МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ИНТЕЛЕКТУАЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

ОТЧЁТ

По лабораторной работе № 3

Выполнил:
Студент группы ИИ-22
Копанчук Евгений Романович
Проверил:
Булей Е. В.

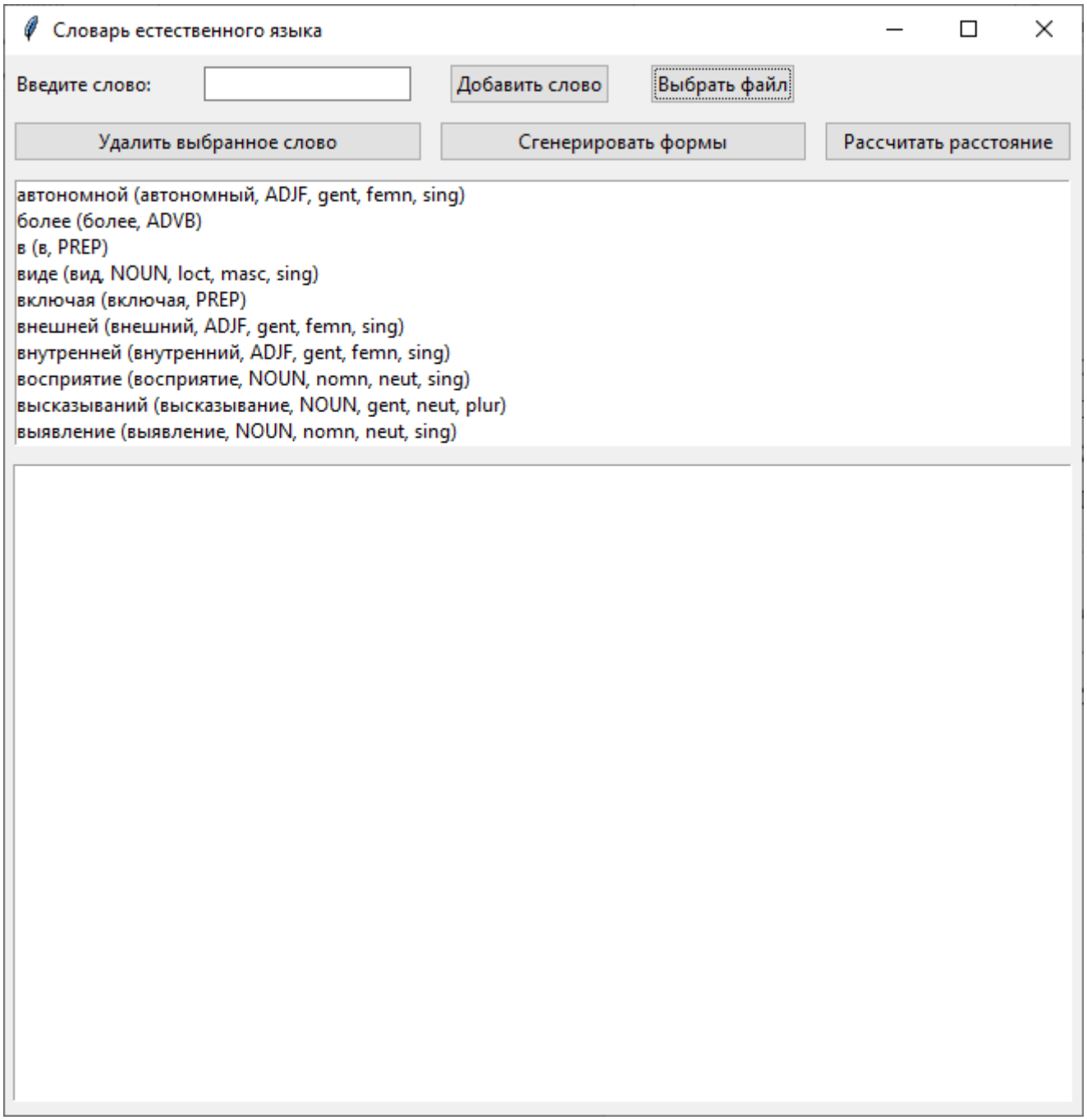
Брест – 2024

Ход работы

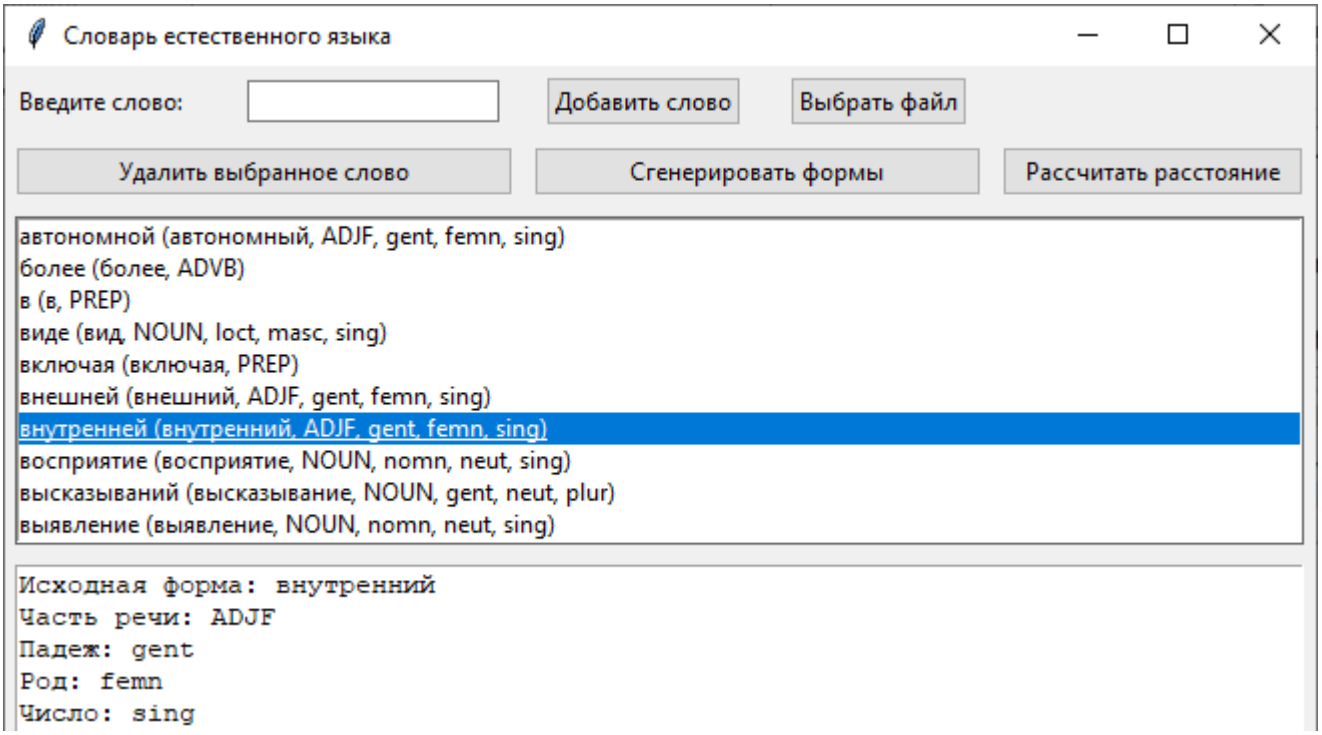
Задание:

1. *Создание программной реализации решения задачи о редакционном расстоянии*
   Необходимо разработать программу, которая будет решать задачу о редакционном расстоянии между словами. Редакционное расстояние между двумя строками определяется как минимальное количество операций вставки, удаления и замены символов, необходимых для преобразования одной строки в другую.

