Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження доцільності використання аугментації тексту в системі машинного перекладу

(тема)

Виконав:

студент II курсу, групи СПм-20-2

Гаврашенко А.О.
(прізвище, ініціали)

Спеціальність

123 «Комп'ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: проф. Фесенко Т.Г
(посада, прізвище, ініціали)

Допускається до захисту

В.о. зав. кафедри ЕОМ _____ Волк М.О.
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп'ютерна інженерія»
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
В.о. зав.
кафедри _____
(підпис)
"____" _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Гаврашенку Антону Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження доцільності використання аугментації тексту в системі машинного перекладу

затверджена наказом по університету від "24" березня 2022 р. № 413Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2022р.

3. Вхідні дані до роботи
Словники, тестові тексти для перекладу на різних мовах різних стилів,

4. Перелік питань, що потрібно опрацювати у роботі
Аналіз предметної області
Аналіз алгоритмів
Реалізація поставлених задач
Тестування та аналіз отриманих результатів.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 17 слайдів

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд та аналіз існуючих рішень	1.04.2022	
2	Створення моделі системи перекладу на		
	основі обернених словників	15.04.2022	
3	Реалізація використання підготовленого		
	словнику для перекладу	30.04.2022	
4	Формування та розширення словників	4.05.2022	
5	Аналіз отриманих результатів	8.05.2022	
6	Оформлення пояснювальної записки. Подання	14.05.2022	
	на перевірку керівникові		

Дата видачі завдання 28 березня 2022 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Фесенко Т.Г.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 95 с., 54 рис., 16 табл., 1 дод., 27 джерел.

ПЕРЕКЛАДАЧ, МОВА ПРОГРАМУВАННЯ, ІНТЕРНЕТ, СЛОВНИК, ОБРОБКА ТЕКСТУ, АУГМЕНТАЦІЯ.

Об’єктом даного дослідження є автоматичний словник, який може бути використаний для перекладу та тлумачення сленгової або штучної лексики, як такої, що створюється на перетині декількох мов і має багато спільних (запозичених) за написанням і значенням слів із різних мов.

Головною метою проекту є створення системи автоматичного перекладу слів та текстів із/на довільні мови, включаючи гібридні, штучні та сленгові мови за допомогою уже відомих словників інших мов.

Для досягнення поставленої мети мають бути вирішені наступні задачі:

- реалізація можливості використання підготовленого словнику для перекладу;
- розширення словнику;
- створення нових словників на основі вже існуючих;
- розробка підходу та створення обернених словників;
- розробка підходу генерування нових словників за допомогою третіх мов.

ABSTRACT

Master’s thesis: 95 pages, 54 figures, 16 tables, 1 appendices, 27 sources

TRANSLATOR, PROGRAMMING LANGUAGE, INTERNET, DICTIONARY, TEXT PROCESSING, AUGMENTATION.

The object of this study is an automatic dictionary that can be used to translate and interpret slang or artificial vocabulary, as it is created at the intersection of several languages and has many common (borrowed) spellings and meanings of words from different languages.

The main goal of the project is to create a system of automatic translation of words and texts from / into arbitrary languages, including hybrid, artificial and slang languages using already known dictionaries of other languages.

To achieve this goal, the following tasks must be solved:

- realization of the possibility of using the prepared dictionary for translation;
- dictionary expansion;
- creation of new dictionaries on the basis of existing ones;
- developing an approach and creating inverted dictionaries;
- development of an approach to generating new dictionaries using third languages.

ЗМІСТ

ВСТУП

У сучасному світі існує велика кількість різних мов, та сленгів. Великі онлайн перекладачі дозволяють перекладати лише основні мови, чого не достатньо в сучасному світі. Так як знайти словник для перекладів сленгів досить важко, було запропоновано генерувати їх за допомогою аугментації тексту. Аугментація (augmentation) – це побудова додаткових даних із вихідних під час вирішення завдань машинного навчання. Зазвичай при аугментації застосовують перетворення вихідних об'єктів, які змінюють їх мітки, але змінюють (іноді істотно) описи.

Сленгами ми можемо вважати любу мову, для якої існує інша мова, з якою перетин однакових слів як за написанням так і значенням досить великий. Наприклад одна і та ж сама мова в різних регіонах країни вважається сленгом. У такому випадку існує основна - літературна мова. Тоді взаємозв’язки цих мов можно представити у вигляді дерева, з коренем в основній мові. Так як не обов’язково що пара сленгів однієї мови між собою також будуть сленгами, то чим далі знаходиться пара мов у дереві, тим менше ймовірність того, що ця пара між собою є сленгами.

У наш час для перекладу сленгу або суржику потрібно витратити дуже багато часу, так як знайти спеціалізований перекладач досить тяжко. Всі популярні перекладачі можуть перекладати лише на певну кількість мов, та досить тяжко дотати нові слова для перекладу. Але кількість сленгів постійно зростає. Раніше нові сленги породжувалися в ході впливу на них сусідніх народів та залежали від певної території, або серед людей певної професії. Та вони були досить схожими на оригінал, з додаванням або зміни зазвичай невеликої кількості слів.

Кожна група створена за інтересами під вливом певної книги, фільму, гри і т.д, може спілкуватися за допомогою свого сленг, який стороння людина може взагалі не може зрозуміти. Це може відлякувати людей які

хочуть приєднатися до цієї групи. Тому, на наш погляд, розробка програми для перекладу сленгів є актуальною в сучасному світі.[1]

Штучні мови – спеціалізовані мови, в яких лексика, фонетика і граматика були спеціально розроблені для втілення певних цілей. Саме цілеспрямованість відрізняє штучні мови від природних. Іноді дані мови називають несправжніми мовами. Таких мов існує вже більше тисячі, і постійно створюються нові.[2]

Причинами для створення штучної мови є: полегшення людського спілкування (міжнародні допоміжні мови, коди), надання художній літературі додаткового реалізму, лінгвістичні експерименти, забезпечення комунікації в вигаданому світі, мовні ігри та отримання задоволення.

Вираз «штучна мова» іноді використовується для позначення планових мов та інших мов, розроблених для спілкування людей. Іноді вважають за краще називати такі мови саме «плановими», так як слово «штучний» може мати зневажливий відтінок в деяких мовах.

Більшість штучних мов створюється однією людиною, наприклад - талосскій. Але є мови, які створені групою людей, так мова Інтерлінгва, розроблена Міжнародною асоціацією допоміжної мови та ложбан, створений Групою Логічних Мов[].

Спільні розробки штучних мов стали поширені в останні роки, оскільки проектувальники штучної мови почали використовувати інтернет-інструменти для координації конструкторських розробок.

Перекладом ми можемо назвати відтворення оригіналу засобами іншої мови із збереженням єдності змісту і форми. Ця єдність досягається цілісним відтворенням ідейного змісту оригіналу в характерній для нього стилістичній своєрідності на іншій мовній основі. Шлях до досягнення такої єдності не лежить через встановлення формальних відповідників. Зіставлення засобів різних мов, навіть найбільш віддалених, можливе лише шляхом зіставлення функцій, які виконують різні мовні засоби. Звідси точність перекладу полягає у функціональній, а не формальній відповідності оригіналу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Лексикографія є дуже старим видом лінгвістичної діяльності, що доводять перші словники, які з’явилися вже наприкінці XII ст. для виконання інформативної (дозволяють найкоротшим способом – через позначення – долучитися до накопичених знань) та нормативної функцій, а також для відображення знань, які має суспільство у певну епоху. Подальший розвиток комп’ютерних та обчислювальних технологій призвів до появи нової області лексикографії – комп’ютерної або електронної лексикографії, яка забезпечує можливість користування лексикографічним ресурсом з комп’ютера або інших електронних пристроїв. Головним об’єктом дослідження комп’ютерної лексикографії є електронний словник. Аналогічними термінами є «машинні словники», «автоматичні словники», «автоматизовані словники», «комп’ютерні словники» [1]. Підґрунтям для створення електронних словників є методи із області лінгвістичного аналізу, обробки текстів, лексикографічних баз даних, машинного перекладу (статистичний машинний переклад, нейронний машинний переклад, машинний переклад на основі правил, гібридний машинний переклад) тощо. Розповсюдження та масовий доступ користувачів до Інтернет поклав початок розвитку такого напрямку лексикографії, як кібернетична лексикографія (кіберлексикографія), яка передбачає використання Інтернет для складання словнику.

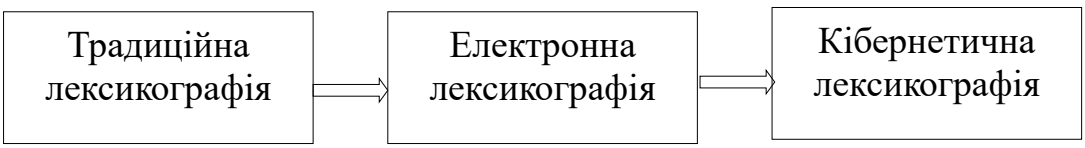

Рисунок 1.1 - Еволюція лінгвістичної діяльності людини

У багатьох джерелах, що присвячені педагогічним або філологічним наукам наводяться детальні класифікації словників. Прикладом є [11].

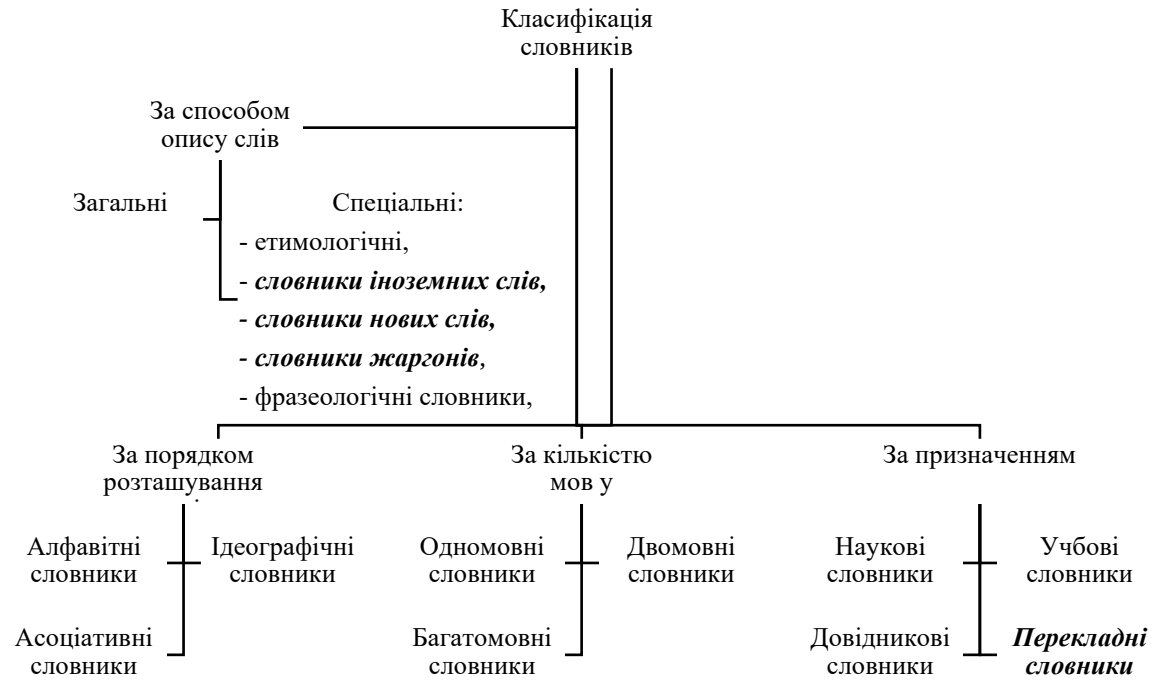

Рисунок 1.2 - Систематизація словників

Використання того чи іншого словнику залежить від поставленої задачі, як то переклад тексту чи тлумачення терміну. Так, перекладні словники створені для зіставлення і переходу з однієї мовної системи в іншу (українсько-білоруський, англійсько-французький і ін.), у той час як спеціальні словники, наприклад, регіональних та приватних діалектів розкривають лексичні значення у рамках одного діалекту або говору (словник жаргонів та сленгів).

1.2. Аналіз аналогів

В наш час для перекладу сленгу або суржику потрібно потратити досить багато часу, так як знайти спеціалізований перекладач досить важко. Всі популярні перекладачі можуть перекладати лише на певну кількість мов (табл.1.1), та досить складно масштабувати такі словники, додаючи нові слова для перекладу.

Таблиця 1.1 – Порівняння з аналогами

Критерій для порівння	Google translate	Яндекс. Перекладач	www.dictionary.com	ABBYY Lingvo
Можливість налаштувати роботу під себе	Користувач отримає декілька результатів серед яких може вибрати кращий для себе.	Користувач отримає декілька результатів серед яких може вибрати кращий для себе.	Працює за принципом довідника. Для кожного запиту буде завчасно відома відповідь.	Працює за принципом довідника. Для кожного запиту буде завчасно відома відповідь
Кількість підтримуємих мов.	104	95	1	20(220 словників)
Доступність	Безкоштовна. Результат можливо отримати за дуже короткий термін. Інтерфейс інтуітивний.	Безкоштовна. Результат можливо отримати за дуже короткий термін. Інтерфейс інтуітивний.	Безкоштовна. Результат можливо отримати за дуже короткий термін. Інтерфейс інтуітивни.	Платна. Для отримання результату необхідно знайти потрібний словник, та зробити пошук по ньому, що потребує час
Платформи на яких працює перекладач	Windows, Linux, для IOS та Android існують окремі додатки.	Windows, Linux, для IOS та Android існують окремі доатки. Використовується для перекладів web-сторінок	Windows, Linux, Android, IOS	Windows, Android, IOS. Для роботи на кожній з платформ необхідно інсталюталювати програму
Можливість працювати онлайн	Лише онлайн.	Лише онлайн.	Лише онлайн.	Можливість працювати в обох режимах

Використовуючи дану таблицю, можемо зрозуміти, що найбільше переваг мають Google translate та Яндекс перекладач, але вони забезпечують можливість перекладу лише з та на офіційні державні мови, не підтримуючи можливості розширення новими штучно створеними словами, та перекладу

сленгової лексики, а також штучних мов. Нові сленги породжуються в ході впливу мов одна на одну та залежать від певної території, то створення окремих словників для кожної з них є не доцільним підходом. Сленгова лексика схожа на вихідну, з додаванням або зміною невеликої кількості слів. Проте, розуміти та вільно спілкуватися із групою людей, що володіють певним сленгом, не завжди зручно. В штучних мовах лексика, фонетика і граматика спеціально розроблені для втілення певних цілей, наприклад з метою спілкування групи людей, що потрапили під вплив певної книги, фільму, гри. Саме цілеспрямованість відрізняє штучні мови від природних

Враховуючи все вищезазначене, створення автоматичного перекладачу сленгів та штучних слів на офіційні мови, є актуальною та затребуваною задачею в сучасному світі.

