Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*
(повна назва)
Кафедра *Системотехніки*
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

ГЮИК.501310.002 ПЗ

Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту
(тема)

Виконав:
Студент 2 курсу, групи *ІТПм-19-1*
Спеціальність *122 – Комп'ютерні науки*
(код і повна назва напряму)
Тип програми *освітньо- професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Інформаційні технології проектування*
(повна назва освітньої програми)

Волобуєва В.В.
(прізвище, ініціали)

Керівник *проф. Колесник Л.В.*
(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ *Гребеннік І. В.*
(підпис) (прізвище, ініціали)

2020 р.

Атестаційна робота оформлена у відповідності до вимог діючих стандартів та методичних вказівок.

Матеріали атестаційної роботи не містять відомостей, що заборонені для опублікування у відкритих виданнях.

Попередній захист проведено 18 грудня 2020 року.

Керівник атестаційної роботи *Л.В.Колесник*

Харківський національний університет радіоелектроніки

Факультет *Комп'ютерних наук*

(повна назва)

Кафедра *Системотехніки*

(повна назва)

Рівень вищої освіти другий (магістерський)

Спеціальність *122 – Комп'ютерні науки*

(код і повна назва)

Тип програми *освітньо-професійна*

(освітньо-професійна або освітньо-наукова)

Освітня програма *Інформаційні технології проектування*

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри *Гребеннік І. В.*

(підпис)

«____»_____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові *Волобуєвій Владлені Віталіївні*

(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту
затверджена наказом по університету від « *02* » 11 20*20* р. № 1517 Ст

2. Термін здачі студентом роботи (проекту) *18 грудня 2020 р.*

3. Вихідні дані до роботи (проекту) Функція: дослідження методів приведення слів до нормальної форми. Організація даних: файлова з прямим доступом. Форма діалогу: веб-система. Перелік використовуваних програмних засобів: ОС Microsoft Windows 7 та вище, браузер Google Chrome, IE11, Mozilla Firefox тощо. Технічне забезпечення: комп'ютер з не менш, ніж 512 Мб оперативної пам'яті.

4. Зміст пояснювальної записки (перелік питань, що потрібно опрацювати в роботі 4.1 Вступ. 4.2 Аналіз предметної області та постановка задачі. 4.3 Дослідження і побудова рішення 4.4 Аналіз

існуючих технологій для розробки програмного засобу. 4.5 Розробка проектних рішень по реалізації системи. 4.6 Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 5.1 Алгоритм роботи стеммінга Портера. 5.2 Алгоритм роботи методу Mystem. 5.3 Результати вимірювання швидкості знаходження стемми для 1 слова. 5.4 Результати вимірювання швидкості знаходження стемми для 100 слів. 5.5 Результати вимірювання швидкості знаходження стемми для 1000 слів. 5.6 Приклад побудови дерева суфіксів. 5.7 Приклад побудови префіксного дерева інвертованих основ.

6. Консультанти розділів роботи(проекту)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спеціальна частина	проф. Колесник Л.В.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання завдання на атестаційне проектування*	*02.11.2020*	
2	*Аналіз завдання, пошук літератури та аналогів з теми атестаційної роботи*	*05.11.2020*	
3	*Постановка задачі та вибір методу її вирішення*	*10.11.2020*	
4	*Проведення експериментальних досліджень*	*25.11.2020*	
5	*Оформлення пояснювальної записки*	*01.12.2020*	
6	*Підготовка презентації*	*10.12.2020*	
7	*Подання закінченої роботи науковому керівникові*	*15.12.2020*	
8	*Подання роботи на рецензування*	*16.12.2020*	
9	*Попередній захист*	*18.12.2020*	
10	*Подання роботи до екзаменаційної комісії*	*22.12.2020*	

Дата видачі завдання *02 листопада 2020 р.*

Студент _____ *Волобуєва В.В .*
(підпис)

Керівник роботи _____ *проф. Колесник Л.В.*
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до атестаційної роботи: 95 с., 15 рис., 1 табл., 4 дод., 36 джерел. Графічний матеріал атестаційної роботи містить 7 плакатів.

ВЕБ-ДОДАТОК, ВЕБ-СИСТЕМА, МОРФОЛОГІЧНИЙ АНАЛІЗ ТЕКСТУ, НОРМАЛІЗАЦІЯ, СТЕММА, СТЕММІНГ, СТЕММЕР ПОРТЕРА, JAVA, MYSTEM, SNOWBALL, STEMKA, TRIEMAP.

Мета атестаційної роботи – дослідити існуючі методи нормалізації слів шляхом виділення стемми слова. Порівняти особливості існуючих методів, виділити їх переваги та недоліки. Провести аналіз проблеми застосування нормалізації слів тексту, запропонувати варіанти вирішення, якщо це можливо. Розробити компоненти веб-системи для змоги продемонструвати вирішення поставленого завдання.

Об’єкт дослідження – семантичний аналіз тексту.

Предмет дослідження – методи приведення слів до нормальної форми для подальшого проведення семантичного аналізу тексту.

Демонстрація об’єкта дослідження відбувається через веб-систему, яка б дозволила у подальшому проводити семантичний аналіз тексту. У даній роботі об’єктом розробки є мова програмування Java.

Результатом атестаційної роботи є розроблений прототип веб-орієнтована системи, що підтверджує працездатність реалізованого методу виділення нормальної форми для корпусу слів російської мови.

Область застосування системи – впровадження методу до пошукових систем для розширення пошукового запиту користувача, використання у SEO процесах.

ABSTRACT

Attestation work: 95 p., 15 pic., 1 table, 36 source, 4 applications. Graphic material attestation work contains 14 poster.

JAVA, LINGUISTICS, MORPHOLOGICAL ANALYSIS OF THE TEXT, MYSTEM, NORMALIZATION, SNOWBALL, STEMKA, STEMMING, TRIEMAP, WEB-APPLICATION, WEB-SYSTEM.

The purpose of the certification work is to investigate the existing methods of word normalization by distinguishing the word's stem. Compare the features of existing methods, compare their advantages and disadvantages. Analyze the problem of word normalization process and if possible to suggest solutions. Develop web-system components in order to demonstrate the solution to the normalization problem.

The object of research is the semantic analysis of the text.

The subject of research – methods of bringing words to normal form for further semantic analysis of the text.

Demonstration of the object of study will be handled through the web system, as it will allow future usage of the semantic analysis. The development object of this work is Java programming language

The result of the certification work is a developed prototype of a web-oriented system, which confirms the efficiency of the implemented method of allocating a normal form for the words of the Russian language corpus.

The scope of the system – method introduction to search engines to expand the user's search query, usage in SEO processes.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Гіпотеза – припущення, щодо нормальної форми слова, яке потребує перевірки.

Ключові слова – це певні слова з тексту, здатні уявити найбільш значущі слова, за якими може вестися оцінка і пошук тексту.

Лема – нормальна (початкова, словникова) форма слова.

Лематизація – процес приведення слова до леми.

Префіксне дерево – структура даних, що дозволяє зберігати асоціативний масив, ключами якого є рядки.

Семантичне ядро – ключові слова і словосполучення, які використовуються для пошуку необхідної інформації.

Стемма – основа слова, незмінна частина слова.

Стеммінг – процес знаходження основи слова для вихідного слова.

Стоп-слово – слово, що не несе ніякого смислового навантаження.

Флективність – словозміна, утворення словоформ тієї ж лексеми, що мають різні граматичні значення.

PPMV – пара потенційних морфологічних варіантів (pair of potential morphological variants).

TrieMaps – це карти, що використовують структуру даних trie, які по суті є деревами.

ВСТУП

На будь-якому етапі свого розвитку людина знаходиться в безперервному потоці інформації, тому природним мати прагнення впорядкувати та автоматизувати її обробку. Існують різні способи представлення даних, проте природні мови залишаються найбільш використовуваними і найбільш складними для автоматичної обробки.

Різні мови мають різні семантичні та граматичні особливості, тому часто алгоритми, успішно використовуються для обробки однієї мови, показують дуже низьку ефективність на іншій мові. Проблема обробки тексту, написаного на природній мові, полягає ще й у тому, що для цього часто потрібно розуміння тексту, доступне тільки людині.

Проте складнощі обробки природної мови не виключають можливості виділення більш вузьких завдань, які вже можна вирішити алгоритмічно: наприклад, визначення частин мови або розбиття текстів на логічні групи. Однак деякі особливості природних мов значно знижують ефективність даних рішень: так, наприклад, облік всіх словоформ для кожного слова в українській чи російській мовах значно збільшує складність обробки текстів.

Приведення всіх слів тексту до нормальної форми сильно спрощує роботу з ним для пошукових систем, систем автоматичного реферування, машинних перекладачів. Дана робота присвячена вивченню та побудові вирішення проблеми приведення слів до нормальної форми на прикладі українських чи російських слів.

У даний час основним джерелом інформації про навколишній світ для людини є Інтернет. Так, наприклад, за даними wordstat.yandex.ru, середня денна аудиторія, яка використовує Google пошук, становить 14.5 мільйонів чоловік тільки по Україні. Використання природної мови для пошукових запитів дозволяє скласти безліч варіантів для одного і того ж запиту, які пошукова система повинна вміти обробляти і при необхідності ототожнювати.

При цьому значна частка пошукових запитів містить помилки і друкарські помилки: по даними https://trends.google.com.ua/ на вересень 2020 року, слово «*морквяний*» зустрічається в запитах в середньому 11753 рази в місяць, «*моркв'яний*» – 1093, «*моркваний*» – 281. Навіть при використанні словників синонімів і перевірки правопису, відповідно до емпіричного законом Хіпс про те, що число різних слів в тексті зростає приблизно як корінь квадратний з кількості всіх слів тексту, швидкість появи нових слів в Інтернеті робить необхідність їх постійного поновлення в великих обсягах, що не є раціональним. Крім того, пошукова система повинна обробляти навіть ті запити, в яких зустрічаються неіснуючі, або існуючі, але дуже рідко використовуються слова, такі наприклад, як слово *рентгеноелектрокардіографічного*, що зустрічається в пошукових запитах близько 67 разів на місяць. При цьому, в якій би формі це слово не вживалося в запиті і в існуючих документах, якісна пошукова система повинна зіставити їх загальної нормальній формі, щоб видати максимально повний результат [1].

Нормалізація слів, використаних в пошуковому запиті, тобто приведення слів до початковій формі, має низькі витрати, але значно покращує якість роботи. Крім того, нормалізовані тексти краще піддаються кластеризації та іншої механічній обробці, в т.ч. машинного перекладу.

Об'єкт дослідження – семантичний аналіз тексту.

Предмет дослідження – методи приведення слів до нормальної форми для подальшого проведення семантичного аналізу тексту.

Завдання про застосуваннях нормалізованих слів у проведенні семантичного аналізу в атестаційній роботі будемо розглядати, як можливе доповнення до існуючих систем. Надалі систему можна буде поліпшити шляхом впровадження до неї компонентів семантичної обробки шляхом застосування штучного інтелекту, а саме машинного навчання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз задачі

Задача даної атестаційної роботи полягає в дослідженні методів нормалізації слів для подальшого використання у семантичному аналізу. Для виконання поставленого завдання необхідно:

- розробити календарний план проведення досліджень;
- вивчити предметну область;
- дослідити процес нормалізації та його складові;
- проаналізувати існуючі методи вирішення поставленого завдання;
- дослідити принципи обробки слів існуючими методами;
- порівняти на знайти найбільш оптимальні способи реалізації;
- розробити прототип системи;
- провести тестування системи на різних вхідних даних.

Досліджувані методи повинні дозволяти проводити обробку слів природних мов. Зокрема працювати з групами флективних синтетичних мов, до яких відносяться російська (українська) мови.

1.2 Огляд і аналіз сучасного стану проблеми

Ми живемо у часі, коли обсяги інформації, що виробляє людина, більше, ніж будь-коли, і кількість цих даних зростає з кожним днем. Однак значну користь з цієї інформації можна отримати лише при правильній обробці цих даних [2].

Зараз щомиті по всьому світу створюються гігабайти нових даних різного виду: робляться нові знімки, відеозапису, пишуться сотні відгуків до товарів в

інтернет-магазинах, тисячі коментарів під записами на Facebook, десятки рецензій до фільмів в Інтернет-кінотеатрах, ціни на акції, які то здіймаються, то падають.

І велика частина цих даних в «сирому» вигляді практично не має сенсу. Щоб витягти з них якусь користь, їх потрібно відфільтрувати і обробити. За часів, коли технології ще не були так розвинені, все це доводилося робити вручну. На це йшли години, дні, тижні, а то і місяці. А якщо врахувати, що раніше і самої інформації для обробки було в рази менше, то нескладно зрозуміти, що зараз обробляти такі обсяги вручну просто неможливо. Тому було розроблено безліч алгоритмів, які дозволяють робити це за допомогою комп'ютерної техніки. Саме такі напрямки, що стосуються обробки природньої мови, буде розглянуто.

(Natural Language Processing, NLP) – обробка природної мови, напрямок в дослідженнях у галузях штучного інтелекту та математичної лінгвістики. NLP вивчає проблеми проведення комп'ютерного аналізу і синтезу природних мов. У штучного інтелекту, аналіз несе сенс розуміння мови, а під синтезом – генерацію граматично вірного тексту. Рішення цих проблем означає створення якомога більш зручної форми взаємодії комп'ютера і людини [3]. Основними напрямами обробки природної мови прийнято вважати такі, як, наприклад, витяг фактів, проведення аналізу тональності тексту, генерація тексту, відповіді на питання, пошук інформації, переклад і т.д. Детальніше про деякі з них.

Витяг інформації або фактів. Під отриманням інформації мається на увазі пошук в неструктурованому або слабо структурованому документі окремих фактів, що саме цікавлять користувача. Наприклад, у вас є величезна кількість статей, в яких фігурує велика кількість різних особистостей, і ви хочете скласти базу даних, яка буде зберігати дані про те, хто з фігурують в даних статтях людей, є чоловіком і дружиною. Даний приклад був використаний для демонстрації [4] можливостей програми під назвою DeepDive, створеної групою студентів і працівників університету Stanford.

Аналіз тональності тексту. Аналіз тональності тексту має на увазі під собою автоматичне визначення емоційного забарвлення тексту і виявлення

ставлення людини, який написав текст, до об'єкта обговорення. Даний тип аналізу може бути використаний, наприклад, продавцями для того, щоб краще зрозуміти, який з проданих ним товарів користується великим успіхом серед покупців, аналізуючи відгуки. Також його можуть використовувати влади для виявлення ставлення до них і їх рішень громадян країни і т.д. У наші часи найбільш використовуваними при дослідженнях методами прийнято вважати методи, що базуються на основі машинного навчання з вчителем. Сенсом таких методів є те, що на початковому етапі відбувається навчання машинного класифікатору (як наприклад, байєсовский) на текстах, які були розмічені заздалегідь, а потім використовувати модель, що була отримана при аналізі нових документів [5].

Відповіді на питання. Під це визначення в цілому можуть підходити і так звані чатботи, які імітують реальне спілкування з людьми за допомогою передачі текстових повідомлень, і спеціальні програми, які спершу аналізують якийсь текст, а після – відповідають на питання, пов'язані з його змістом. Результати одного з останніх досліджень на цю тему, описані в статті [6], під авторством John Ball.

Переклад тексту. Також одним з найбільш відомих і часто використовуваних напрямків обробки природного тексту є його переклад з однієї природної мови на іншу. Однією з найбільш просунутих методів, використовуваних наразі для досягнення правильного перекладу, є використання нейронних мереж типу «Seq2Seq» [7] з «Увагою» [8], що розшифровується, як «sequence to sequence», або «послідовність в послідовність».

Індексації автоматично пропускаються для економії простору баз даних. Оптимізаторам цей факт необхідно обов'язково враховувати при складанні пошукових запитів і визначенні щільності ключових слів в контекстному наповненні сторінок.

Також при обробці слів природньої мови слід пам’ятати про таке явище як «стоп-слова». Стоп-слова – це термін з області SEO (Search engine optimization), позначає слова в тексті, які не несуть смислового навантаження. Також їх називають «*шумові слова*».

Кожна пошукова система має свою власну базу стоп-слів. Пошукі системи постійно розвивають і удосконалюють свої алгоритми, тому ці списки періодично оновлюються і змінюються. Але, так чи інакше, при написанні текстів варто звертати увагу на входження стоп-слів з перерахованих вище категорій і їх співвідношення із загальною масою слів і ключових слів.

