МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
_____
(підпис)
червня 2024 р.

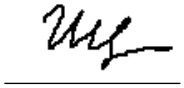
КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Алгоритмічне та програмне забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації»
здобувача групи ІН - 01 Слатвицького Дениса Едуардовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.


(підпис) Денис СЛАТВИЦЬКИЙ

Керівник,
кандидат фізико-математичних наук, доцент Сергій ШАПОВАЛОВ


(підпис)

Суми – 2024

Сумський державний університет

Факультет електроніки та інформаційних технологій

вечірньої форм навчання

Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Ігор ШЕЛЕХОВ

_____

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»

здобувача групи ІН-01 Слатвицького Дениса Едуардовича

1. Тема роботи: «Алгоритмічне та програмне забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації»
затверджую наказом по СумДУ від *«22» квітня 2024 р. № 0414-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 14 липня 2024 року*
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд та аналіз алгоритмів «стемінгу», «токенізації» та обробки слова. 3) Розробка алгоритму та моделі нормалізації тексту української мови. 4) Тестування та аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис Дати	
		Задання видав	Завдання прийняв

7. Дата видачі завдання «06» травня 2024 р.

Завдання прийняв до виконання _____ Керівник _____

(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз проблеми предметної області, постановка й формування завдань дослідження		
2	Огляд та аналіз алгоритмів «стемінгу», «токенізації» та обробки слова		
3	Розробка алгоритму та моделі нормалізації тексту української мови		
4	Тестування та аналіз результатів		
5	Оформлення пояснювальної записки до кваліфікаційної роботи		

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 44 стр., 4 таб., 7 рис., 3 додатків, 16 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є надзвичайно актуальною, оскільки присвячена розв’язанню важливої практичної задачі створення інструментів для обробки текстів українською мовою, адже обсяг текстової інформації невпинно зростає, що викликає потребу в ефективних та точних алгоритмах для обробки та аналізу текстових даних.

Об’єкт дослідження — процес нормалізації текстів в системах аналізу та обробки україномовної інформації.

Мета роботи — розробка алгоритмічного та програмного забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації.

Методи дослідження — алгоритми обробки природної мови (NLP), методи стемінгу, інструменти для морфологічного аналізу, орфографічної та граматичної корекції, а також програмні бібліотеки для роботи з текстовими даними і створення користувацьких інтерфейсів.

Результати — розроблено модель нормалізації тексту, який зчитує фрагменти українського тексту, обробляє їх, методами аналізу допомагає визначити певні морфологічні, орфографічні, та граматичні помилки, здійснює стемінг, для пошуку кореневої частини, обробляючи та стемінгуючи вхідні тексти. Створено простий додаток з користувацьким інтерфейсом, що дозволяє користувачу, напряму взаємодіяти з отриманими даними. Проведено тестування алгоритму, оцінка його ефективності, його адаптування під українську мову, виділенні основні неточності, та проблеми даної обробки.

МОДЕЛЬ НОРМАЛІЗАЦІЇ ТЕКСТУ УКРАЇНСЬКОЇ МОВИ, JAVA, GUI, STEMMING.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи. У зв’язку з активним та ефективним розвитком різних інструментів, автоматичної обробки текстів, та прогресом нейронних мереж, що можуть обробляти тексти, виникає необхідність у створенні різних інструментів для обробки текстів української мови. В основі більшості існуючих моделей використовуються різні підходи текстового аналізу, проте етап токенізації лежить в основі роботи абсолютно всіх нейронних мереж, тому виникає потреба у розвитку в даному напряму, та дослідження саме методу стемінгу для державної мови. За початок такого дослідження обрано розробка саме модуля нормалізації, та на його базі проводити дослідження та доопрацювання алгоритму обробки слова, виділяючи основні проблеми, та неточності.

Актуальність. Тема кваліфікаційної роботи є надзвичайно актуальною у зв'язку з необхідністю розробки ефективних інструментів для обробки українських текстів. Розвиток інформаційних технологій та значне збільшення обсягу текстової інформації вимагають надійних алгоритмів для аналізу та корекції текстів. Українська мова має свої специфічні особливості, які потребують розробки окремих методів та алгоритмів для її коректної обробки. На сьогодні існує обмежена кількість якісних інструментів, що здатні виконувати морфологічний аналіз, корекцію орфографічних і граматичних помилок, а також визначення частин мови для української мови. За основу взятий доволі перспективний у своїй основі алгоритм, який у свою чергу може стати основою у розв’язанні такої проблеми, як повна автоматизація обробки текстів української мови, визначення помилок, та їх повне або часткове виправлення.

Об’єкт дослідження. Процес нормалізації текстів в системах аналізу та обробки україномовної інформації.

Предмет дослідження. Алгоритми та програмне забезпечення для нормалізації українських текстів, зокрема стемінг, морфологічний аналіз, корекція орфографічних і граматичних помилок.

Мета дослідження. Розробка алгоритмічного та програмного забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації. Завданням роботи є розробка адаптація та дослідження алгоритму стемінгу, створення модулю нормалізації тексту на його основі.

Методи дослідження. Методи дослідження включають алгоритми обробки природної мови (NLP), методи стемінгу, інструменти для морфологічного аналізу, орфографічної та граматичної корекції, а також програмні бібліотеки для роботи з текстовими даними і створення користувацьких інтерфейсів.

Структура та обсяг роботи. Робота складається з вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 75 сторінок, включаючи 15 ілюстрацій та 5 таблиць.

Практичне значення отриманих результатів. Практичне значення роботи полягає у створенні інтерактивного додатку, який дозволяє користувачам здійснювати морфологічний аналіз, корекцію орфографічних та граматичних помилок українських текстів, а також використовувати стемінг для пошуку спільнокореневих слів. Цей додаток може бути використаний як у навчальних закладах, так і в інших установах для покращення якості обробки українських текстів. Адаптування алгоритму може допомогти у подальшому розвитку обробки тексту українською мовою. Результати тестування даного додатку, нагально показують сутність та проблематику дослідження таких інструментів для державної мови, слугують основою для подальшої роботи в цій сфері.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Аналіз існуючих рішень

Розробка інформаційної системи для нормалізації текстів українською мовою є актуальним завданням у сучасних умовах розвитку інформаційних технологій. Існуючі рішення для обробки текстів часто орієнтовані на англомовний контент, що створює значні труднощі для обробки текстів іншими мовами, включаючи українську. Відсутність якісних інструментів для аналізу та обробки україномовних текстів знижує ефективність роботи з великими обсягами даних, ускладнює навчання та вивчення мови, а також обмежує можливості автоматизації процесів у різних галузях. Варто також зорієнтуватись на існування методів обробки текстів, які використовують розповсюдженні нейромережі. Адже основою їх роботи є велика кількість хмарних баз даних наповнені різного виду тексту, і методами токенізації тексту, підбираються найбільш популярні частинки слів та текстів. В залежності від ступеня навчання такої системи відповідь на запит користувача може бути більш точною, однак це не відміняє фактів гарантії правильності текстів, виправлення орфографічних та морфологічних помилок.

Стемінг («Stemming») - це процесом обробки тексту, спрямованим на визначення і видалення афіксів (приставок, суфіксів, закінчень) зі слова, залишаючи його основу або корінь [9]. Одним з перспективних напрямків є розробка модулів нормалізації тексту, які включають алгоритми стемінгу, морфологічного аналізу, корекції орфографічних та граматичних помилок. Важливим аспектом таких систем є їхня дружність до користувача, що передбачає наявність інтуїтивно зрозумілого інтерфейсу та можливості інтеграції з іншими програмними продуктами. Користувач сам може ознайомитись зі структурою слова, отримати основні результати обробки слова, порівняти з можливими варіантами. Алгоритм обробки слова дозволяє бути основою для різного виду інструментарію, в напрямах вивчення української мови, та пришвидшення автоматичної обробки текстів.

1.2 Системи нормалізації тексту української мови

Одним з розповсюджених програм що має за мету вирішення схожої проблеми є онлайн платформа «Grammarly», розробники якої такий функціонал, а тим паче програмний код даного функціоналу, у вільному доступі не відкрила. Проте у вільному доступі є для української мови алгоритм «токенізації тексту», який і лежить в основі роботи «Штучного інтелекту – асистенту онлайн платформи Grammarly». [1]

Spacy-Ukrainian є модулем для обробки текстів українською мовою, який базується на бібліотеці SpaCy. Він включає токенізацію, частиномовний аналіз і залежність синтаксису, але обмежений підтримкою для морфологічного аналізу і нормалізації форм слів. [2]

LemmatizerUA або «Lemmatization and Morphological Analyzer for Ukrainian» ще один інструмент, спеціалізований на лематизації української мови, обробляє слово та ділить його на частини, мето якого є отримання основної інформації щодо його творення. Є більш наближенним, проте основною проблемою даної системи, в тому що його функціонал обмежений, він знаходиться в стадії розробки, не враховує контексту, і синтаксичних залежностей.

