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**

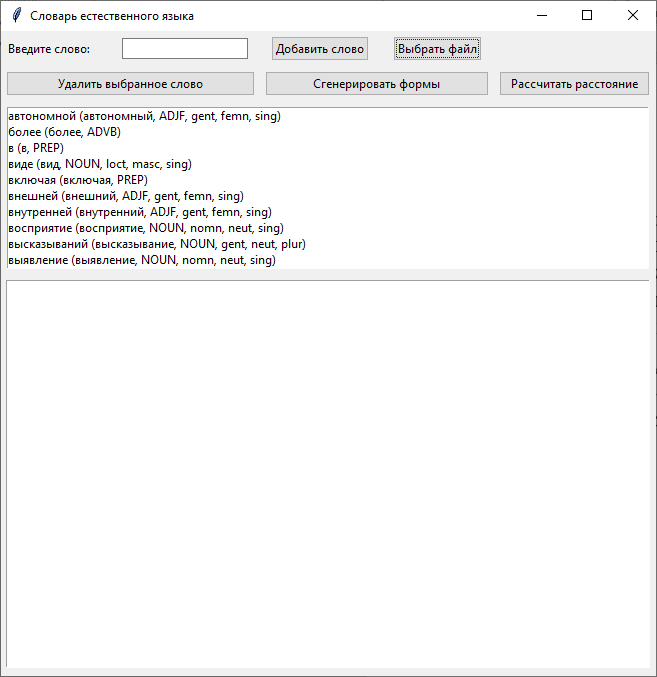
**Задание:**

1. *Создание программной реализации решения задачи о редакционном расстоянии*

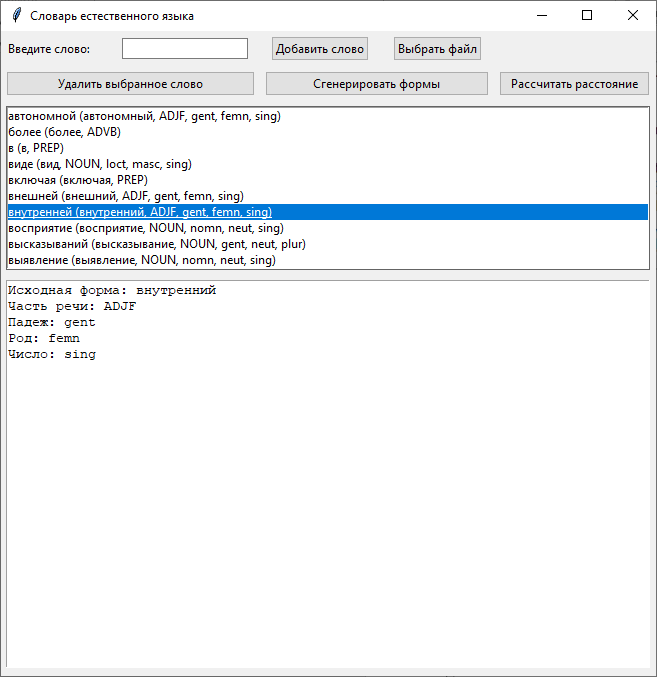
Необходимо разработать программу, которая будет решать задачу о редакционном расстоянии между словами. Редакционное расстояние между двумя строками определяется как минимальное количество операций вставки, удаления и замены символов, необходимых для преобразования одной строки в другую.

1. *Проектирование внешнего интерфейса автоматизированной системы*

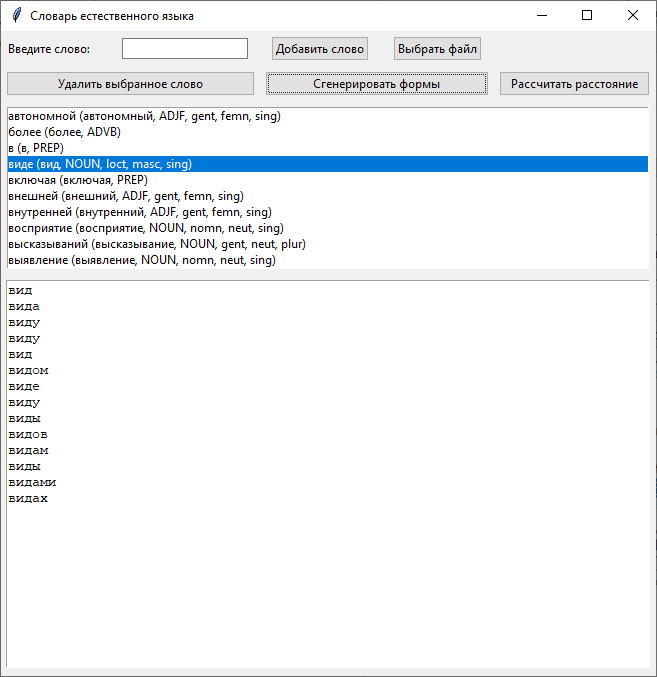
Необходимо спроектировать внешний интерфейс автоматизированной системы, которая будет осуществлять решение задачи о редакционном расстоянии.



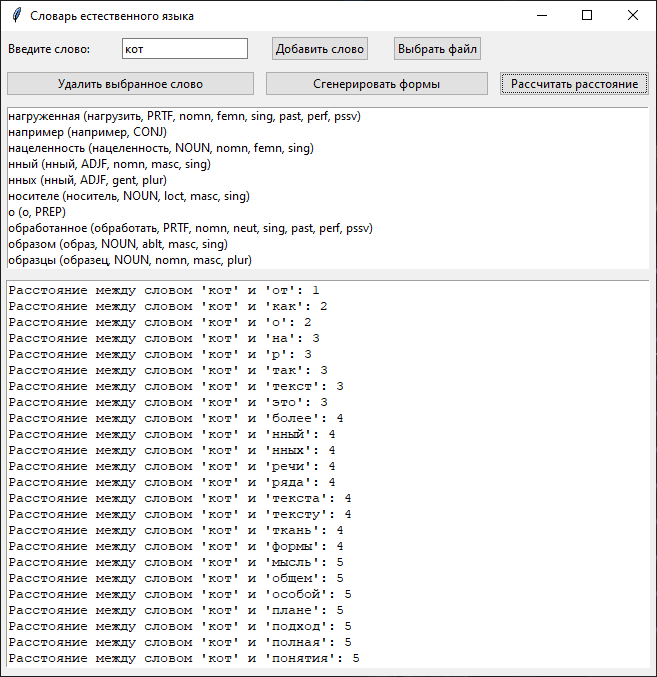
*Приложение с загруженным фалом docx*



*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()