Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

За шестой семестр

По дисциплине: «Естественно-языковой интерфейс ИС»

# Тема: «Разработка автоматизированной системы формирования словаря естественного языка»

Выполнил:

Студент 3 курса

Группы ИИ-23

Романюк А. П

Проверил:

Булей Е. В.

Брест 2025

**Ход работы**

Список слов, упорядоченный по алфавиту и включающий только лексемы с дополнительно оформленными записями о месте и роли данного слова в составе предложения. К такой информации относится описание того, каким членом предложения может быть данное слово и в какой форме (падеж, число, время и т.п.). Например, если это существительное в именительном падеже, то оно может выступать в роли подлежащего; если это существительное в родительном падеже, то оно может быть дополнением; если это прилагательное, то оно может быть определением и т.п.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

**Код программы:**

**Gui.py**

import re  
  
from PySide6.QtWidgets import (QApplication, QFileDialog, QHBoxLayout,  
 QHeaderView, QMainWindow, QMessageBox,  
 QPushButton, QTableWidget, QTableWidgetItem,  
 QTextEdit, QVBoxLayout, QWidget)  
  
from data\_manager import DataManager  
from text\_processor import TextProcessor  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("Лексический анализатор с ролями слов")  
 self.setGeometry(100, 100, 800, 600)  
 self.text\_processor = TextProcessor()  
 self.original\_data = None  
 self.init\_ui()  
  
 def init\_ui(self):  
 *"""Инициализация пользовательского интерфейса с улучшенным расположением элементов."""* # Поле для ввода текста  
 self.text\_edit = QTextEdit(self)  
 self.text\_edit.setPlaceholderText(  
 "Введите текст или выберите файл для обработки..."  
 )  
 self.text\_edit.setMinimumHeight(100)  
  
 # Кнопки управления файлами  
 self.open\_file\_button = QPushButton("Выбрать файл", self)  
 self.process\_button = QPushButton("Обработать текст", self)  
 self.save\_button = QPushButton("Сохранить результат", self)  
 self.load\_button = QPushButton("Загрузить словарь", self)  
  
 self.open\_file\_button.clicked.connect(self.open\_file\_dialog)  
 self.process\_button.clicked.connect(self.process\_text)  
 self.save\_button.clicked.connect(self.save\_results)  
 self.load\_button.clicked.connect(self.load\_results)  
  
 # Поле поиска и кнопка  
 self.search\_edit = QTextEdit(self)  
 self.search\_edit.setPlaceholderText("Поиск...")  
 self.search\_edit.setFixedHeight(30) # Делаем поле поиска маленьким  
 self.search\_button = QPushButton("Искать", self)  
 self.search\_button.clicked.connect(self.search\_results)  
  
 # Таблица для отображения результатов  
 self.table = QTableWidget(self)  
 self.table.setColumnCount(8)  
 self.table.setHorizontalHeaderLabels(  
 [  
 "Лексема",  
 "Словоформа",  
 "Часть речи",  
 "Падеж",  
 "Число",  
 "Время",  
 "Роль в предложении",  
 "Частота",  
 ]  
 )  
 #self.table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)  
  
 # Размещение кнопок в горизонтальный layout  
 file\_buttons\_layout = QHBoxLayout()  
 file\_buttons\_layout.addWidget(self.open\_file\_button)  
 file\_buttons\_layout.addWidget(self.process\_button)  
 file\_buttons\_layout.addWidget(self.save\_button)  
 file\_buttons\_layout.addWidget(self.load\_button)  
  
 search\_layout = QHBoxLayout()  
 self.reset\_search\_button = QPushButton("Сбросить поиск", self)  
 self.reset\_search\_button.clicked.connect(  
 lambda: self.search\_edit.setText("") or self.search\_results()  
 )  
  
 search\_layout.addWidget(self.search\_edit, 2)  
 search\_layout.addWidget(self.search\_button, 1)  
 search\_layout.addWidget(  
 self.reset\_search\_button, 1  
 ) # Новая кнопка сброса поиска  
  
 # Основной layout  
 layout = QVBoxLayout()  
 layout.addWidget(self.text\_edit)  
 layout.addLayout(file\_buttons\_layout) # Кнопки работы с файлами  
 layout.addLayout(search\_layout) # Поиск  
 layout.addWidget(self.table) # Таблица занимает оставшееся пространство  
  
 container = QWidget()  
 container.setLayout(layout)  
 self.setCentralWidget(container)  
  
 def process\_text(self):  
 *"""Обработка текста и вывод результатов в таблицу."""* text = self.text\_edit.toPlainText().lower()  
 if text:  
 words = self.extract\_words(text)  
 sorted\_lemmas = self.text\_processor.process\_text(words)  
 self.original\_data = sorted\_lemmas  
 self.update\_table(self.original\_data)  
 else:  
 QMessageBox.warning(self, "Ошибка", "Поле ввода текста пусто!")  
  