1.3. Мета роботи

Об’єктом даного дослідження є автоматичний словник, який може бути використаний для перекладу та тлумачення сленгової або штучної лексики, як такої, що створюється на перетині декількох мов і має багато спільних (запозичених) за написанням і значенням слів із різних мов.

Головною метою проекту є створення системи автоматичного перекладу слів та текстів із/на довільні мови, включаючи гібридні, штучні та сленгові мови за допомогою уже відомих словників інших мов.

Для досягнення поставленої мети мають бути вирішені наступні задачі:

- реалізація можливості використання підготовленого словнику для перекладу;
- розширення словнику;
- створення нових словників на основі вже існуючих;
- розробка підходу та створення обернених словників;
- розробка підходу генерування нових словників за допомогю третіх мов.

2. АНАЛІЗ АЛГОРИТМІВ

2.1 Аугментація тексту

Аугментація (augmentation) – це побудова додаткових даних із вихідних під час вирішення завдань машинного навчання. Зазвичай при аугментації застосовують перетворення вихідних об'єктів, які змінюють їх мітки, але змінюють (іноді істотно) описи. Наприклад, якщо ми, тренуючи нейромережу, яка повинна відрізняти фотографії кішок від фотографій собак, будемо крутити, розтягувати, змінювати яскравість і контрастність вихідних зображень, то це не змінить того, що на них зображено, але дасть можливість навчитися мережі на «поганих», деформованих фотографіях, а також на ракурсах, яких можуе бути не достатньо в навчальній вибірці.

Аугментація текстів трохи складніша за аугментацію зображень. По-перше, перетворюючи текст більше шансів спотворити його сенс (або взагалі отримати безглуздий текст). По-друге, тут перетворення "менш автоматичні". Наприклад, щоб повернути фотографію не треба бути фотографом або знати закони оптики, а ось щоб перефразувати якусь пропозицію треба бути принаймні носієм мови (а також знати синоніми, контекст тощо).

2.1.1 Зворотній переклад

За наявності хороших автоматичних перекладачів часто текст перекладають іншою мовою, а потім перекладають «назад» на вихідну. Зрозуміло, що при цьому виходить перефразування вихідної фрази. Такий метод використовувався, наприклад, у роботі Xie та ін. "Unsupervised Data Augmentation", а також переможцем Kaggle-змагання "Toxic Comment Classification Challenge".

Є кілька прийомів, які застосовуються при зворотному перекладі, які

збільшують кількість можливих аугментацій. Перший — переклад можна здійснювати різними мовами. Другий — можна грати з налаштуванням мовної моделі, що формує текст перекладу (генеруючи трохи менш ймовірні, з погляду LM, тексти, які можуть бути вдалими перефразування).

2.1.2 Використання уявлень слів

Випадкові слова замінюємо на близькі до них у просторі уявлень (Word Embeddings). Крім того тут не завжди використовуються синоніми. Часто слова, які вживаються у схожих контекстах або разом із словом, що замінюється. Щоб убезпечити себе від небажаних замін, можна замінювати лише словом тієї самої частини мови. Таку аугментацію застосовували у роботі Wang and Yang[].

2.1.3 Кросовер

Досить оригінальний та простий прийом аугментації запропонований у Franco M. Luque.

Для генерації нових об'єктів класу беремо два його представники: A та B. Кожен із цих текстів ділимо навпіл, отримуємо тексти $A = A1 + A2$, $B = B1 + B2$, де плюс означає конкатенацію. Після цього тексти $A1 + B2$ та $B1 + A2$ додаються до навчання.

Зрозуміло, що описаний метод застосовується тільки в задачах з розміченими даними, а також з досить великими текстами (наприклад, в задачі класифікації фраз діалогу тексти складаються з 1-2 речень, тому застосування кросовера не розумне). В оригінальній статті кросовер не впливав на точність класифікації завдання аналізу сентименту, але збільшував F1-міру.

2.2. Алгоритми пошуку слів в тексі та словнику

2.2.1 Прямий пошук

Прямий пошук – найпростіший алгоритм для пошуку усіх входжень слова в рядку або тексті. Для пошуку в рядку потрібно порівняти кожне зі слів з шуканим словом. Для пошуку підрядку переберемо кожний можливий підрядок з вибраним рядком. Складність даного алгоритму буде: O(N*M) за часом та O(N+M) за пам'яттю. Де N – довжина тексту, M – довжина шуканого рядка, O() - обмеження асимптотичної складності зверху, з точністю до константного множника або нотація Ландау[].

2.2.2 Кнут-Моріс-Пратт

Алгоритм заснований на використанні перфікс-фукції [21].

Префікс-функція визначається наступним чином: це така найбільша довжина найбільшого власного суфікса підрядка s[0..i], що збігається з її префіксом (власний суфікс – значить не збігається з усією рядком). Зокрема, значення $\pi[0]$ вважається рівним нулю.

Математично визначення префікс-функції можна записати в такий спосіб (2.1):

$$\pi[\mathrm{i}] = \max_{k=0..i} \{\mathrm{k} : \mathrm{s}[0...\mathrm{k}-1] = \mathrm{s}[\mathrm{i}-\mathrm{k}+1...\mathrm{i}]\} \qquad (2.1)$$

Алгоритм префікс-функції буде наступним:

Вважати значення префікс-функції $\pi[\mathrm{i}]$ будемо по черзі: від i = 1 до i = n-1 (значення $\pi[0]$ просто дамо рівним нулю).

Для підрахунку поточного значення $\pi[\mathrm{i}]$ ми заводимо змінну j, що позначає довжину поточного розглянутого зразка. Спочатку $j = \pi[\mathrm{i}-1]$.

Тестуємо зразок довжини j, для чого порівнюємо символи s [j] і s [i]. Якщо вони збігаються – то вважаємо π[i] =j+1 і переходимо до наступного індексу i+1. Якщо ж символи відрізняються, то зменшуємо довжину j, вважаючи її рівною π[j−1], і повторюємо цей крок алгоритму з початку.

Якщо ми дійшли до довжини j=0 і так і не знайшли збіги, то зупиняємо процес перебору зразків і вважаємо π[i] = 0 і переходимо до наступного індексу i+1.

Сам алгоритм Кнута-Морріса-Пратта вирішує наступну задачу.

Подано текст t і рядок s, потрібно знайти і вивести позиції всіх входжень рядка s в текст t.

Утворюємо рядок s + # + t, де символ # – це роздільник, який не повинен ніде більше зустрічатися. Порахуємо для цього рядка префікс-функцію. Тепер розглянемо її значення, крім перших n + 1 (які, як видно, відносяться до рядка s та роздільника).

За визначенням, значення π[i] показує найбільшу довжину підрядка, що закінчується в позиції i, та збігається з префіксом. Але в нашому випадку це π[i] – фактично довжина найбільшого блоку збігається з рядком s і закінчується в позиції i. Більше, ніж n, ця довжина бути не може – за рахунок роздільника. А ось рівність π[i] = n (там, де воно досягається), означає, що в позиції і закінчується шукане входження рядка s (тільки не треба забувати, що всі позиції відраховуються в склеєної рядку s + # + t).

Складність алгоритму: O(N+M) за часом, O(N+M) пам’ять.

2.2.3 Алгоритм Рабіна-Карпа

Алгоритм Рабіна-Карпа — алгоритм пошуку рядка запропонований Рабіном і Карпом. Алгоритм показує високу продуктивність на практиці, а також дозволяє узагальнення на інші споріднені задачі.

Ідея алгоритму полягає в заміні текстових рядків числами, порівняння

яких можна виконувати значно швидше.

Для простоти припустимо, що алфавіт складається з десяткових цифр Σ = {0,1,…,9}. (В загальному випадку можна припустити, що кожний символ — це цифра в системі числення з основою d, де d = |Σ|.) Після цього, рядок з k символів, можна розглядати як число довжини k. Тобто символьний рядок «12345» відповідає числу 12345.

Для заданого зразка P[1..m] позначимо через p відповідне йому десяткове значення. Аналогічно, для заданого тексту T[1..n] позначимо через ts десяткове значення підрядка T[s+1..s+m] довжини m при s = 0,1,…,n-m. Очевидно, що t_s=p тоді і тільки тоді, коли T[s+1..s+m]=P[1..m]; таким чином, s — допустимий зсув тоді і тільки тоді, коли t_s=p.

Якщо значення p можна обчислити за O(m) а значення ts за сумарний час O(n-m+1), то усі допустимі зсуви можна було б знайти за час O(m) + O(n-m+1) = O(n) шляхом порівняння p з кожним з можливих t_s. (Покищо до уваги не береться той факт, що величини p і t_s можуть виявитись дуже великими.)

З допомогою схеми Горнера величину p можна обчислити за час O(m):

$$p = P[m]+10(P[m-1]+10(P[m-2]+...+10(P[2]+10P[1]))). \qquad (2.2)$$

Значення t0 можна обчислити з масиву T[1..n] аналогічним способом за час O(m). В той же час, знаючи величину t_s величину t_{s+1} можна обчислити за фіксований час:

$$t_{s+1} = 10(t_s - 10\,\hat{}\,(m-1)T[s+1]) + t[s+m+1] \qquad (2.3)$$

Отже, всі ts можна обчислити за час O(n).

В цій процедурі пошуку наявна складність, пов'язана з тим, що значення p і t_s можуть виявитись занадто великими і з ними буде незручно працювати. Якщо зразок P складається з m цифр, то припущення про те, що арифметичні операції з числом p (до якого входить m цифр) займають «фіксований час», не відповідає дійсності. Ця проблема має просте

вирішення: обчислення значень p і ts за модулем деякого числа q. Оскільки обчислення проводяться рекурентно, то знаходження p можна виконати за O(m), а всіх ts відповідно за O(n). Значення q звичайно обирають таким, щоб величина dq не перевищувала максимальну величину комп'ютерного слова.

Тоді, співвідношення для t_{s+1} приймає вигляд:

$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1) \bmod q \qquad (2.4)$$

де $h \equiv d^{m-1} (\bmod q)$ — значення, що приймає цифра «1» поставлена в старший розряд m-значного текстового рядка.

Робота по модулю q має свої недоліки, оскільки з $ts \equiv p (\bmod q)$ не випливає, що ts=p. З іншого боку, якщо $ts \neq d^{m-1} (\bmod q)$, то обов'язково виконується співвідношення $ts \neq p$ і можна зробити висновок, що зсув s неприпустимий. Таким чином, співвідношення $ts \equiv p (\bmod q)$ можна використовувати як швидкий евристичний тест, що дозволяє виключити із розгляду деякі неприпустимі зсуви. Усі зсуви, для яких співвідношення виконується, треба додатково перевірити. Якщо q достатньо велике, то можна сподіватися, що хибні зсуви будуть зустрічатися досить рідко і час додаткової перевірки буде малим.

Складність алгоритму: O(N+M) за часом, O(N+M) пам’ять.

2.2.4 Пошук в словнику за допомогою бінарного пошуку

Для використання бінарного пошуку, нам необхідно мати відосортований словник. Для цього потрібно або завчасно додавати слова в словниковому порядку, або додати всі слова та потім відсортувати їх зі складністю O(N*logN)(якщо не відомо щось про порядок слів що допоможе відсортувати їх швидше).

Алгоритм пошуку слова буде відповідати стандартному бінарному

пошуку. Розглянемо спочатку весь масив слів, та порівняємо шукне слово із середнім в цьому масиві. Знаючи яке зі слів менше в лексикографічному порядку, будемо розглядати лише потрібну нам половину масиву. Повторимо цей процес доки не залишиться одне слово. Якщо це слово співпадає з шуканим, то ми його знайшли, інакше його не існує в нашому масиві. Так як на кожному кроці область масиву на якій ми проводимо пошук зменшується в 2 рази, то складність пошуку O(logN).

Для об’єднання двох словників, замість сортування можна використати метод двох вказівників.

Так як словники повинні вже бути відосортованими, то найменшим в алфавітному порядку в сумарному словнику буде одне з найменших слів цих двох словників. Вибравши його ми можемо порівняти наступне слово після вибраного, з найменшим із іншого словника. Таким чином замість сортування за O(N*logN) ми зможемо об’єднати словники за O(N), бо для кожного слова із результуючого словника нам потрібно буде зробити лише одну перевірку.

Складність алгоритму:

- O(N*logN) час побудови словника, де N – кількість слів в словнику;
- O(N) додання одного слова;
- O(logN) – пошук слова;
- O(N) – об’єднання двох дерев;
- O(N*M) – максимальний розмір необхідної пам’яті, де M-середній розмір слова.

2.2.5 Пошук заснований на хеш таблицях

Для цього алгоритму знайдемо хеш кожного зі слів словника. Як саме знаходити хеш описано в пункті 2.2.3.

Цей хеш буде слугувати нам ключем для первинної перевірки наявності слова в словнику, бо якщо його нема, то такого слова точно не

існує в словнику. Для більш точного пошуку ми можемо використати метод хешування по двох ключах. Для цього проведемо хешування за двома різними простими основами, та ключем буде не значеннння хешу, а значення пари хешів по різним основам. Це допопможе зменшити ймовірність фальшивого зпрацювання. Для того щоб швидко дізнатися чи є вибраний хеш в нашому словнику ми можемо скористатися різними методами, такими як бінарний пошук (аналогічний методу 2.2.4, але будемо зберігати не самі рядки, а їх хеші), метод бінарного збалансованого дерева(аналогічний методу з пункту 2.2.6), або методу таблиці.

Для методу таблиці потрібно виділити масив розміром рівним основі на базі якого проводилося хешування. Після цього для кожного зі слів потрібно порахувати хеш, та додати в відповідну комірку необхідне нам слово. Кожна комірка має свій алогоритм пошуку слів, але вважаємо, що ми можемо підібрати базу для хешу та модуль так, щоб всі слова розподілялись між комірками майже рівномірно. Метод вигідно застосовувати тоді і тільки тоді, коли величина прискорення яка не перевищує P(де P- модуль за яким проводиться хешування), була значно вищою ніж час затрачений на хешування, який пропорційний довжині слова.

Якщо хешувати за двома модулями, то необхідно буде виділяти пам’ять рівну P1*P2, тому прості числа повинні бути не настільки великими як в попердньому випадку, або потрібно буде застосовувати інший метод для швидкого пошуку пар.

Складність алгоритму:

- O(N*M) час побудови таблиці, де N – кількість слів в словнику, M-довжина слова;
- O(M) додання одного слова;
- O(1) – пошук слова;
- O(P) – об’єднання двох таблиць, де P- модуль хешування;
- O(N*M+P) – максимальний розмір необхідної пам’яті, де M-середній розмір слова.

2.2.6 Пошук заснований на бінарному збалансованому дереві

Збалансоване дерево — це такий різновид двійкового дерева пошуку, яке автоматично підтримує свою висоту, тобто кількість рівнів вершин під коренем є мінімальною.

Так як для кожної пари слів ми можемо визначити операції < та > за алфавітним порядком, то ми можемо використати пошук за допомогою бінарного збалансованого дерева.

