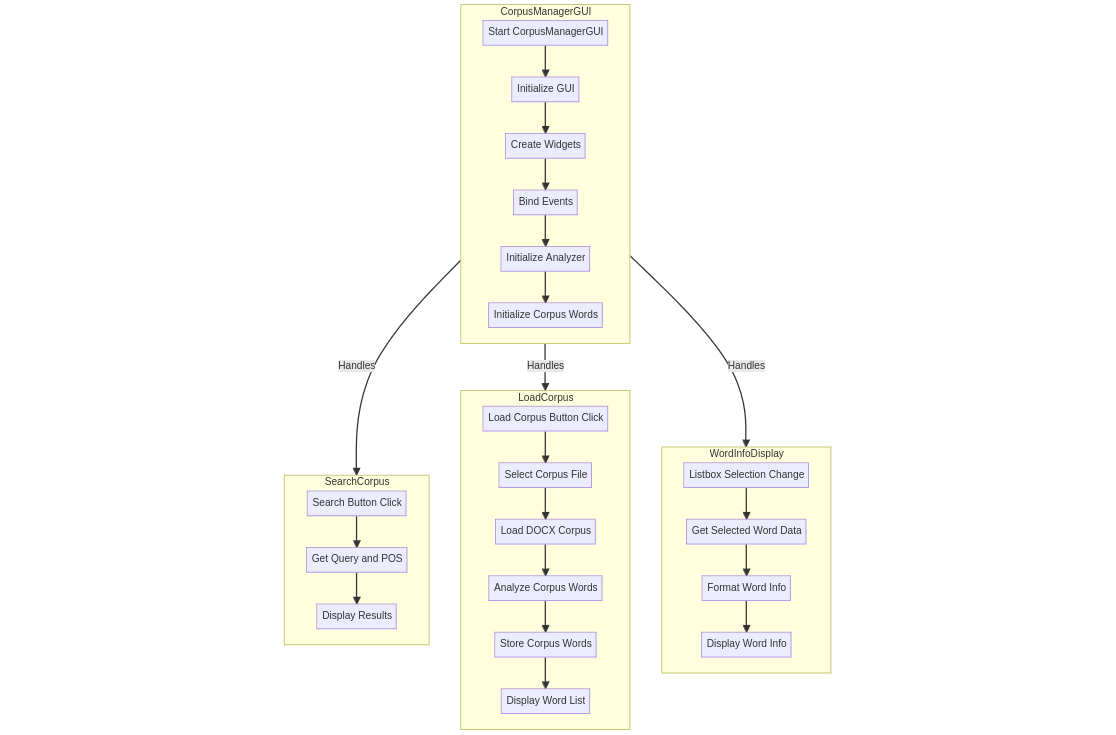
Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

*Поиск по вхождению подстроки в лексеме*

**Код программы:**

from collections import defaultdict  
import re  
import pymorphy3  
from PySide6.QtCore import Qt  
from PySide6.QtWidgets import QProgressDialog  
  
class TextProcessor:  
 def \_\_init\_\_(self, parent=None):  
 self.morph = pymorphy3.MorphAnalyzer()  
 self.parent = parent # Для взаимодействия с UI  
  
 @staticmethod  
 def clean\_text(text):  
 *"""Удаляет лишние пробельные символы (\n, \t и т. д.)"""* return re.sub(r'\s+', ' ', text).strip()  
  
 def parse\_word(self, word):  
 *"""Кэшированный анализ слова"""* return self.morph.parse(word)[0]  
  
 def generate\_concordance(self, word, corpus):  
 *"""Генерирует первое вхождение конкордансного списка для слова"""* corpus = self.clean\_text(corpus)  
 pattern = r'\b' + re.escape(word) + r'\b'  
 sentences = re.split(r'(?<=[.!?])\s+', corpus)  
 return next((s for s in sentences if re.search(pattern, s)), None)  
  
 def process\_text(self, words, corpus):  
 *"""Обрабатывает текст без многопоточности, с прогресс-баром"""* word\_info\_dict = {} # Храним информацию о словах  
 parsed\_words = {} # Сохраняем результаты морфологического анализа  
 word\_freq, lemma\_freq = defaultdict(int), defaultdict(int)  
  
 # Прогресс-бар  
 progress = QProgressDialog("Обработка текста...", "Отмена", 0, len(words)\*2, self.parent)  
 progress.setWindowTitle("Прогресс")  
 progress.setWindowModality(Qt.WindowModal)  
 progress.show()  
  
 # Первый проход: сохраняем результаты морфологического анализа и подсчитываем частоты  
 for word in words:  
 parsed\_word = self.parse\_word(word)  
 parsed\_words[word] = parsed\_word # Сохраняем результаты для каждого слова  
 word\_freq[word] += 1  
 lemma\_freq[parsed\_word.normal\_form] += 1  
 progress.setValue(progress.value() + 1)  
  
 # Второй проход: создаем информацию о словах  
 for index, word in enumerate(words):  
 if progress.wasCanceled():  
 break  
  
 parsed\_word = parsed\_words[word] # Извлекаем результаты из кэша  
 lemma = parsed\_word.normal\_form  
  