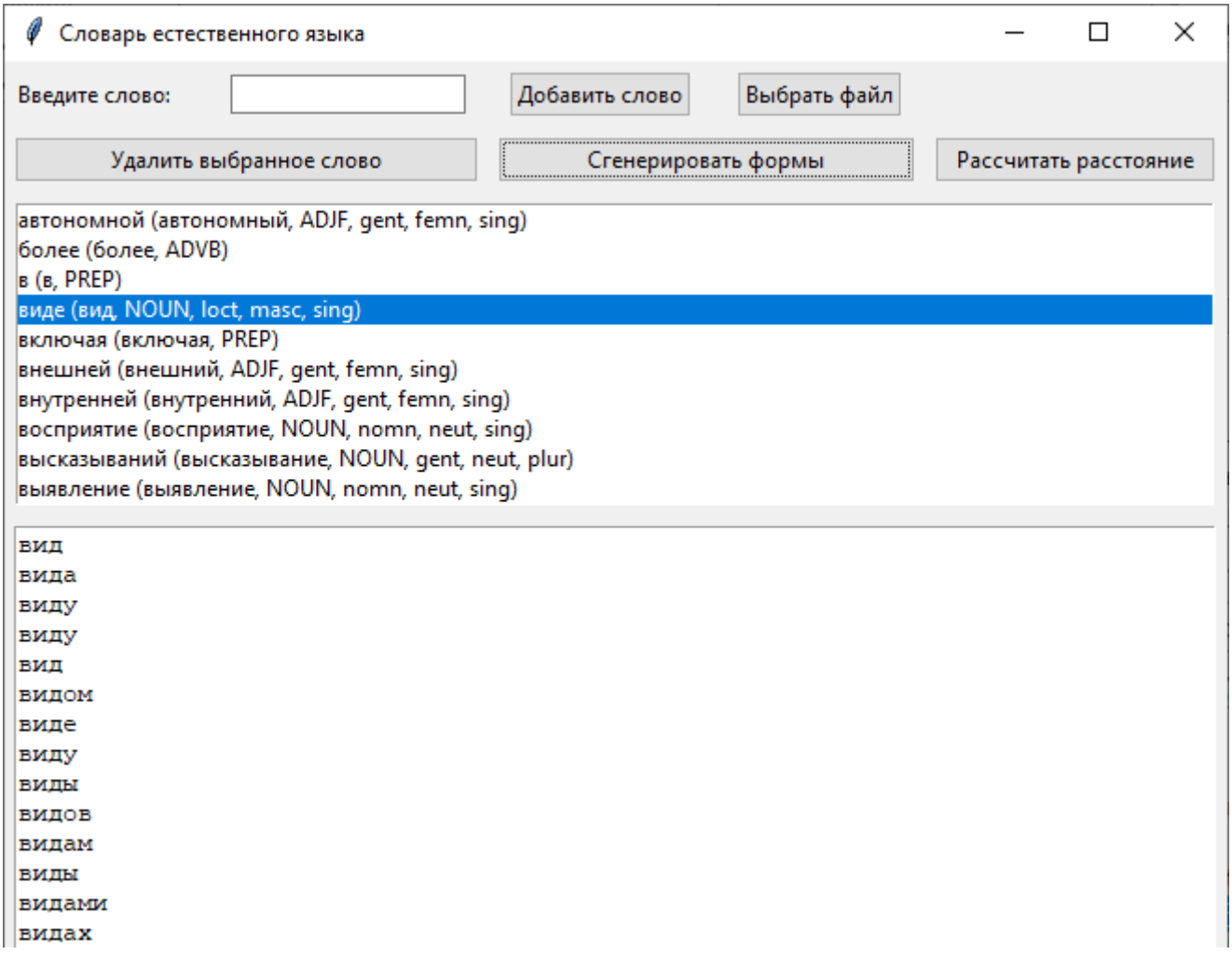
2. *Проектирование внешнего интерфейса автоматизированной системы*
   Необходимо спроектировать внешний интерфейс автоматизированной системы, которая будет осуществлять решение задачи о редакционном расстоянии.