Для того щоб підтримувати збалансування дерева, при доданні елементу ми можемо використати наступний метод. Порівнявши наш рядок із корневим, ми можемо з'ясувати в яку строну дерева нам потрібно піти. Знаючи це ми можемо порахувати як зміняться розміри піддерев. Якщо розмір одного з піддерев більше ніж на 2 чим розмір іншого, то ми можемо зменшити розмір цього піддерева зробивши його корінь корнем усього дерева, змінивши структуру дерева. Таким чином ми зможемо підтримувати його збалансованиість.

Складність алгоритму:

- O(N*logN*M) час побудови дерева, де N – кількість слів в словнику, M- довжина слова;
- O(M*logN) додання одного слова;
- O(M*logN) – пошук слова;
- O(N*logN*M) – об'єднання двох таблиць, де P- модуль хешування;
- O(N*M) – максимальний розмір необхідної пам'яті, де M-середній розмір слова.

Кожна операція потребує додаткової складності, так як порівняння двох рядків відбувається за O(M). Для зменшення цього можна використовувати наприклад хеші.

2.2.7 Пошук на основі бору

Що таке бор? Строго кажучи, бор - це дерево, в якому кожна вершина позначає якийсь рядок (корінь позначає нульовий рядок - ε). На ребрах між вершинами написана 1 літера, таким чином, добираючись по ребрах з кореня в якусь вершину і контангенуючи літери з ребер у порядку обходу, ми отримаємо рядок, що відповідає цій вершині. З визначення бору як дерева випливає також єдиність шляху між коренем і будь-якою вершиною, отже — кожній вершині відповідає рівно один рядок (надалі ототожнюватимемо вершину і рядок, який вона позначає).

Будувати бор будемо послідовним додавання вихідних рядків. Спочатку ми маємо 1 вершину, корінь (root) — порожній рядок. Додавання рядка відбувається так: починаючи з кореня, рухаємося по нашому дереву, вибираючи щоразу ребро, що відповідає черговій літері рядка. Якщо такого ребра немає, ми створюємо його разом із вершиною. Ось приклад побудованого бору для рядків: 1) acab, 2) accc, 3) acac, 4) baca, 5) abb, 6) z, 7) ac.

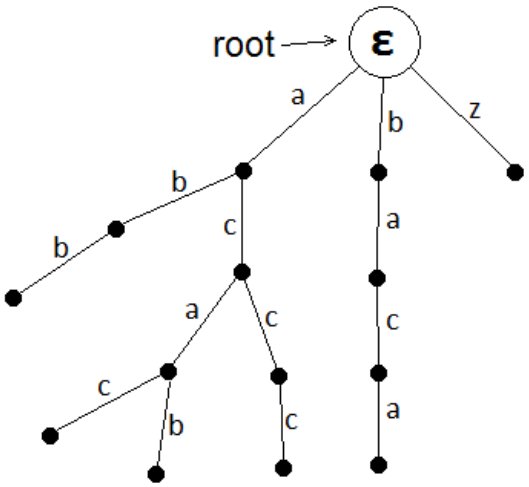

Рисунок 2.1 – Побудований бор

Звернемо увагу на те, що при доданні рядка 7, нам не потрібно взагалі додавати вершини, тому потрібно додати признак того, чи є дана вершина кінцем якогось рядка.

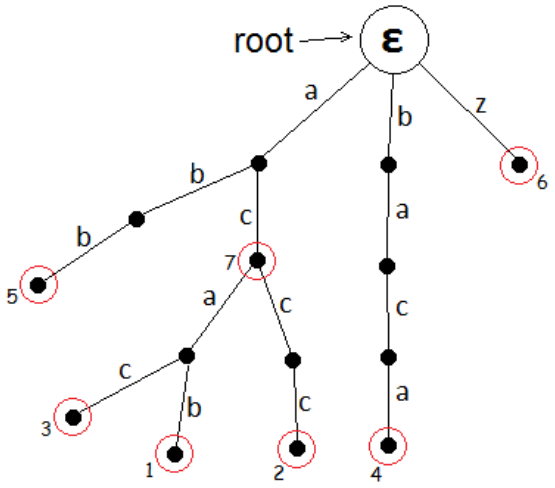

Рисунок 2.2 – Побудований бор з додатковими признаками

Очевидно, що пошук слова буде виконуватися аналогічно додаванню нового слова. Якщо шляху не існує, то замість додавання вершини ми повертаємо відповідь, що слова не існує. Якщо при закінчені слова в останній вершині не було додаткового признаку, то також повертаємо цю саму відповідь.

Складність:

O(N*M) час побудови бору, де N – кількість слів в словнику, M-середня довжина слова.

O(M) додання одного слова.

O(M) – пошук слова.

O(M*K) – об’єднання двох дерев.

O(M*K) – максимальний розмір необхідної пам’яті, де K-потужність алфавіту(максимальна кількість різних літер які використовуються).

2.2.8 Порівняння алгоритмів

Порівняємо алгоритми пошуку слів у словнику за допомогою наступної таблиці:

Таблиця 2.1. – Порівняння алгоритмів пошуку слів

	Бінарний пошук	Хеш таблиці	Бінарне дерево	Бор
Створення словника	NlogN	N*M	NMlogN	N*M
Об’єднання двох словників	N	P	NMlogN	M*K
Додання нового слова	N	M	MlogN	M
Пошук слова	logN	1	MlogN	M
Використана пам’ять	N*M	N*M+P	N*M	M*K

Як можемо бачити, кожен з алгоритмів має свої переваги та недоліки. Якщо потрібно зробити застосунок максимально швидко, та час роботи або занята пам’ять не важливі, то краще за все використовувати збалансоване дерево, тому що воно зазвичай вже реалізоване в мовах програмування. Наприклад std::map в c++ або dictionary в pyton.

Для швидкого пошуку якщо немає обмежень на пам’ять, ми можемо використовувати хешування поєднане з бінарним пошуком, що дасть гарний приріст продуктивності.

Якщо потрібно часто саме знаходити слова, та ми можемо дозволити собі використати складну структуру, рекомендується скористатися методом бора, яка дозволить як швидко додавати слова, так і швидко їх знаходити за час близький до довжини слова. Але при використанні великого алфавіту метод втрачає свою потужність.

2.3 Методи машинного перекладу

Машинний переклад (МП) – технології автоматизованого перекладу текстів (письмових та усних) з однієї природної мови на іншу за допомогою комп'ютера; напрямок наукових досліджень, пов'язаний з побудовою систем автоматизованого перекладу.[4,5]

На базовому рівні, робота комп'ютерних програм для перекладу полягає у заміні слів чи словосполучень з однієї мови на слова чи словосполучення з іншої. Однак тоді виникає проблема, що така заміна не може забезпечити якісний переклад тексту, адже потрібне визначення та розпізнання слів та цілих фраз з мови оригіналу. Це спонукає активну наукову діяльність у галузі комп'ютерної лінгвістики. Наразі, для вирішення неоднозначностей при перекладі, використовуються багатомовні онтологічні ресурси, такі як WordNet та UWN.

Машинний переклад – одна з підгруп комп’ютерної лінгвістики, яка досліджує використання програмного забезпечення для перекладу тексту з однієї мови на іншу. На початковому рівні МТ виконує звичайну заміну слів з однієї мови на слова з іншої мови, але, зазвичай, переклад здійснений таким чином не є дуже якісним, адже для того, щоб повністю передати сенс речення, та знайти найспорідненіший аналог в «цільовій» (target language) – потрібній перекладачу мові, часто потрібно здійснювати переклад цілої фрази.

Вирішення цієї проблеми з статистичними (statistical) та нейронними (neural) системами перекладу є швидко зростаючою галуззю, яка веде до покращення перекладу, усунення різниці в лінгвістичній типології, перекладу ідіом та виділенню аномалій.

Сучасне програмне забезпечення для машинного перекладу має функцію зміни налаштувань за доменом (domain) – галуззю або професійною діяльністю (напр. метеорологічні звіти). Обмежуючи сферу допустимих замін/заміщень ми маємо змогу отримати кращий результат перекладу.

Цей метод є особливо ефективним у сферах, де використовується формальна чи шаблонна мова. Це означає, що машинний переклад, наприклад, урядових та юридичних документів є більш якісним, ніж переклад розмовних чи будь-яких менш стандартизованих текстів.

Підвищення якості кінцевого продукту може також бути досягнуто шляхом людського втручання: наприклад деякі системи зможуть надати більш точний переклад, якщо користувач заздалегідь позначить які слова в тексті є власними іменами. За допомогою цих методів, МТ проявив себе як знаряддя, що дійсно допомагає перекладачам, а іноді, у дуже рідкісних випадках і сам може слугувати високоякісним перекладачем, здійснюючи переклад, який не потребує коректівів. З моменту виникнення машинного перекладу (кінець 50-х років XX ст.) і до сьогодення науковці сперечаються щодо його прогресу та потенціалу.

Починаючи з 1950-х років ряд дослідників поставили під сумнів той факт, що автоматично здійснений переклад може бути високої якості.

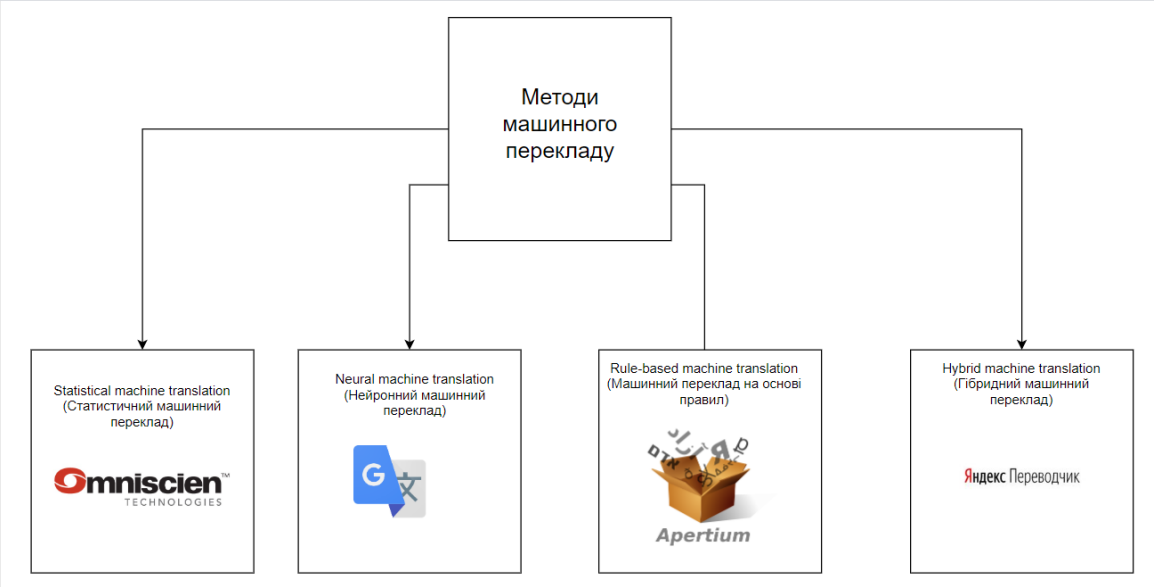

Рисунок 2.3 – Методи машинного перекладу

Деякі критики стверджують що існують перешкоди що унеможливлюють повну комп’ютеризацію процесу перекладу.

Існують два принципово різних підходи до побудови алгоритмів машинного перекладу: заснований на правилах (rule-based) і статистичний, або заснований на статистиці (statistical-based). Перший підхід є традиційним і використовується більшістю розробників систем машинного перекладу (ПРОМТ в Росії, SYSTRAN у Франції, Linguatec у Німеччині тощо).

В подальшому розглянемо такі методи машинного перекладу:

- statistical machine translation (статистичний машинний переклад);
- neural machine translation (нейронний машинний переклад);
- rule-based machine translation (машинний переклад на основі правил);
- hybrid machine translation (гібридний машинний переклад).

2.3.1 Статистичний машинний переклад

Статистичний машинний переклад – це різновид машинного перекладу тексту, заснований на порівнянні великих обсягів мовних пар. Мовні пари – тексти, що містять речення однією мовою і відповідні речення іншою, можуть бути як варіантами написання двох речень людиною – носієм двох мов, так і набором речень та їх перекладів, виконаних людиною. Таким чином статистичний машинний переклад володіє властивістю «самонавчання». Чим більше в розпорядженні програми є мовних пар і чим точніше вони відповідають один одному, тим кращий результат статистичного машинного перекладу.

Під поняттям «статистичного машинного перекладу» мається на увазі загальний підхід до вирішення проблеми перекладу, який заснований на пошуку найімовірнішого перекладу речення з використанням даних, отриманих з двомовної сукупності текстів.

Як приклад двомовної сукупності текстів можна назвати парламентські звіти, які являють собою протоколи дебатів в парламенті. Двомовні парламентські звіти видаються в Канаді, Гонконгу та інших країнах; офіційні документи Європейського економічного співтовариства

видаються 11 мовами, а Організація Об'єднаних Націй публікує документи на декількох мовах. Як виявилося, ці матеріали є безцінними ресурсами для статистичного машинного перекладу.

Дана система базується на статистичному вирахуванні вірогідності збігів. Задля виконання перекладу програма повинна мати доступ до сотень мільйонів документів, які заздалегідь були перекладені людьми. Такі документи слугують для системи шаблонами, на основі яких вона і здійснює переклад. Чим більше документів, тим вища ймовірність більш якісного перекладу.

На початку свого існування, з 2006 року, Google Translate базувався саме на статистичному методі машинного перекладу, і здійснений ним переклад був дуже низької якості, і вважався одним з найгірших варіантів перекладу, який може здійснити онлайн-перекладач. Сьогодні Google використовую «нейронний» метод МП і складає серйозну конкуренцію комерційним підприємствам, продукція яких не є безкоштовною.

Перші ідеї статистичного машинного перекладу були введені Уорреном Вівером у 1949 році, включаючи ідеї застосування теорії інформації Клода Шеннона. Статистичний машинний переклад був знову введений в кінці 1980-х – початку 1990-х років дослідниками дослідницького центру Томаса Дж. Уотсона IBM і сприяв значному пожвавленню інтересу до машинного перекладу в останні роки. До впровадження нейронного машинного перекладу це був найбільш широко вивчений метод машинного перекладу.

Ідея статистичного машинного перекладу походить від теорії інформації. Документ перекладається відповідно до розподілу ймовірностей $p(e|f)$, що рядок e на цільовій мові (наприклад, англійська) є перекладом рядка f мовою джерела (наприклад, французькою).

До проблеми моделювання розподілу ймовірностей $p(e|f)$ розглянуто декілька способів. Один із підходів, який добре піддається впровадженню комп'ютера, – це застосувати теорему Байєса, тобто $p(e|f) = f(f|e)p(e)$, де модель перекладу $p(f|e)$ – це ймовірність того, що вихідний рядок є

перекладом цільової рядка, а мовна модель $p(e)$ – ймовірність бачити цей цільовий рядок мови. Це розкладання є привабливим, оскільки розбиває проблему на дві підпрограми. Пошук найкращого перекладу e здійснюється шляхом вибору того, який дає найбільшу ймовірність.