 # Получаем частоту для слова и леммы  
 wordform\_freq = f"{word\_freq[word]}/{len(words)}"  
 lemma\_freq\_value = f"{lemma\_freq[lemma]}/{len(words)}"  
  
 if word in word\_info\_dict:  
 word\_info\_dict[word]["frequency"]["wordform\_frequency"] = wordform\_freq  
 word\_info\_dict[word]["frequency"]["lemma\_frequency"] = lemma\_freq\_value  
 progress.setValue(progress.value() + 1)  
 continue  
  
 concordance = self.generate\_concordance(word, corpus)  
  
 stem = parsed\_word.normalized.word # Основа (лемма)  
 ending = word[len(stem):]  
  
 word\_info\_dict[word] = {  
 "word": word,  
 "lemma": lemma,  
 "part\_of\_speech": self.determine\_pos(parsed\_word.tag.POS),  
 "grammatical\_categories": {  
 "gender": self.determine\_gender(parsed\_word.tag.gender),  
 "number": self.determine\_number(parsed\_word.tag.number),  
 "case": self.determine\_case(parsed\_word.tag.case),  
 "tense": self.determine\_tense(parsed\_word.tag.tense),  
 "role": self.determine\_role(parsed\_word.tag.POS, parsed\_word.tag.case, parsed\_word.tag.tense)  
 },  
 "morphological\_characteristics": {  
 "stem": stem,  
 "ending": ending  
 },  
 "frequency": {  
 "wordform\_frequency": wordform\_freq,  
 "lemma\_frequency": lemma\_freq\_value  
 },  
 "concordance": concordance  
 }  
  
 # Обновление прогресса  
 progress.setValue(progress.value() + 1)  
  
 progress.setValue(len(words)\*2)  
 return sorted(word\_info\_dict.values(), key=lambda x: x["word"])  
  
 @staticmethod  
 def determine\_role(pos, case, tense):  
 if pos == "NOUN": # Существительное  
 if case == "nomn": # Именительный падеж  
 return "Подлежащее"  
 if case in [  
 "gent",  
 "datv",  
 "accs",  
 "ablt",  
 "loct",  
 ]: # Родительный, дательный, винительный, творительный, предложный  
 return "Дополнение"  
 elif pos == "ADJF": # Полное прилагательное  
 return "Определение"  
 elif pos == "ADJS": # Краткое прилагательное  
 return "Именная часть составного сказуемого"  
 elif pos == "COMP": # Компаратив  
 return "Определение (сравнительная степень)"  
 elif pos == "VERB": # Глагол (личная форма)  
 if tense == "pres" or tense == "futr": # Настоящее или будущее время  
 return "Сказуемое"  
 if tense == "past": # Прошедшее время  
 return "Сказуемое (прошедшее время)"  
 elif pos == "INFN": # Инфинитив  
 return "Сказуемое (инфинитив)"  
 elif pos == "PRTF": # Полное причастие  
 return "Определение (причастный оборот)"  
 elif pos == "PRTS": # Краткое причастие  
 return "Именная часть составного сказуемого"  
 elif pos == "GRND": # Деепричастие  
 return "Обстоятельство (деепричастный оборот)"  
 elif pos == "NUMR": # Числительное  
 if case == "nomn": # Именительный падеж  
 return "Подлежащее (числительное)"  
 return "Дополнение (числительное)"  
 elif pos == "ADVB": # Наречие  
 return "Обстоятельство"  
 elif pos == "NPRO": # Местоимение-существительное  
 if case == "nomn": # Именительный падеж  
 return "Подлежащее (местоимение)"  
 return "Дополнение (местоимение)"  
 elif pos == "PRED": # Предикатив  
 return "Сказуемое (предикатив)"  
 elif pos == "PREP": # Предлог  
 return "Предлог"  
 elif pos == "CONJ": # Союз  
 return "Союз"  
 elif pos == "PRCL": # Частица  
 return "Частица"  
 elif pos == "INTJ": # Междометие  
 return "Междометие"  
 else:  
 return "-"  
  
 @staticmethod  
 def determine\_pos(pos):  
 *"""Переводит часть речи на русский"""* pos\_map = {  
 "NOUN": "имя существительное",  
 "ADJF": "имя прилагательное (полное)",  
 "ADJS": "имя прилагательное (краткое)",  
 "COMP": "компаратив",  
 "VERB": "глагол (личная форма)",  
 "INFN": "глагол (инфинитив)",  
 "PRTF": "причастие (полное)",  
 "PRTS": "причастие (краткое)",  
 "GRND": "деепричастие",  
 "NUMR": "числительное",  
 "ADVB": "наречие",  
 "NPRO": "местоимение-существительное",  
 "PRED": "предикатив",  
 "PREP": "предлог",  
 "CONJ": "союз",  
 "PRCL": "частица",  
 "INTJ": "междометие",  
 }  
 return pos\_map.get(pos, "-")  
  
 @staticmethod  
 def determine\_case(case):  
 *"""Переводит падеж на русский"""* case\_map = {  
 "nomn": "именительный",  
 "gent": "родительный",  
 "datv": "дательный",  
 "accs": "винительный",  
 "ablt": "творительный",  
 "loct": "предложный",  
 "voct": "звательный",  
 "gen2": "второй родительный",  
 "acc2": "второй винительный",  
 "loc2": "второй предложный",  
 }  
 return case\_map.get(case, "-")  
  
 @staticmethod  
 def determine\_number(number):  
 *"""Переводит число на русский"""* number\_map = {"sing": "единственное", "plur": "множественное"}  
 return number\_map.get(number, "-")  
  
 @staticmethod  
 def determine\_tense(tense):  
 *"""Переводит время на русский"""* tense\_map = {"pres": "настоящее", "past": "прошедшее", "futr": "будущее"}  
 return tense\_map.get(tense, "-")  
  
 @staticmethod  
 def determine\_gender(gender):  
 *"""Определяет род на русском"""* gender\_map = {  
 "masc": "мужской",  
 "femn": "женский",  
 "neut": "средний"  
 }  
 return gender\_map.get(gender, "неизвестно")



**Вывод:** в ходе выполнения лабораторной работы освоил принципы построения корпусов текстов, виды разметки и способы аннотирования, инструменты работы с корпусами текстов.