DeepPavlov є комфортним конструктором діалогових систем і машинного навчання, який включає моделі для обробки текстів різними мовами, включаючи українську. Однак точність і надійність для української мови можуть бути нижчими порівняно з більш популярними мовами, такими як англійська чи німецька. [4]

Автором алгоритму «Stemming» є Портер Стеммер, який у 1980 році був опублікований автором, для англійської мови. З чого був народжений проект, назва якого співпадає з назвою широкорозповсюдженою бібліотекою для природної обробки мови «Snowball», даний алгоритм був переведений на різні мови, без алгоритмічної зміни, у тому числі для мови «ворожого сусіда». [7-8]

На жаль аналогів систем для нормалізації текстів української мови що основані на процесі обробки слова «Stemming» , у загальному доступі майже не існує. Проте існує велика кількість програм, що основані на NLP (Обробка природної мови), та алгоритму стемінгу для обробки текстів англійської мови, та їх не ідеальні аналоги для обробки інших мов у тому числі мов слов'янської групи. [5]

1.3 Постановка задачі

Інтернет джерела та форуми (блоги програмістів) подають різні відповіді на запит «Алгоритм stemming», і можна знайти певні адаптації під українську мову, але варто зазначити неточність та невідповідність таких алгоритмів, щодо базових правил словотворення української мови. Тому як висновок, такої пошукової роботи, прийнято рішення про адаптацію алгоритму стемінгу під обробку тексту українською мовою, що буде виконувати ряд функціоналу, що продемонструє принцип обробки слова державної мови.

Отже основним завданням роботи стало адаптація та дослідження алгоритму Портера Стеммера для обробки слова української мови. Роботу було розподіллено на етапи та під завдання:

- реалізація базового алгоритму стемінгу: переробити алгоритм, який буде відповідати основним принципам стемінгу, зокрема заснований на алгоритмі Портера з адаптаціями для української мови, врахувати особливості української морфології, такі як різні форми слова в залежності від частини мови та морфологічні особливості суфіксів та префіксів;

- імплементація алгоритму в програмний код: написати програму, яка здійснює введення слова для стемінгу та послідовну обробку кроків стемінгу відповідно до розробленого алгоритму;

- реалізація патернів тестування: провести набір тестів для перевірки правильності та ефективності роботи алгоритму, ключити в тести варіанти з різними типами слів (іменників, дієслів, прикметників тощо) для перевірки коректності стемінгу;

- додавання визначення частин мови: реалізувати можливість визначення частини мови для під час стемінгу, імплементувати модуль, який автоматично визначає частину мови слова для подальшого використання у стемінгу;

- інтеграція графічного інтерфейсу: створити графічний інтерфейс користувача для тестування та демонстрації роботи програми, забезпечити можливість введення слова через інтерфейс, відображення результатів стемінгу та виправлення введених слів у реальному часі.

- тестування та вдосконалення алгоритму: провести інтенсивне тестування розробленого програмного забезпечення для виявлення потенційних проблем та недоліків стемінгу, здійснити необхідні корекції та вдосконалення алгоритму на основі результатів тестування.

- встановити певний результат аналізу, підбити підсумки, та скорегувати подальше можливе використання програми в майбтньому.

У результаті виконання всіх завдань буде створено програмне забезпечення для стемінгу текстів українською мовою, яке відповідатиме сучасним стандартам ефективності та точності. Це дозволить полегшити автоматизацію обробки української мови та покращити роботу з великими обсягами тексів у сучасних інформаційних системах.

2 МЕТОДИ ВИРІШЕННЯ ЗАДАЧІ

2.1 Stemming Algorithm, NLP, шаблони, токенізація слів

Алгоритм стемінгу використовується для зменшення слова до його базової форми (стему) шляхом видалення аффіксів. Це дозволяє знизити різноманітні варіації одного слова до єдиної форми, що спрощує подальший аналіз тексту. Основним завданням є створення максимально ефективного алгоритму який буде виконувати дану функцію, для слова української мови, адаптуючи існуючий алгоритм Портера Стеммера під орфографічні та морфологічні правила мови. [7-8]

Даний алгоритм вирішує ряд завдань:

- **у**згодженість даних: зменшення кількості варіацій слів до стемів допомагає покращити узгодженість даних для подальшого аналізу та обробки, у вбільшості випадків стемінг відбувається до корня, проте це не є основою алгоритму;

- зменшення розміру словника**:** стемінг дозволяє зменшити розмір словника, що покращує швидкість і ефективність обробки тексту;

- пошукові запити: використання стемінгу у пошукових системах дозволяє знайти усі варіанти слова, включаючи його різноманітні форми.

NLP (Обробка природної мови, Natural Language Processing) використовується для розуміння та генерації природної мови людьми або комп'ютерами. Вона дозволяє виконувати наступні завдання:

- токенізація: розбиває текст на окремі слова або токени;

- аналіз семантики: визначення значення слів та їх взаємозв’язків;

- класифікація тексту: категорій та типів тексту. [13]

У роботі даний алгоритм і є частковим інструментом роботи NLP, покриваючи функції часткового скорочення слова, що може бути токеном для майбутнього пошуку, аналіз тексту, виявлення морфологічних ознак слова, що може надати інформацію про синтаксис та контекст, який так потрібен для роботи нейронних мереж. Гібрид такого алгоритму та інформаційної системи

обробки природної мови може стати досить ефективним інструментом для швидкого аналізу слів та виправлення різного типу помилок.

Основою роботи також є Шаблони (Regular Expression), які активно використовуються всюди для обробки текстів, знаходження висновків, та сталих патернів. Розробники широко використовують регулярні вирази для виділення та аналізу текстів, фільтрації даних, та перевірки форматів. [14]

Усі ці методи, інструменти, та поняття так чи інакше застосовуються для розробки програмного забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації.

2.2 Мова програмування

На етапі обробки та пошуку теоретичних аспектів алгоритму, його приклади, та аналізу документації, було прослідковано, що основним інструментом для написання схожих інструментів є Python про те варто розуміти, що як правило такі системи одразу інтегровані до бібліотек вже певної обробки слів, і фактично для української мови можуть виникнути певні неточності. Інші мови програмування є теж доволі ефективними у написанні та розробки подібних алгоритмів, однак якщо обирати наприклад мову C++, яка була б швидша за мову Java, вона б потребувала набагато більшого поглиблення в бібліотеки, щоб ускладнило процес коригування самого алгоритму, та створення користувацького інтерфейсу.

Обираючи мову Java можна виділити такі основні переваги:

- Java має добру продуктивність і підтримує багатопотоковість, що робить її ефективною для обробки великих обсягів даних, надає потужні інструменти для роботи з колекціями даних, а також для виконання операцій паралельного програмування;

- Java має широкі можливості для реалізації графічних і консольних інтерфейсів, для графічного інтерфейсу часто використовуються бібліотеки Swing або JavaFX, які надають розширені можливості для створення інтерактивних додатків;

- Java відома своєю «крос-платформенністю», що означає, що програми, написані на Java, можуть працювати на різних операційних системах без змін у коді. Вона підтримується на Windows, macOS і різних дистрибутивах Linux.

Саме тому ця мова підходить для розробки програмного забезпечення, яке має обробляти великі дані, підтримувати графічні і консольні інтерфейси, і бути крос-платформним.

2.3 Середовище розробки

Використано було доволі популярну IDE для Java, IntelliJ IDEA. Це середовище надає багато функціональних можливостей для комфортної розробки, включаючи автодоповнення коду, налагодження, підтримку систем контролю версій та інтеграцію з іншими інструментами розробки.