Однією з основних проблем, з якими стикаються при обробці слів природних мов є мовні відмінності. Хоча більша частина ранніх академічних робіт в області стеммінгу була зосереджена на англійській мові (зі значним використанням алгоритму Портера Стеммер), багато інших мов були досліджені. Іврит і арабська досі вважаються складними для дослідження мовами. Англійські стеммери досить тривіальні (з рідкісними проблемами, наприклад, *сушить* – це форма третьої особи однини від дієслова *сухий*, *сокири* – це множина від *сокири*, а також від *осі*); але розробка стеммеру стає все важче, оскільки морфологія, орфографія і кодування символів цільової мови стають більш складними. Наприклад, італійська мова з наявністю парадигм є більш складною, ніж англійська мова (через велику кількість дієслівних форм), російська мова ще складніша (через більшу кількість іменникових відмінювань), мова іврит для порівняння є ще більш складною (з nonconcatenative морфологією, система письма без голосних і вимога видалення префікса: основи івриту можуть складатися з двох, трьох або чотирьох символів, але не більше) і так далі.

Існує також таке поняття як багатомовний стеммінг. Він застосовує морфологічні правила двох або більше мов одночасно замість правил тільки однієї мови при інтерпретації пошукового запиту. Існують комерційні системи, що використовують багатомовний стемінг [9].

В алгоритмах виділення стемм є два виміри помилок: оверстеммінг і підстеммінг. Оверстеммінг – це помилка, при якій два окремих слова із змінною формулою пов'язані з одним і тим же коренем, але не повинні були бути – хибнопозитивні. Підкреслення – це помилка, при якій два окремих слова із змінною схильністю повинні бути пов'язані з одним коренем, але це не так – помилково

негативні результати. Алгоритми виведення намагаються мінімізувати кожен тип помилки, хоча зменшення одного типу може привести до збільшення іншого. Наприклад, широко використовуваний стеммер Портера переводить слова *universal*, *university* та *universe* в *universal*. Це випадок надмірного визначення меж: хоча ці три слова етимологічно пов'язані, їх сучасні значення знаходяться в самих різних областях, тому розгляд їх як синонімів в пошуковій системі, ймовірно, знизить релевантність результатів пошуку. Прикладом нижнього стебла в Стеммер Портера є *випускника* → *випускник*, *випускнику* → *випускник*, *випускник*/*випускники* → *випускник*. Це англійське слово зберігає латинську морфологію, тому ці майже синоніми об'єднуються.

Розглянемо предметну область проблеми обробки слів природної мови.

Деякі IR-дослідження показали дуже незначне поліпшення ефективності пошуку із застосуванням морфологічного аналізу або його відсутність

Однією зі складових обробки слів є проведення морфологічного аналізу. Морфологічний аналіз являє собою процес визначення граматичного значення словоформи і виділення її основи, або, іншими словами, виділення ключових слів у потоці тексту [10].

Будь-який алгоритм морфологічного аналізу складається з двох основних компонентів: декларативного і процедурного. При цьому декларативний компонент має на увазі таблиці структурованих даних, необхідних для аналізу, а процедурний компонент містить самі алгоритми аналізу і допоміжні процедури [11].

У зв'язку з особливостями російської (української) мови і наявністю великої кількості слів винятків в ньому здійснення морфологічного аналізу може бути ускладнене. Для подолання цих труднощів існує можливість вибору методу морфологічного аналізу, серед яких найбільш відомими є три основні.

Першим методом є складання морфологічного словника для конкретного підприємства вручну, з урахуванням всіх ключових слів, здатних вказати на витік інформації. Даний спосіб доцільно використовувати при невеликому обсязі

можливих ключових слів, аналізуючи корінь даних слів. Якщо інформаційна система підприємства (організації) складна, містить велику кількість різноманітних ресурсів, то використання даного методу буде важко, особливо на початкових етапах.

Особливістю другого методу є використання алгоритму стеммінг, суть якого полягає у виділенні основи слова, а не його кореня.

Метою роботи алгоритму є приведення кожного слова вхідного тексту до нормальної форми. Нормальною формою слова називається та форма, в якій це слово наводиться в словнику (наприклад, форма однини називного відмінка для іменників або інфінітив для дієслів). Слово, яке перебуває в нормальній формі, називається лема. При обробці слово поділяється на дві частини: основу (стемм, не обов'язково збігається з коренем або морфологічної основою слова) і формоутворювальну морфему (цю частину слова будемо називати суфіксом за аналогією з англомовними статтями, в російській мові також використовуються назви афікс або формант). Процес знаходження стемм називається стеммінг.

Як правило, більшу точність дає стеммінг по формотворчим (inflectional), а не словотворчим (derivational) морфемам, тому застосування таких алгоритмів для ізолюючих мов (мов, в яких кожній морфемі відповідає окреме слово: наприклад, в'єтнамський або класичний китайський) буде неефективним. Більшість алгоритмів розраховані на синтетичні мови – ті, в яких переважає формоутворення з використанням морфем. Англійська мова вважається аглютинативною: де кожна форма має єдине значення. Російська мова відноситься до групи флективних синтетичних, тобто мов, в яких домінує словотвір з використанням суфіксів, що поєднують відразу кілька граматичних значень (наприклад: хороший:-ий вказує одночасно на однина, чоловічий рід і називний відмінок), тому допускає використання стеммінгових алгоритмів нормалізації.

1.3 Постановка задачі

Метою даної роботи є дослідження методів приведення слів до нормальної форми для можливості їх подальшого використання у проведенні семантичного аналізу тексту.

Для досягнення поставленої мети необхідно реалізувати ряд задач. У першу чергу треба розглянути існуючі рішення задачі нормалізації слів природних мов. Провести порівняння особливостей існуючих методів та визначити проблеми, що є в цих методах, виділити шляхи можливості їх вирішення. Виділити основні характеристики методів, по яким буде відбуватися порівняння.

Більшість використовуваних алгоритмів реалізують лематизацію (приведення до нормальної форми) з використанням стеммінга і різними надбудовами на кшталт визначення частини мови або частотних словників. Таким чином, процес нормалізації можна розбити на дві підзадачі:

– визначення незмінної частина слова – стеммінг;

– синтез нормальної форми: визначення суфікса нормальної форми для даної стемми і передбачуваної моделі формоутворення.

Особливий інтерес представляє перша частина, тому що по-перше, синтез нормальної форми безпосередньо залежить від способу отримання стемми, а по-друге, більшість реалізацій синтезує всі можливі леми, не вибираючи з них єдиного результату, або зупиняється на визначенні стемми.

Для вирішення проблеми вибору нормальної форми слова в тих випадках, коли не має однозначно виділеної стемми, необхідно реалізувати ранжування всіх потенційних гіпотез слова. Таким чином, ймовірність виділення коректної стемми значно зросте.

Також необхідно при розробці алгоритму стеммінгу слів російської (українськьої) мови врахувати можливість роботи зі словами, що не входять до словникової бази, мати можливість виділення гіпотетичних стемм.

Розробити систему для демонстрації працездатності розглянутого методу. Для цього необхідно визначити засоби програмування та тестування. Провести тестування системи та скласти проектну документацію.

1.3.1 Опис вхідної та вихідної інформації

В якості вхідної інформації буде виступати текст, написаний на природній мові, що міститиме як існуючі так і неіснуючі слова, щоб перевірити, як алгоритм буде себе поводити з різними даними.

Вихідною інформацією є результат аналізу, що подається у вигляді таблиці – слово, що аналізується у першому стовпці, та нормальна форма відповідного слова – у другому стовпці. Для слів що не мають чітко визначеної нормальної форми приводиться знайдена кількість гіпотез.

1.4 Огляд існуючих методів

Розглянемо деякі існуючі розв’язки задачі нормалізації слів природних мов шляхом виділення стемми. Три найбільш популярних реалізації стеммера, що ґрунтуються на різних принципах і допускають обробку неіснуючих слів для російської (української) мови:

– Snowball (стеммер Портера);
– Stemka;
– Mystem.

1.4.1 Snowball (стеммер Портера)

Snowball – реалізація алгоритму, розробленого Мартіном Портером в 1979 році для англійської мови. Після того як Мартін звільнився від розробки в 2014 році, Snowball ства підтримуватися як суспільний проект. Мартін спочатку обрав

ім'я Snowball як данину пам'яті SNOBOL, мові обробки строкових даних з 1960-х років. Зараз це також служить метафорою того, як проект росте за рахунок збора внесків з часом.

Оригінальний стеммер був написаний на що нині не використовувану мову BCPL (в даний час є реалізації на більшості популярних мов програмування, включаючи Java, C, Perl, Common Lisp і python). Згодом був створений проект Snowball і на основі оригінального алгоритму були написані стеммери для більшості індоєвропейських мов, в т.ч. російської (української).

Основна ідея стеммера Портера полягає в тому, що існує обмежена кількість формо- і словотворчих суфіксів, і стеммінг слова відбувається без використання будь-яких баз основ: тільки безліч існуючих суфіксів (при цьому складні складові суфікси розбиваються на прості) і вручну задані правила [12].

Алгоритм складається з п'яти кроків. На кожному кроці відсікається формо- або словотворчий суфікс і решта перевіряється на відповідність правилам. Необхідно пам’ятати що для кожної мови існують свої правила належності отриманої словоформи до основи слова. Наприклад, для російських (українських) слів основа повинна містити не менше однієї голосної. Якщо отримане слово задовольняє правилам, відбувається перехід на наступний крок. якщо немає – алгоритм вибирає інший суфікс для відсікання. Згідно з офіційним сайтом проекту, на першому кроці відсікається максимальний формотворчий суфікс, на другому – буква -*і*-, на третьому – словотворчий суфікс, на четвертому – суфікси форм найвищого ступеня, -*ь*- і одна з двох -*н*-.

На рисунку 1.1 схематично зображено алгоритм Snowball (стеммер Портера). Пояснення до рисунку 1.1 наведено нижче:

1. Цифрою *1* позначений блок операцій для відсікання формотворчих суфіксів.
2. Цифрою *2* позначений блок операцій для відсікання закінчень -*і.*
3. Цифрою *3* позначений блок операцій для відсікання словотворчих суфіксів.

4. Цифрою *4* позначений блок операцій для відсікання суфіксів чудовою форми, закінчень на *-ь* і подвоєних *-н-*.

5. У результаті виконання алгоритму виходить необхідна для впізнання частина слова.

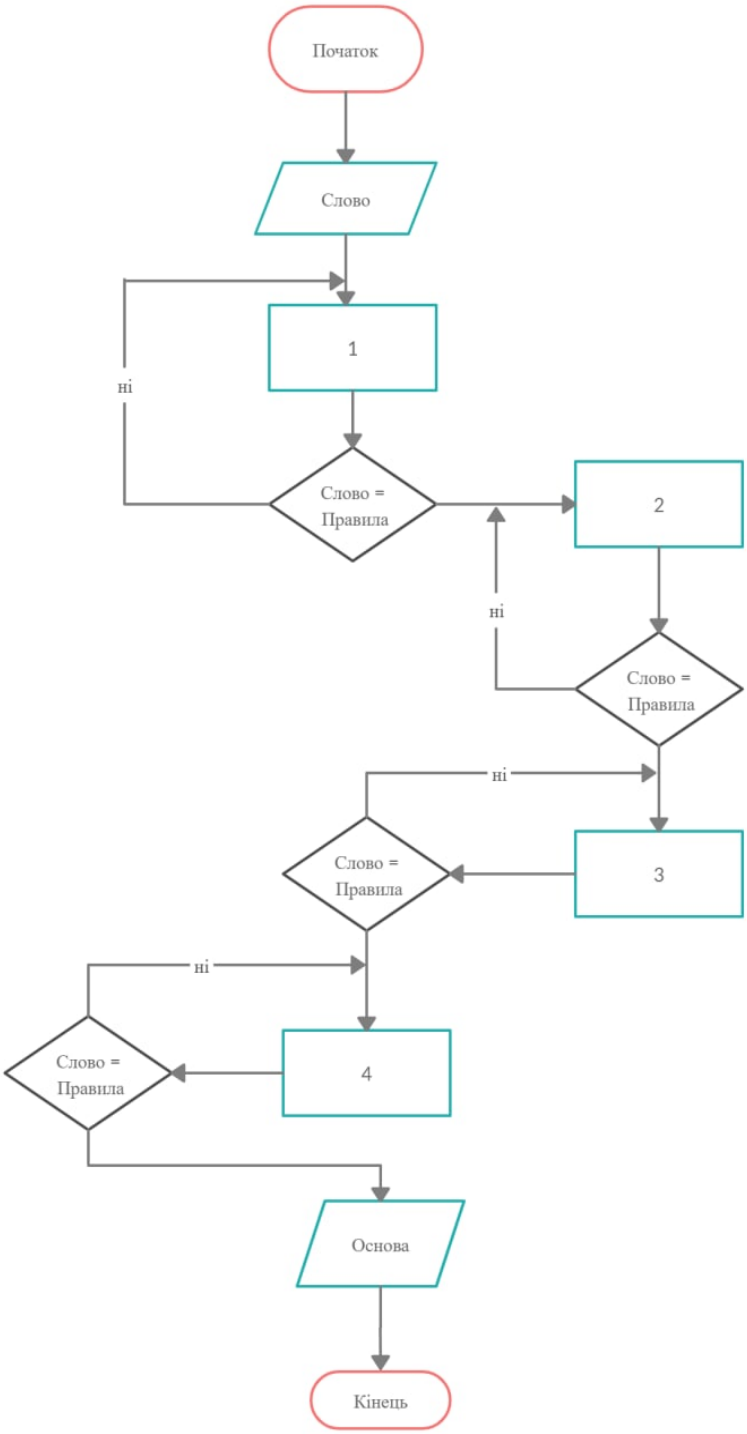

Рисунок 1.1 – Алгоритм роботи стеммінга Портера

Те, що стеммер Портера не використовує ніяких словників і баз основ, є плюсом для швидкодії і спектра застосування (він непогано справляється з неіснуючими словами), і одночасно мінусом з точки зору точності виділення стемм. Алгоритм часто обрізає слово більше необхідного, що ускладнює синтез нормальної форми по отриманій стеммі: *кровать → крова* (при цьому реально незмінна частина слова – *кроват,* але стеммер обрізає найбільш довгу морфему) і не справляється з випадаючими голосними в корені, наприклад: *кошек → кошек, кошками → кошк*. Крім того, до мінусів стеммера Портера часто відносять людський фактор: те, що правила для перевірки задаються вручну і іноді пов'язані з граматичними особливостями мови, збільшує ймовірність помилки [13].

1.4.2 Stemka

Другий відомий алгоритм стемінгу, написаний спеціально для російської мови – stemka (Андрій Коваленко, 2002). Російсько-український ідентифікатор морфології був створений за допомогою спеціально розробленого морфологічного модуля. Цей стеммер заснований на ймовірнісній моделі: слова з навчальної тексту розбираються аналізатором на пари *останні дві букви основи* + *суфікс* і якщо така пара вже присутня в моделі – збільшується її вага, інакше вона додається в модель. Після чого отриманий масив даних ранжується за зменшенням ваги і моделі, ймовірність реалізації яких становить менше 1 \ 10000, відсікаються. Результат – набір потенційних закінчень з умовами на попередні символи – інвертується для зручності сканування словоформ «справа наліво» і представляється у вигляді таблиці переходів кінцевого автомата. При розборі слово сканується по побудованим таблиць переходу [14].

Для уявлення автоматичних правил усічення слів була обрана модель зберігання можливого закінчення з двома попередніми буквами незмінної частини слова. Так, наприклад, словоформа *словарями* породжує правило, яке дозволяє відщеплення закінчення -ями за умови, що йому передує послідовність *-ар-*.

Аналогічно, зустрівши словоформу *морями*, ми породимо правило про можливе відщепленні того ж закінчення *-ами* за умови, що воно зустрілося після фрагмента *-ор-*.

Далі застосовується розроблена програма обробки масивів повнотекстової інформації, яка, розбивши текст на слова, виконувала обробку чергової потенційної словоформи точним морфологічним аналізатором; при цьому невідомі словниковому аналізатору рядки ігнорувалися, що є допустимою похибкою, оскільки, маючи базу більш 150,000 основ і розпізнаючи понад чотири мільйони різних форм російських слів, аналізатор ігнорує менш одного відсотка рядків що зустрілися, які здебільшого виявляються або орфографічними помилками, або абревіатурами, або екзотичними назвами або іменами власними [15]. Для впізнаних ж словоформ виділялася їх точна основа, тобто частина слова, що залишається незмінною при зміні або відмінюванні; виділене, у такий спосіб, закінчення укупі з останніми двома символами формальної основи надходило в накопичувач, який або реєстрував нове правило, або збільшував вагу вже існуючого правила відщеплення закінчення.

По завершенні роботи сканера текстів отриманий масив даних було проранжовано відповідно до убування ймовірності зустріти кожну з присутніх моделей словозміни, після чого моделі, ймовірність реалізації яких становила менше однієї десятитисячної, були відкинуті як рідкісні і потенційно небезпечні, тобто здатні породити надмірний шум .