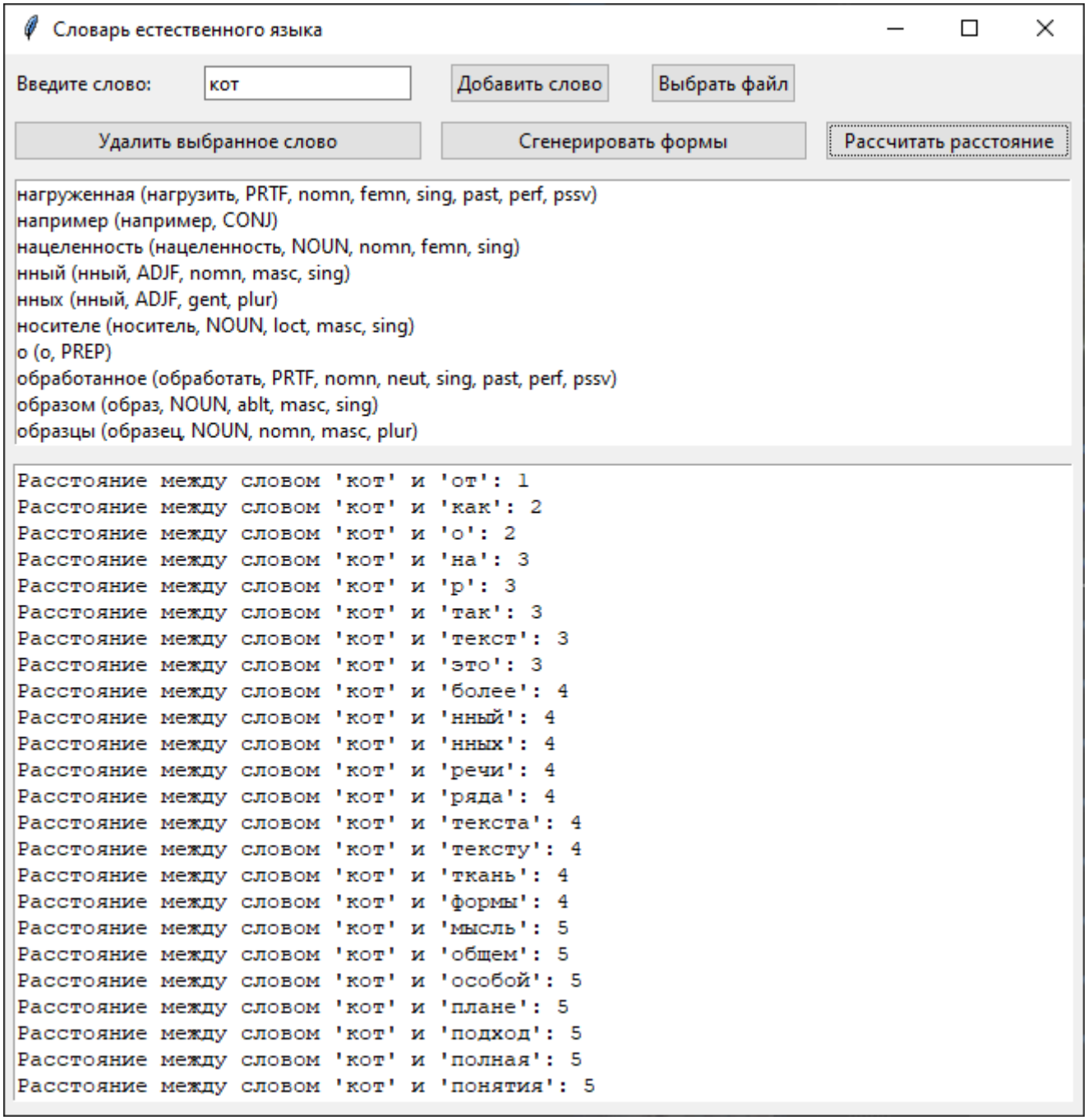
Приложение с загруженным фалом docx



Характеристики выбранного слова



Формы выбранного слова



Расстояние Левенштейна между введённым словом и всеми в словаре

Код программы:

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
import pymorphy3
import docx2txt
import re
import Levenshtein

class Word:
    def __init__(self, word):
        self.word = word
        self.morphy = []

        morph_analyzer = pymorphy3.MorphAnalyzer()
        parsed_word = morph_analyzer.parse(word)[0]

        tags = [
            {'name': 'Исходная форма', 'value': parsed_word.normal_form},
            {'name': 'Часть речи', 'value': parsed_word.tag.POS},
            {'name': 'Падеж', 'value': parsed_word.tag.case},
```

```
            {'name': 'Род', 'value': parsed_word.tag.gender},
            {'name': 'Число', 'value': parsed_word.tag.number},
            {'name': 'Время', 'value': parsed_word.tag.tense},
            {'name': 'Вид', 'value': parsed_word.tag.aspect},
            {'name': 'Лицо', 'value': parsed_word.tag.person},
            {'name': 'Наклонение', 'value': parsed_word.tag.mood},
            {'name': 'Залог', 'value': parsed_word.tag.voice}
        ]

        for tag in tags:
            if tag['value'] is not None:
                self.morphy.append(tag)

class Application(tk.Tk):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.title("Словарь естественного языка")
        self.geometry("655x645")

        self.words = []

        self.word_label = ttk.Label(self, text="Введите слово:")
        self.word_label.grid(row=0, column=0, padx=5, pady=5, sticky=tk.W)

        self.word_entry = ttk.Entry(self)
        self.word_entry.grid(row=0, column=1, padx=5, pady=5)

        self.add_word_button = ttk.Button(self, text="Добавить слово", command=self.add_word)
        self.add_word_button.grid(row=0, column=2, padx=5, pady=5)