У дослідженні було використано бібліотеку Swing для створення графічного інтерфейсу користувача (GUI) у Java. Swing є стандартною бібліотекою для створення графічних додатків у Java, яка входить до складу JDK. Swing надає набір компонентів GUI, таких як кнопки, текстові поля, панелі, етикетки, і багато інших, що дозволяє створювати складні та зручні для користувача інтерфейси. Простота гнучкість в створенні графічного інтерфейсу переважають над об’ємом коду. Єдиним недоліком такої бібліотеки в порівнянні з більш сучасними це її потужність, але наша програма є наглядною і доволі скромною у дизайні.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Алгоритм та його адаптація

Розробка програмного забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації включає кілька етапів: переведення алгоритму на українську мову, пошук основних правил та напрямів обробки слів на правилами української мови, попереднє тестування, підбиття проміжних висновків, удосконалення алгоритму, відповідно до отриманих даних, порівняння його з токенізацією, створення графічного інтерфейсу, підбиття підсумків, аспектів розвитку та використання подібного модуля.

Аналізуючи основний алгоритм Портера Стеммера створений для мови js виділено основні моменти обробки слова: [15]

- перевіряється кожен крок довжина основи, вона має містити щонайменше 1 голосну літеру та 3 літери в основі;

- одразу відкидаються закінчення «s», «sses», «ies», найпростіший спосіб позбутися деяких дієслів, форм іменників та іменників множини;

- видалення закінчень «ing», «eed», «ed» - відкидаються майже всі прикметники та дієприкметники, а також дієслова третьої форми;

- відкидання інших афіксів, які з точки зору словотворення української мови є суфіксами іменників та прикметників;

- та додаткові поодинокі випадки видалення певних букв відповідно до правил подвоєння та подібного. (Таблиця 3.1)

Таблиця 3.1 Поетапна обробка слова англійської відповідно до алгоритму Портера Стеммера. Дані взяті з [7-8, 15]

Етап обробки слова	**Патерни частин мови**	**Опис**
Попередня обробка	-	Перевірка довжини слова. Зміна першої літери "y" на велику.
Крок 1a: Видалення закінчень множини та третьої особи однини	s, sses, ies	Відкидання закінчень дієслів та іменників множини.
Крок 1b: Видалення закінчень минулого часу та дієприкметників	ing, eed, ed	Видалення закінчень дієслів, прикметників та дієприкметників.
Крок 1c: Обробка слів, що закінчуються на y	y	Заміна y на i, якщо попередня частина слова містить голосну.
Крок 2: Заміна специфічних закінчень	ational, tional, enci, anci, izer, bli, alli, entli, eli, ousli, ization, ation, ator, alism, iveness, fulness, ousness, aliti, iviti, biliti, logi	Заміна специфічних суфіксів на коротші форми.
Крок 3: Додаткове видалення закінчень	icate, ative, alize, iciti, ical, ful, ness	Видалення закінчень прикметників та іменників.
Крок 4: Видалення певних закінчень	al, ance, ence, er, ic, able, ible, ant, ement, ment, ent, ou, ism, ate, iti, ous, ive, ize	Видалення закінчень, що є суфіксами іменників та прикметників.
Додаткові випадки видалення	ll, e	Видалення певних букв відповідно до правил подвоєння та видалення закінчення e.
Повернення до початкової літери	-	Зміна першої літери на малу, якщо вона була змінена на велику на початку.

Звісно такий алгоритм не нагромаджує і доволі простий у розборі та використанні у швидшому пошуку слів, стемінгуючи їх, спрощуючи до основи. Отриману базову частинку можна використовувати у токенізації, для подальшого пошуку, і вирішення проблем лексичної, морфологічної, орфографічної та граматичної перевірки слова.

Однак адаптація алгоритму під українську мову викликає велику кількість проблем, які до кінця досі залишаються не вирішеними. Пов’язане це з тим що українська мова має набагато ширший запас афіксів, варіантів словотворення, існування винятків, та різних форм відмінків і тд.

Основою для створення алгоритму для обробки слова українською мовою постало правила творення закінчень для різних частин мови. Наступним етапом є виділення афіксів, при чому префікси якщо такі присутні варто відкидати спочатку, проте існує вірогідність відкидання частини основи, що відноситься до недоліку даної адаптації. Результат адаптації подано у Таблиці 3.2.

Таблиця 3.2 Обробка слова української мови, результат адаптивності під правила української мови. (Дані зібрані автором, відповідно до правил словотворення української мови [16], та аналізу алгоритму з різних джерел [7-8, 15]. Та використовуються у коді програми Додаток А.)

Етап обробки слова	**Патерни частин мови**	**Опис**
Попередня обробка	-	Зміна слова на нижній регістр, видалення апострофів та заміна деяких символів.
Крок 1: Видалення префіксів	зі, з, зо, с, пре, при, прі, роз, без, через, від, од, між, над, об, перед, під, понад, пред, із	Видалення префіксів зі слова.
Крок 2: Видалення закінчень доконаного виду (perfectiveground)	ив, ивши, ившись, ыв, ывши, ывшись, `((?<=[ая])(в	вши
Крок 3: Видалення прикметників та дієприкметників	ими, ій, ий, а, е, ова, ове, ів, є, їй, єє, еє, я, ім, ем, им, ім, их, іх, ою, йми, іми, у, ю, ого, ому, ої	Видалення закінчень прикметників.

Продовження таблиці 3.2

Крок 4: Видалення закінчень дієприкметників	ий, ого, ому, им, ім, а, ій, у, ою, ій, і, их, ими, их	Видалення закінчень дієприкметників.
Крок 5: Видалення закінчень дієслів	сь, ся, ив, ать, ять, у, ю, ув, ав, али, учи, ячи, вши, ши, е, ме, ати, яти, є	Видалення закінчень дієслів.
Крок 6: Видалення закінчень іменників	а, ев, ов, е, ями, ами, еи, и, ей, ой, ий, й, иям, ям, ием, ем, ам, ом, о, у, ах, иях, ях, ы, ь, ию, ью, ю, ия, ья, я, і, ові, ї, ею, єю, ою, є, еві, ем, єм, ів, їв, ю	Видалення закінчень іменників.
Крок 7: Видалення суфіксів	ість	Видалення суфіксу "ість".
Крок 8: Видалення закінчення и	и$	Видалення закінчення "и".
Крок 9: Видалення суфіксів, якщо вони відповідають деривативному патерну	[^аеиоуюяіїє][аеиоуюяіїє]+[^аеиоуюяіїє]+[аеиоуюяіїє].*(?<=о)сть?$	Видалення суфіксу "ість", якщо слово відповідає деривативному патерну.
Крок 10: Видалення закінчення ь	ь$	Видалення закінчення "ь".
Крок 11: Видалення закінчення ейше або нн	ейше?$, нн$	Видалення закінчення "ейше" або зміна "нн" на "н".
Повернення до початкової літери	-	Якщо перша літера була змінена на велику, змінюється назад на малу.

Виходячи з отриманої покрокової нормалізації, більшість випадків вже можна покрити відповідними патернами, а етапи обробки допомагають на певних етапах та в основних випадках визначити, частину мову та основу слова, яка як правило і є коренем слова.

3.2 Регулярні вирази

Відповідно до поетапної обробки використовуються покроково патерни, які представлені у вигляді регулярних виразів, завдання яких є на певних етапах отримання інформації про частину мови слова, та покроковий стемінг відповідно до вказаних регулярних виразів.

Основними з них є:

- голосні букви, для легшої перевірки існування закінчень, та базової частки слова;

- доконаний вид дієслова, та інші форми відповідно до закінчень, цей етап відкидає закінчення які зустрічаються тільки в цьому випадку, а отже одразу можна виділити, частини мов які мають ці закінчення в словотвореннях;

- рефлексивне дієслово, якщо ми пропустили в минулому кроці дієслово, програма визначає його за таким закінченням;

- основні закінчення прикметників, проте далеко не всі, словотворення та відмінювання залишається проблемою, яку все ще потребує дослідження та коригування;

- основні суфікси та закінчення дієприкметників, теж регулярний вираз який потребує доопрацювання;

- після певних етапів стемінгу суфіксів та закінчень, потрібна більш обширна перевірка на дієслово, вказані основні закінченні та дієслова. Такий підхід є більш ефективним, однак проблема зміни закінчення та суфіксів дієслова залишається відкритим;

- всі префікси, на цьому етапі стемінг слова відбувається зліва, алгоритмічно не вирішена проблема щодо подвійних та гібридних префіксів, адже часто зустрічаються випадки коли нібито префікс забирається у кореневої частини слова;

- основні закінчення та суфікси іменника, даний етап потребує подальшого дослідження, та адаптації до слів іншомовного запозичення. Як правило стемінг

приводить слова до іменника, або кореня з якого найпростіше зробити іменник. Адже велика кількість іменників не мають суфіксів;

- паттерн деривацій них суфіксів містить ряд закінчень, які можуть свідчити про існування ще одного суфікса, якщо виконується умова присутності більше одного складу в перевірочному слові.