При цьому stemka, на відміну від snowball, повертає набір з декількох стемм, що задовольняють таблицями. Тому при порівнянні з іншими алгоритмами прийнято розглядати два режими роботи стеммера: максимальний в глибину (агресивний) і мінімальний (консервативний). Примітно, що в процесі налагодження в цей алгоритм було додано правило, використовується також і в стеммер Портера: необхідність наявності хоча б однієї голосної в основі [16].

Оскільки він виробляє кілька варіантів основи для кожного слова, ми розглядаємо два крайні варіанти, які воно виробляє: найглибший або агресивний і

найповерховіший або консервативний. При реалізації алгоритму Stemka формується набір потенційних закінчень з урахуванням умови їх використання та представляється у вигляді таблиці переходів. Алгоритм MyStem виконує порівняння усіченого слова з існуючими у словнику. Тобто для обох реалізацій необхідне створення деякої опорної бази, що потребує додаткових вимог до об’ємів пам’яті.

1.4.3 Mystem

І третій стеммер, який розглядається, – розробка Іллі Сегаловича (Яндекс, 1998) Mystem. Модель будується в вигляді лісу інвертованих префіксних дерев суфіксів і інвертованого префіксного дерева для основ, для цього використовується словник з перерахуванням всіх граматичних форм (парадигми) слова. На першому кроці за допомогою дерева суфіксів у вхідному слові визначаються можливі межі між стеммою і суфіксом, після чого для кожної потенційної стемми (починаючи з найдовшої) бінарним пошуком по дереву основ перевіряється її наявність в словнику або знаходження найбільш близьких до неї основ (мірою близькості є довжина загального «хвоста»). Якщо слово словникове – алгоритм закінчує роботу, інакше – переходить до наступного розбиття [17].

Якщо варіант основи не збігається ні з однією з «найближчих» словникових основ, то це означає, що аналізоване слово з даними варіантом основи в словнику відсутній. Тоді за наявною основою, суфіксом і моделлю «найближчої» словникової основи генерується гіпотетична модель зміни даного слова. Гіпотеза запам'ятовується, а якщо вона вже була побудована раніше – збільшує свою вагу. Якщо слово так і не було знайдено в словнику – довжина необхідного загального закінчення основ зменшується на одиницю, йде перегляд дерева на предмет нових гіпотез. Коли довжина загального «хвоста» досягає 2, пошук зупиняється і йде ранжування гіпотез по продуктивності: якщо вага гіпотези в п'ять і більше разів менше найбільшого ваги – така гіпотеза відсіюється.

Приклад опції, що може підтримувати третя версія Mystem. Вказуються за правилами UNIX – до імен файлів, при цьому можна склеїти, об'єднати тощо:

– -*n* – порядковий режим; кожне слово друкується на новому рядку;

– -*c* – копіювати все введення на вивід. Тобто, не тільки слова, а й проміжки між словами. Опція необхідна для повернення до повного відображенню тексту;

– у разі порядкового виводу (коли задана опція n) пробілом між словами проміжки витягуються в один рядок, символи перекладу рядка замінюються на \ *r* і / або \ *n*;

– пробіл для більшої видимості змінюється на підкреслення. Символ \ змінюється на \\, підкреслення на \ _. Таким чином можна однозначно відновити вихідний текст;

– -*w* – друкувати тільки словникові слова;

– -*l* – не друкувати вихідні словоформи, тільки леми і граммеми (граматичні значення як компонентів граматичної категорії);

– -*i* – друкувати граматичну інформацію, розшифровка нижче;

– -*g* – склеїти інформацію словоформ при одній леммі (тільки при включеній опції -i);

– -*s* – друкувати маркер кінця пропозиції (тільки при включеній опції -c);

– -*e* – кодування введення / виведення.. Можливі варіанти: cp866, cp1251, koi8-r, utf-8 (за замовчуванням);

– -*d* – застосувати контекстне зняття омонімії;

– -*eng-gr* – друкувати англійські позначення граммем;

– -*filter-gram* – будувати розбори лише із зазначеними граммемами;

– -*fixlist* – використовувати файл з призначеним для користувача словником;

– -*format* – формат виведення. Можливі варіанти: text, xml, json. Значення за замовчуванням - *text*;

– -*generate-all* – генерувати всі можливі гіпотези для несловникових слів;

– -*weight* – друк ймовірності лемми, незалежно від контексту.

Результатом роботи стеммеру є отриманий набір гіпотез для неіснуючого або одна гіпотеза для словникового слова. Крім того, остання версія Mystem для кожного варіанта початкової форми пропонує всю граматичну інформацію (просинтезовану і для неіснуючих слів) і частоту його використання в IPM (instances per million) (якщо частота невідома – виводиться *0.00*) – ці дані можна використовувати в подальшому для вибору однієї нормальної форми з безлічі, запропонованого програмою [18].

Слід зауважити, що з трьох розглянутих алгоритмів тільки в останньому реалізований етап синтезу. Тобто алгоритм не тільки виділяє стемму, але й приводить до нормальної форми. Наприклад синтезує слово «*морями*» до нормальної форми «*море*».

2 ДОСЛІДЖЕННЯ І ПОБУДОВА РІШЕННЯ

2.1 Дослідження і порівняння методів

Проведемо порівняння методів Snowball, Stemka та Mystem за такими показниками як швидкодія, близькість пар за семантичним значенням, можливість приведення до нормальної форми тощо.

2.1.1 Швидкодія

Одна з характеристик, за якими буде проходити порівняння методів стеммінгу, це вимірювання швидкодії роботи алгоритмів.

Для проведення цього експерименту алгоритми було запущено на корпусі словникових слів російської мови у середовищі Intellij Idea. Було проаналізовано та виміряно, за який час алгоритми знаходили стемму 1, 100, 1000 слів.

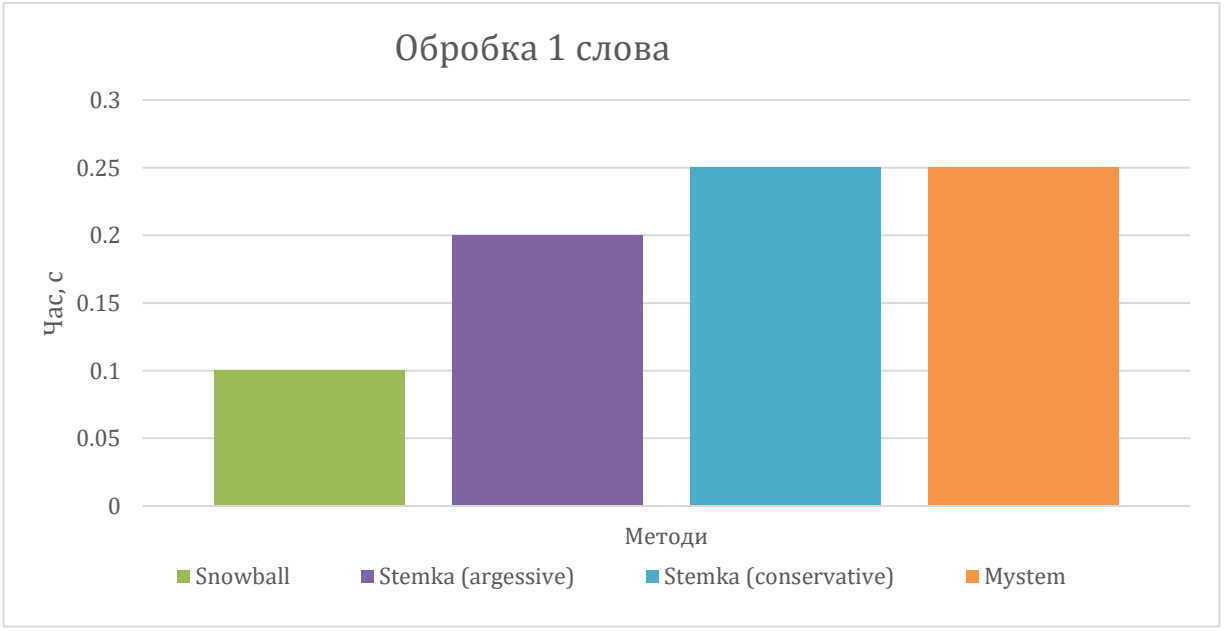

Рисунок 2.1 – Результати вимірювання швидкості знаходження стемми для 1 слова

На прикладі обробки одного слова помітно, з рисунку 2.1 помітно, що алгоритм Snowball швидше за усі інші алгоритми трохи більш ніж у 2 рази. Алгоритми Stemka та Mystem показали дуже близькі результати – близько 0,25 с на обробку одного слова. Проте, слід зазначити, що коли слово, яке береться для аналізу, має складну будову (більше двох складів) – декілька суфіксів чи складену основу, то в такому випадку алгоритм Mystem швидше порався з обробкою слова порівняно з Стемкою.

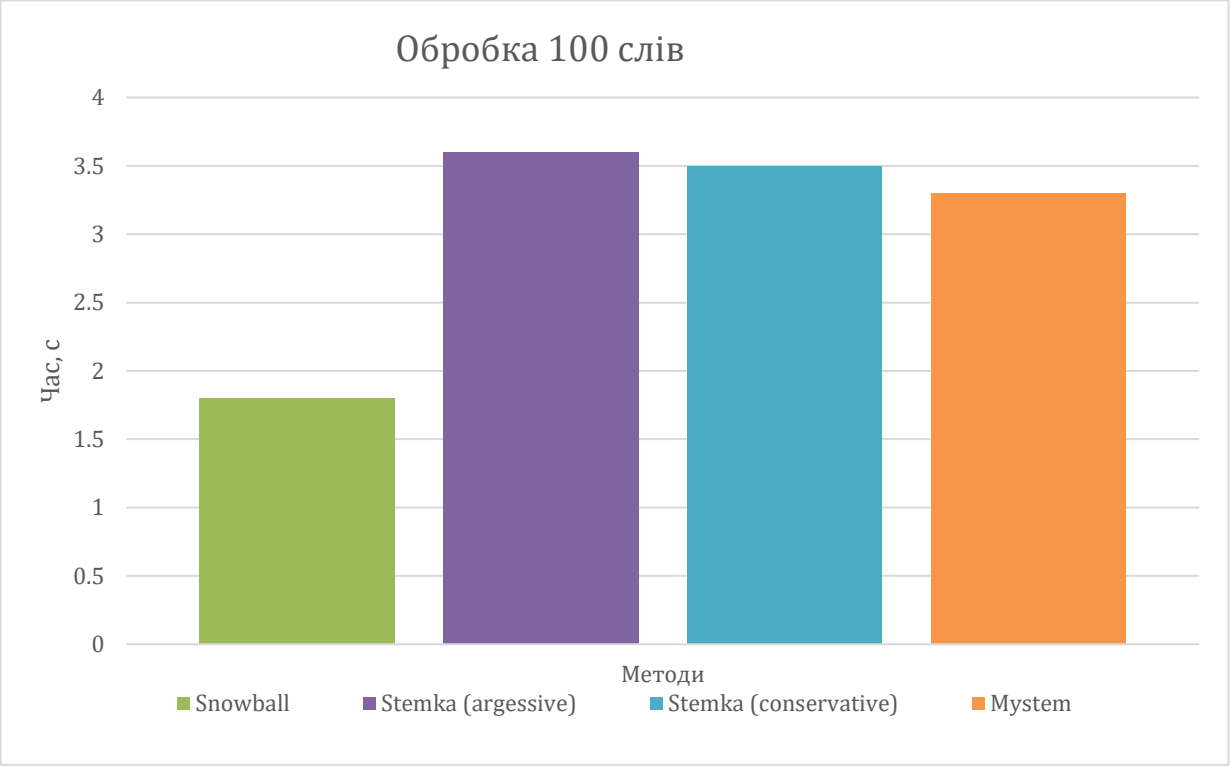

Рисунок 2.2 – Результати вимірювання швидкості знаходження стемми для 100 слів

За дослідом, приведеним на рисунку 2.2 спостерігалось наступне. При збільшенні кількості для аналізу до 100 результати швидкодії алгоритмів кардинально не змінились. Snowball як і раніше показав найшвидший результат, близько 2 секунд. Хоча на цей раз, в досліді з більшою кількістю слів, алгоритм Mystem показав дещо кращий результат за алгоритми Stemka. Можна привести припущення, що через наявність стоп-слів чи слів іншомовного походження в

даному фрагменті тексту, алгоритм Mystem впорався краще, бо в ньому передбачено обробку таких слів.

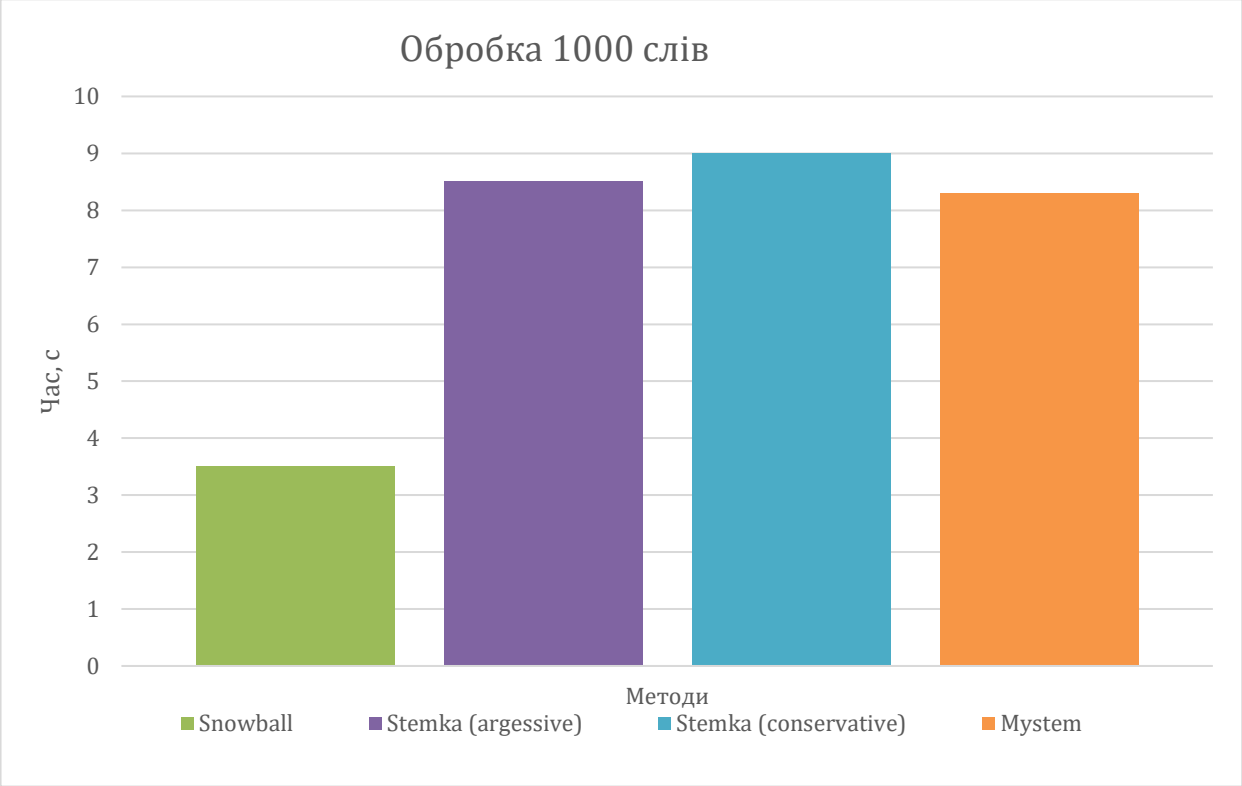

Рисунок 2.3 – Результати вимірювання швидкості знаходження стемми для 1000 слів

Як бачимо з результатів досліджень швидше за всіх зі знаходженням стемми справляється алгоритм Snowball. У середньому він показав результати в 2 рази швидше за Mystem та Stemka. Інші алгоритми маюсь приблизно схожий результат, проте помітно що для Stemka потребується трохи більше часу, в порівнянні з іншими алгоритмами. Такий надлишок в часі для двох останніх алгоритмів пояснюється тим що в них затрачується додатковий час на роботу зі словниковою базою та на ранжування потенційних стемм. Результати даного спостереження зображено на рисунку 2.3.

2.1.2 Семантична подібність

Друга характеристика, за якою буде проходити порівняння методів стеммінгу це вимірювання семантичного подібності між варіантами PPMV (PPMV – pair of potential morphological variants). Якщо дві різні форми досить близькі, вони повинні бути близькі з точки зору результатів пошуку пошукової системи (на прикладі системи Google, що є хорошою універсальною мірою тому, що вона не використовує морфологічну обробку).