Для суворої реалізації цього потрібно було б виконати вичерпний пошук, пройшовши всі рядки e* рідною мовою. Ефективне виконання пошуку – це робота дешифратора машинного перекладу, який використовує сторонні рядки, евристику та інші методи, щоб обмежити пошуковий простір і одночасно зберігати прийнятну якість. Цей компроміс між якістю та часом використання може бути також розпізнаний у мовленні.

Оскільки системи перекладу не в змозі зберігати всі рідні рядки та їхні переклади, документ, як правило, перекладається речення за реченням, але навіть цього недостатньо. Мовні моделі, як правило, наближаються до згладжених n-грамових моделей, і подібні підходи застосовуються до моделей перекладу, але є додаткова складність через різну довжину речення та порядки слів у мовах.

Моделі статистичного перекладу спочатку були засновані на словах , але значні успіхи були досягнуті впровадженням моделей на основі фрази. Останні роботи включають синтаксичні чи квазісинтаксичні структури.

2.3.2 Нейронний машинний переклад

Нейронний машинний переклад Google (GNMT) – це система нейронного машинного перекладу (NMT), розроблена компанією Google і представлена в листопаді 2016 року, яка використовує штучну нейронну мережу для підвищення швидкості і якості перекладу в Google Перекладачі.[6,7]

GNMT підвищує якість перекладу, застосовуючи метод машинного перекладу що базується на прикладах, в якому система "навчається на мільйонах прикладів". Архітектура системного навчання GNMT була вперше

випробувана на більш ніж ста мовах, що підтримуються Google Перекладачем. Завдяки потужній внутрішній структурі, система з часом вчиться створювати більш якісні і природні переклади. GNMT здатна перекладати речення в цілому, а не частинами. GNMT здатна виконувати інтерлінгвальний машинний переклад, кодуючи семантику речення замість того, щоб запам'ятовувати переклади окремих фраз.

Проект Google Brain був створений у 2011 році в «секретній дослідницькій лабораторії Google X» Джеффом Діном, співробітником Google, Грегом Коррадо, дослідником з Google, та Ендрю Іном, професором комп'ютерних наук Стенфордського університету. Робота Іна лягла в основу одного з найбільших технологічних проривів в Google і Стенфорді.

У вересні 2016 року дослідницька група Google оголосила про розробку системи перекладу GNMT, і до листопада Google Перекладач почав використовувати нейронний машинний переклад (NMT) замість колишніх статистичних методів (SMT), які використовувалися з жовтня 2007 року всередині власної закритої SMT системи.

Система NMT всередині Google Перекладача використовує велику штучну нейронну мережу, придатну для глибинного навчання. Вивчаючи мільйони прикладів, GNMT покращує якість перекладу, використовуючи більш широкий контекст для виведення найбільш якісного перекладу. Потім результат перебудовується і пристосовується для відповідності граматиці людської мови. GNMT не створила свою внутрішню універсальну мову, а скоріше прагнула знайти спільне між багатьма мовами, що має бути цікаво більше для психологів і лінгвістів, ніж для фахівців в галузі інформатики. Новий механізм перекладу був включений в обидві сторони для дев'яти мов: англійської, французької, німецької, іспанської, португальської, китайської, японської, корейської та турецької в 2016 році.

У березні 2017 року були додані ще три мови: російська, хінді і в'єтнамська. У тому ж місяці за допомогою спільноти Google Перекладача була додана підтримка івриту та арабської мови. Далі в кінці квітня 2017

року було додано підтримку дев'яти індійських мов, а саме: хінді, бенгалі, маратхі, гуджараті, пенджабі, тамільської, телугу і малаялам.

Стверджується, що система GNMT краще попереднього варіанту Google Перекладача тим, що вона може виконувати "прямий переклад", тобто перекладати з однієї мови на іншу безпосередньо (наприклад, з японської на корейську). Раніше Google Перекладач спочатку перекладав з початкової мови на англійську, а потім з англійської на кінцеву мову замість прямого перекладу з однієї мови на іншу.[8,9]

2.3.3 Машинний переклад на основі правил

Машинний переклад на основі правил характеризується використанням і ручним створенням лінгвістичних правил. Ефективність систем машинного перекладу на основі правил визначається якістю двомовних словників та точністю заданих правил, а їх створення потребує довготривалої роботи.[10]

Перші системи машинного перекладу створювалась для конкретних пар мов і ґрунтувались на складних процесах моделювання мови, основу яких становили методи аналізу, трансферу, синтезу й інтерлінгви. Системи машинного перекладу першого покоління працювали за методом прямої заміни слів мови оригіналу словами мови перекладу. Системи другого покоління аналізували структури мови оригіналу, а потім на основі трансферу синтезували їх в еквівалентні структури мови оригіналу. Третім поколінням були системи машинного перекладу на основі формальної мови-посередника – інтерлінгви. Ця концепція передбачала перетворення слів на мову-посередника, яка є універсальною мовою, створеною для системи, незалежною від залучених у процес перекладу мов. Цей підхід використовує два методи: аналізу і синтезу.[11]

Системи машинного перекладу на основі правил ґрунтуються на різних рівнях лінгвістичного опрацювання мовної пари:

- морфологічному: лематизація лексичних одиниць, пошук лексичних

одиниць у словнику, аналіз морфем, розпізнавання контекстного граматичного класу лексичних одиниць, відмінків, флексій тощо;

- синтаксичному: розпізнавання типів синтаксичних структур, реляційних зв’язків між окремими елементами синтаксичної структури тощо;

- семантичному: виокремлення лексичного значення багатозначних лексичних одиниць та афіксів, визначення їхньої семантичної функції, синтез їхньої синтаксичної однозначності на основі семантичного аналізу.

Системи машинного перекладу на основі правил не потребують доступу до баз паралельних текстів, їх можна налаштовувати, що поліпшує якість перекладу спеціалізованих текстів.

Системи на основі правил можуть мати справу з багатьма мовними явищами і зручні в супроводі. Проте винятки в граматиці додають труднощів, що потребує розроблення нових алгоритмів і покращення раніше створених.

Основним недоліком систем машинного перекладу на підставі правил є те, що для підвищення якості цих систем потрібно покращувати раніше створені й розробляти нові алгоритми, що є дуже ресурсоємним.[13]

2.3.4 Гібридний машинний переклад

Останніми роками все більшої популярності набирає Гібридний МП (Hybrid machine translation [HMT]).

ГМП – використовує сильні сторони обох систем машинного перекладу, в результаті користувач отримує якісний переклад, який забезпечує RBMT та високу швидкість, яку надає статистичний метод.

Кілька компаній які займаються МП, наприклад Omniscien Technologies (колишня Asia Online), LinguaSys, SYSTRAN, PROMT і т.п. стверджують, що використовують саме гібридний вид МП.

Види гібридного МП різняться між собою:

- статистична корекція після виконання перекладу системою RBMT: спочатку переклади здійснюються системою RBMT, а після цього, з метою

виправлення помилок, або ж внесення власних коректівів, застосовується система статистичного МП;

- статистичний метод, що керується правилами:

Правила використовують для попередньої обробки даних, задля здійснення кращого управління статистичним механізмом. Правила також використовують для обробки даних після здійснення статистичного перекладу з метою виконання такої функції як нормалізація.

Цей метод перекладу має багато переваг: він є більш потужним, гнучким (тобто здійснює якісний переклад в багатьох сферах діяльності). Система також контролює процес обробки контенту як при здійсненні завчасного перекладу (напр. розподілу вмісту та термінів що не перекладаються), так і після здійснення перекладу (корегування та виправлення).

Нещодавно, з появою Нейронного МП, з'явилася нова версія Гібридного МП, яка поєднує в собі переваги 3 видів машинного перекладу: RB, статистичного та нейронного. Такий підхід дозволяє користуватися перевагами NMT та SMT, які в процесі перекладу контролюються правилами RBMT. Єдиним недоліком цієї системи перекладу є невід'ємна складність такої роботи, яка робить його нагідним лише для специфічних випадків використання. Одним з прихильників такого методу для складних випадків є Omniscien Technologies.

Завдяки здатності статистичних та гібридних систем МП навчатися внаслідок накопичення мовних даних якість перекладу в них підвищується з кожним наступним перекладеним текстом.[14]

2.3.5 Порівняння алгоритмів

Для порівняння використаємо наступні параметри:

- швидкість навчання – кількість часу необхідна для отримання перших результатів перекладу;

- якість перекладу – з якою ймовірністю переклад може допускати помилки, чи потрібна обробка людини після перекладу, на скільки переклад машиною відрізняється від людського перекладу;

- розширюваність – чи можливо додати нові слова до перекладача, змінити вірогідності перекладу;

Таблиця 2.2 – Порівняння з алгоритмів

Критерій для порівняння	Статистичний машинний переклад	Нейронний машинний переклад	Машинний переклад на основі правил	Гібридний машинний переклад
Швидкість навчання	Потрібно найменше часу	Потрібно досить багато часу	Потрібно досить багато часу	Потрібно найбільше часу
Якість перекладу	Висока	Найкраща.	Висока	Досить висока
Розширюваність	Передбачена	Передбачена	Є можливість розширювання, але тяжче ніж в інших алгоритмах	Є можливість розширювання, але тяжче ніж в інших алгоритмах

Після порівняння можемо зробити висновок, що кожен з типів машинного перекладу можливо застоувати, але найкращими є нейронний та гібридний машинний переклад, які зараз застосовуються найчастіше.

2.4 Результат аналізу проблемної області

У результаті аналізу проблемної області, для вирішення поставленої задачі у роботі запропонована узагальнена модель системи автоматичного перекладу текстів з довільної на довільну мови (рисунок 2.4.). Дана система

насамперед передбачає можливість виконувати переклад навіть із/на штучні мови, наприклад, есперанто, талосську, інтерлінгва, ложбан та інші.

Наведена модель показує взаємодію та часткову взаємозалежність модулів створення та коригування словника та модулю перекладу, що пояснюється підходом зворотнього розповсюдження помилки перекладання.

Тексти, що подаються на вхід запрпонованої моделі мають бути представлені на двох довільних мовах, мають бути однаковими за кількістю речень. Речення рахуються за кількістю знаків крапка(«.»), знак оклику(«!»), знак питання(«?»), три крапки(«…»). Крім того, кожне речення повинно мати хоча б одне слово. Якщо, наприклад, між знаком оклику та знаком питання будуть лише проміжки, коми, лапки та інші розділові знаки, то це не буде вважатися реченням.

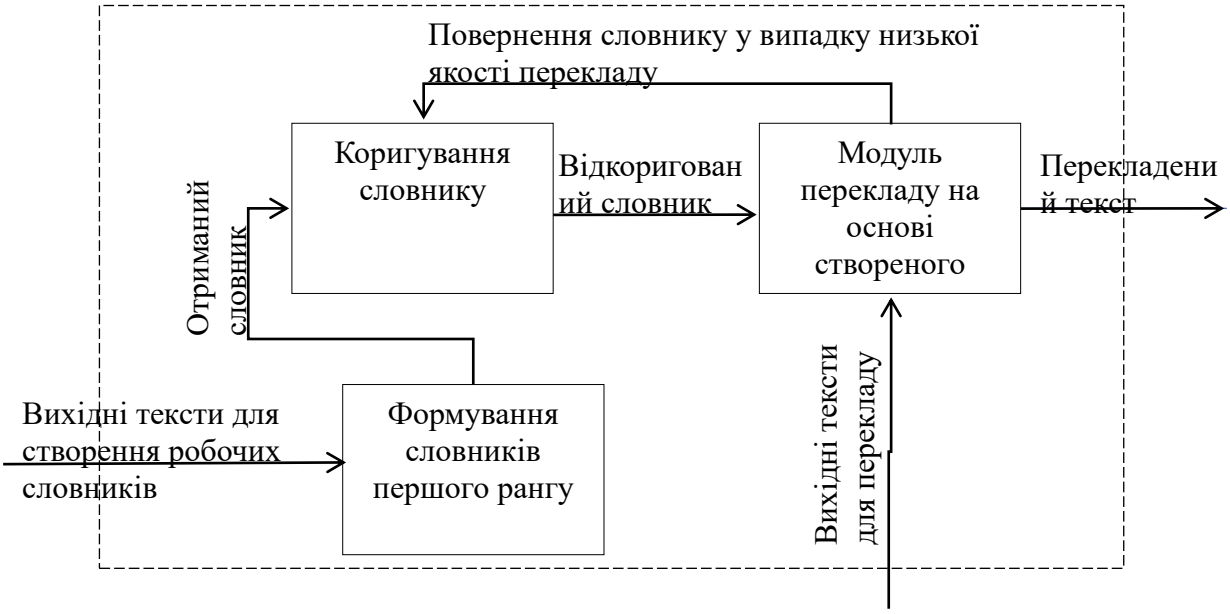

Рисунок 2.4 – Узагальнена модель системи автоматичного перекладу текстів з довільної на довільну мови

Крім того, речення з однаковими порядковими номерами будуть рахуватися перекладами одне одного, тобто змінювати порядок речень для правильної роботи програми не можна.

Пояснимо роботу модулю створення робочих словників, які будуть

основою для перекладу вихідних текстів.

Для побудови словників використовується алгоритм, який розраховує відносну вірогідність перекладу на базі місцезнаходження слів у реченні, відсотки відповідних речень в яких зустрічались обидва слова, та їх схожість за допомогою методу триграм та алгоритму Кнута-Моріса-Прата.

Проте, так як знайти один і той самий текст на 2-ох вибраних мовах не завжди можливо, у роботі запропоновано побудувати словник через уже існуючі.

Нехай мови 1 та 2 відомі мови, мови 3 та 4 є сленгами. Крім того між мовами 1–2 є двосторонній словник, а між мовами та їх сленгами також є двосторонній словник. Позначимо це на графі, де кожна мова вершина, а словник –ребро (рисунок 2.5)

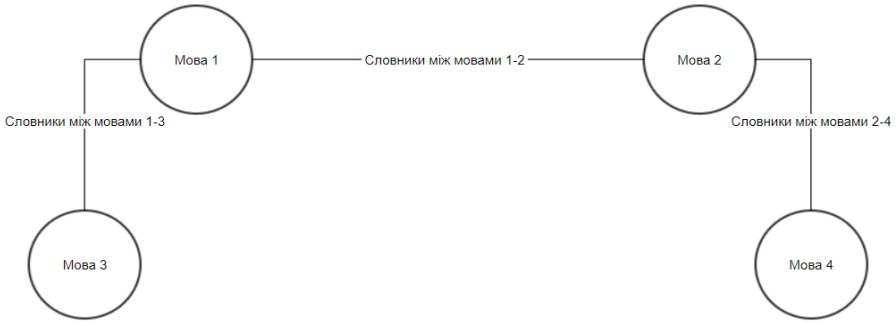

Рисунок 2.5 – Зображення графу мов та словників між ними в початковому положенні

Так як між мовами 3 та 1 та між 1 та 2 є ребра, то для кожного слова з 3-ої мови спробуємо знайти переклад на мову 2 за допомогою словника з першої на другу мову. Так як це не точний переклад, то ймовірності перекладів буде перемножено між собою, та помножено на корегувальний коефіцієнт k, $0.8 \leq k \leq 1$ ($k = 1$ у випадку якщо всі ймовірності будуть не більше одиниці). Позначимо новий словник між мовами 2 та 3 пунктирною лінією на схемі (рисунок 2.6).