Таблиця 3.3 Патерни мовою Java та їх опис функції в програмі

Патерн мовою Джава	Опис
private final String vowel = "аеиоуюяіїє";	Патерн vowel містить голосні літери українського алфавіту. Використовується для перевірки наявності голосних літер у слові.
private final String perfectiveground = "(ив|ивши|ившись|ыв|ывши|ывшись ((?<=[ая])(в|вши|вшись)))$";	Патерн perfectiveground визначає закінчення, що вказує на доконаний вид дієслова. Використовується для перевірки наявності цього закінчення та видалення його з слова.
private final String reflexive = "(с[яьи])$";	Патерн reflexive розпізнає закінчення, що вказує на рефлексивний відмінок дієслова. Використовується для видалення цього закінчення з слова.
private final String adjective = "(ими|ій|ий|а|е|ова|ове |ів|є|їй|єє|еє|я|ім |ем|им|ім|их|іх|ою|йми |іми|у|ю|ого|ому|ої)$";	Патерн adjective визначає закінчення, що вказує на прикметник. Використовується для перевірки наявності прикметникового закінчення та його видалення.
private final String participle = "(ий|ого|ому|им|ім |а|ій|у|ою|ій| і|их|ими|их)$";	Патерн participle розпізнає закінчення, що вказує на дієприкметник. Використовується для видалення дієприкметникового закінчення з слова.
private final String verb = "(сь|ся|ив|ать|ять|у|ю |ав|али|учи|ячи|вш и|ши|е|ме|ати|яти|є)$";	Патерн verb визначає закінчення, що вказує на дієслово. Використовується для перевірки наявності дієсловного закінчення та його видалення.

3.3 Алгоритм Стемінгу

Вхідними даними для безпосередньої моделі нормалізації тексту є слово, а очікуваним результатом є набір афіксів та закінчень, а також базової частини слова, яке в нашому випадку може бути кореневою частиною слова, або основою. [9]

Алгоритм включає кілька етапів попередньої обробки, перевірок та змін. Поетапно відбуваються така підготовка слова: (див. рис.3.1)

- перетворення слова в нижній регістр;
- відкидання апострофа;
- перевірка на наявність літер «мови сусіда»;

```
private String ukstemmerSearchPreprocess(String word) {
    word = word.toLowerCase();
    String originalWord = word; // Оригінальне слово

    word = word.replace( target: "'", replacement: "");
    if (!word.equals(originalWord)) {
        final_messsage += "replace '";
    }

    originalWord = word;

    word = word.replace( target: "ë", replacement: "e");
    if (!word.equals(originalWord)) {
        final_messsage += " ë";
    }

    originalWord = word;

    word = word.replace( target: "ъ", replacement: "ї");
    if (!word.equals(originalWord)) {
        final_messsage += " ъ";
    }
```

Рисунок 3.1 -Метод функція якого підготувати слово до стемінгу

Під час кожного етапу стемінгу є перевірка, існування голосних літер в основі слова, адже коренева частина слова, відповідно до основ методів «токенізації тексту» має містити в собі склад, а склад в українській мові, має включати в себе голосну букву, дана перевірка використовує патерн голосних літер. Ще однією умовою перевірки слова є наявність щонайменше 3х символів, адже коренева частина, як правило, довша за 2 літери.

```
if (!word.matches(".*[" + vowel + "].*")) {
    result = word; // Зберегти слово як результат
    final_messsage += "Нема голосних, скоріш за все слова не існує";
    return word;
}
```

Алгоритм стемінгу включає наступні етапи:

- перевірка на наявність префіксу;

```
Pattern prefixPattern = Pattern.compile(prefix);
Matcher prefixMatcher = prefixPattern.matcher(word);

// Check if the prefix pattern is found
if (prefixMatcher.find()) {
    String prefix = word.substring(0, prefixMatcher.end());
    // Remove the prefix from the word
    word = word.substring(prefixMatcher.end());
    System.out.println(word);
    // Update the result
}
```

- виділення кореневої частини слова

```
Pattern pattern = Pattern.compile(rvre);
Matcher matcher = pattern.matcher(word);

// Якщо у слові знайдено патерн, виконуємо алгоритм стемінгу
if (matcher.find()) {
    String start = word.substring(0, matcher.end());
    RV = word.substring(matcher.end());
```

- перевірка відповідності до патернів певних частин мови:

1) якщо RV відповідає патерну `perfectiveground` (доконаний вид), видаляється відповідне закінчення;

2) якщо RV відповідає патерну `adjective`, видаляються закінчення для прикметників;

3) якщо RV відповідає патерну `participle`, видаляються закінчення для дієприкметників;

4) якщо RV відповідає патерну `verb`, видаляються закінчення для дієслів;

5) якщо RV відповідає патерну `noun`, видаляються закінчення для іменників.

```
if (!s(RV, perfectiveground, "")) {
    // Крок 1: Перевірка чи слово відповідає патерну для perfectiveground
(доконаний вид)
    if (s(RV, adjective, "")) {
        // Перевірка чи слово відповідає патерну для прикметників
        s(RV, participle, ""); // Видалення закінчень для дієприкметників
        final_messsage += "прикметник";
    } else {
        if (!s(RV, verb, "")) {
            s(RV, noun, ""); // Видалення закінчень для іменників
            final_messsage += "іменник";
        }if (!s(RV, verb, "")) {
```

```
            s(RV, noun, ""); // Видалення закінчень для іменників
            final_messsage += "іменник";
        }
    }
    if(final_messsage.isEmpty()){
        final_messsage += "дієслово/друга частина мови";
    }
}
```

- видалення закінчення «и»;

- перевірка наявності суфіксів та подвоєнь (подвоєння скорочується до однієї літери);

```
// Крок 2: видаляємо закінчення "и", якщо воно є
s(RV, "и$", "");

// Крок 3: перевіряємо наявність суфіксів і змінюємо, якщо потрібно
if (RV.matches(derivational)) {
    s(RV, "ість$", "");
    final_messsage += "суфікс ість";
}

// Крок 4: змінюємо закінчення "ь", "ейше", "нн", якщо вони є
s(RV, "ь$", "");
s(RV, "ейше?$", "");
s(RV, "нн$", "н");
```

- формування фінального результату та повернення основи та закінчень.

Відповідно до перевірки патернів, йде певне припущення щодо 3х основних частин мови, для того аби достовірніше розуміти недоліки та додаткові проблеми обробки слів. Відповідно до патернів та обраного умовного оператора, в окрему змінну зберігається результат перевірки, якщо знайшлись співпадіння.

3.4 Пошук стемінгом

Наступним етапом дослідженням алгоритму – це здійснення пошуку використовуючи обробку слова. Принцип такої роботи закладається в тому що отриману основу, ми використовуємо як лексичне розуміння вхідного слова. Аби зрозуміти чи існує тлумачення такого слова, програма перевіряє певний об’єм перевірених даних (перевірені дані – це файли що місять підручники та різні публікації у тому числі в науково-публіцистичному стилі, що пройшли редакцію та видавалися тим чи іншим українським виданням), та порівнює основи слів, якщо співпадіння є вона поверне всі знайдені слова, та словотворення.

Основною метою такого пошуку є показати яка ефективність такого пошуку щодо саме українських слів та текстів, та особисто оцінити відповідність щодо очікуваного результату.

Для вирішення поставленої задачі, було створено примітивний графічний інтерфейс, де користувач робить запит, та як результат отримує перелік слів зі спільною основою, а також саму основу (очікувано кореневу частину слова) та перелік відкинутих афіксів та закінчень. (див. рис. 3.2)

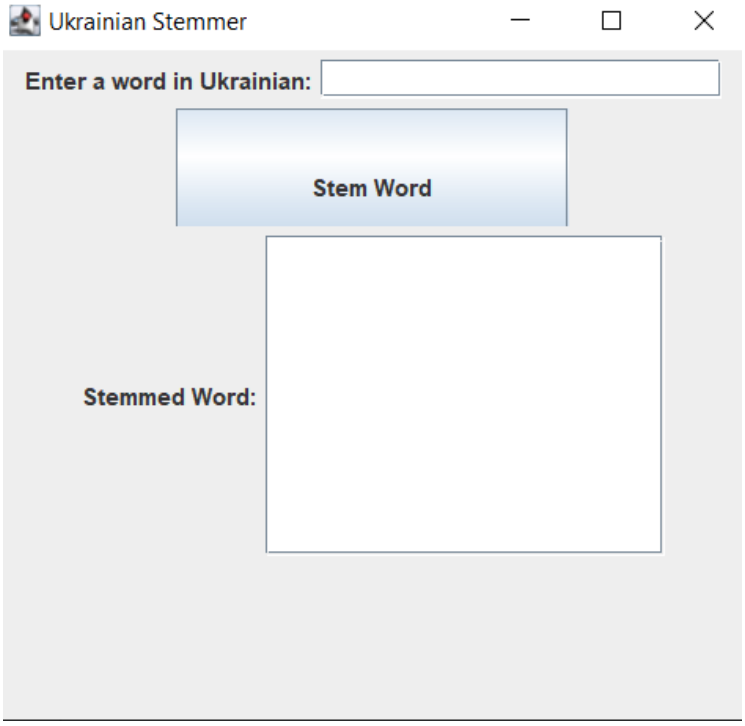


Рисунок 3.2 – Графічний інтерфейс програми

3.5 Результати роботи програми

На даному етапі роботи в основу лягає тестування, та збір отриманих даних, їх аналіз та виявлення наявної ефективності алгоритму та проблем, які треба вирішити в майбутньому.