Порівняння даних стеммерів узято зі статті Іллі Сегаловича. Критерієм, за яким порівнюються алгоритми, виступає кількість потенційних морфологічних пар *{форма + основа}* (т. зв. PPMV – pair of potential morphological variants), побудованих за результатами аналізу тексту. PPMV в цьому розумінні – це все пари, які відповідають одному і тому ж текстовому поданню слова, леммі (нормальній формі) або основі. Усі алгоритми запускалися на корпусі https://ruscorpora.ru/ російської мови [19].

Таблиця 2.1 – Порівняння стеммерів за семантичною подібністю

Module	*Total PPMVs*	*PPMVs with 10-100 freq*
Stemka (aggressive)	628 154	82 895
Stemka (conservative)	108 248	18 014
Mystem	519 262	73 863
Snowball	218 216	43 879
Canonical	484 462	70 028

У першому рядку таблиці 2.1 – кількість реально розмічених в корпусі пар. Щоб зменшити вибірку, залишили тільки словоформи, що зустрічаються в корпусі з частотами від 10 до 100, що свідчить про більшу специфічність терміну (10-100 ppm).

У результаті порівняння стеммерів помітно, що більше за всі алгоритми запропоновано морфологічних пар було алгоритмом Стемка (з консервативним підходом). Таких пар було знайдено більше ніж в два рази більше за кількість пар, розмічену у корпусі, що може свідчити про надлишковість алгоритму Стемка. Хоча Mystem також знайшов деяку кількість зайвих пар, проте на відміну від інших алгоритмів, додані Mystem пари найбільш семантично близькі. Також у Mystem отримано мінімальну кількість втрачених пар з усіх алгоритмів і найкращу семантичну близькість.

2.1.3 Виділення стемми

Для практичного порівняння правильності виділення стемми було запущено три алгоритми на корпусі російської мови.

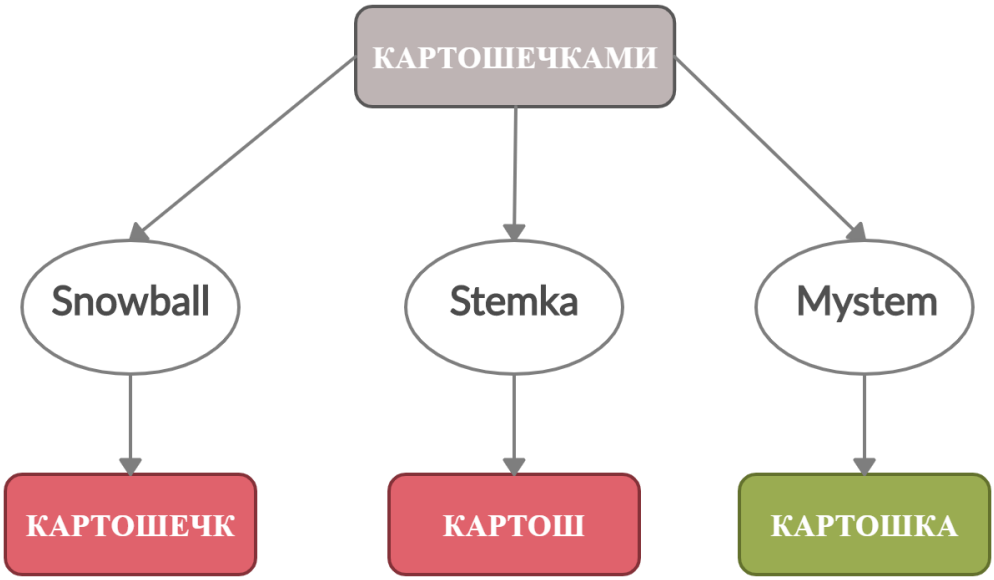

Рисунок 2.4 – Виділення стемми слова зі змінним закінченням

Для прикладу було взято слово «*картошечками*», що прийшло для аналізу в формі множини, непрямий відмінок (це свідчить про те, що слово знаходиться не в нормальній формі). Після проведення різних методів над цим словом отримали наступні результати, що показані на рисунку 2.4: методи Snowball та Stemka лише дали нам приближену стемму, проте в першому методі стемма була виділена не остаточно. Алгоритм Mystem не тільки привів слово до нормальної форми – число однини, називний відмінок, а й запропонував саме початкову форму «*картошка*», замість «*картошечка*». Запропонована гіпотеза «*картошка*» мала найбільшу вагу, через це й була запропонована алгоритмом Mystem.

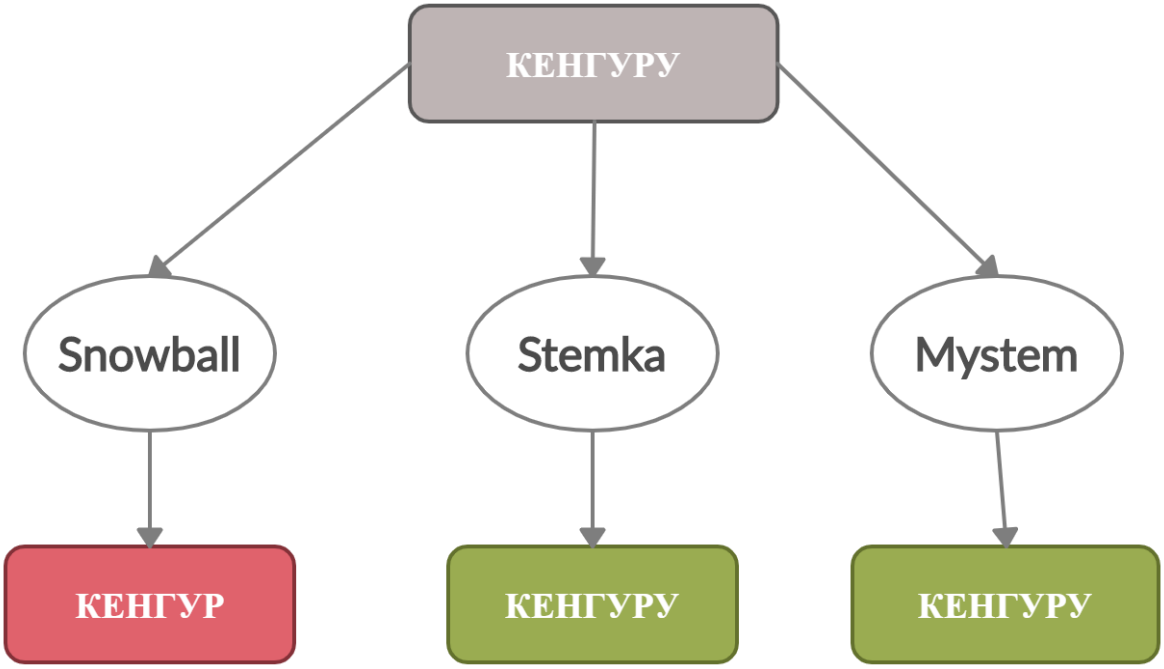

Рисунок 2.5 – Виділення стемми слова з незмінним закінченням

Також було проведено експеримент, зображений на рисунку 2.5, по виділенню стемми для слова з незмінним закінченням. Наприклад слово «*кенгуру*» це не є непрямий відмінок слова «*кенгура*». Та не зважаючи на це алгоритм Snowball не має можливості розрізнити незмінні слова, тож ним було виділено

стемму «*кенгур*». Це не є критичною помилкою, проте якщо порівнювати з результатами роботи методів Mystem та Stemka алгоритмам вдалось розпізнати незмінну форму слова та подати її в якості стемми (що тут також є нормальною формою) – «*кенгуру*».

Правильність виділення стемми в даному випадку може бути досягнута завдяки використанню словникової бази та оцінюванню потенційних гіпотез.

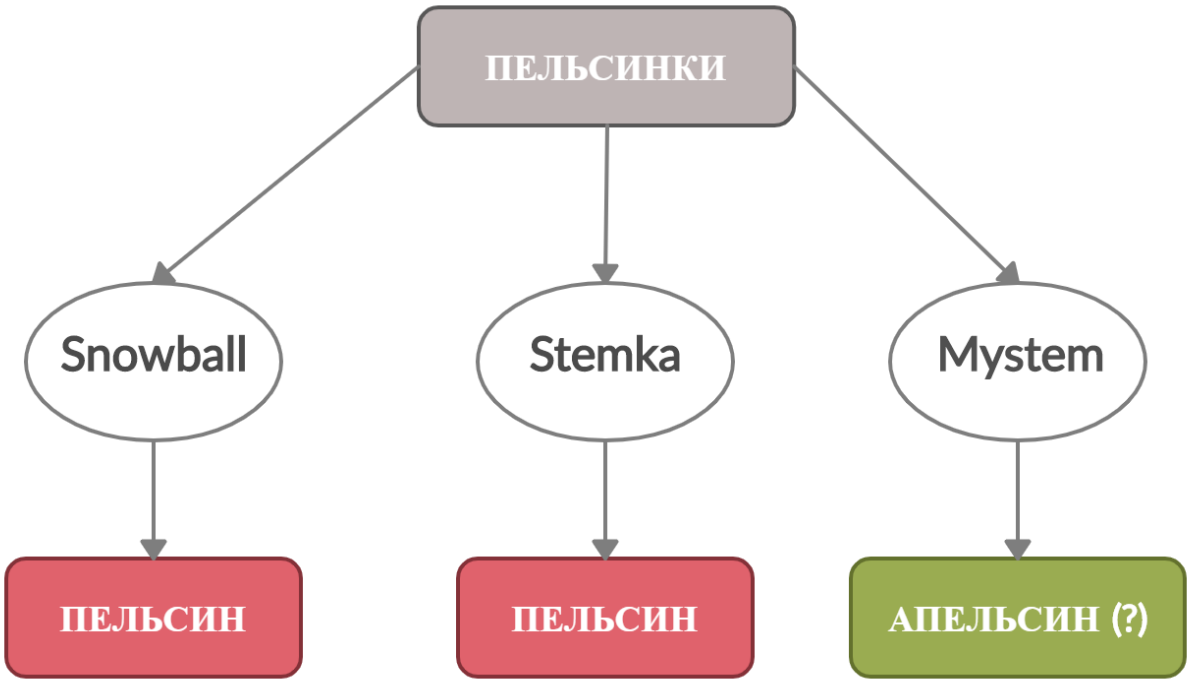

Рисунок 2.6 – Виділення стемми неіснуючого слова

Робота алгоритмів по обробці неіснуючих слів було проведено на прикладі слова «*пельсинки*». Як бачимо з рисунку 2.6 методами Snowball та Stemka було відсічено суфікс -*ки* від неіснуючого слова: «*пельсинки*» → «*пельсин*». З точки зору правильності роботи методу, алгоритм відпрацював корректно. В цьому слові дійсно можна вважати запропоновану стемму незмінною частиною. Проте якщо провести аналіз цього слова, можно привести припущення, що при наборі слова «*апельсинки*» була допущена помилка та не надруковано першу літеру. Алгоритм Mystem як результат своєї роботи подав саме варіант «апельсин», як набільш

вірогіднішу гіпотезу з усіх запропонованих. Досягти такого результату вдалось завдяки тому, що при пошуку передбачуваної стемми з дерева стемм, був зроблений ще один крок до знаходження вірогідної форми слова, отримані гіпотези було проранжовано та після порівняння зі словниковою базою будо запропоновано найбільш ймовірнісний варіант.

2.2 Обґрунтування вибору методу стеммінга

Було розглянуто три методи для знаходження стемми слова, а саме: Snowball, Stemka та Mystem.

Для подальшої роботи та розвитку за основу було обрано алгоритм Mystem. Саме цей алгоритм продемонстрував найкращі результати при дослідженні семантичної подібності та найточніше виділенні стемми. Так Mystem поступається швидкістю тому ж Snowball, через те що працює зі словниковою базою, проте цей показник має нижчий пріоритет та серйозність, тому деякою повільністю роботи алгоритму можна знехтувати. У подальшому алгоритм Mystem може підлягати оптимізації задля покращення швидкості опрацювання слів.

Також при незначній модифікації алгоритм дозволяє робити припущення про суфікс нормальної форми по підібраній парадигмі, тобто зачіпає другий етап нормалізації. Лише в алгоритмі Mystem реалізовано приведення припущень щодо нормальної форми слова (Називний, прямий, відмінок, форма однини). Цей фактор є основним та вирішальним, так як в алгоритмах Snowball та Stemka не вирішується основна проблема нормалізації – приведення слова до нормальної форми, а лише знаходиться стемма слова.

2.3 Розгляд етапів нормалізації

Як вже зазначалось раніше, процес нормалізації слів природної мови можна розділити на дві частини: визначення стемми і підбір відповідного суфікса для

нормальної форми. В якості стеммера було реалізовано дещо модифікований алгоритм пошуку основи, використаний в Mystem.

Схематично роботу алгоритму приведено в вигляді схеми алгоритму на рисунку 2.7.

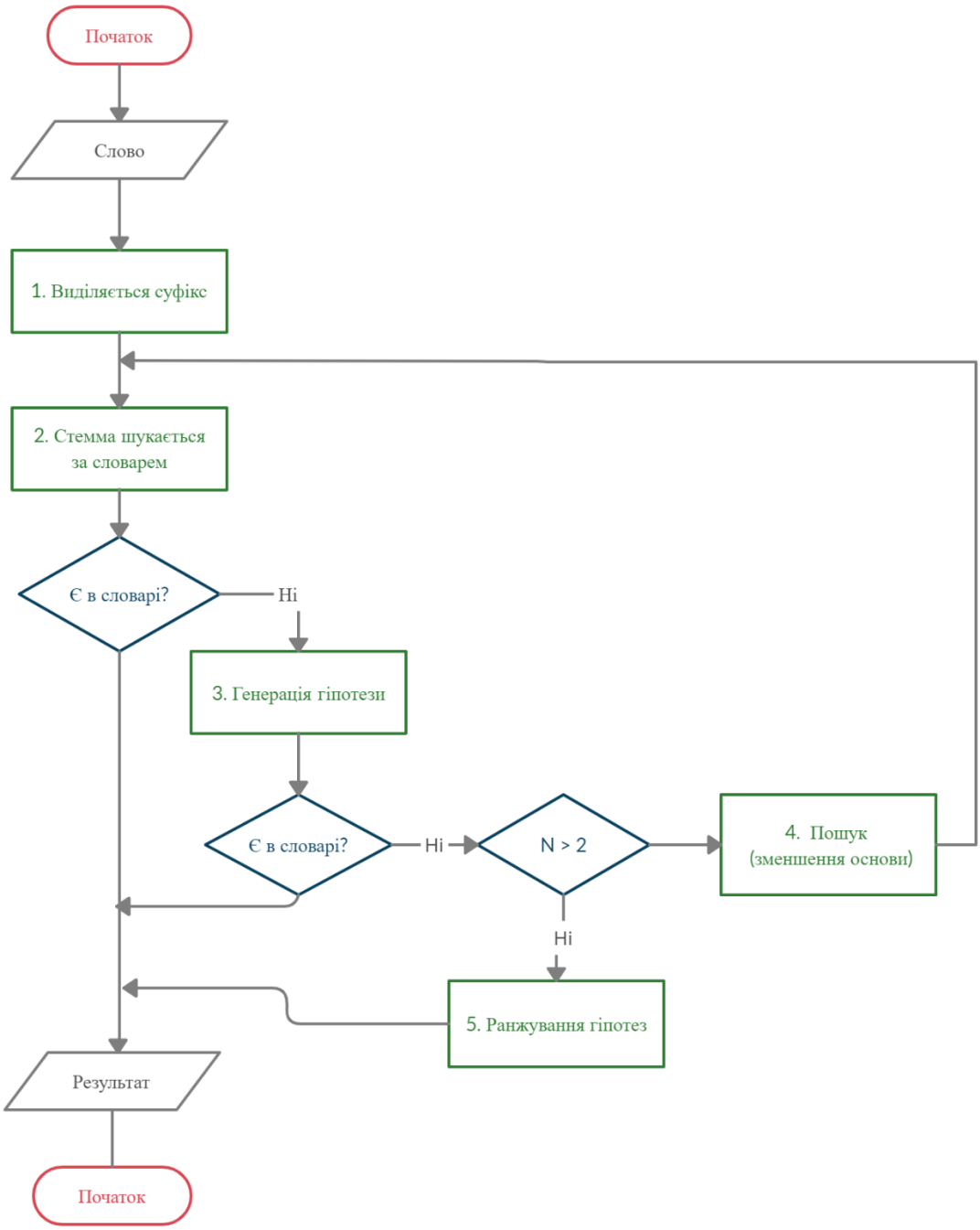

Рисунок 2.7 – Алгоритм роботи методу Mystem

Для вирішення проблеми нормалізації розглянемо наступні етапи.

1. Побудова моделі.

Для побудови словникової моделі була використана повна парадигма по А. А. Залізняку [20].