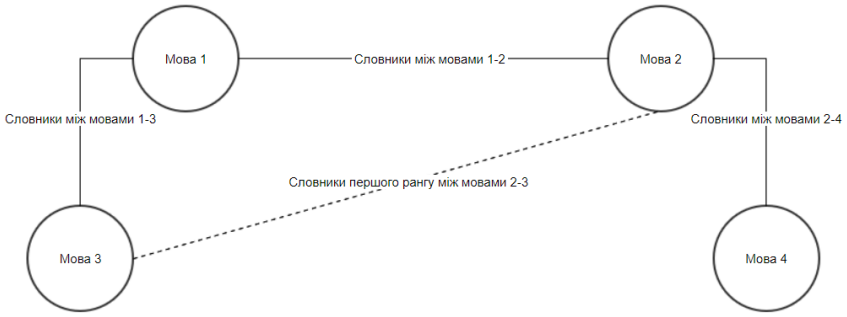

Рисунок 2.6 – Зображення графу мов та словників між ними після побудови словника з 3 в 2

Використовуючи ту саму ідею побудуємо словник через мови 3–2–4. Після чого отримаємо словник з коригуючим коефіцієнтом k=0.64.

Рисунок 2.7 – Зображення графу мов та словників між ними після побудови словника з 3 в 4

Далі цей словник також можна використовувати для побудови нових, але не рекомендується це робити, оскільки словники із рангом 3 або більше, містять в собі дуже низькі вірогідності відносних перекладів, та їх використання не є доречним (ранг словника визначається як сумма рангів словників, які його побудували, плюс один).

Для підвищення якості побудованих нових словників обовязковим та необхідним є доповнення готового словника новими текстами. Для цього на попередніх ітераціях необхідно зберігати не лише слово, можливі переклади та вірогідність цих перекладів, а й кількість оброблених слів та сумарний розмір речень, на основі яких будувався словник.

Після побудови словника, ми можемо за його допомогою перекладати тексти. При генерації словника можливо побудувати не лише словник перекладів слово в слово, а й слово в словосполучення та навпаки. Для того щоб вибрати кращу можливість перекладу словосполучень, було використано наступний алгоритм(рисунок 2.8).

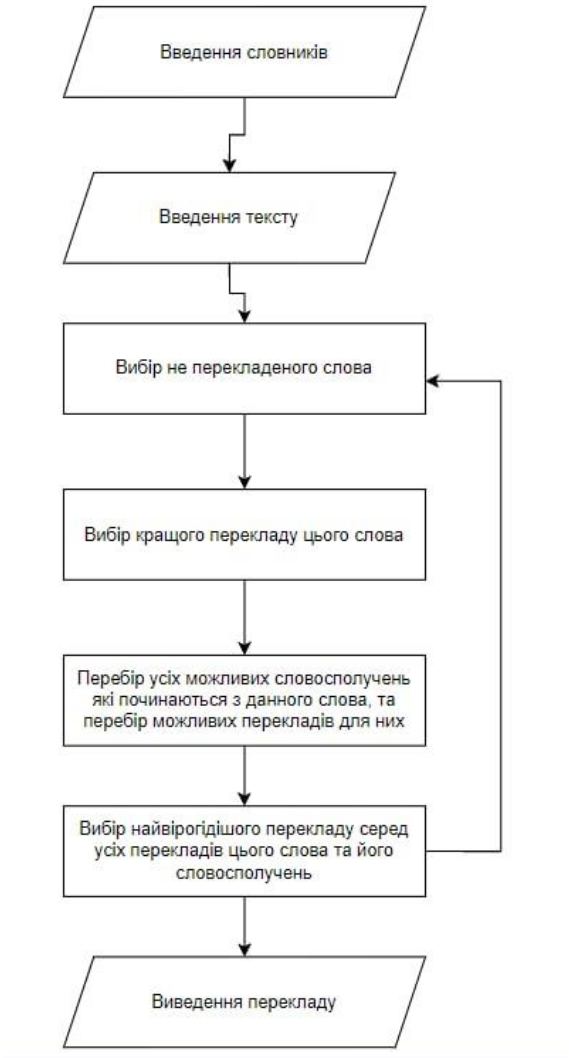

Рисунок 2.8 – Алгоритм перекладу тексту зі словником словосполучень

У результаті роботи алгоритму, вхідний текст буде перекладено за допомогою словників, які були подані на вхід перекладача. Алгоритм намагається максимізувати загальну вірогідність коректного перекладу за домогою максимізації суми вірогідностей відносного перекладу кожного слова або словосполучення, які були записані в словниках.

3 РЕАЛІЗАЦІЯ ПОСТАВЛЕНИХ ЗАДАЧ

3.1 Аналіз структури додатку

Для виконання поставлених задач була розроблена система з наступними входами, цілями та обмеженняим.

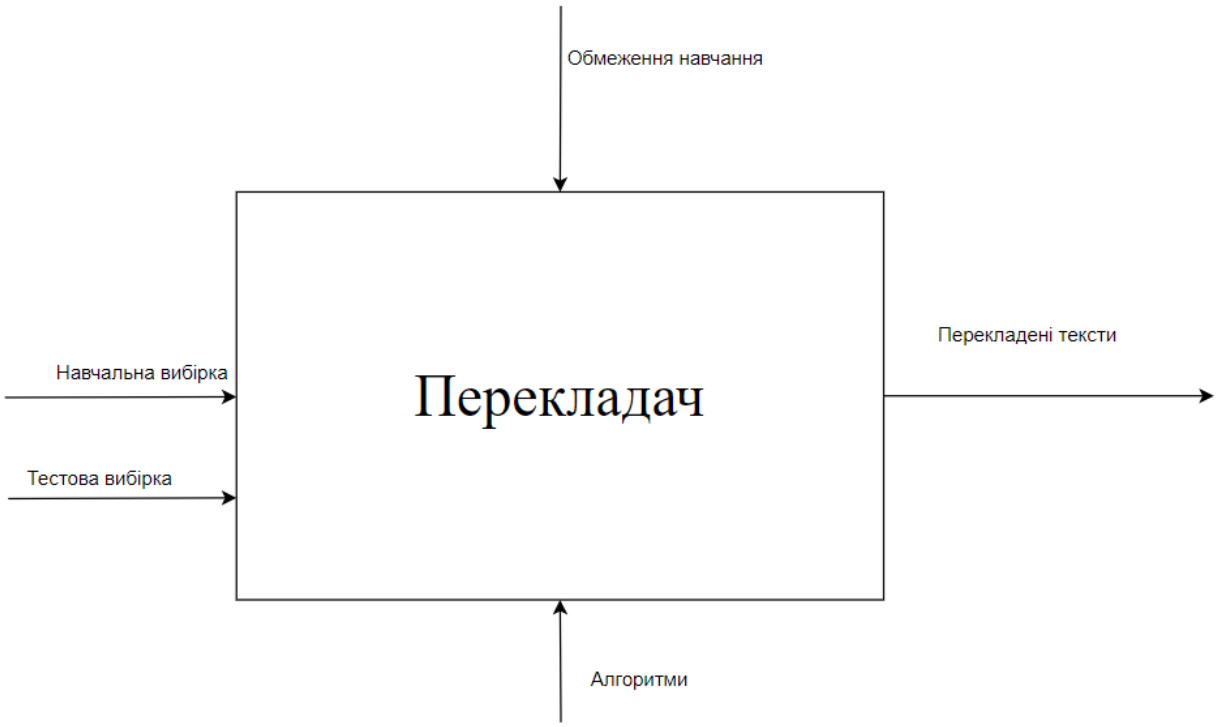

Рисунок 3.1 – Узагальнена модель запропонованої системи машинного перекладу

Розглянемо додаток на більш детальному рівні.

Додаток складається з 3 основних частин:

- дотаток для генерування словників на основі вхідних текстів;
- дотаток для генерування словників на основі словників інших мов;
- програма перекладач для перекладення текстів за допопомогою словників.

Детальна структура зображена на рисунку 3.2.

Рисунок 3.2- Детальна модель запропонованої системи машинного перекладу

Розглянемо блоки до яких ми зможемо застосувати певні покращення та проведемо над ними експерементів.

3.2 Аналіз алгоритму обробки тексту

Щоб почати робити перекладач необхідно створити словник.

Для створення словника необхідний текст перекладений на обидві мови, або набір перекладених не пов'язаних між собою речень. З метою створення початкового словника використовується наступний принцип.

Кожне слово яке зустрічається менше N разів, де N – число, яке обирає користувач. Оптимальними зззначеннями для N є числа від 5 до 10. При даних значеннях швидкість роботи майже не змінюється, але результат перекладу є більш логічним. При N менше 5, переклад вважається не вивченим у повній мірі, та результат перекладу для цього слова може не відповідати дійсності. Для решти слів починається переклад.

Для кожного слова вибираються лише ті речення, в яких це слово фігурувало. Всі слова з перекладу цих речень стають потенційними

перекладами, та для кожного з них вираховується вірогідність того, що це слово дійсно є перекладом.

Ця вірогідність залежить від наступних параметрів:

- відсоток речень в яких зустрічалась ця пара слів, відносно вибраних речень. Кожне речення, в якому зустрічається перекладене слово, та в його перекладі зустрічається потенційний переклад збільшують вірогідність того, що перекладене слово перкладається саме так;

взаємне розташування цих слів у реченні. Вирахуємо вірогідність того, що перекладене слово перекладається саме як потенційний переклад відносно лише одного речення. Для цього знайдемо розміри цього речення (так, як речення перекладено на 2 різні мови, ця величина не завжди співпадає), місце розташування перекладеного слова у реченні та місце розташування потенційного перекладу в перекладі цього речення. Використовуючи ці значення знайдемо відсоткове місце цих слів.

Тоді вірогідність перекладу відносно цього речення буде обернено пропорційна квадрату відстані між цими словами;

- врахування ваги речення в залежності від кількості слів у ньому.

Так, як різні речення можуть мати різну довжну, то необхідно ввести поправочний коефіцієнт, який буде враховувати цю довжину.

Для роз’яснення розглянемо два наступних речення:

- «Світає, край неба палає; соловейко в темнім гаї сонце зустрічає. »
- «Світає».

У другому випадку ми маємо речення лише з одного слова. Його перекладом має бути лише аналогічне коротке речення. Зустрівши таке речення, ми можемо зрозуміти, що його переклад має бути дослівним перекладом, та ми можемо його використовувати без змін з великою ймовірністю.

У першому реченні слів набагато більше. Так як слова могли в перекладі змінити своє розташуваня, кожен з можливих перекладів повинен оцінюватися з меншою ймовірністю. Тому для збаласування впливу цих

речень будемо використовувати поправочний коефіцієнт. Сумма цих коефіцієнтів має бути рівна одиниці, та кожен з них буде залежати від загальної довжини речень та довжини вибраного речення.

Після цієї обробки потенційні переклади які не потрапили до 5 найвірогідніших перекладів, та мають вірогідність перекладу нижче певної відмітки (рекомендується використовувати значення 0.1) відсіюються, як неможливі. Всі інші переклади запам'ятовуються, та потрапляють до словника.

3.2.1 Покращення обробки за допомогою двосторонньої обробки

Так як деякі слова, такі як:

- слова паразити;
- прийменники, частки, сполучники;
- артиклі в іноземних мовах;

зустрічаються майже в кожному реченні, то вірогідність того, що вони будуть вибрані як переклад, після використовування алгоритму з пункту 1 досить висока. Ці слова мають назву шуми. Для зменшення впливу шуму необхідно поставити певний фільтр.

Використуємо наступну ідею. Зробимо переклад не лише з першої мови на другу, а і оберенений переклад на цьому самому тексті. Тоді отримавши 2 словники, ми зможемо знайти такі слова, та зменшити їх вірогідність перекладу.

Алгоритм працює наступним чином. Переберемо кожну пару слів. Будемо розглядати кожну пару окремо, тобто не будемо враховути інші переклади цих слів. Знайдемо вірогідність перекладу слова з першої мови на другу та навпаки. Після чого знайдемо різницю за модулем між цими величинами. Зменшення величин обох ймовірностей буде зменшуватися пропорційно модулю різності між цими величинами.

За замовчуванням це зменшення буде рівне половині від модулю

різниці між цими вірогідностями. Тоді якщо обидва слова будуть мати можливості переводитися одне на одного, то ймовірність буде зменшена не більше ніж в 2 рази.

Крім того, якщо одне слово може перекладатися як інше, але навпаки не виконується, то за замовчуванням ця вірогідність буде зменшена в чотири рази, що дозволяє відсіювати шуми більш ефективно. Якщо вибраний переклад взагалі не існує в другому словнику, то вірогідність буде зменшена в 8 разів.

Це зроблено для захисту від вибору не вірного словника, так як в такому випадку жоден переклад не буде знайдений і всі вірогідності будуть пропорційно зменшені і не змінять загальної картини.

Крім того, якщо проводити нормалізацію ймовірностей (раніше ймовірності вважалися незалежними одна від одної, лежали в границях від 0 до 1 і могли в суммі давати любе число. Нормалізація приведе до відображення відносної вірогідності, тобто їх сумма буде рівна 1, або 0 якщо не існує перекладів), то це зменшення взагалі не вплине на результат.

3.2.2 Покращення часу роботи за допомогою паралелізму

Для того щоб перевірити доцільність використання паралельної обробки даних змінимо трохи введення та виведення даних до алгоритму.

Замість того, щоб вводити данні з файлу будемо генерувати текст одразу у програмі. Генерування буде проходити наступним чином.

Вводимо кількість речень для тестування. Після цього згенеруємо дану кількість слів, та для кожного з цих слів згенеруємо 10 позицій в тексті де вони стояли, та саме слово. Крім того згенеруємо текст іншої мови, необхідної кількості речень, в кожному з яких буде по 10 слів. Кожне зі слів буде довжиною 10 символів (для того щоб алгоритми порівняння н-грам та Кнута-Моріса-Прата не закічувалися моментально, а впливали на час роботи алгоритму).