Запустивши програму ми отримаємо вікно де є можливість ввести певне слово, натиснути кнопку «Stem Word» та отримати певні результати пошуку, стемінгу та частини яку відкинули. (див. рис. 3.3)

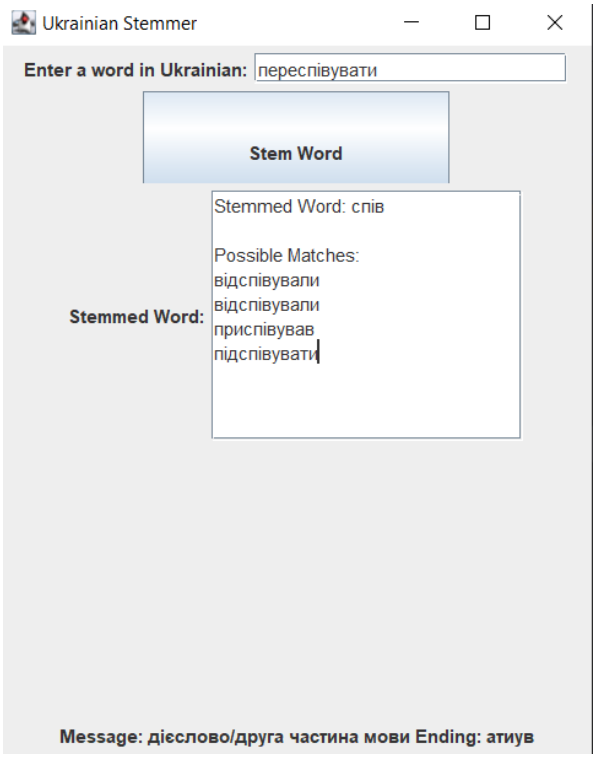


Рисунок 3.3 – Результат пошуку стемінгом

В ході тестування стає зрозумілим які помилки, неточності та конфлікти мають ті чи інші результати запитів, адже в основі нормалізації йде логічне й послідовне прийняття логічних рішень відповідно до виставлених регулярних виразів. Зустрічаються й випадки коли слово обробилось але пошук нічого не показав, а отже слів зі знайденою основою у джерелах не існує. (див. рис. 3.4)

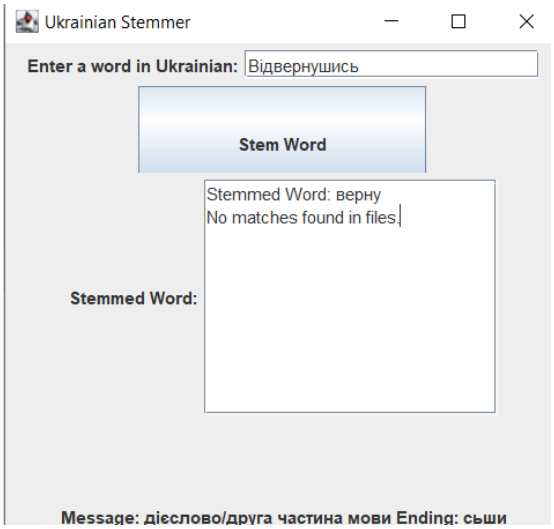


Рисунок 3.4 – Приклад не вдалого пошуку

Простіш представити вибірку певних слів у таблиці та визначити вручну які є помилки та які моменти, даний алгоритм вирішив.

Таблиця 3.4 Результати ручної обробки результатів.

Вхідне слово	Корінь вхідного слова	Помилка/ Неточність	Stemmed слово	Афікси та закінчення що були відкинуті	Результат пошуку за стемінгом	Частина мови
переспівувати	спів	-	спів	ати + ув	відспівували відспівували приспівував підспівувати	дієслово
розсмішити	сміх	х - ш "ти" має бути раніше	смішит	и	-	іменник
терпкий	терп	суфікс к, 3 приголосні поряд	терпк	ий	Терпкий терпкий Терпке терпкі	прикметник
розпираючий	пир	-	пир	ий аюч	спираючись Спираючись спираючись спираючись Спираючись Спираючись Спираючись підпираючи спираючись	прикметник
брати	брат	-	брат	и	брат братами брати братом зібрати брати брати зібрати брати	іменник

Продовження таблиці 3.4

тринадцять	тринадцят	Чисельник	тринадц	я + ть	тринадцять	іменник
слідкуючий	слід	учи ючи суфікси + с тут не префікс	лідкуюч	ий	слідкуючи	прикметник
п'ятий	п'ят	чисельник	пят	апостроф ий	п'ятий п'ятій п'ять п'ятою п'ята п'ять п'ять п'ять п'ят	прикметник
перспектива	перспект	суфікс ив / іменник	перспектив	а	перспектива перспектива перспектива перспектива перспективою	прикметник
відповідальність	відповід	суфікс альн + іст	повідальність	ь	відповідальність відповідальність відповідальність відповідальність відповідальність відповідальністю відповідальність	Іменник

Для полегшення аналізу роботи програми, запропоновано поділити результати на кілька видів, до позитивних віднести:

- стемінг пройшов успішно, основа відповідає кореню, очікуваний результат, пошук показав певну кількість спільнокореневих слів. Даний варіант є найбільш вигідним та виграшним, адже пошук по кореню слова відбувається максимально ефективно пропонуючи користувачу велику кількість спільнокореневих слів;

- стемінг пройшов успішно, основа в неповній мірі відповідає кореню, проте афікси та закінчення виділенні, а пошук видав певний результат. Даний результат вважається теж ідеальним, адже основа алгоритму не є отримання саме кореню, а отримання такої частини слова, яке б могло бути частиною інших слів;

- стемінг пройшов успішно, отримано корінь, або майже корінь слова, відкинута певну кількість афіксів та закінчень, проте пошук не видав результатів. Даний варіант показує наскільки стемінг був ефективним, в залежності

наближеності до кореня слова, можна зробити висновок, або слів з таким коренем не існує, або отримано не занадто гарний стемінг, результати такого виду перевірки мають зберігатися.

Дані варіанти є позитивному тому що основна мета стемінгу та пошуку відбулася успішна, про те в залежності від специфіки слова ми отримуємо різні види результатів.

Розглядаючи негативні результати роботи моделі, варто віднести:

- стемінг не відбувся взагалі, або довжина отриманої основи більша за 6 символів, корень слова як правило не може перевищувати 2 склади, тоюто в середньому 6 літер;

- стемінг не відбувся, пошук не відбувся, такі випадки виносяться окремо, адже можливо такого слова й не існує, а можливо, порушено алгоритмічність, і через конфлікти між різними логічними аналізами, стемінг взагалі не відбувся.