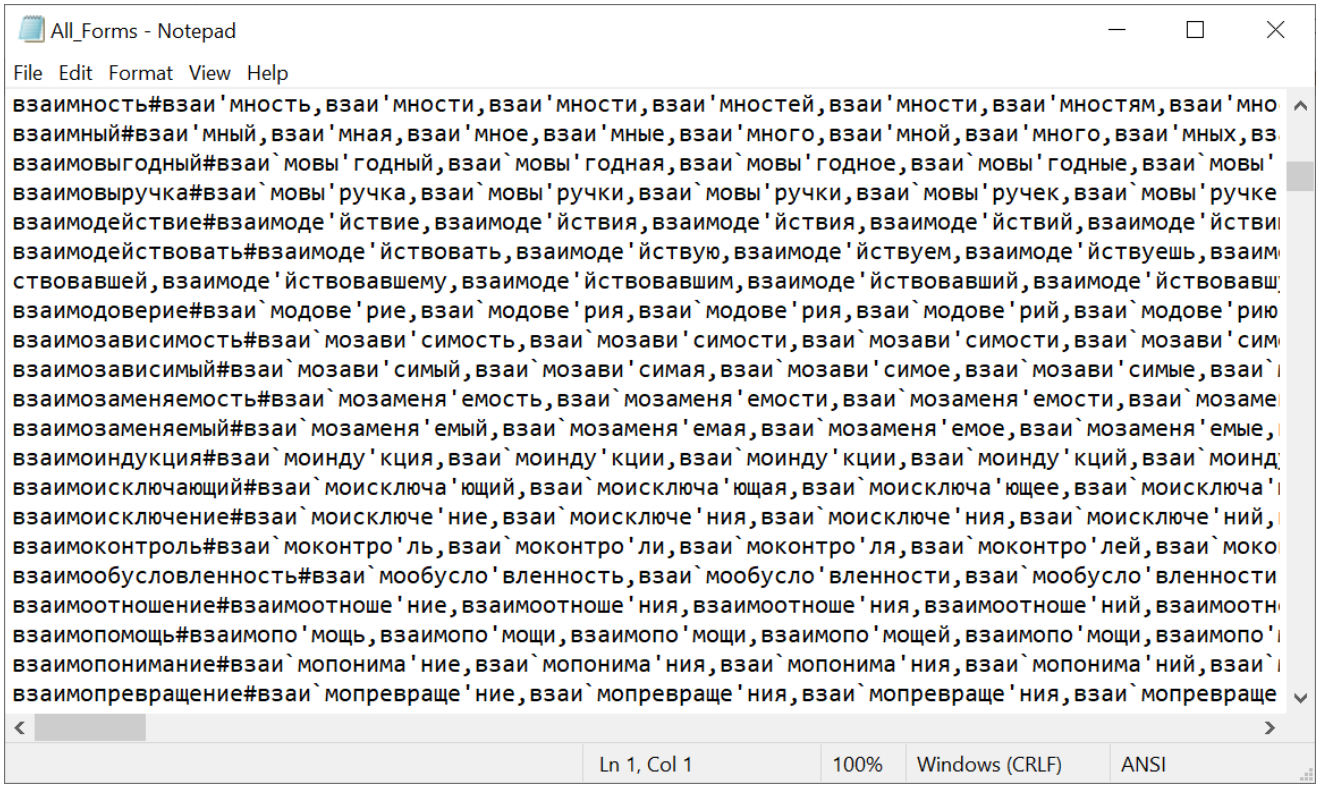

Рисунок 2.8 – Варіант подання парадигми по А.Залізняку

На рисунку 2.8 приведено приклад файлу, що використовується в якості словникової бази, з перерахуванням словникових моделей у вигляді: *порядковий номер* + *нормальна форма* + *частина мови* + *стемм* + *можливі суфікси*. Частина мови обчислюється по наявності тих чи інших суфіксів в парадигмі. Так, наприклад, всі іменники в формі місцевого відмінка множини мають закінчення *-ах / -ях*. У суперечливих випадках обирається частина мови, що має велику частоту використання (відповідно до даних, представленими ruscorpora.ru).

Прийнято рішення розділити частини мови на 4 формальні групи: іменник, дієслово, прикметник, незмінне слово – в порядку зменшення частоти використання. Таке розбиття задовольняє використовуваному алгоритму і не

викликає проблем при використанні. Дієприкметник і дієприслівник вважаються формами дієслова, порядкові числівники належать до прикметників, кількісні – до імен іменників [21].

2. Побудова внутрішнього уявлення для моделі.

У відповідності до записаного файлу словникової моделі аналогічно алгоритму Mystem будується префіксне дерево інвертованих основ і аналогічне дерево суфіксів. У тих вузлах дерева стемм, які відповідають першій букві основи, зберігається безліч номерів флективних моделей, які містять дану стемму. Будується внутрішнє представлення моделі в вигляді масиву структур, кожна з яких містить всю інформацію про парадигму, визначену в пункті 1.

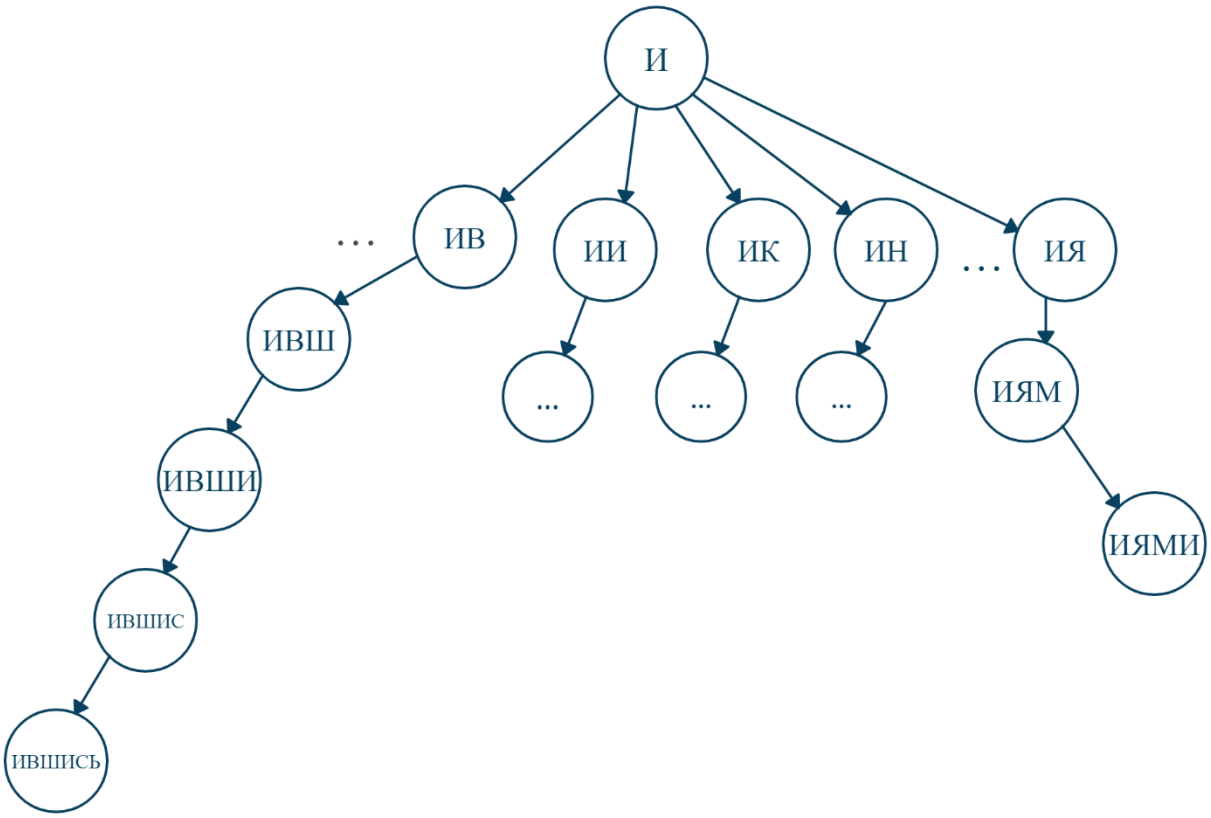

Рисунок 2.9 – Приклад побудови дерева суфіксів

На рисунку 2.9 зображено наочний приклад як може виглядати дерево суфіксі. Пошук розпочинається к кореня дерева, в даному випадку з суфіксу *-и*, і далі отримувати різні варіанті суфіксів на першому рівні глибинного пошуку по

дереву – *-ив, -ик, -ии, -ин, -ия* тощо. Якщо зробити ще один крок, то можна отримати такі варіанти як *-ивш, -иям*. Пошук по дереву може продовжуватись до того моменту поки не будо досягнуто листа, від якого далі не йде розгалуження. На даному фрагменті дерева суфіксів маємо два листа: суфікси *-ившись* та *-иями*. Наприклад для слова «*утопившись*» буде знайдено нормальну форму «*утопить*» завдяки тому, що буде визначено суфікс *-ившись* та стему «*утоп*».

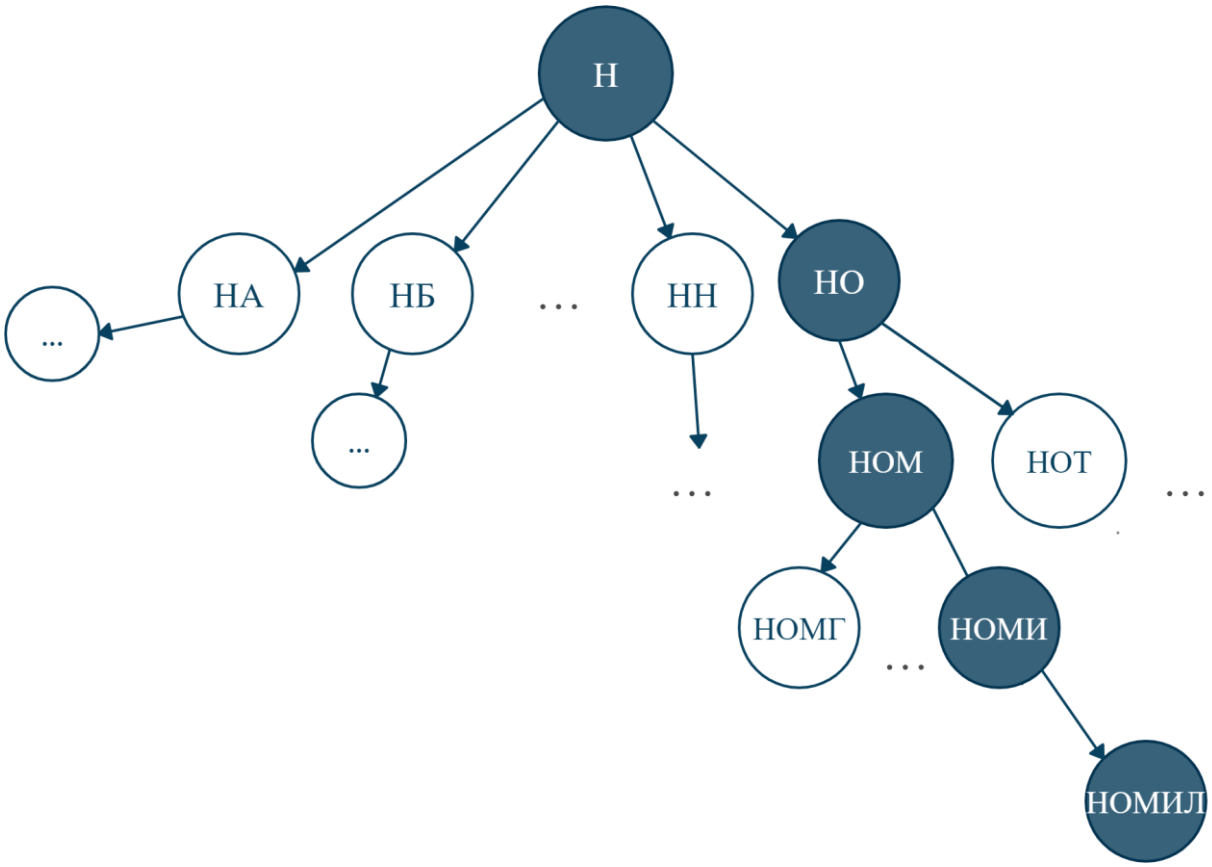

Рисунок 2.10 – Приклад побудови префіксного дерева інвертованих основ

На рисунку 2.10 відображено приклад побудованого префіксного дерева, що містить інвертовані основи слова. Основи слов беруться зі словникової моделі побудованій на повній парадигмі по А.А. Зализняку. Ці основи інвертуються та за принципом префіксного дерева розбиваються та утворюють вузли, тим самим формуючи дерево стемм. На наведеному прикладі бачимо як проходить пошук стеми слова «*лимон*». Початок пошуку починається з літери *«и»*, далі по дереву

проходить пошук потрібний ключ, бінарним порівнянням зіставляються два варіанти, серед них обирається потрібний або пошук продовжується далі. У процесі пошуку однієї основи можна отримати чи зібрати будь які варіант, як на даному випадку можливо отримати стемму «*тон*» чи «*гном*», які також представлені у словниковій базу. У даному випадку пошук необхідної стемми по дереву суфіксів продовжувався до того моменту, поки не було досягнуло листа «*номил*», що є інвертованою основою слова «лимон».

3. Підготовка вхідного тексту.

У вхідному тексті зчитується почергово кожен стовпець, де кожен рядок розбивається на слова, видаляються знаки пунктуації чи спрощується, *«ё»* замінюється на *«е».*

Так як алгоритм передбачає можливе розширення у майбутньому на корпус слів української мови то пропонується покрити одразу й особливості, що притаманні для специфіки цієї мови. Наприклад виключення із слова символу апострофа та рівнозначність букв *«ґ»* та *«г».*

4. Отримання гіпотези про нормальну форму для кожного слова.

Кожне слово в рядку перевіряється на приналежність списку «стоп-слів».

Стоп-слова - це слова-зв'язки. До них відносяться:

– союзи та союзні слова (або, але, щоб, потім, потім, як тільки);

– займенники (він, ми, його, ви, вам, вас, її, що, який, їх, все, вони, я, весь, мені, мене, таким);

– прийменники (для, на, по, з, від, до, без, над, під, за, при, після, під);

– частинки (ні, ж, то, б, все, все, навіть, так, ні);

– вигуки (ой, ого, ех, браво, здрастуйте, спасибі, вибачте);

– цифри і числівники (1, 2, 3 один, два, три, сто, перший, другий, третій, сотий і т. д.);

– розділові знаки і спеціальні символи (., - _ = + /!;:%? *);

– вставні слова (скажімо, може, припустимо, чесно кажучи, наприклад, насправді, однак, взагалі, в загальному, ймовірно;

– окремо написані літери (а, б, в, і т. д.);

– невизначені союзи, прислівники і деякі звичайні прислівники (щось, якийсь, десь, якось, далі, ближче, раніше, пізніше, коли-то);

– слова-підсилювачі (дуже, мінімально, максимально, абсолютно, величезний, гранично, сильно, слабо, самий);

– ряд деяких іменників, дієслів, прислівників (наприклад, сайт, давати, завжди, однак і ін.).

Без стоп-слів практично неможливо написати зв'язний, добре читабельний текст. Текст, в якому зовсім немає стоп-слів, сухий і виглядає неприродньо. Але якщо їх більше, ніж потрібно, це погано. Насичений стоп-словами текст гірше сприймається. Стоп-слова не несуть смислового навантаження і можуть заважати чітко побачити основну інформацію в тексті. Крім того, всі слова, що мають довжину менше трьох символів, також не обробляються. Якщо слово є «стоп-словом», воно записується в рядок результату без змін, алгоритм переходить до обробки наступного слова [22].

Для кожного слова, що не є «стоп-словом», запускається алгоритм виділення стемм. За допомогою дерева суфіксів від слова відсікається суфікс і відбувається пошук передбачуваної стемм в дереві стемм. якщо стемма знаходиться в дереві – перевіряємо, чи можливо поєднання даної стемми і даного суфікса, чи задовольняє отримана модель необхідної частини мови. Якщо так – повертається лема, відповідна даної моделі. Якщо немає – проводиться пошук стемм, найближчих до даної, і вибираються ті з них, які поєднуються з даними суфіксом.

На цьому ж етапі для кожної «найближчої» стемми запам'ятовується ступінь близькості – це буде використано при виборі єдиною леми з безлічі. По кожній з обраних моделей синтезуються нормальні форми для оброблюваного слова. Якщо розглянута стемма містить хоча б одну голосну – від вихідного слова відбувається відсікання наступного за довжиною суфікса і пошук повторюється.

5. Синтез нормальної форми для знайдене слів.

Від леми, відповідної використовуваної для синтезу моделі, відсікається стемм і отримане закінчення приписується до передбачуваної стемми оброблюваного слова. До кожної нормальній формі з безлічі додається префікс у вигляді довжини загальної частини стемми використаної моделі з оброблюваних словом (ступінь близькості). Якщо це слово є словниковим, то цей крок пропускається.