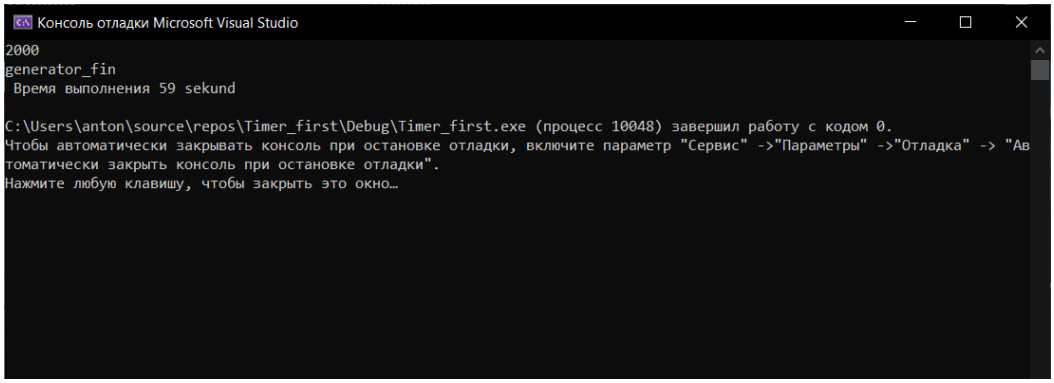

Рисунок 3.3 – Приклад результату роботи алгоритму

Таблиця 3.1 – Результати роботи послідовного алгоритму

Розмір вхідних даних (речення)	100	500	1000	1500	2000
Час роботи(секунди)	3	14	28	44	59

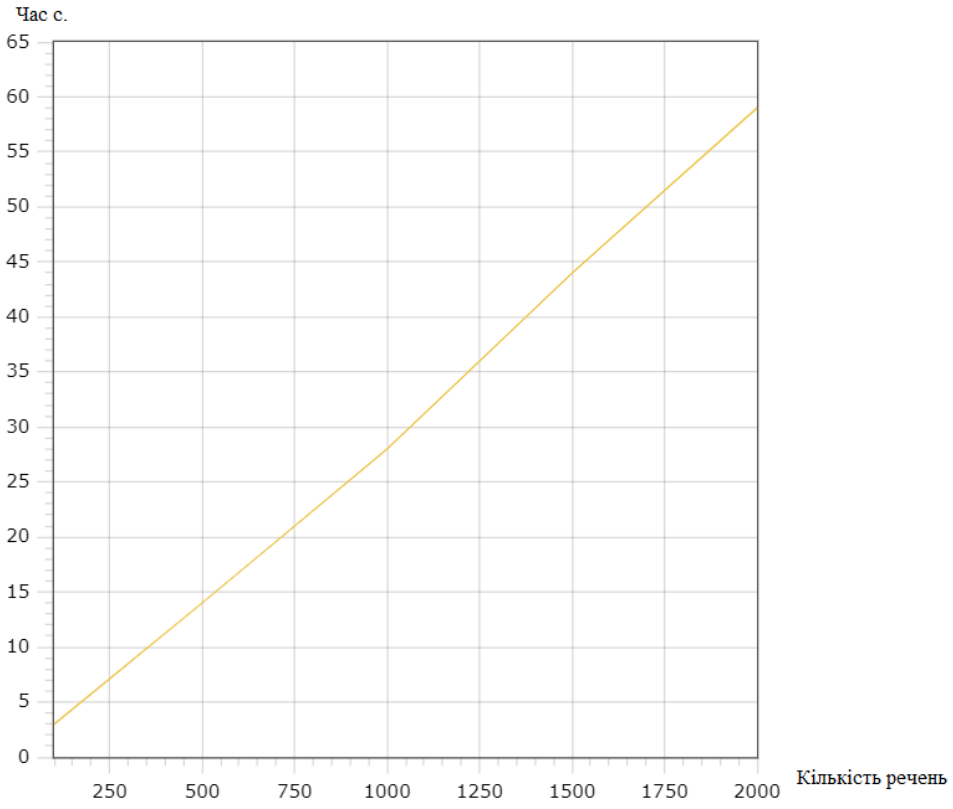

Рисунок 3.4. – Залежність часу роботи від розміру вхідних даних

Аналіз результатів, наведених на рисунку 3.4, показав, що алгоритм не є дуже ефективним в данній версії, та подребує його прискорення.

3.2.2.1 Паралельний алгоритм на CPU

Для того щоб розпаралелити алгоритм змінимо його. Замість того щоб послідовно оброблювати речення, знайдемо для кожного слова його позиції та будемо обробляти кожне слово не залежно одне від одного.

Тоді нам потрібно буде лише зчитувати інформацію про слова в відповідних реченнях, але так як вони не змінюються, то нас не буде проблеми з синхронізацією.

Для того щоб записати результат, виділимо місце завчасно, та для кожного слова це місце буде зарезервоване. Тому нам не потрібно буде витрачати ніякі ресурси для синхронізації. Для того щоб розпралелити використаємо openmp.

Алгоритм буде виглядати як наведено на рисунку 3.5.

Для параелізму використовувся OpenMP, який реалізує паралельні обчислення за допомогою багатопотоковості, в якій «головний» (master) потік створює набір підлеглих (slave) потоків і завдання розподіляється між ними. Передбачається, що потоки виконуються паралельно на машині з декількома процесорами (кількість процесорів не обов'язково має бути більше або дорівнювати кількості потоків).

Завдання, що виконуються потоками паралельно, так само як і дані, необхідні для виконання цих завдань, описуються за допомогою спеціальних директив препроцесора відповідної мови — прагм.

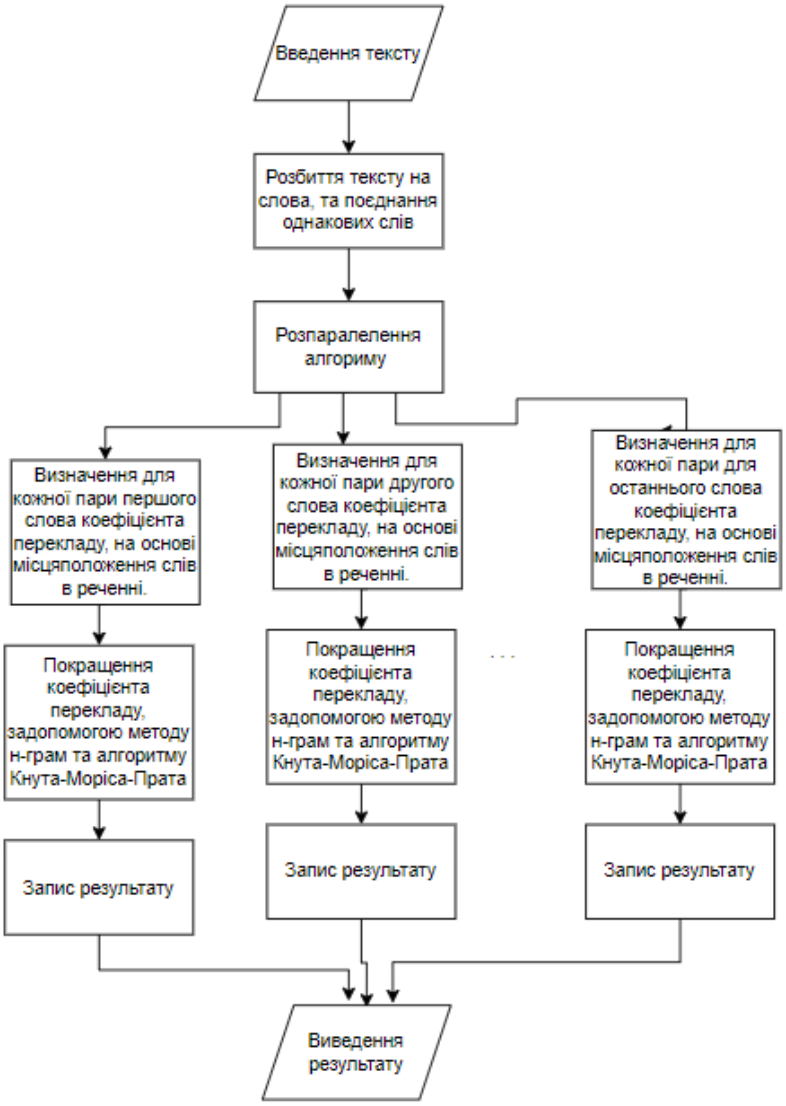

Рисунок 3.5 – Паралельний алгоритм

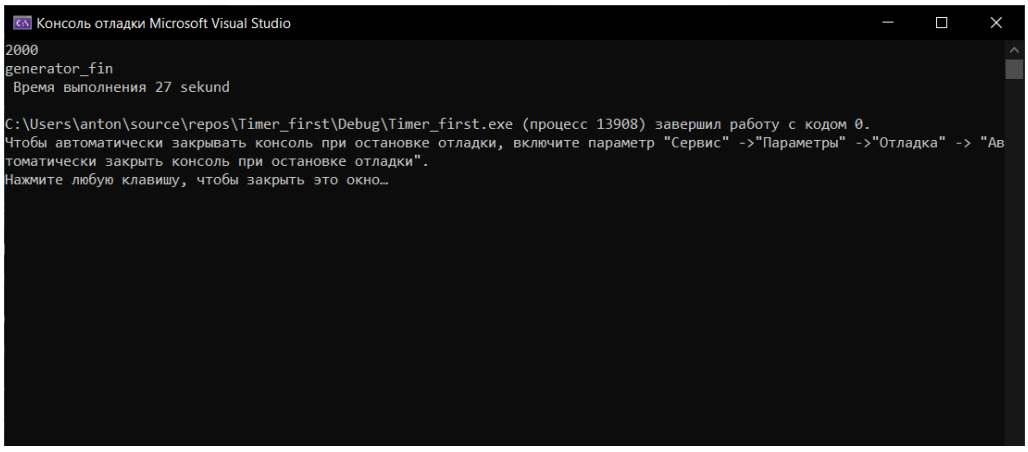

Рисунок 3.6 – Приклад результату роботи алгоритму

Для виконання розпаралелювання алгоритму використовувався процесор Intel® Core™ i5-9300H CPU @ 2.40 GHz, який має 4 ядра 8 потоків, та підтримує типи пам’яті DDR4-2666, LPDDR3-2133.

Таблиця 3.2 - Результати роботи паралельного алгоритму

Розмір вхідних даних(речення)	100	500	1000	1500	2000
Час роботи (секунди)	1	7	12	19	27

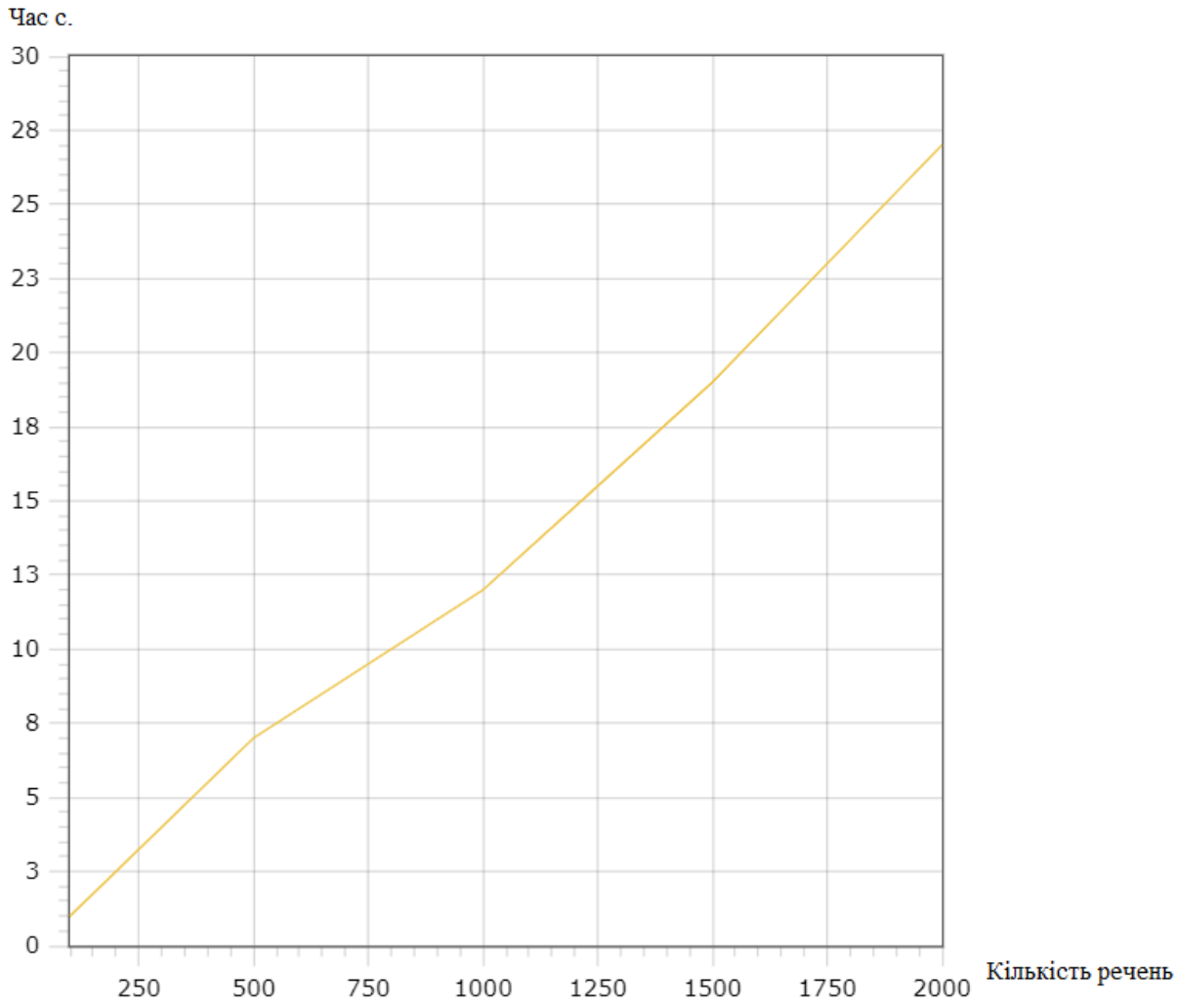

Рисунок 3.7– Залежність часу роботи від розміру вхідних даних

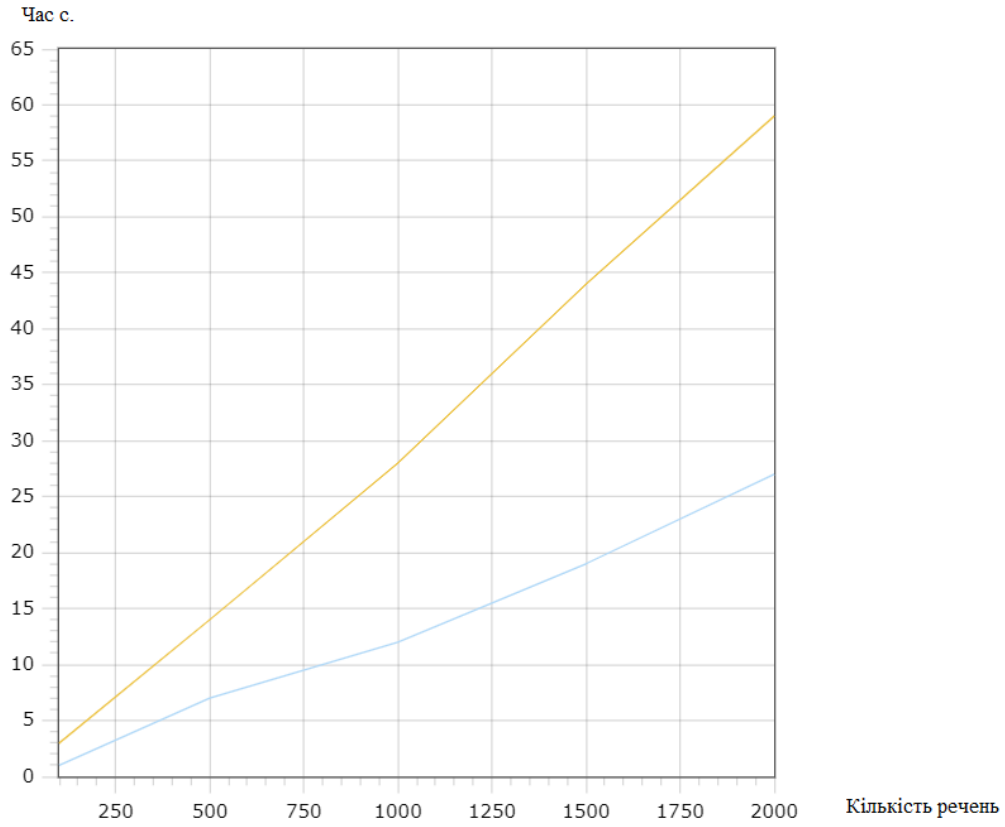

Рисунок 3.8. – Порівняння графіків послідовного та паралельного алгоритму

Як можемо бачити, паралельний метод дає прискорення до 3 разів. Так як було використано 4 паралельних потоки, ми можемо прийти до висновку, що паралельний алгоритм справляється із поставленими завданнями.