Створивши інструмент для автоматичного тестування слів з файлу, та провівши тестування виявилося, що майже половину слів із вибірки тексту з джерел отримали негативний результат роботи моделі, тобто або не відбувся стемінг, або довжина отриманої основи більша з потрібну. (див. рис. 3.5)

Рисунок 3.5 – Результат роботи моделі нормалізації обробки 1000+ слів з джерел. (позитивний результат 340, не обробляються 250, негативний 410)

3.6 Гібридний пошук

Для більш вираженої різниці пошуків спробуємо для наших слів за допомогою токенізації тексту української мови, зробити пошук, та порівняти результати. В основі програми основаній на токенізації тексту, йде простий аналіз, та надання кожному слову з обробленого джерела відповідного токену, якщо відповідний токен вже існує, слово або його частина дане слово вже не підлягає токенізації. Пошук відбувається таким чином що вхідному слову теж надається певний токен, якщо його токен співпаде з токеном вхідного словаЮ програма покаже відповідність.[13]

Звісно такий спосіб пошуку не є ефективним адже в залежності від об'єму джерел, та довжини слова, програма поверне скоріше негативний результат. Даний алгоритм адаптують та доопрацьовують, аби пошук відбувався ще меншими частинками слів, а токени більш потрібні для утримання контексту, та сенсу текстів. (див. рис. 3.6)

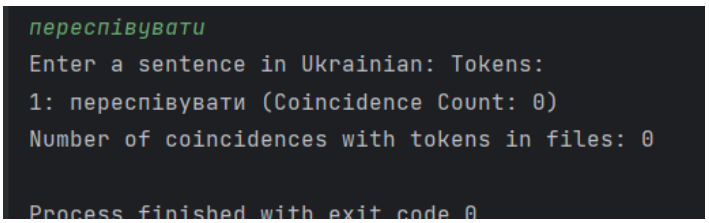


Рисунок 3.6 – Пошук слова «Переспівувати» по масиву токенів, який обробила програма. (Додаток Б)

Одним із варіантів адаптування такого алгоритму є пошук за тією ж токенізацією але не звичайного слова а слова, що пройшло нормалізацію моделі, та зазнало стемінгу. Адже тепер вірогідність того, що одним з токенів буде розповсюджена частинка слова (в прикладі корень), набагато вища, аніж пошук серед токенів уього слова.

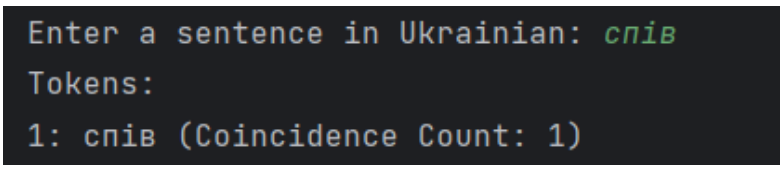


Рисунок 3.7 – Результат гібридного пошуку, перше слово «Переспівувати» спочатку зазнало стемінгу.

ВИСНОВКИ

В бакалаврській кваліфікаційній роботі досліджено алгоритмічне та програмне забезпечення модуля нормалізації тексту в системах аналізу та обробки україномовної інформації. Адаптовано алгоритм Портера Стеммера для обробки слова української мови, та проведено дослідження щодо ефективності такої адаптації. Результатом дослідження є:

1. Створено алгоритм стемінгу слова української мови, відповідно до основ орфографічних та морфологічних правил словотворення.
2. Розроблено графічний інтерфейс та створено додаток, де користувач може перевірити слово відповідно до своїх потреб, виконати стемінг, та пошук за отриманою основою спільнокореневих, або слів зі спільною основою.
3. Проведено ручне та автоматичне тестування ефективності такого алгоритму, отримані результати сформовано таблично та графічно, дані показують основні аспекти проблем у розробці та адаптації подібних алгоритмів для нормалізації текстів української мови.
4. Показаний приклад використання власної адаптації у гібридних алгоритмах обробки природньої мови NLP, де ефективність пошуку слів по токенам, очевидно краща після стемінгу слова.

Дане дослідження, розробка алгоритмів, та їх адаптація, отримані дані тестування, дозволяють продовжити роботу у цій сфері удосконалення алгоритму обробки тексту, та застосування даних алгоритмів у різних інструментах вивчення української мови, або у сфері нейронних мереж, інших інструментів автоматичної обробки текстів української мови.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GitHub – grammarly/ua-gec/blob. Gramarly. (n.d.). UA-GEC, 2023. GitHub https://github.com/grammarly/ua-gec .
2. GitHub - kurnosovv/ukr-spacy. ukr-spacy. (n.d.). GitHub. https://github.com/grammarly/ua-gec .
3. ANTLR 4 Documentation [Електронний ресурс]. – Режим доступу: GitHub. https://github.com/antlr/antlr4/blob/4.8/doc/index.md
4. DeepPavlov Documentation [Електронний ресурс]. – Режим доступу: https://docs.deeppavlov.ai/en/master/
5. Ömer YILMAZ, Mastering Stemming Algorithms in Natural Language Processing: A Complete Guide with Python Implementation. Medium. 2024. [Електронний ресурс]. – Режим доступу: https://medium.com/@omrylmzz35/mastering-stemming-algorithms-in-natural-language-processing-a-complete-guide-with-python-e7fd12089a69
6. Hobson Lane, Cole Howard, Hannes Hapke, Natural Language Processing in Action, 2019.
7. Мартін Портер, The Porter Stemming Algorithm. (n.d.). – Режим доступу: https://tartarus.org/martin/PorterStemmer/
8. Singh J., Gupta V. Text Stemming. ACM Computing Surveys. 2016. Vol. 49, no. 3. P. 1–46. URL: https://doi.org/10.1145/2975608 (date of access: 13.06.2024).
9. Hobson Lane, Cole Howard, Hannes Hapke, Natural Language Processing in Action, 2019.
10. Davy Cielen, Arno Meysman, Mohamed Ali, Introducing Data Science: Big Data, Machine Learning, and more, using Python tools, 2016
11. Ніколенко С. І., Кадурін А. А., Архангельська Є. О, Глибоке навчання. Занурення в світ нейронних мереж, 2018.
12. Comprehensive N. NLP: The New Technology of Achievement. Harper Paperbacks, 1996. 352 p.
13. Sharan K. Regular Expressions. Beginning Java 8 Fundamentals. Berkeley, CA, 2014. P. 519–542. URL: https://doi.org/10.1007/978-1-4302-6653-2_14 (date of access: 13.06.2024).
14. GitHub - jedp/porter-stemmer. jedp. (n.d.). GitHub. https://github.com/jedp/porter-stemmer
15. Тараненко О. О. Словотворення української мови в аспекті її сучасних системно-нормотворчих тенденцій (кінець XX - початок XXI ст.). Мовознавство. 2015. № 1. С. 3–32.

16. Aggarwal E., Nair S. NLP TOKEN MATCHING ON DATABASE USING BINARY SEARCH. INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY. 2012. Vol. 3, no. 1. P. 140–143. URL: https://doi.org/10.24297/ijct.v3i1c.2766 (date of access: 14.06.2024).

Додаток А

Програмні файли

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class UkrainianStemmer {
    private String word;
    private final String vowel = "аеиоуюяіїє";
    private final String perfectiveground =
"(ив|ивши|ившись|ыв|ывши|ывшись((?<=[ая])(в|вши|вшись)))$";
    private final String reflexive = "(с[яьи])$";
    private final String adjective =
"(ими|ій|ий|а|е|ова|ове|ів|є|їй|єє|еє|я|ім|ем|им|ім|их|іх|ою|йми|іми|у|ю|ого|ому
|оїй)$";
    private final String participle =
"(ий|ого|ому|им|ім|а|ій|у|ою|ій|і|их|йми|их)$";
    private final String verb =
"(сь|ся|ив|ать|ять|у|ю|ув|ав|али|учи|ячи|вши|ши|е|ме|ати|яти|є)$";
    private final String prefix =
"^(зі|з|зо|с|пре|при|прі|роз|без|через|від|од|між|над|об|пере|під|понад|пред|із)
";
    private final String noun =
"(а|ев|ов|е|ями|ами|еи|и|ей|ой|ий|й|иям|ям|ием|ем|ам|ом|о|у|ах|иях|ях|ы|ь|ию|ью|
ю|ия|ья|я|і|ові|ї|ею|ею|ою|є|еві|ем|єм|ів|їв|ю)$";
    private final String rvre = "[аеиоуюяіїє]";
    private final String derivational =
"[^аеиоуюяіїє][аеиоуюяіїє]+[^аеиоуюяіїє]+[аеиоуюяіїє].*(?<=о)сть?$";
    private String RV;

    // Глобальна змінна для результату
    private String result = "";
    private StringBuilder ending = new StringBuilder();
    private String final_messsage = "";
    public String getEnding() {
        return ending.toString() + " ";
    }
    public void setFinal_messsage(String final_messsage) {
        this.final_messsage = final_messsage;
    }

    public String getFinal_messsage() {
        return final_messsage;
    }

    public UkrainianStemmer(String word) {
        this.word = word;
    }

    public void setResult(String result) {
        this.result = result;
    }

    private String ukstemmerSearchPreprocess(String word) {
        word = word.toLowerCase();
```