6. Вибір єдиної нормальної форми.

До кожної нормальної форми, отриманої на попередньому кроці, додається префікс з номера відповідної частини мови. Якщо в процесі пошуку виявилося, що слово не є словниковим (тобто стемм досягла мінімальної довжини, а пошук не завершений), з результуючої множини вибирається лема, утворена за моделлю, найбільш близькою до вихідного слова.

Якщо таких моделей декілька, серед тих моделей, які мають найбільш пріоритетну в отриманому безлічі частина мови, вибирається мінімальна у лексикографічному сенсі лема. Отримані леми окремо запам’ятовуються та ранжуються відповідно до більш вірогідних в даному контексті. Для словникових слів вибір відбувається аналогічно, минаючи етап вибору найбільш близьких стемм.

7. Отримання результату.

Нормалізовані слова записуються в рядок у вихідному порядку, повністю оброблений рядок записується в файл результату.

Використання примітивної розмітки частин мови допомагає скоротити кількість гіпотез для знайдене слів, тому що найчастіше ці гіпотези відносяться до різних частин мови. Крім того, визначення потрібної частини мови допомагає уникнути досить часто зустрічається проблеми даного алгоритму стеммінг: в разі, коли *стемма + суфікс* збігаються зі стеммою деякого іншого словникового слова («*замок*» → «*замокнуть*»).

Для вибору між лемами, що відносяться до однієї частини мови, необхідний вже аналіз сенсу контексту, що не завжди можливо однозначно зробити. Класичний приклад: «*Эти типы стали есть в литейном цехе*» – в залежності від змісту даного речення, слово «*стали*» може бути формою дієслова «*стать*», а може входити в парадигму слова «*сталь*».

3 АНАЛІЗ ІСНУЮЧИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАСОБУ

3.1 Вибір технологій реалізації

Для зручної та зрозумілої демонстрації роботи алгоритму та порівняння отриманих результатів нормалізації було обрано веб-орієнтовану систему реалізації. Оскільки у рамках роботи розробляється саме веб-система. то вона повинна бути реалізована із застосуванням веб-технологій.

Веб-додаток – це обраний тип програми, побудований на архітектурі клієнт-сервер. Його особливість полягає в тому, що саме веб-додаток знаходиться і виконується на сервері, а клієнт отримує тільки результати своєї роботи. Робота такого додатка заснована на отриманні запитів від користувача, їх обробці та видачі результатів запиту. Передача запитів і наслідки їх обробки здійснюються через Інтернет. Однією з переваг цього підходу є те, що клієнти не залежать від операційної системи конкретного користувача, що робить веб-додатки багатоплатформовими.

Конкретний додаток, що зветься браузер (Google Chrome, Safari, Opera і ін.) – зазвичай займається відображенням результатів запитів клієнтів, а також отриманням від нього даних і відправкою їх на сервер. Одна з основних функцій браузера – відображати дані, зібрані з Інтернету, у вигляді сторінки, описаної в HTML. Результат, що передається клієнту сервером, має бути представлений цією мовою. Веб-додаток на серверній стороні працює за допомогою спеціального програмного забезпечення (веб-сервера), яке приймає запити користувачів, обробляє їх, генерує відповідь у вигляді сторінки, описаної в HTML, і відправляє його клієнту [23].

Архітектура веб-системи являє собою особливу модель взаємодії між життєво важливими «учасниками» веб-додатками, як внутрішніми, так і

зовнішніми. Веб-додатки працюють за принципом взаємодії клієнт-сервер, припускаючи одночасну роботу двох програм. Один з них обробляє запити користувача через веб-браузер, показуючи їм результати пошуку. Інший, заснований на сервері, обробляє відправлений HTTP-запит.

Оскільки і браузер (клієнт), і респондент (сервер) працюють з певними кодами, вибір мови програмування повинен ґрунтуватися на взаємодії цих кодів – чи буде код сервера успішно взаємодіяти з кодом браузера? Фахівці, що працюють в цій галузі, на стороні сервера зазвичай використовують такі мови:

– JavaScript;
– PHP;
– Python;
– Ruby on Rails і т. д.

Що стосується коду на стороні користувача, найбільш поширеними комп'ютерними мовами є:

– HTML;
– JavaScript;
– CSS.

Слід зазначити, що код «браузера» може бути відкритий і змінений клієнтом, на відміну від «серверних» скриптів. Браузер працює тільки з HTTP-запитом, відповідаючи на введені користувачем дані.

3.2 Технології реалізації клієнтської частини

Клієнтська складова – це те, що бачить користувач на екрані браузера. Веб-проект будується з тексту, зображень, посилань, списків, форм введення даних, таблиць і тому подібне. По суті, веб-розробка інтернет-сайту полягає в розміщенні всіх його текстових і графічних елементів в потрібному місці і надання їм необхідних форм / властивостей.

Наприклад, простий текст можна оформити у вигляді списку або заголовка і опублікувати його посередині екрана. Зображення можна збільшити або зменшити, зробити його посиланням і помістити в потрібному місці. Таким чином, створення клієнтської частини веб – це редагування тексту / графіки веб-сторінки, тільки не традиційним способом (як це роблять мишею), а через написання спеціального коду на HTML, CSS і JavaScript.

3.2.1 Мова гіпертекстової розмітки HTML

HTML – це мова розмітки гіпертексту, яка дає змогу користувачу створювати розділи, структурувати параграфи, блоки, заголовки, посилання для веб-сторінок і додатків.

HTML не є мовою програмування, через те що ця мова немає можливості для програмування динамічних функцій. Проте не зважаючи на це, властивості мови дозволяють нам робити організацію та форматування документів, дещо схожим до можливостей Microsoft Word. При роботі з HTML використовуються прості структури коду (такі як атрибути, теги), задля того, щоб розмітити сторінку веб-сайту.

HTML-документи – це файли, які закінчуються розширенням .html або .htm. Ви можете переглядати його за допомогою всіляких веб-браузерів (наприклад, Google Chrome, IE11 чи Opera). Браузер може зчитувати HTML-файл, відображати його вміст, для того що користувачі інтернету мали можливість його переглядати.

Зазвичай середньо-статистичний веб-сайт складається з кількох різних HTML-сторінок. Наприклад: домашні сторінки, так звані стартові, звичайні сторінки, елементи навігації, меню, контактів матимуть окремі HTML-документи.

Кожна окрема HTML-сторінка складається з набору тегів (також відомих як набір елементів), які можна вважати за будівельні блоки веб-сторінок. Вони створюють ієрархію, яка структурує контент за параграфами, розділами, заголовкам та іншим блокам контенту.

Більшість елементів HTML мають відкриття і закриття, в яких використовується синтаксис <tag> </ tag>. HTML-теги мають два основних типи: блок-рівень і вбудовані теги [24].

Елементи рівня блоку займають весь вільний простір і завжди запускають новий рядок в документі. Параграфи та заголовки – це один з найяскравіших прикладів блокових тегів.

Вбудовані елементи займають стільки місця, скільки їм потрібно, і не запускають новий рядок на сторінці. Вони зазвичай служать для форматування внутрішнього вмісту елементів рівня блоку. Посилання та підкреслені рядки – хороші приклади вбудованих тегів.

До переваг HTML можна віднести такі:

– широко використовується мова з великою кількістю ресурсів і величезним співтовариством;

– сторінки здатні запускатися в кожному веб-браузері;

– знаходиться у відкритому доступі та є цілком безкоштовним;

– чиста і послідовна розмітка;

– офіційні веб-стандарти підтримуються консорціумом World Wide Web;

– легко інтегрується з базовими мовами, такими як PHP і Node.js.

Серед недоліків HTML, яких слід зазначити небагато, можна виділити такі:

– переважно використовується для статичних веб-сторінок. Для динамічної функціональності може знадобитися використовувати JavaScript або бекенда-мову, такий як PHP;

– не дозволяє користувачеві реалізувати логіку. В результаті всі веб-сторінки потрібно створювати незалежно, навіть в тому випадку, якщо вони використовують однакові елементи, як наприклад, заголовки і параграфи;

– деякі браузери повільно зчитують нові функції;

– іноді поведінку браузера складно передбачити (наприклад, старі браузери не завжди створюють нові теги).

3.2.2 Таблиця каскадних стилів CSS

Технологія CSS (англ. Cascading Style Sheets – каскадні таблиці стилів) з'явилася як розвиток технології HTML, коли образотворчих можливостей розмітки стало недостатньо. Більш того, було вирішено відокремити завдання семантичної розмітки тексту від її візуального представлення. Наприклад, текст «Війна і мир» – це заголовок першого рівня і це визначає код HTML <h1> Війна і мир </ h1>, а то, що текст забарвлений в зелений колір – визначає CSS h1 {color: green;}. Відділення форми від змісту спрощує підтримку коду і обробку тексту пошуковими машинами. CSS відповідає тільки за зовнішній вигляд документа. Змінюючи код CSS, один і той же документ можна представити в різному вигляді, змінити дизайн сайту не змінюючи його зміст [25].

Визначимо таки переваги застосування CSS:

– мова CSS є простою в своєму синтаксисі, разом з цим дає можливість скоротити час на розробку дизайну та підтримку сайту;

– наявні кілька варіантів дизайну сторінки для перегляду на різних пристроях. Наприклад, дизайн на екрані комп'ютера розрахований на одну ширину, і буде повністю виводитися на екран, а на мобільних пристроях він буде підлаштовуватися до розмірів екрану і деякі елементи будуть виключені від показу, також і при друці, буде друкуватися потрібний текст, без зайвого (наприклад, без шапки меню);

– збільшується швидкість завантаження сторінок web-сайту за рахунок того, що правила представлення даних переносяться в окремий CSS-файл. Завдяки цьому браузер підгружає тільки основну структуру документа, а також дані, що зберігаються на сторінці. Представлення цих даних завантажується браузером тільки один раз і може кешуватись – завдяки чому відбувається зменшення трафіку, часу завантаження, а також навантаження на сервер;

– простота зміни дизайну. Один CSS керує відображенням безлічі HTML-сторінок. Коли виникає необхідність змінити дизайн сайту, то нема чого

правити кожну сторінку. Для зміни дизайну в майбутньому всього лише слід змінити CSS-файл, і як результат, зміна дизайну робиться швидше;

– CSS забезпечує додатковими можливостями форматування, про які при використанні тільки самих атрибутів навіть і не мріяли;

– сумісність з різними платформами забезпечується завдяки використанню web-стандартів [26].

Серед недоліків CSS можна виділити такі:

– різні відображення верстки в різних браузерах. Якщо браузери застарілі, то можливо, що одні й ті ж дані CSS по-різному ними інтерпретуються;

– при необхідності внесення виправлень, буде задіяно не тільки один CSS-файл, але й HTML-теги. Часто на практиці трапляється необхідність виправляти не тільки один CSS-файл, але й залежні теги HTML, що пов'язані з селекторами CSS. Іноді це значно підвищує час редагування, а також і тестування.

3.2.3 JavaScript

JavaScript – це мова програмування, що є прототипно-орієнтованою. Вона наслідує мову ECMA-Script, чиїм прототипом спочатку і була. Перша версія мови з'явилася в 1995 році і з тих пір має тенденцію до постійного вдосконалення, поки не набула нинішнього стану.

Найчастіше ця мова використовується в розробці додатків і браузерах з метою надання їм інтерактивності і «жвавості». За допомогою неї доступні до виконання такі функції:

- можливість змінювати сторінки браузерів;
- додавання або видалення тегів;
- зміна стилів сторінки;
- інформація про дії користувача на сторінці;
- запит доступу до випадкової частини вихідного коду сторінки;

– внесення змін до цього код;

– виконання дії з cookie-файлами.

Область застосування цієї мови дивно обширна і нічим не обмежена: серед програм, які використовують JS, присутні і тестові редактори і додатки (як для комп'ютерів, так і мобільні і навіть серверні), і прикладне програмне забезпечення (ПЗ) [27].

Виділимо такі переваги JavaScript:

– ні один сучасний браузер не існує без підтримки JavaScript;

– використанню плагінів і скриптів, що написані на JavaScript, під силу навчитися навіть без фахової підготовки;

– корисні функціональні настройки;

– мова увесь час удосконалюється – зараз проходить розробка бета-варіації проекту JavaScript2;

– взаємодія з додатком може здійснюється навіть через текстові редактори – Microsoft Office і Open Office;

– перспектива використання мови в процесі навчання програмуванню і інформатиці.

3.3 Технології проектування серверної частини

Серверна частина веб-додатку – це програма чи скрипт, який виконується на стороні сервера, що оброблює запити користувача (а саме, запити браузера). Кожного разу як користувач переходить по посиланню, браузер відправляє запит до сервера. Сервер виконує обробку цього запиту, викликаючи деякий скрипт, що формує веб-сторінку, яка описана мовою розмітки, і відсилає клієнтові результат по мережі. Браузер в цей час відображає отриманий результат у вигляді відповідної веб-сторінки [28].

Найчастіше серверна частина веб-додатки програмується на таких мовах програмування як PHP, Java та ASP.NET.

3.3.1 PHP

PHP – є мовою програмування, яка виконується на стороні WEB-сервера для динамічної генерації HTML-сторінок. Про це каже і сам переклад назви та її розшифровка з англійської: PHP – (препроцесор гіпертексту) Personal HyperText Processor.

PHP – є однієї з небагатьох мов програмування, що було створено саме для розробки веб-додатків. Через це в цій мові передбачені всі функції, які необхідні для роботи з веб-серверам. При цьому ця мова позбавлена надмірності, як властивої багатьом іншим мовам програмування.

Примітною особливістю PHP є те, що команди заключні в HTML-сторінки за допомогою спеціальних тегів, що і спонукають машину виконувати необхідні дії на сервері [29]. Програми, що пишуться на PHP не мають необхідності в спеціальних CGI-директорії з правами на доступи. Крім того, «простий» HTML і PHP-код може довільно чергуватися на одній сторінці.

Синтаксис у цієї мови подібний до мови C. Такі елементи, як наприклад масиви асоціацій, запозичені з Perl. Робота програми може стартувати з оператора PHP, описувати змінні не потрібно.

Мова компілює код всередині <? Php?> обмежувачів. Дані поза цих тегів виводяться в стандартній розмітці HTML-документа. Ім'я змінної починається з символу $ і є регістр залежним, як і імена функцій, класів і констант. Змінні зчитуються взяті в подвійні лапки або апострофи рядки, а також тих, що були створені за допомогою <<< оператора [30]. Скрипти опрацьовуються інтерпретатором в порядку, який забезпечує кроссплатформенність додатків.

Проте серед недоліків можна зазначити протиріччя в коді. Коли шаблонізатор був тільки створений, все програмне забезпечення розроблялися за допомогою C. Тому в мові було застосовано безліч синтаксису з нього. У той же час, сучасна аудиторія більше сконцентрована на Java. У підсумку, код

переповнений різними залишками з різних мов. І всі вони можуть навіть бути сконцентровані в одному вираженні коду .

3.3.2 Java

Java є об'єктно-орієнтованою мовою. Вона може підтримувати поліморфізм, спадкування, статичну типізацію. Об'єктно-орієнтований підхід вирішує завдання з побудови великих, але водночас гнучких і розширюваних додатків.

На відміну від багатьох інших мов, включаючи C і C ++, мова Java, коли була створена, компілювалась в незалежній від платформи байт-коді. Цей код розповсюджується всюди по інтернету і інтерпретується в Java Virtual Machine (JVM), на якій він в працює.

Також цю мову визнано вважати архітектурно-нейтральною. Не залежить від реалізації частин специфікацій – все це робить Java портативною.

Технологія Java Servlets (сервлети) винайдена компанією Sun Microsystems, задля використання переваг Java платформи для змоги розв’язання питань розробки веб-додатків. Проблема продуктивності вирішується завдяки багатопотоковості, виконуючи в одному процесі всі запити як потоки. Сервлети не залежать від платформи бо виконуються всередині Java Virtual Machine (JVM), і можуть легко розпаралелити ресурси [31].

Модель безпеки Java робить забезпечує управління рівнем доступу, а обробка виключень робить сервлети надійнішим засобом. Ця технологія володіє широкими функціональними можливостями, і велика кількість бібліотек надає найрізноманітніші засоби, необхідні в розробці.

Сервлет є класом Java повинен бути виконаний всередині Java VM. «Контейнер» WEB додатки типу Tomcat завантажує клас сервлета при першому зверненні до нього, або відразу при запуску сервера згідно настройки конфігурації. Далі сервлет залишається завантаженим для обробки запитів, поки він не вивантажується явно, або до зупинки контейнера.