3.2.2.2 Паралельний алгоритм на GPU

Для цього алгоритму будемо використовувати CUDA. Так як не зрозуміло як саме загрузити данні замість паралелізму усього алгоритму, будемо лише паралелити тільки метод н-грам. Для кожного зі слів, запишемо всі можливі варіанти(яких 100), загрузимо їх в CUDA ядра, паралельно виконаємо алгоритм, та повернемо результат. Так як у нас буде досить довга підготовка данних, та їх копіювання, а сам алгоритм займає не так багато часу, то результат буде не дуже хороший. Алгоритм буде мати вигляд, наведений на рисунку 3.9.

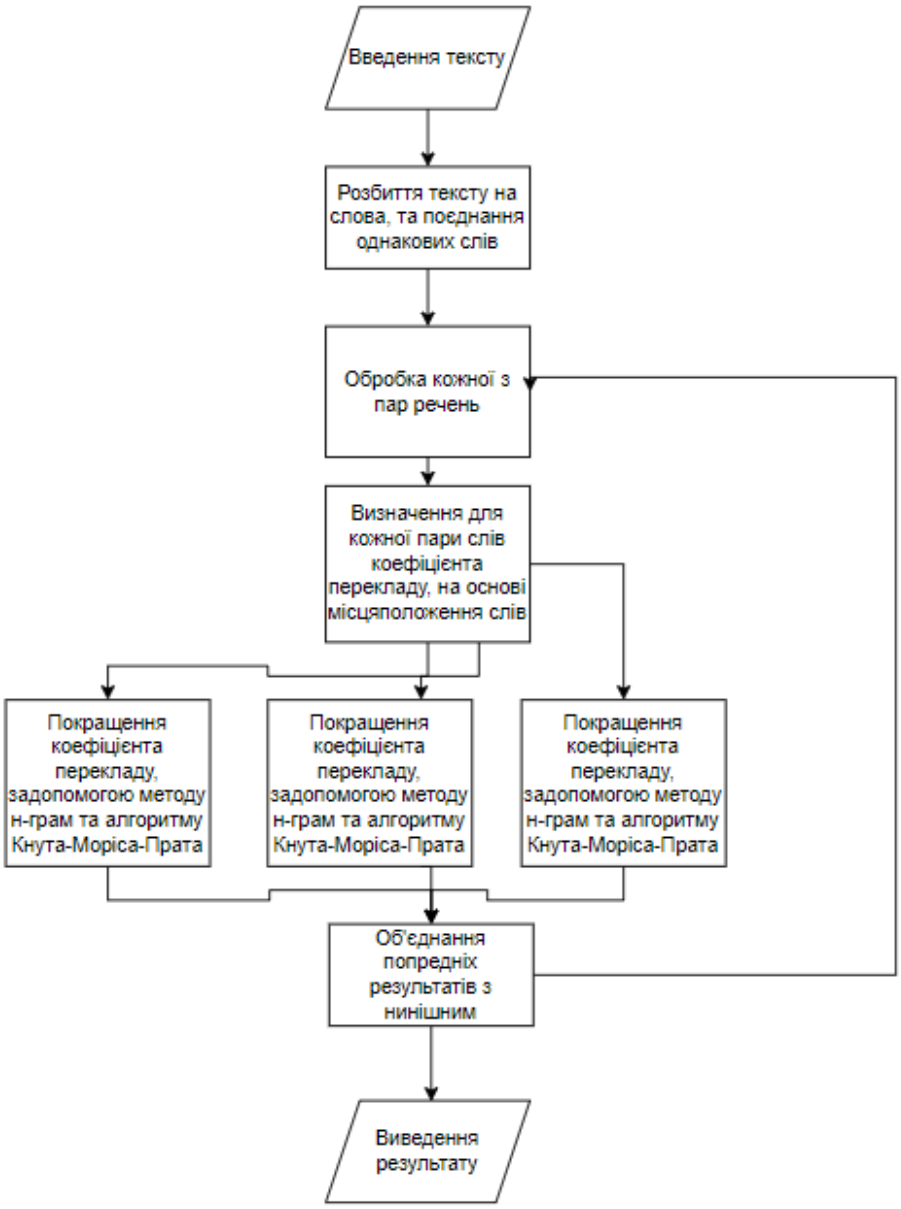

Рисунок 3.9 – Паралельний на GPU алгоритм

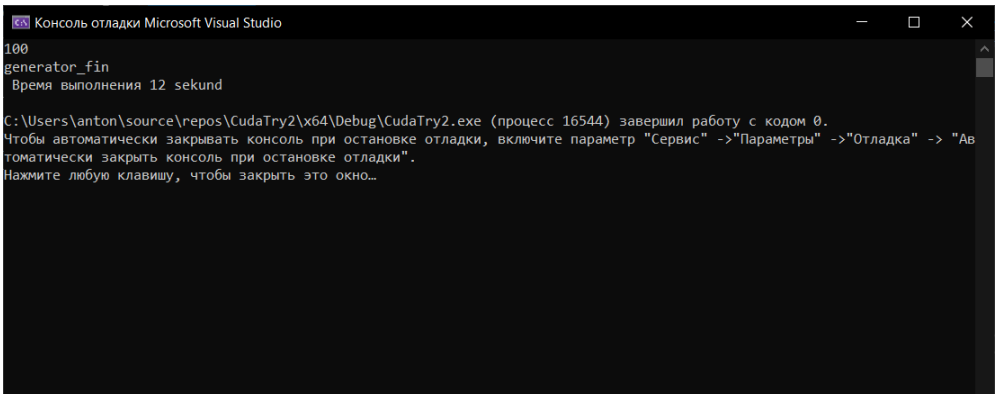

Рисунок 3.10 – Приклад роботи алгоритму

Для проведення цього експерименту використовувалася наступна відеокартка:

- відеокарта: Nvidia geforce gtx 1660 ti;
- сімейство ГПУ:TU116;
- архітектура ГПУ:Turing;
- кількість TPC: 3;
- кількість SM:24;
- кількість SP:(ядер CUDA)1536;
- об'єм пам'яті:6144MiB;
- розмір варпа:32.

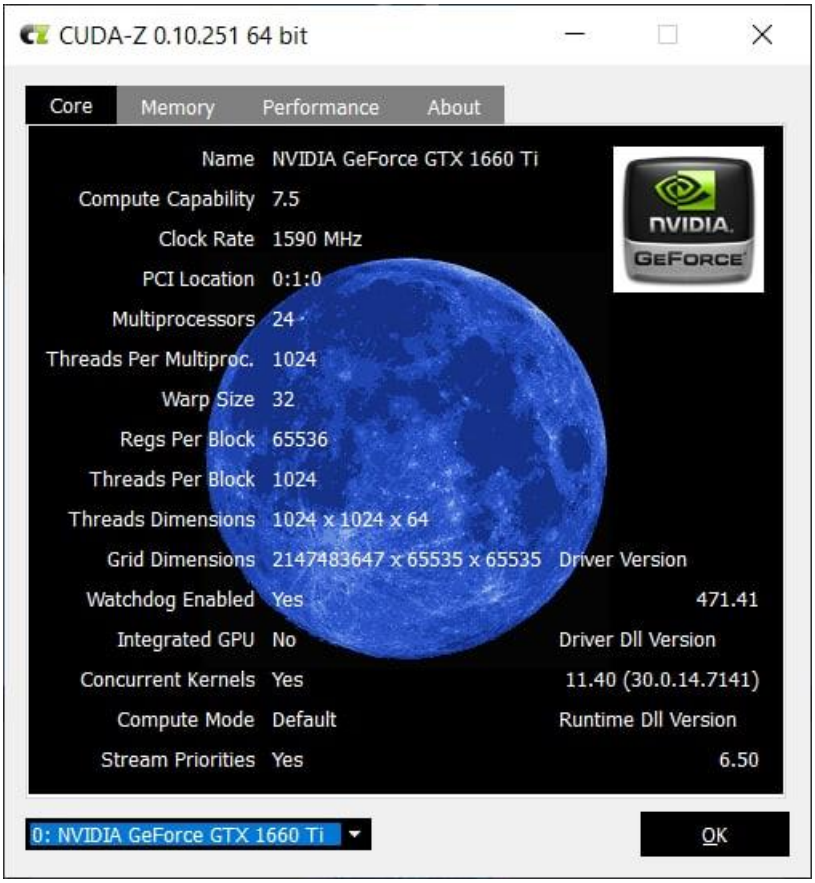

Рисунок 3.11 – Данні про відеократу з програми CUDA-Z

Рисунок 3.12 – Данні про відеократу з програми CUDA-Z

В результаті роботи отримаємо наступні результати:

Таблиця 3.3 - Результати роботи паралельного алгоритму

Розмір вхідних даних(речення)	100	500	1000	1500	2000
Час роботи (секунди)	12	22	32	45	55

Для того щоб виконати розпараелювання, використовувалася Cuda 11.5 Runtime, яка допомагає задіяти cuda ядра для використання у якості паралельного обчислювача.

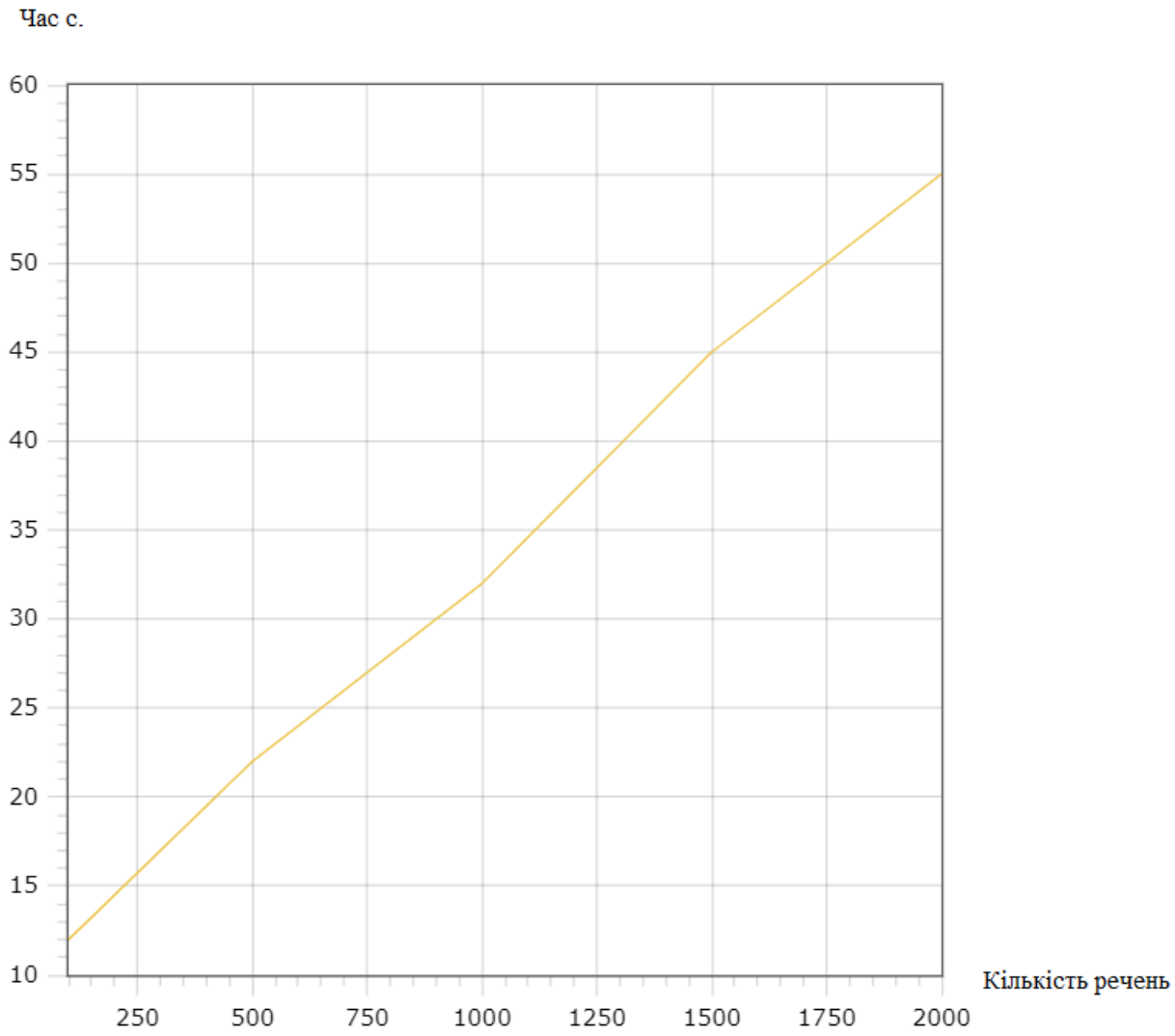

Рисунок 3.13– Залежність часу роботи від розміру вхідних даних

Як бачимо паралельний алгоритм на основі GPU дає незначне покращення на малому розмірі даних, яке поступово зникає, що означає те, що алгоритм не потрібно використовувати.

3.2.2.3 Порівняння алгоритмів

Для порівняння алгоритмів зведемо результати роботи в спільну таблицю.

