Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №4

По дисциплине: «Естественно-языковой интерфейс ИС»

Тема: «Разработка автоматизированной системы синтаксического анализа текста естественного языка»

Выполнил:

Студент 3 курса

Группы ИИ-23

Романюк А. П.

Проверил:

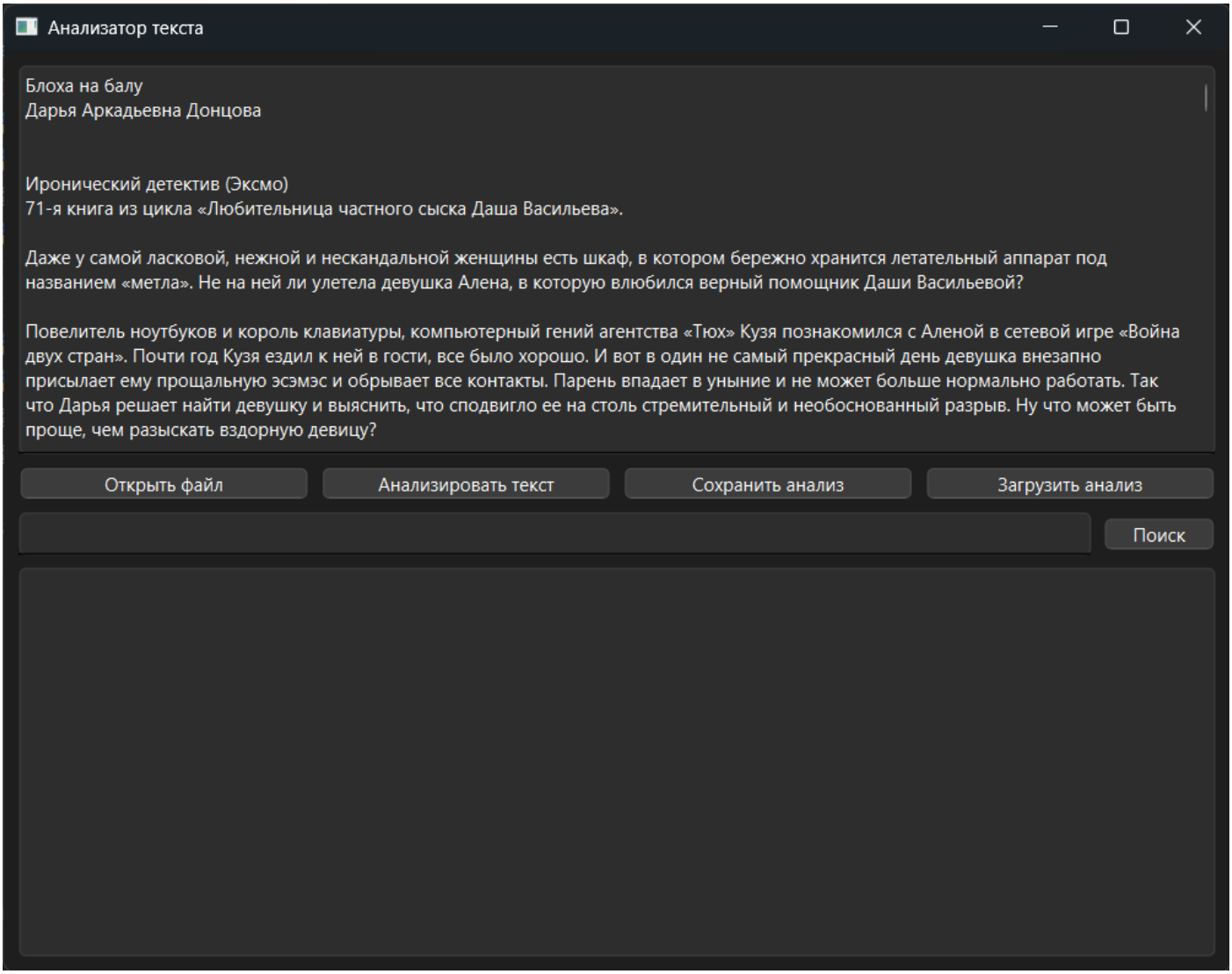
Булей Е. В.

Брест 2025

Ход работы

Задание:

1. *Входные данные* – текст заданного естественного языка;
2. *Выходные данные* – структуры, полученные при проведении автоматического синтаксического анализа предложений входного текста
3. Взаимодействие с пользователем посредствам графического интерфейса (интерфейс должен быть интуитивно-понятным и дружественным пользователю)