```
        String originalWord = word; // Оригінальне слово

        word = word.replace("'", "");
        if (!word.equals(originalWord)) {
            final_messsage += "replace '";
        }

        originalWord = word;

        word = word.replace("ë", "e");
        if (!word.equals(originalWord)) {
            final_messsage += " ë";
        }

        originalWord = word;

        word = word.replace("ъ", "и");
        if (!word.equals(originalWord)) {
            final_messsage += " ъ";
        }

        return word;
    }

    private boolean s(String st, String reg, String to) {
        String orig = st;
        RV = st.replaceAll(reg, to);
        if (!orig.equals(RV)) {
            ending.append(orig.substring(RV.length()));
        }
        return !orig.equals(RV);
    }

    public String stemWord() {
        // Попередній обробник вхідного слова (українська кирилиця, апострофи, і
т.д.)
        word = ukstemmerSearchPreprocess(word);
        // Якщо слово не містить жодного голосного символу, то не проводимо
жодних змін і повертаємо його без змін
        if (!word.matches(".*[" + vowel + "].*")) {
            result = word; // Зберегти слово як результат
            final_messsage += "Нема голосних, скоріш за все слова не існує";
            return word;
        } else {
            Pattern prefixPattern = Pattern.compile(prefix);
            Matcher prefixMatcher = prefixPattern.matcher(word);

            // Check if the prefix pattern is found
            if (prefixMatcher.find()) {
                String prefix = word.substring(0, prefixMatcher.end());
                // Remove the prefix from the word
                word = word.substring(prefixMatcher.end());
                System.out.println(word);
                // Update the result
            }

            Pattern pattern = Pattern.compile(rvre);
            Matcher matcher = pattern.matcher(word);

            // Якщо у слові знайдено патерн, виконуємо алгоритм стемінгу
            if (matcher.find()) {
                String start = word.substring(0, matcher.end());
                RV = word.substring(matcher.end());
```

```
                // Крок 1: перевіряємо правила для кореневої частини слова
                if (!s(RV, perfectiveground, "")) {
                    // Крок 1: Перевірка чи слово відповідає патерну для
perfectiveground (доконаний вид)
                    if (s(RV, adjective, "")) {
                        // Перевірка чи слово відповідає патерну для
прикметників
                        s(RV, participle, ""); // Видалення закінчень для
дієприкметників
                        final_messsage += "прикметник";
                    } else {
                        if (!s(RV, verb, "")) {
                            s(RV, noun, ""); // Видалення закінчень для
іменників
                            final_messsage += "іменник";
                        }if (!s(RV, verb, "")) {
                            s(RV, noun, ""); // Видалення закінчень для
іменників
                            final_messsage += "іменник";
                        }
                    }
                    if(final_messsage.isEmpty()){
                        final_messsage += "дієслово/друга частина мови";
                    }
                }

                // Крок 2: видаляємо закінчення "и", якщо воно є
                s(RV, "и$", "");

                // Крок 3: перевіряємо наявність суфіксів і змінюємо, якщо
потрібно
                if (RV.matches(derivational)) {
                    s(RV, "ість$", "");
                    final_messsage += "суфікс ість";
                }

                // Крок 4: змінюємо закінчення "ь", "ейше", "нн", якщо вони є
                s(RV, "ь$", "");
                s(RV, "ейше?$", "");
                s(RV, "нн$", "н");

                // Повертаємо об'єднаний результат
                result += start + RV; // Зберегти результат
                System.out.println(result);
                return result;
            } else {
                // Якщо ж патерн не знайдено, повертаємо слово без змін
                final_messsage += "Не було проведено ніяких перетворень";
                result = word; // Зберегти слово як результат
                return word;
            }
        }
    }


    // Функція для друку результату
    public void printResult() {
        System.out.println("Stemmed Word: " + result);
    }
    public String getResult() {
        return result;
```

```
    }
}
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

public class UkrainianStemmerGUI {
    private JTextField inputField;
    private JTextArea outputArea;
    private JLabel infoLabel;

    public UkrainianStemmerGUI() {
        // Create the main frame
        JFrame frame = new JFrame("Ukrainian Stemmer");

        // Create input panel
        JPanel inputPanel = new JPanel();
        JLabel inputLabel = new JLabel("Enter a word in Ukrainian:");
        inputField = new JTextField(20);
        JButton stemButton = new JButton("Stem Word");

        // Create output panel
        JPanel outputPanel = new JPanel();
        JLabel outputLabel = new JLabel("Stemmed Word:");
        outputArea = new JTextArea(10, 20);
        outputArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(outputArea);

        // Create info panel
        JPanel infoPanel = new JPanel();
        infoLabel = new JLabel();

        // Set layout manager
        frame.setLayout(new BorderLayout());
        inputPanel.setLayout(new FlowLayout());
        outputPanel.setLayout(new FlowLayout());
        infoPanel.setLayout(new FlowLayout());

        // Add components to input panel
        inputPanel.add(inputLabel);
        inputPanel.add(inputField);
        inputPanel.add(stemButton);

        // Add components to output panel
        outputPanel.add(outputLabel);
        outputPanel.add(scrollPane);

        // Add components to info panel
```

```
        infoPanel.add(infoLabel);

        // Add panels to the main frame
        frame.add(inputPanel, BorderLayout.NORTH);
        frame.add(outputPanel, BorderLayout.CENTER);
        frame.add(infoPanel, BorderLayout.SOUTH);

        // Set action listener for the stem button
        stemButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                stemWordAndDisplayResult();
            }
        });

        // Set default close operation
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Set frame size and visibility
        frame.setSize(400, 500);
        frame.setVisible(true);

        // Set preferred size for the button
        Dimension buttonSize = new Dimension(200, 80);
        stemButton.setPreferredSize(buttonSize);
    }

    private void stemWordAndDisplayResult() {
        String input = inputField.getText().toLowerCase(); // Convert input to
lowercase

        if (clearInput(input) != null) {
            UkrainianStemmer stemmer = new UkrainianStemmer(input);
            stemmer.stemWord(); // Call the stemWord function
            String stemmedResult = "Stemmed Word: " + stemmer.getResult();
            outputArea.setText(stemmedResult);

            // Find possible full words matching the stemmed root in files
            List<String> possibleMatches =
findPossibleMatches(stemmer.getFinal_messsage(), stemmer.getResult());

            // Append possible matches to the output area
            if (!possibleMatches.isEmpty()) {
                outputArea.append("\n\nPossible Matches:\n");
                for (String match : possibleMatches) {
                    outputArea.append(match + "\n");
                }
            } else {
                outputArea.append("\nNo matches found in files.");
            }
//"Entered msg: " + input +
            // Set info label text
            String infoText = " Message: " + stemmer.getFinal_messsage() +
                    " Ending: " + stemmer.getEnding();
            infoLabel.setText(infoText);
        } else {
            outputArea.setText("Something wrong");
        }
    }

    private static String clearInput(String input) {
        // Check if the input has more than one Ukrainian letter
        if (input.matches(".*[" + "аеиоуюяіїє" + "].*")) {
```

```
            // Remove extraneous characters (digits, special symbols, etc.) and
keep only Ukrainian letters
            return input.replaceAll("[^аеиоуюяіїє]+", "");
        } else {
            // If the input does not have more than one Ukrainian letter, return
null or handle it as needed
            return null;
        }
    }

    private static List<String> findPossibleMatches(String stemmedRoot, String
word_main) {
        String stem_word = word_main;
        List<String> possibleMatches = new ArrayList<>();

        // Assuming files are located in the project directory
        File projectFolder = new
File("F:\\study\\sumdu\\4th_course\\program\\Практика\\gui_ye_mova\\src\\source\
\good");
        // Iterate through files in the project folder

        for (File file : Objects.requireNonNull(projectFolder.listFiles())) {
            if (file.isFile()) {
                try {
                    // Read the content of each file
                    String content = new
String(Files.readAllBytes(file.toPath()));

                    // Tokenize the content into words
                    String[] words = content.split("\\s+");