Технологія сервлетів є поширеною і може бути використана з усіма популярними WEB серверами, які виконують функції контейнера сервлетів (Apache Tomcat, Java Web Server від Sun).

Програмний інтерфейс дозволяє сервлетам обробляти запити на низькому рівні (заголовки запитів, їх тип, і т.д). Це забезпечує більшу гнучкість при розробці нестандартних оброблювачів, наприклад при роботі з двійковим або мультимедійним вмістом.

3.3.3 ASP.NET

Технологія .NET є останньою розробкою компанії Microsoft і заявлена як новітній етап в розвитку засобів між додатками взаємодії. На даний час вона доступна як доповнення .NET Framework до сімейства операційних систем (ОС) Microsoft Windows. В цей час продовжуються роботи по створенню та вдосконаленню технології .NET Framework на інших операційних системах.

Основою .NET є Common Language Runtime (CLR загальне середовище виконання мов), яке базується на системних службах ОС і керує виконанням коду, який написан на будь-якій мові програмування. Набір базових класів дає доступ до сервісів платформи, які використовуються при програмуванні. CLR і базові класи є складовою основи .NET платформи [32].

NET поставляє такі високорівневі сервіси для розробки WEB додатків як: ASP .NET – нова версія ASP, яка дає можливість використовувати будь-яку (.NET сумісну) мову для програмування Web сторінок; Windows Forms і Web Forms – набір класів для побудови інтерфейсу користувача, що призначено для локальних і WEB-орієнтованих додатків.

У технологію ASP.NET закладено все, для забезпечення всього циклу розробки WEB додатків швидшим, а підтримку простішою. Разом зі створенням технології ASP.NET з'явилися нові можливості у галузі розробки WEB систем, що задовольняють всім сучасним вимогам та дозволяють у значній мірі пришвидшити

і спростити розробку великих програм. Однак ASP.NET сильно прив'язана сервера IIS, і, хоча архітектура .NET дозволяє перенести додатки ASP.NET на іншу платформу, на даний момент реальна можливість відсутня.

Таким чином, багатоплатформеність поки ще не може бути задоволена платформою .NET. Однак необхідно відзначити, що така система повинна мати можливості інтеграції з платформою .NET (особливо WEB сервіси).

3.4 Обґрунтування вибору технологій програмування клієнтської частини

У якості мови реалізації була обрана Java, що є кросплатформною мовою програмування, відповідною для розробки вбудованого в систему модуля для обробки тексту. До того ж обране рішення для вирішення завдання не вимагає безпосередньої роботи з пам'яттю.

Як реалізації префіксного дерева був використаний TrieMap (http://www.superliminal.com/sources/TrieMap.java.html), який є вільним програмним забезпеченням і модифікований для зберігання в вузлах множин моделей.

4 РОЗРОБКА ПРОЕКТНИХ РІШЕНЬ ПО РЕАЛІЗАЦІЇ СИСТЕМИ

4.1 Основні функції системи

Результатом розробки є прототип веб-додатку, який дозволяє виділяти нормальну форму слова із застосуванням методу MyStem з урахуванням можливих стоп-слів та висуванням гіпотез для неіснуючих слів. Веб-система реалізує такі функцій:

– введення тексту для аналізу;

– отримання слів після нормалізації;

– отримання списку стоп-слів;

– отримання загальної кількості слів;

– отримувати таблиці з потенційними гіпотезами.

4.2 Опис архітектури системи

Основна схема роботи алгоритму, що була реалізована мовою Java є наступною :

– зчитування рядка з вхідного тексту, обробка кожного слова в рядку;

– якщо слово є службовим (належить списку «стоп-слів»), воно записується в результат та відбувається перехід до наступного слова;

– для неслужбового слова будується безліч можливих лем;

– за ступенем близькості до оброблюваного слова і пріоритетності частин мови з безлічі можливих лем вибирається єдина;

– слово разом з лемою записується в результуючий рядок;

– рядок записується у вигляді результуючої таблиці, якщо опрацьований не весь текст з введеного користувачем, відбувається перехід до першого пункту.

Для реалізації системи було написано слідуючи класи для реалізації основних функцій та процесів. Далі приведено стислий опис архітектури системи та застосованих класів.

Клас *Normalizer* є основним класом, в якому відбувається попередня обробка тексту і запуск побудови моделі або обробки слів вхідного тексту.

Внутрішній клас *TrieMap* реалізує структуру префіксного дерева, з можливістю вставки і видалення елементів, отримання значень, що зберігаються в вузлах і визначення «найближчих» до заданого ключів.

Функція *ModelBuilder()* будує модель за словником.

Функція *InnerImage(model_filename)* перетворює модель з введеного тексту у внутрішнє уявлення.

Функція *Normalization()* виробляє нормалізацію тексту , що вводить користувач та повертає опрацьований результат.

4.2.1 TrieMap

Для реалізації структури побудованого префіксного дерева інвертованих основ і аналогічного дерева суфіксів та змоги проводити пошук по цим деревам було вибрано технологію TrieMap [33].

TrieMap (або як його ще називають – префіксне дерево) – це впорядкована деревоподібна структура даних, яка використовується для зберігання динамічного набору або асоціативного масиву, де ключі зазвичай є рядками.

Являє собою кореневе дерево, де кожне ребро позначається деяким символом так, що для будь-якого вузла всі ребра, що з'єднують цей вузол з його синами, позначені різними символами. Деякі вузли префіксного дерева виділені і вважається, що префіксне дерево містить цей рядок-ключ тоді і тільки тоді, коли цей рядок можна прочитати на шляху з кореня до деякого (єдиного для цього рядка) виділеного вузла.

Навантажене дерево відрізняється від звичайних n-арних дерев тим, що в його вузлах не зберігається ключі. Замість них в вузлах зберігаються односимвольні мітки, а ключем, який відповідає якомусь вузлу є шлях від кореня дерева до цього вузла, а точніше рядок складена з міток вузлів, що зустрілися на цьому шляху. У такому випадку корінь дерева, очевидно, відповідає порожньому ключу.

У деяких додатках зручно вважати всі вузли дерева виділеними. На відміну від двійкового дерева пошуку, жоден вузол в дереві не зберігає ключ, пов'язаний з цим вузлом; замість цього його позиція в дереві визначає ключ, з яким він пов'язаний; тобто значення ключа розподілені по структурі. Всі нащадки вузла мають загальний префікс рядка, пов'язаного з цим вузлом, а корінь пов'язаний з нового рядка. Ключі зазвичай пов'язані з листям, хоча деякі внутрішні вузли можуть відповідати цільовим ключам. Отже, ключі не обов'язково пов'язані з кожним вузлом [34].

Так як навантажене дерево реалізує інтерфейс асоціативного масиву, в ньому можна виділити три основні операції, а саме вставку, видалення і пошук ключа. Як і багато дерев, навантажене дерево має властивість подібності самого до себе, тобто будь-яке його піддерево також є повноцінним навантаженим деревом. Легко помітити, що всі ключі в такому піддереві мають загальний префікс, (звідки і пішла назва «префіксне дерево») а значить можна виділити специфічну для цього дерева операцію – отримання всіх ключів дерева з заданим префіксом за час $O(|Prefix|)$.

Як вже було сказано, ключ, відповідний вузлу – конкатенація міток вузлів, що містяться на шляху від кореня до даного вузла. З цієї властивості логічно розглядати алгоритм пошуку ключа (як, втім, і алгоритми додавання і видалення).

Нехай дано ключ *Key*, який необхідно знайти в дереві. Будемо спускатися з кореня дерева на нижні рівні, кожен раз переходячи в вузол, чия мітка збігається з черговим символом ключа. Після того як оброблені всі символи ключа, вузол, в якому зупинився спуск і буде шуканим вузлом. Якщо в процесі спуску не знайшлося вузла з міткою, що відповідає черговому символу ключа, або спуск

зупинився на проміжній вершині (вершині, яка не має значення), то шуканий ключ відсутній в дереві.

Часова складність цього алгоритму, дорівнює *O(| Key |)*.

Алгоритм додавання ключа в дерево дуже схожий на алгоритм пошуку.

Нехай дана пара з ключа *Key* і значення *Value*, яку потрібно додати. Як і в алгоритмі пошуку ключа, будемо спускатися з кореня дерева на нижні рівні, кожен раз переходячи в вузол, чия мітка збігається з черговим символом ключа. Після того як оброблені всі символи ключа, вузол, в якому зупинився спуск і буде вузлом, яким має бути присвоєно значення *Value* (також, зрозуміло, вузол повинен бути позначений як має значення). Якщо в процесі спуску відсутній вузол з міткою, що відповідає черговому символу ключа, то слід створити новий проміжний вузол з потрібною міткою і призначити його нащадком поточного.

Часова складність додавання ключа – *O(| Key |)* [35].

Наведемо приклад програмної реалізації структури TrieMap. Для початку було створено класс *TrieMapv()* та оголошені його змінні:

Лістинг 4.1 – Фрагмент програмного коду реалізації префіксного дерева

```
  private static class TrieMapv {
    public String mStr;
    public Object mVal;
    public Leafv(String str, Object val) {
      mStr = str;
      mVal = val;
    }

public void insert (String word) {
TrieMapv current = root;
    for (char l: word.toCharArray()) {
```

```
      current = current.getChildren().computeIfAbsent(l, c ->
new TrieNode());
      }
   current.setEndOfWord(true); }
```

4.3 Розробка інтерфейсу користувача системи

Розроблений веб-додаток має бути доступним з будь-яких пристроїв, на яких встановлено такі браузери як Internet Explorer, Opera, Mozilla Firefox і т.п. та запроваджений доступ до Інтернету.

Вірно спроектований інтерфейс може бути розглянутий як набір методів та рішень, які дозволяють користувачам досягнути своїх цілей за мінімальних витрат часу та сил.

Основною вимогою до веб-додатків є те, що вони мають бути інтуїтивно зрозумілими для користувачів. Якщо користувач, відкриваючи веб-сторінку не зрозуміє, які наступні дії від нього очікуються аби отримати необхідний йому результат, то є велика ймовірність, що довго на цьому сайті людина не затримається і покине ресурс.

У веб-системі, що розробляється інтерфейс є інтуїтивно зрозумілим:

– навігація на головну сторінку сайту, вкладка інформаційної сторінки реєстрація та авторизація знаходиться у верхній частині сайту;

– нижче знаходиться текстове поле для вводу тексту, яка є зрозумілою за рахунок вказівок, які повідомляють користувача про те, що повинно бути введено у це поле;

– всі поля введення підписані і виводять повідомлення в разі неправильного введення даних;

– при натисканні кнопки «Analys», на веб-сторінці з’являється результат аналізу.

Головна сторінка сайту зображена на рисунку 4.1.

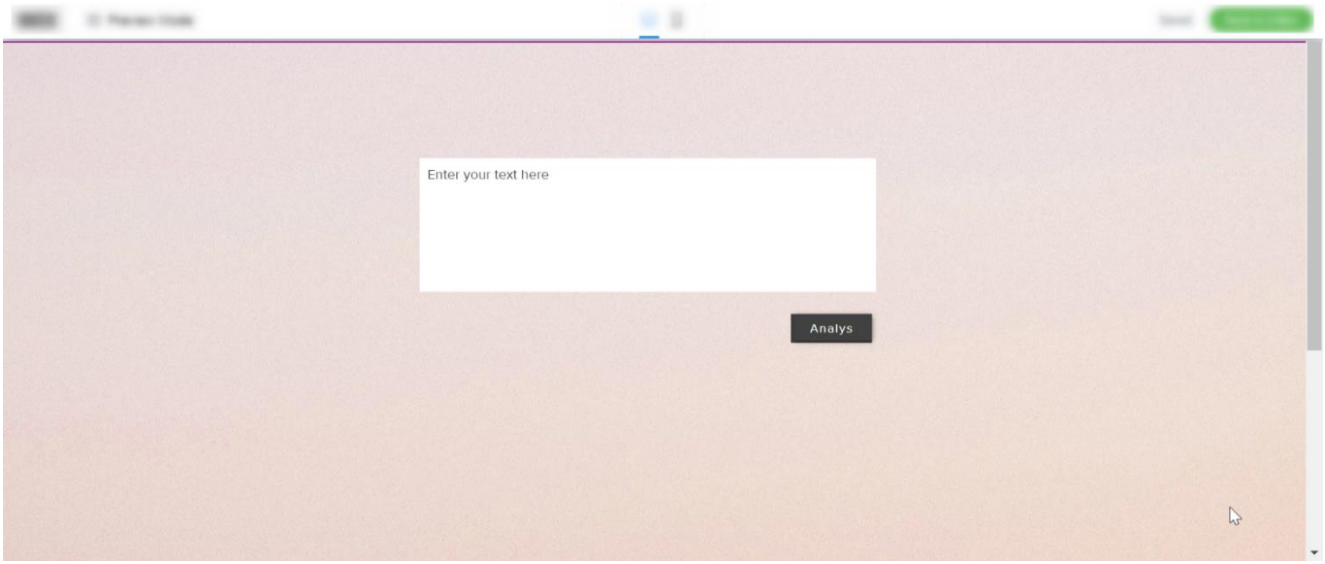

Рисунок 4.1 – Головна сторінка

Усі функціональні елементи на сторінці, текстові поля та кнопки мають інтуїтивно зрозумілі назви. На текстових полях присутній хінт-текст, що виступає в ролі інструкції користувачу.

Інтерфейс розробленого прототипу є мінімалістичним, так як занадто захаращений інтерфейс є великою перешкодою для його розуміння користувачем. Зайві елементи загружають сторінку і заважають користувачеві ефективно взаємодіяти з додатком. У додатку, що розробляється мінімалістичний інтерфейс і пригнічена колірна гамма – світлий фон, темні шрифти.

Також веб-сторінка повинна швидко завантажуватись та так само швидко видавати результати пошуку користувачеві. Так як повільне завантаження інтерфейсу може дратувати і відштовхувати користувача. Швидкодія була досягнута шляхом уникнення застосування «важких» елементів на сайті. Та оптимізації звернення до словникової бази на стороні серверу. Так як додаток розгортається на віддаленому сервері (що дозволяє значно зменшити навантаження на пристрій користувача), доступ до системи здійснюється через відповідну доменну адресу.

Роботу методу нормалізації продемонстровано на рисунку 4.2 на прикладі фрагменту тексту, що вводиться користувачем.

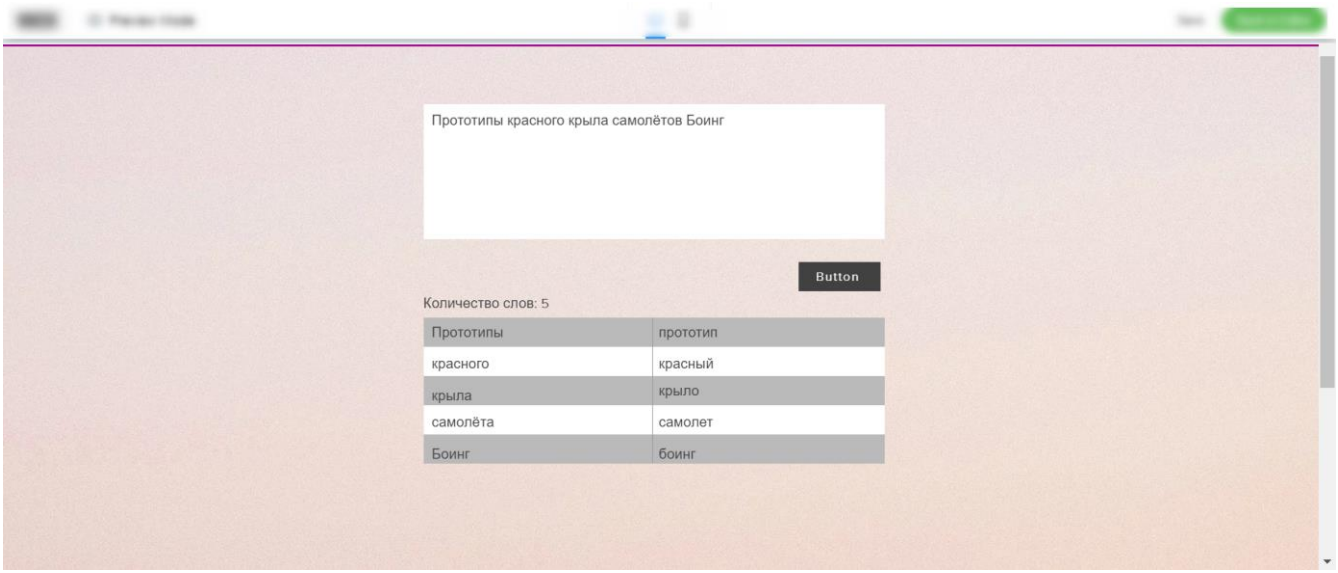

Рисунок 4.2 – Приклад обробки фрагменту тексту

Поки користувачем не буде введено текст для аналізу у текстове поле, кнопка «Analys» буде неактивна. Як тільки текст було введено, кнопка стає активною.