Приложение с загруженным русским текстом

Анализ предложения

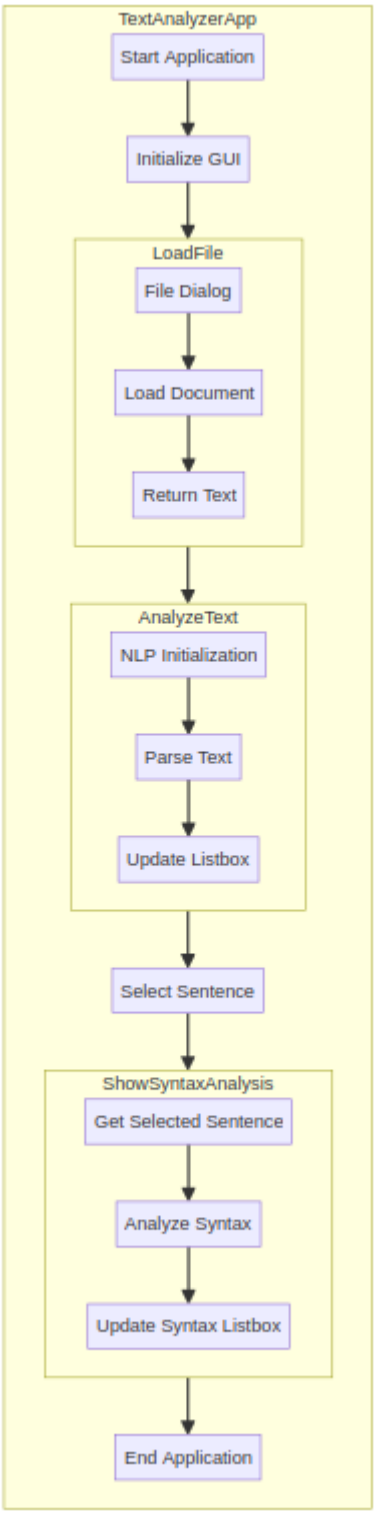
Предложение: Ну что может быть проще, чем разыскать вздорную девицу?

	Слово	Лемма	Часть речи	Падеж	Число	Время	
1	ну	ну	частица	-	-	-	Частица
2	что	что	союз	-	-	-	Союз
3	может	мочь	глагол (личная форма)	-	единственное	настоящее	Сказуемое
4	быть	быть	глагол (инфинитив)	-	-	-	Сказуемое (ин
5	проще	простой	компаратив	-	-	-	Определение
6	чем	чем	союз	-	-	-	Союз
7	разыскать	разыскать	глагол (инфинитив)	-	-	-	Сказуемое (ин
8	вздорную	вздорный	имя прилагательное (полное)	винительный	единственное	-	Определен
9	девицу	девица	имя существительное	винительный	единственное	-	Дополнени

Синтаксический анализ выбранного предложения

Код программы:

```
 from collections import defaultdict
import re
import pymorphy3


class TextProcessor:
    def __init__(self):
        self.morph = pymorphy3.MorphAnalyzer()

    def process_text(self, text):
        """Обрабатывает текст, разбивая на предложения и слова"""
        sentences = self.split_into_sentences(text)
        processed_sentences = []

        for sentence in sentences:
            words = self.extract_words(sentence)
            processed_words = []

            for word in words:
                parsed_word = self.morph.parse(word)[0]
                lemma = parsed_word.normal_form
                pos = parsed_word.tag.POS
                case = parsed_word.tag.case
                number = parsed_word.tag.number
                tense = parsed_word.tag.tense

                processed_words.append({
                    "word": word,
                    "lemma": lemma,
                    "pos": self.determine_pos(pos),
                    "case": self.determine_case(case),
                    "number": self.determine_number(number),
                    "tense": self.determine_tense(tense),
                    "role": self.determine_role(pos, case, tense)
                })

            processed_sentences.append({
                "sentence_text": sentence,
                "words": processed_words
            })

        return processed_sentences

    @staticmethod
    def split_into_sentences(text):
        """Разбивает текст на предложения"""
```



```
    return [s.strip() for s in re.split(r'(?<=[.!?])\s+', text) if s.strip()]

@staticmethod
def extract_words(sentence):
    """Извлекает слова из предложения"""
    return re.findall(r'\b[а-яА-ЯёЁ]+\b', sentence.lower())

@staticmethod
def prepare_data(lemmas_info):
    """Метод для сборки обработанных слов в красивый словарик"""
    result = []
    for lemma, word_forms in lemmas_info.items():
        lemma_info = {"lemma": lemma, "word_forms": []}
        for word, forms in word_forms.items():
            for (pos, case, number, tense, role), count in forms.items():
                lemma_info["word_forms"].append(
                    {
                        "word": word,
                        "pos": pos,
                        "case": case,
                        "number": number,
                        "tense": tense,
                        "role": role,
                        "count": f"{count}/{len(lemmas_info)}",  # Частота встречаемости
                    }
                )
        result.append(lemma_info)
    result.sort(key=lambda x: x["lemma"])
    return result

def determine_role(self, 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 "-"
```

```
def determine_pos(self, 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, "-")

def determine_case(self, 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, "-")

def determine_number(self, number):
    """Переводит число на русский"""
    number_map = {"sing": "единственное", "plur": "множественное"}
    return number_map.get(number, "-")

def determine_tense(self, tense):
    """Переводит время на русский"""
    tense_map = {"pres": "настоящее", "past": "прошедшее", "futr": "будущее"}
    return tense_map.get(tense, "-")
```

Вывод: в ходе выполнения лабораторной работы освоил принципы разработки прикладных сервисных программ для решения задачи анализа текста естественного языка.