        self.select_file_button = ttk.Button(self, text="Выбрать файл",
command=self.select_file)
        self.select_file_button.grid(row=0, column=3, padx=5, pady=5)

        self.delete_word_button = ttk.Button(self, text="Удалить выбранное слово",
command=self.delete_word)
        self.delete_word_button.grid(row=1, column=0, columnspan=2, padx=5, pady=5,
sticky=tk.W+tk.E)

        self.generate_forms_button = ttk.Button(self, text="Сгенерировать формы",
command=self.generate_forms)
        self.generate_forms_button.grid(row=1, column=2, columnspan=2, padx=5, pady=5,
sticky=tk.W+tk.E)

        self.calculate_distance_button = ttk.Button(self, text="Рассчитать расстояние",
command=self.calculate_distance)
        self.calculate_distance_button.grid(row=1, column=4, padx=5, pady=5, sticky=tk.W+tk.E)

        self.word_listbox = tk.Listbox(self, selectmode=tk.SINGLE)
        self.word_listbox.grid(row=2, column=0, columnspan=5, padx=5, pady=5,
sticky=tk.W+tk.E+tk.N+tk.S)
        self.word_listbox.bind("<<ListboxSelect>>", self.show_word_info)

        self.word_info_text = tk.Text(self, wrap="word")
        self.word_info_text.grid(row=3, column=0, columnspan=5, padx=5, pady=5,
sticky=tk.W+tk.E+tk.N+tk.S)

    def add_word(self):
        word = self.word_entry.get().strip()
        if word:
            if word not in [w.word for w in self.words]:
                word_object = Word(word)
                self.words.append(word_object)
                self.sort_words_list()
                word_info = f"{word} ({', '.join(tag['value'] for tag in word_object.morphy)})"
                self.show_word_info()

    def delete_word(self):
        selected_index = self.word_listbox.curselection()
        if selected_index:
            del self.words[selected_index[0]]
            self.word_listbox.delete(selected_index)

    def generate_forms(self):
        selected_index = self.word_listbox.curselection()
```

```
        if selected_index:
            word = self.words[selected_index[0]].word
            morph_analyzer = pymorphy3.MorphAnalyzer()
            parsed_word = morph_analyzer.parse(word)[0]

            forms = parsed_word.lexeme
            self.word_info_text.delete(1.0, tk.END)
            for form in forms:
                self.word_info_text.insert(tk.END, f"{form.word}\n")

    def calculate_distance(self):
        input_word = self.word_entry.get().strip()
        if input_word:
            distances = []
            for word_object in self.words:
                distance = Levenshtein.distance(input_word, word_object.word)
                distances.append((word_object.word, distance))
            distances.sort(key=lambda x: x[1])  # Сортировка по возрастанию расстояния
            result = "\n".join(f"Расстояние между словом '{input_word}' и '{word}': {distance}"
for word, distance in distances)
            self.word_info_text.delete(1.0, tk.END)
            self.word_info_text.insert(tk.END, result)

    def select_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Word files", "*.docx"), ("All
files", "*.*")])
        if file_path:
            self.load_from_docx(file_path)

    def load_from_docx(self, filename):
        text = docx2txt.process(filename).lower()

        pattern = re.compile(r'[а-яА-Я]+')
        words = pattern.findall(text)

        for word in words:
            if word not in [w.word for w in self.words]:
                word_object = Word(word)
                self.words.append(word_object)
                self.sort_words_list()
                word_info = f"{word} ({', '.join(tag['value'] for tag in word_object.morphy)})"
                self.word_listbox.insert(tk.END, word_info)

    def sort_words_list(self):
        self.words.sort(key=lambda x: x.word.lower())
        self.word_listbox.delete(0, tk.END)
        for word_object in self.words:
            word_info = f"{word_object.word} ({', '.join(tag['value'] for tag in
word_object.morphy)})"
            self.word_listbox.insert(tk.END, word_info)

    def show_word_info(self, event):
        selected_index = self.word_listbox.curselection()
        if selected_index:
            if selected_index[0] < len(self.words):
                word = self.words[selected_index[0]]
                self.word_info_text.delete(1.0, tk.END)
                for tag in word.morphy:
                    self.word_info_text.insert(tk.END, f"{tag['name']}: {tag['value']}\n")

if __name__ == "__main__":
    app = Application()
    app.mainloop()
```