                    // Check if the stemmed root matches any word in the file
                    for (String word : words) {
                        UkrainianStemmer fileStemmer = new
UkrainianStemmer(word);
                        String fileStemmedWord = fileStemmer.stemWord();
                        if (fileStemmedWord.equals(stem_word)) {
                            possibleMatches.add(word);
                        }
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return possibleMatches;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new UkrainianStemmerGUI();
            }
        });
    }
}
```

Додаток Б

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Scanner;

public class UkrainianTokenizer {

    public static class Token {
        private int id;
        private String value;
        private int coincidenceCount;

        public Token(int id, String value) {
            this.id = id;
            this.value = value;
            this.coincidenceCount = 0;
        }

        public int getId() {
            return id;
        }

        public String getValue() {
            return value;
        }

        public int getCoincidenceCount() {
            return coincidenceCount;
        }

        public void incrementCoincidenceCount() {
            coincidenceCount++;
        }
    }

    public static void main(String[] args) {
        // Read tokens from files in the specified folder and store them in a
HashSet
        HashSet<String> fileTokens = readTokensFromFolder("src\\good");
        if (fileTokens.isEmpty()) {
            System.out.println("No tokens found in the files.");
            return;
        }

        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a sentence in Ukrainian: ");
        String input = scanner.nextLine().toLowerCase(); // Convert input to
lowercase
        scanner.close();

        ArrayList<Token> tokens = tokenize(input);
        System.out.println("Tokens:");
        int coincidences = 0;

        for (Token token : tokens) {
            if (fileTokens.contains(token.getValue().toLowerCase())) { //
```

```
Convert token to lowercase
                token.incrementCoincidenceCount();
                coincidences++;
            }
            System.out.println(token.getId() + ": " + token.getValue() +
                    " (Coincidence Count: " + token.getCoincidenceCount() +
")");
        }

        System.out.println("Number of coincidences with tokens in files: " +
coincidences);
    }

    private static HashSet<String> readTokensFromFolder(String folderPath) {
        HashSet<String> tokens = new HashSet<>();

        File folder = new File(folderPath);
        if (!folder.exists() || !folder.isDirectory()) {
            System.out.println("Invalid folder path or the folder does not
exist.");
            return tokens;
        }

        File[] files = folder.listFiles();
        if (files == null || files.length == 0) {
            System.out.println("No files found in the folder.");
            return tokens;
        }

        for (File file : files) {
            if (file.isFile() && file.getName().endsWith(".txt")) {
                try (BufferedReader reader = new BufferedReader(new
FileReader(file))) {
                    String line;
                    while ((line = reader.readLine()) != null) {
                        String[] words = line.split("\\s+");
                        for (String word : words) {
                            tokens.add(word.toLowerCase()); // Convert token to
lowercase
                        }
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return tokens;
    }

    private static ArrayList<Token> tokenize(String text) {
        String[] words = text.split("\\s+");
        ArrayList<Token> tokens = new ArrayList<>();

        for (int i = 0; i < words.length; i++) {
            String word = words[i];
            // Check if the token already exists in the ArrayList
            boolean exists = false;
            for (Token token : tokens) {
                if (token.getValue().equals(word)) {
                    exists = true;
                    break;
                }
```

```
            }

            // If the token does not exist, add it to the ArrayList
            if (!exists) {
                tokens.add(new Token(tokens.size() + 1, word));
            }
        }

        return tokens;
    }
}
```

Додаток В

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class UkrainianStemmer {
    private String word;
    private final String vowel = "аеиоуюяіїє";
    private final String perfectiveground =
"(ив|ивши|ившись|ыв|ывши|ывшись((?<=[ая])(в|вши|вшись)))$";
    private final String reflexive = "(с[яьи])$";
    private final String adjective =
"(ими|ій|ий|а|е|ова|ове|ів|є|їй|єе|еє|я|ім|ем|им|ім|их|іх|ою|йми|іми|у|ю|ого|ому
|ої)$";
    private final String participle =
"(ий|ого|ому|им|ім|а|ій|у|ою|ій|і|их|йми|их)$";
    private final String verb =
"(сь|ся|ив|ать|ять|у|ю|ув|ав|али|учи|ячи|вши|ши|е|ме|ати|яти|є)$";
    private final String prefix =
"^(зі|з|зо|с|пре|при|прі|роз|без|через|від|од|між|над|об|пере|під|понад|пред|із)
";
    private final String noun =
"(а|ев|ов|е|ями|ами|еи|и|ей|ой|ий|й|иям|ям|ием|ем|ам|ом|о|у|ах|иях|ях|ы|ь|ию|ью|
ю|ия|ья|я|і|ові|ї|ею|ею|ою|є|еві|ем|єм|ів|їв|ю)$";
    private final String rvre = "[аеиоуюяіїє]";
    private final String derivational =
"[^аеиоуюяіїє][аеиоуюяіїє]+[^аеиоуюяіїє]+[аеиоуюяіїє].*(?<=о)сть?$";
    private String RV;

    private String result = "";
    private StringBuilder ending = new StringBuilder();
    private String finalMessage = "";

    public UkrainianStemmer(String word) {
        this.word = word;
    }

    private String ukstemmerSearchPreprocess(String word) {
        word = word.toLowerCase();
        String originalWord = word;
        word = word.replace("'", "");
        word = word.replace("ё", "е");
        word = word.replace("ъ", "ї");
        return word;
    }

    private boolean s(String st, String reg, String to) {
        String orig = st;
        RV = st.replaceAll(reg, to);
        if (!orig.equals(RV)) {
            ending.append(orig.substring(RV.length()));
        }
        return !orig.equals(RV);
    }
```

```
    public String stemWord() {
        word = ukstemmerSearchPreprocess(word);
        if (!word.matches(".*[" + vowel + "].*")) {
            result = word;
            finalMessage += "Нема голосних, скоріш за все слова не існує";
            return word;
        } else {
            Pattern prefixPattern = Pattern.compile(prefix);
            Matcher prefixMatcher = prefixPattern.matcher(word);
            if (prefixMatcher.find()) {
                word = word.substring(prefixMatcher.end());
            }

            Pattern pattern = Pattern.compile(rvre);
            Matcher matcher = pattern.matcher(word);
            if (matcher.find()) {
                String start = word.substring(0, matcher.end());
                RV = word.substring(matcher.end());

                if (!s(RV, perfectiveground, "")) {
                    if (s(RV, adjective, "")) {
                        s(RV, participle, "");
                        finalMessage += "прикметник";
                    } else {
                        if (!s(RV, verb, "")) {
                            s(RV, noun, "");
                            finalMessage += "іменник";
                        }
                    }
                    if(finalMessage.isEmpty()){
                        finalMessage += "дієслово/друга частина мови";
                    }
                }

                s(RV, "и$", "");
                if (RV.matches(derivational)) {
                    s(RV, "ість$", "");
                    finalMessage += "суфікс ість";
                }

                s(RV, "ь$", "");
                s(RV, "ейше?$", "");
                s(RV, "нн$", "н");

                result += start + RV;
                return result;
            } else {
                finalMessage += "Не було проведено ніяких перетворень";
                result = word;
                return word;
            }
        }
    }

    public String getResult() {
        return result;
    }

    public String getFinalMessage() {
        return finalMessage;
    }

    public static void main(String[] args) {
```

```
        List<String> words = new ArrayList<>();
        List<String> positiveResults = new ArrayList<>();
        List<String> negativeResults = new ArrayList<>();

        try {
            words = Files.readAllLines(Paths.get("words.txt"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        for (String word : words) {
            UkrainianStemmer stemmer = new UkrainianStemmer(word);
            String stemmedWord = stemmer.stemWord();
            String message = stemmer.getFinalMessage();

            // Simulate a search (replace with actual search implementation)
            boolean searchResult = simulatedSearch(stemmedWord);

            if (searchResult) {
                if (message.contains("прикметник") ||
message.contains("іменник")) {
                    positiveResults.add("Word: " + word + ", Stemmed: " +
stemmedWord + ", Message: " + message);
                } else {
                    positiveResults.add("Word: " + word + ", Stemmed: " +
stemmedWord + ", Message: " + message);
                }
            } else {
                if (stemmedWord.length() > 6) {
                    negativeResults.add("Word: " + word + ", Stemmed: " +
stemmedWord + ", Message: " + message);
                } else {
                    negativeResults.add("Word: " + word + ", Stemmed: " +
stemmedWord + ", Message: " + message);
                }
            }
        }

        try (FileWriter writer = new FileWriter("results.txt")) {
            writer.write("Positive Results:\n");
            for (String result : positiveResults) {
                writer.write(result + "\n");
            }
            writer.write("\nNegative Results:\n");
            for (String result : negativeResults) {
                writer.write(result + "\n");
            }
            writer.write("\nStatistics:\n");
            writer.write("Total Words: " + words.size() + "\n");
            writer.write("Positive Results: " + positiveResults.size() + "\n");
            writer.write("Negative Results: " + negativeResults.size() + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static boolean simulatedSearch(String stemmedWord) {
            return Math.random() > 0.3;
    }
}
```