 @staticmethod  
 def extract\_words(text):  
 *"""Извлекает слова из текста."""* pattern = re.compile(r"[а-яА-Я]+")  
 return pattern.findall(text)  
  
 def open\_file\_dialog(self):  
 *"""Открытие диалога выбора файла."""* file\_dialog = QFileDialog(self)  
 file\_dialog.setNameFilter("Текстовые файлы (\*.txt \*.rtf \*.pdf \*.doc \*.docx)")  
 file\_dialog.setViewMode(QFileDialog.Detail)  
 if file\_dialog.exec():  
 file\_paths = file\_dialog.selectedFiles()  
 if file\_paths:  
 file\_path = file\_paths[0]  
 self.load\_file(file\_path)  
  
 def load\_file(self, file\_path):  
 *"""Загрузка текста из файла."""* try:  
 text = self.read\_file(file\_path)  
 self.text\_edit.setText(text)  
 except Exception as e:  
 QMessageBox.critical(self, "Ошибка", f"Не удалось загрузить файл: {str(e)}")  
  
 def load\_results(self):  
 *"""Загрузка данных из JSON в таблицу."""* filename, \_ = QFileDialog.getOpenFileName(  
 self, "Открыть файл", "", "JSON файлы (\*.json)"  
 )  
 if filename:  
 try:  
 data = DataManager.load\_data(filename)  
 self.original\_data = data  
 self.update\_table(self.original\_data)  
 QMessageBox.information(self, "Успех", "Данные успешно загружены.")  
 except Exception as e:  
 QMessageBox.critical(  
 self, "Ошибка", f"Не удалось загрузить данные: {str(e)}"  
 )  
  
 def search\_results(self):  
 *"""Поиск данных по ключевому слову"""* keyword = self.search\_edit.toPlainText().strip()  
 if not keyword:  
 self.update\_table(self.original\_data)  
 return  
 filtered\_data = DataManager.search\_data(self.original\_data, keyword)  
 self.update\_table(filtered\_data)  
  
 def read\_file(self, file\_path):  
 *"""Чтение текста из файла."""* if file\_path.endswith(".txt"):  
 with open(file\_path, "r", encoding="utf-8") as file:  
 return file.read()  
 elif file\_path.endswith(".pdf"):  
 import PyPDF2  
  
 with open(file\_path, "rb") as file:  
 reader = PyPDF2.PdfReader(file)  
 return "".join(page.extract\_text() for page in reader.pages)  
 elif file\_path.endswith(".docx"):  
 import docx  
  
 doc = docx.Document(file\_path)  
 return "\n".join([para.text for para in doc.paragraphs])  
 elif file\_path.endswith(".rtf"):  
 import striprtf  
  
 with open(file\_path, "r", encoding="utf-8") as file:  
 rtf\_text = file.read()  
 return striprtf.rtf\_to\_text(rtf\_text)  
 else:  
 raise ValueError("Неподдерживаемый формат файла")  
  
 def save\_results(self):  
 *"""Сохранение результатов в JSON в формате, аналогичном выходному формату process\_text."""* filename, \_ = QFileDialog.getSaveFileName(  
 self, "Сохранить файл", "", "JSON файлы (\*.json)"  
 )  
 if filename:  
 try:  
 DataManager.save\_data(self.original\_data, filename)  
 QMessageBox.information(self, "Успех", "Данные успешно сохранены.")  
 except Exception as e:  
 QMessageBox.critical(  
 self, "Ошибка", f"Не удалось сохранить данные: {str(e)}"  
 )  
  
 def update\_table(self, sorted\_lemmas):  
 *"""Обновление таблицы с результатами."""* self.table.setRowCount(0)  
 for lemma\_info in sorted\_lemmas:  
 lemma = lemma\_info["lemma"]  
 for form\_info in lemma\_info["word\_forms"]:  
 row\_position = self.table.rowCount()  
 self.table.insertRow(row\_position)  
 self.table.setItem(row\_position, 0, QTableWidgetItem(lemma)) # Лексема  
 self.table.setItem(  
 row\_position, 1, QTableWidgetItem(form\_info["word"])  
 ) # Словоформа  
 self.table.setItem(  
 row\_position, 2, QTableWidgetItem(form\_info["pos"])  
 ) # Часть речи  
 self.table.setItem(row\_position, 3, QTableWidgetItem(form\_info["case"]))  
 self.table.setItem(  
 row\_position, 4, QTableWidgetItem(form\_info["number"])  
 )  
 self.table.setItem(  
 row\_position, 5, QTableWidgetItem(form\_info["tense"])  
 )  
 self.table.setItem(  
 row\_position, 6, QTableWidgetItem(form\_info["role"])  
 ) # Роль в предложении  
 self.table.setItem(  
 row\_position, 7, QTableWidgetItem(str(form\_info["count"]))  
 ) # Частота  
 self.table.resizeColumnsToContents()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication([])  
 window = MainWindow()  
 window.show()  
 app.exec()

**Вывод:** в ходе выполнения лабораторной работы освоил принципы разработки прикладных сервисных программ для решения задачи автоматического лексического и лексико-грамматического анализа текста естественного языка.