Як бачимо з рисунку 4.2 введений текст, що складається з 5 слів було проаналізовано й отримано наступні результати: вірно отримані нормальні форми наступних слів: «*прототипы*» → «*прототип*», «*красного*» → «*красный*» і тому подібне. Зауважимо, що слово «*боинг*» не є словниковим, проте в даному випадку після опрацювання алгоритмом було правильно зазначено стемму, яка в даному випадку і являється нормальною формою слова.

Приклад приведення гіпотез о нормальній формі для неіснуючих слів приведено на рисунках 4.3 та 4.4.

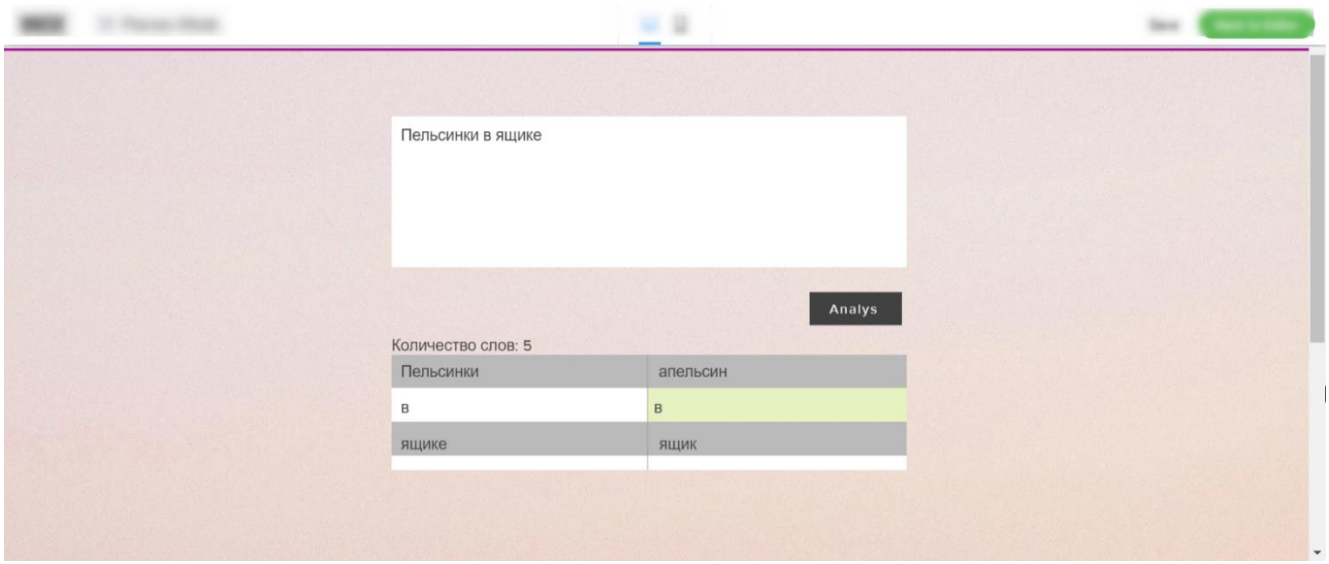

Рисунок 4.3 – Приклад обробки фрагменту тексту

Стоп-слова система виділяє в таблиці результатів жовтим кольором. Слово «*пельсинки*» було зведено до найбільш ймовірної гіпотези – «*апельсин*». Цей результат цілком задовольняє, адже саме така стемма й очікувалась.

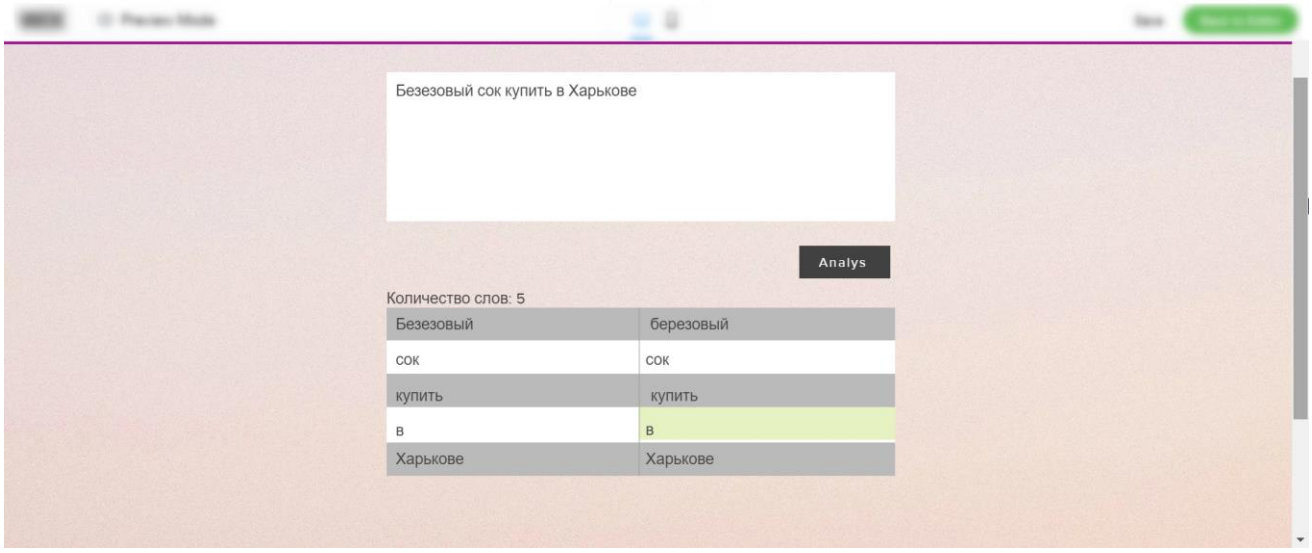

Рисунок 4.4 – Приклад обробки фрагменту тексту

В даному тесті для обробки було запропоновано неіснуюче слово «*безезовый*», та насправді очікувалось слово «*березовий*». Алгоритмом було проаналізовано потенційні гіпотези, та знайдено найближче слово зі словникової

бази за морфологічною близькістю. Проте слід зазначити, що стеммер в даному випадку не впорався зі словом «Харькове», не дивлячись на те що його не має в словниковій базі, в даному випадку стема залишилась така сама як і початкова форма слова. Це доводить, що не можна сказати, що стеммер 100% вірно знаходить нормальні форми для всіх слів.

4.4. Характеристики функціонування системи

Зважаючи на особливості обробки природних мов, питання точності і повноти даного алгоритму не можна розглядати однозначно. І якщо для словникових слів, за винятком випадків граматичної омонімії (наприклад, «*пила*» (дієслово) і «*пила*» (іменник)), завжди можна визначити, чи правильний результат вийшов на виході, то для неіснуючих слів це часто неможливо навіть при аналізі контексту.

Для прикладу можна розглянути фразу, пропоновану для ілюстрації роботи Mystem з неіснуючими словами: «*В мурелкі тьопають пельсіскі*». Оскільки ми не знаємо точного значення слова «*пельсіскі*», будь-який з варіантів початкових форм «*пельсісок*», «*пельсіска*», «*пельсіск*» і «*пельсіскі*» може вважатися вірним. Таким чином, не можна однозначно оцінити точність і повноту даного алгоритму для неіснуючих слів, але можна порівняти його з іншими стеммерами. Завдяки роботі зі словником, та реалізації гіпотез даний нормалізатор має розширений спектр функцій та фундаменту для подальшого навчання. Так як ні один з інших стеммерів, що було розглянуто не пропонував потенційно вірні гіпотези для слів, що написані з помилкою, які автоматично приписуються до ряду неіснуючих. Реалізований нормалізатор показав чудові результати передбачення вірної стемми. Також було вирішено проблему синтезує невірних форм для словникових слів, яка була у стеммера Портера (*«кровать»* → *«крова+ть»* → початкова форма: *«кроватить»*) для нього не існує.

Оскільки алгоритм стеммінгу, використаний при побудові даного нормалізатора, базувався на алгоритмі Mystem, результат роботи, за рідкісним винятком (близько 4% випадків), міститься в множині, що пропонується в якості результату Mystem.

Як правило, до невідповідних випадків відносяться словоформи, утворені з повною зміною основи: *«человек»* → *«людьми»*. Надалі, цю проблему можна вирішити, якщо розраховувати для кожної моделі не одну стемму, як було зроблено в даній роботі, а як набір можливих стемм: це кілька підвищить складність алгоритму, але збільшить його ефективність.

ВИСНОВКИ

Перед виконанням теоретичної та експериментальної частини роботи була докладно проаналізована тематична література. Опрацьовано теоретичний матеріал, в якому розглянуто актуальність проблеми нормалізації слів природньої мови, види методів для знаходження нормальної форми слова, а саме метод стеммінгу. Було зроблено огляд найактуальніших на даний час стеммерів, їх можливостей. Проведено порівняння існуючих методів та визначені проблеми, що є присутні в цих методах.

На базі існуючих рішень був побудований підхід для вирішення проблеми вибору нормальної форми слова в тих випадках коли не має однозначно виділеної стеми. Цю проблему вдалось дещо мінімізувати шляхом впровадження ранжування всіх потенційних гіпотез слова. Таким чином, ймовірність виділення коректної стемми порівняно зросла Також була врахована та реалізована можливість роботи зі словами, що не входять до словникової бази, в даному дослідженні на корпусі російської мови, задля того, щоб можливість виділення гіпотетичних стемм.

У ході виконання атестаційної роботи було розроблено компонент веб-системи для демонстрації працездатності розглянутого методу. Розроблений метод є більш оптимальним рішенням для нормалізації слів адже він покриває обробку несловникових слів та висування гіпотез щодо потенційної нормальної форми слова в суперечливих випадках.

Розроблена система впроваджена мовою Java, яка дає змогу розгорнути сервер додатку на будь-якій платформі, що є її додатковою перевагою.

Подальшим напрямком розвитку розглянутого методу нормалізації слів може бути розширення функціоналу на опрацювання різних мов, зробити систему мультимовною. Адже для цього слід враховувати не тільки лексичні особливості мови, а й її формат написання та можливість машинної обробки. Та вирішити

проблему омонімії, що є досі не 100% вирішеною для слів на корпусі російської мови.

Розроблені компоненти системи готові до впровадження до системи семантичної обробки текстів, як реалізація етапу попередньої обробки слова та зведення його до нормальної форми.

Результати даного дослідження апробовано на *II* Міжнародній науково-практичній конференції «Priority directions of science and technology development» (Київ, Україна) [36].

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Загальні відомості про списки лідируючих запитів в Google Trends. URL: https://trends.google.com.ua/ (дата звернення: 02.10.2020).

2. Imangulova Z., Kolesnyk L. An algorithm for building a project team considering interpersonal relations of employees / Eastern-European Journal of Enterprise Technologies. 2016. Vol. 6. P. 19–25.

3. D. Jurafsky, James H. Martin. Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition / Prentice-Hall, 2000. P. 179–185.

4. DeepDive Tutorial. Extracting mentions of spouses from the news – URL: http://deepdive.stanford.edu/example-spouse (дата звернення: 10.10.2020).

5. Landauer T., Foltz P. W., Laham D. Introduction to Latent Semantic Analysis / Discourse Processes 25 , 1998. P. 259-284.

6. John S. Ball Using NLU in Context for Question Answering: Improving on Facebook's bAbI, 2017. URL: https://arxiv.org/ftp/arxiv/papers/1709/1709.04558.pdf (дата звернення: 12.10.2020).

7. Neural Machine Translation (seq2seq) Tutorial. URL:https://www.tensorflow.org/tutorials/seq2seq (дата звернення: 12.10.2020).

8. Word2vec. URL: https://medium.com/@zafaralibagh6/simple-tutorial-on-word-embedding-and-word2vec-43d477624b6d (дата звернення: 12.10.2020).

9. Філипчик А.В. Стемминг слов в лінгвістичній інформаційно-пошуковій системі. 2016 р. С. 65-66.

10. Шабуров А.С., Журілова Е.Е., Лужний В.С. Технічні аспекти впровадження DLP-системи на основі Falcongaze Secure Tower / Вісник Пермського національного дослідницького політехнічного університету. 2015. С. 57-67.

11. Пруцков А.В., Розанов А.К. Методи морфологічної обробки текстів / Прикаспійський журнал: управління та високі технології. 2014. URL: http://prutzkow.com/pdf/114.pdf (дата звернення: 22.10.2020).

12. M. F. Porter. An algorithm for suffix stripping. 1980. P. 130-137.

13. P. Willett. The Porter stemming algorithm: then and now / Program: Electronic Library and Information Systems. 2006. P. 219-223.

14. Коваленко А. Імовірнісний морфологічний аналізатор російської та української мов. URL: http://www.keva.ru/stemka/stemka.html (дата звернення: 03.11.2020).

15. Астапова О.П. Дослідження і розробка методів нормалізації слів російської мови. / Москва, МГУ ім. Ломоносова, 2012. С. 12-18.

16. Голомазов Д.Д. Виділення термінів з колекції текстів із заданим тематичним поділом. / Москва, МГУ ім. М.В. Ломоносова, 2008. С. 121-122.

17. Марчук Ю.Н. Основи комп’ютерної лінгвістики. 2000. С. 17-23.

18. I. Kuznetsov, E. Kozerenko. The system for extracting semantic information from natural language texts / Proceeding of International Conference on Machine Learning. Las Vegas US, 23 – 26 June 2003. P. 75-80.

19. I. Segalovich, M. Maslov. Russian Morphological Analysis and Synthesis With Automatic Generation of Inflection Models For Unknown Words. / Dialog'98 (in Russian). URL: http://company.yandex.ru/articles/article1.html (дата звернення: 03.11.2020).

20. Ilya Segalovich. A fast morphological algorithm with unknown word guessing induced by a dictionary for a web search engine». 2003. P. 88-94

21. Залізняк А.А. Граматичний словник російської мови (словозміна) URL: http://speakrus.ru/dict/all_forms.rar (дата звернення: 21.11.2020).

22. Сегаловіч І. Швидкий морфологічний алгоритм підбору невідомого для пошукової системи слова за допомогою словника. 2014. URL: http://wseob.ru/seo/morphological-algorithm (дата звернення: 06.11.2020).

23. Гращенко Л. А. Про модельний стоп-словник 2013. С. 40-46.

24. Berners-Lee, Tim; Connolly, Daniel. Hypertext Markup Language (HTML): A Representation of Textual Information and MetaInformation for Retrieval and Interchange. 2017. P. 215-219.

25. CSS developer guide. Mozilla Developer Network. Archived from the original on. 2015.

26. Ситніков Д.Е., Марьин С.А., Коваленко А.И. Узагальнення понять за ознаками: методи теорії приблизних множин / Молодий вчений. 2016. С. 35-38.

27. Сухотін Б.В. Виділення морфем в текстах без пропусків між словами. М., 1999. С. 56-58.

28. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin Attention Is All You Need – URL: https://arxiv.org/pdf/1706.03762.pdf (дата звернення: 10.11.2020).

29. Zhang, X., Junbo Zhao, Yann LeCun. Character-level convolutional networks for text classification / In Advances in Neural Information Processing Systems. 2015. P. 649-657.

30. Creating a module for Sentiment Analysis with NLTK. URL: https://pythonprogramming.net/sentiment-analysis-module-nltk-tutorial/ (дата звернення: 11.11.2020).

31. Sitnikov D. Assessment of extended aggregated association rules / Sitnikov D., Ryabov O., Titova O., Kovalenko A. / Proceedings of 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies, DESSERT’2018, 24 27 May, 2018, Kyiv, Ukraine. P 95-99.

32. ASP.NET is part of a great open source .NET community. Microsoft. Microsoft. May 11, 2020. URL: https://dotnet.microsoft.com/platform/open-source (дата звернення: 11.11.2020).

33. Bentley, Jon; Sedgewick, Robert. Ternary Search Trees. / Dr. Dobb's Journal. Dr Dobb's. 2008. P 06-23.

34. Kageura K., Umino B. Methods of automatic term recognition: A review. Terminology. 1996. Vol. 3, no. 2. P. 259-289.

35. Wattenberg, Martin. A Note on Space-Filling Visualizations and Space-Filling Curves". In Stasko, John T.; Ward, Matthew O. (eds.). / IEEE Symposium on Information Visualization (InfoVis 2005), 23-25 October 2005, Minneapolis, MN, USA (PDF). IEEE Computer Society. P.24.

36. Волобуєва В.В., Колесник Л.В. Тези доповіді за темою «Дослідження методів приведення слів до нормальної форми для проведення семантичного аналізу тексту». / II Міжнародній науково-практичній конференції «Priority directions of science and technology development» (Київ, Україна), 2020. С. 237-241.