Таблиця 3.4 – Порівняння алгоритмів

Розмір вхідних даних	100	500	1000	1500	2000
Послідовний алгоритм	3	14	28	44	59
Паралельний на CPU	1	7	12	19	27
Праралельний на GPU	12	22	32	45	55

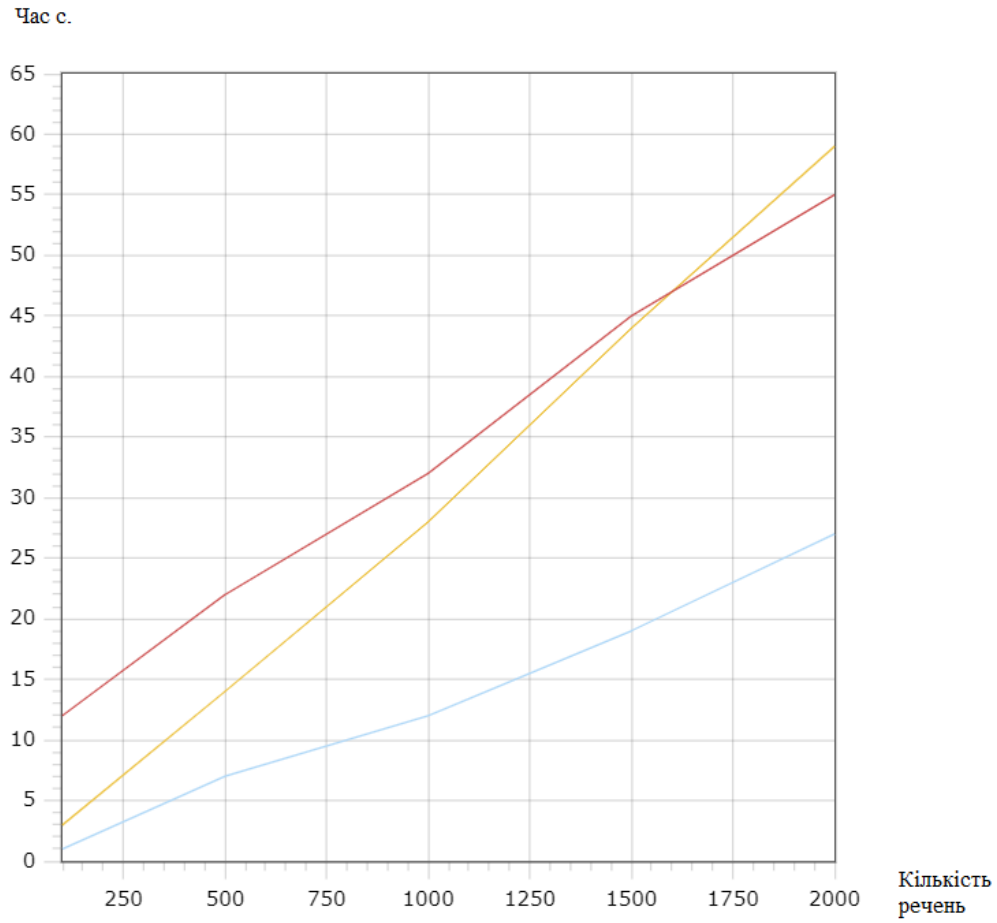

Рисунок 3.14 – Порівняння часу роботи усіх алгоритмів

На рисунку 3.14 ми бачимо порівняння усіх методів. Нижня лінія – паралельний алгоритм на CPU. Середня – паралельний алгоритм на GPU. Верхня – звичайний алгоритм.

За допомогою графіка ми можемо бачити, що при великій кількості вхідних даних, паралельний алгоритм на GPU втрачає свою перевагу.

Взагалі ми можемо прийти до висновку, що потрібно використовувати метод паралелізації на CPU, який дає прискорення порівняне з кількістю використаних ядер.

Метод паралелізації на GPU не дає бажаного прискорення, так як за структурою програми ми можемо розпаралелити тільки одну її частину, що не дає портібного результату.

3.3 Перевірка якості побудови словників за допомогою інших словників

Для того щоб перевірити можливість застосування алгоритму аугментації тексту на основі методу зворотнього словника, первіримо його на таких тестових прикладах:

- ідеальний варіант(абсолютна точність словника);
- звичайний варіант(середнья точність словника);
- найгірший варіант(низька точність словника);
- словник на основі уже згенерованого словника.

3.3.1 Побудова словників за допомогою словників з абсолютною точністю

Для перевірки дієвості методу використає не згенеровані програмою словники, а звичайні словники, та запустимо метод на ньому. Для цього прикладу використаємо українсько-російкий та російсько-англійський словники, для генерування українсько -англійського словника.

```
бавовна : хлопок 0.81667 |
батьківщина : родина 0.85909 |
батько : отец 0.825 |
брат : брат 0.95 |
затишок : уют 0.83333 |
мати : мать 0.9 |
навчання : учеба 0.82 |
обручка : кольцо 0.81667 |
перемога : победа 0.81667 |
печатка : печать 0.92619 |
ранок : утро 0.825 |
сім’я : семья 0.82 |
сестра : сестра 0.96667 |
сонце : солнце 0.87 |
шафа : шкаф 0.85 |
школа : школа 0.96 |
```

Рисунок 3.15 – Використаний українсько -російський словник

```
брат : brother 0.8 |
кольцо : ring 0.8 |
мать : mother 0.8 |
отец : father 0.8 |
печать : seal 0.8 |
победа : victory 0.8 |
родина : motherland 0.8 |
семья : family 0.8 |
сестра : sister 0.8 |
солнце : sun 0.8 |
утро : morning 0.8 |
учеба : studies 0.8 |
уют : cosiness 0.8 |
хлопок : clap 0.8 cotton 0.4 |
шкаф : closet 0.8 |
школа : school 0.8 |
```

Рисунок 3.16 – Використаний російсько-англійський словник

У результаті роботи отримали наступний словник:

```
бавовна : clap 0.8 cotton 0.4 |
батьківщина : motherland 0.8 |
батько : father 0.8 |
брат : brother 0.8 |
затишок : cosiness 0.8 |
мати : mother 0.8 |
навчання : studies 0.8 |
обручка : ring 0.8 |
перемога : victory 0.8 |
печатка : seal 0.8 |
ранок : morning 0.8 |
сім’я : family 0.8 |
сестра : sister 0.8 |
сонце : sun 0.8 |
шафа : closet 0.8 |
школа : school 0.8 |
```

Рисунок 3.17 – Отриманий словник

При аналізі отриманого словника ми дізналися, що для кожного зі слів було запропоновано потрібний варіант, але не завжди потрібний варіант був найкращим. Це пояснюється тим, що всі переклади були завчасно введені, та мали однакову вірогідність перекладу(так як вони дійсні переклади), але так як ми не розподіляли слова які мають різний наголос або омоніми, для них не завжди вибирався головний вірний переклад.

Прикладом такого виключення може бути слово “бавовна”. Російський переклад – “хлопок” має омонім. Одне зі значень “cotton”, яке і є вірним в даному прикладі, інше “clap” яке і було використано.

Для подальшого тестування такі слова були видалені.

3.3.2 Побудова словників за допомогою словників з високою точністю

Для наступного експеременту використаємо генерування словника за допомогою програми описаної в пункті 3.2. Для того щоб це виконати виберемо один і той самий текст на трьох мовах, українській, російській та англійській. Проведемо генерацію українсько-російкого словника, російсько-англійського, проведемо за їх допомогою генерацію українсько-англійського словника. Після цього проведемо генерацію українсько-англійського словника та порівняємо їх якість.

```
Світ такий, яким він є.
Важко припустити, щоб світ був створений єдино для задоволення наших потреб.
Це було б дивом з див.
Світ нейтральний.
Він не ласкавий і не ворожий людині.
Вам вселили, що людина народжується для того, щоб померти, і що ви повинні все життя мучитися цією думкою.
Навіщо?
Смерть - не факт свідомості.
"Сенс роздумів про смерть в тому, що вони позбавлені сенсу", - писав Монтерлан.
Смерть близьких приголомшує нас.
А наша власна?
Боятися її - значить уявляти собі і світ, де ми є, і світ, де нас немає.
Ці два образи несумісні.
Вам вселили, що ми живемо на краю прірви, але навіть якщо ми йдемо по краю прірви, ніщо не штовхає нас вниз.
Вам вселили, що старі моральні цінності відійшли в минуле.
Це брехня.
Я нагадаю вам для початку кілька старих як світ істин.
```

Рисунок 3.18 – Вхідний український текст

```
Мир таков, каков он есть.
Трудно предположить, чтобы мир был создан единственно для удовлетворения наших потребностей.
Это было бы чудом из чудес.
Мир нейтрален.
Он не дружествен и не враждебен человеку.
Вам внушили, что человек рождается для того, чтобы умереть, и что вы должны всю жизнь терзаться этой мыслью.
Чего ради?
Смерть - не факт сознания.
"Смысл раздумий о смерти в том, что они лишены смысла", - писал Монтерлан.
Смерть близких потрясает нас.
А наша собственная?
Бояться ее - значит представлять себе и мир, где мы есть, и мир, где нас нет.
Эти два образа несовместимы.
Вам внушили, что мы живем на краю пропасти, но даже если мы идем по краю пропасти, ничто не толкает нас вниз.
Вам внушили, что старые моральные ценности канули в прошлое.
Это ложь.
Я напомню вам для начала несколько древних как мир истин.
```

Рисунок 3.19 – Вхідний російський текст

```
The world is what it is.
It is difficult to imagine that the world was created solely to meet our needs.
It would be a miracle of miracles.
The world is neutral.
It is neither friendly nor hostile to man.
You have been taught that a person is born to die, and that you should be tormented by this thought all your life.
What for?
Death is not a fact of consciousness.
"The meaning of thinking about death is that they are meaningless," Monterlant wrote.
The death of loved ones shocks us.
What about our own?
To be afraid of it means to imagine both the world where we are and the world where we are not.
These two images are incompatible.
You have been told that we live on the edge of an abyss, but even if we walk along the edge of the abyss, nothing pushes us down.
You have been told that the old moral values have sunk into the past.
It's a lie.
I will remind you to begin with a few ancient as the world of truths.
```

Рисунок 3.20 – Вхідний англійський текст

Для того щоб у кожного зі слів був переклад, та для того щоб мати змогу коректно оцінити якість зегенрованого словника, було використано один і той самий текст, але різними мовами(рисунки 3.18, 3.19, 3.20).

```
а : а 0.9 наша 0.55833 власна 0.28333  |
близких : близьких 0.89286 приголомшує 0.60909 смерть 0.6 нас 0.4  |
бояться : боятися 0.88571 її 0.75 значить 0.67857 уявляти 0.62857  |
бы : б 0.9 було 0.69167 дивом 0.66667 це 0.53333 з 0.53333  |
был : був 0.83333 світ 0.72727 створений 0.72727 щоб 0.68788 єдино 0.65455  |
было : було 0.85 б 0.76667 це 0.66667 дивом 0.55333 з 0.4  |
в : в 0.9 що 0.51624 смерть 0.36923 тому 0.36923 про 0.33846  |
вам : вам 0.93333 вселили 0.57143 що 0.51429 для 0.32444 ми 0.22143  |
вам_внушили : вам 0.90909 вселили 0.79481 що 0.68571 ми 0.27857 на 0.25317  |
вниз : вниз 0.95 нас 0.79524 штовхає 0.7381 не 0.73571 ніщо 0.67262  |
внушили : вселили 0.85714 вам 0.77619 що 0.74286 ми 0.29127 на 0.26587  |
внушили_что : вселили 0.85195 що 0.79286 вам 0.77619 ми 0.29127 на 0.26587  |
враждебен : ворожий 0.81429 людині 0.70238 і 0.57143 ласкавий 0.46964 не 0.39286  |
всю : все 0.86667 повинні 0.76984 ви 0.76111 життя 0.75556 мучитися 0.72361  |
вы : ви 0.85 повинні 0.76984 все 0.74444 і 0.71111 померти 0.66667  |
где : де 0.775 світ 0.625 і 0.575 собі 0.525 уявляти 0.475  |
даже : навіть 0.81667 але 0.79524 якщо 0.7619 прірви 0.72381 краю 0.71071  |
два : два 0.93333 образи 0.61667 ці 0.6 несумісні 0.4  |
для : для 0.93333 вам 0.43259 щоб 0.38249 світ 0.30303 я 0.28667  |
должны : повинні 0.81429 ви 0.75556 все 0.75556 життя 0.73111 і 0.66667  |
древних : старих 0.83333 кілька 0.72 як 0.72 світ 0.665 початку 0.64  |
дружествен : ласкавий 0.8125 не 0.73571 і 0.68571 він 0.60476 ворожий 0.47143  |
единственно : єдино 0.86909 створений 0.7697 для 0.76061 задоволення 0.69091 був 0.68788  |
ее : її 0.8 боятися 0.75 значить 0.7 уявляти 0.65  |
если : якщо 0.8 навіть 0.7619 але 0.75714 йдемо 0.74381 прірви 0.70238  |
есть : є 0.8 світ 0.43 він 0.32 яким 0.24 такий 0.18  |
живем : живемо 0.94333 ми 0.8119 на 0.7619 що 0.72381 краю 0.72381  |
жизнь : життя 0.84 мучитися 0.76806 все 0.75556 повинні 0.7254 ви 0.71667  |
значит : значить 0.95238 її 0.7 боятися 0.66429 уявляти 0.76429  |
и : і 0.7375 ми 0.3875 світ 0.3375 де 0.3375 собі 0.3125  |
идем : йдемо 0.885 по 0.7619 якщо 0.72381 навіть 0.68571 але 0.68095  |
из : з 0.9 див 0.7 дивом 0.68667 б 0.53333 було 0.4  |
истин : істин 0.92 світ 0.745 як 0.64 старих 0.59333 кілька 0.48  |
как : як 0.85 старих 0.73667 світ 0.72 кілька 0.67333 істин 0.64  |
```

Рисунок 3.21 – Отриманий українсько-російський словник

```
близких : loved 0.8 the 0.68571 of 0.68571 death 0.57143 ones 0.45714  |
бояться : afraid 0.8 be 0.7619 to 0.72381 of 0.68571 it 0.64762  |
бы : be 0.8 would 0.68571 a 0.68571 it 0.57143 miracle 0.57143  |
был : was 0.8 to 0.74667 that 0.74667 difficult 0.69333 the 0.69333  |
было : would 0.8 it 0.68571 be 0.68571 a 0.57143 miracle 0.45714  |
в : in 0.62198 the 0.61978 about 0.4 moral 0.4 old 0.37143  |
вам : you 0.78667 have 0.56988 been 0.53975 that 0.4795 the 0.35143  |
вам_внушили : you 0.8 have 0.75983 been 0.71967 that 0.63934 told 0.44762  |
вниз : along 0.74286 walk 0.71429 nothing 0.68571 if 0.65714 pushes 0.65714  |
внушили : have 0.8 you 0.75983 been 0.75983 that 0.6795 told 0.47619  |
внушили_что : suggest 0.8 you 0.75983 been 0.75983 that 0.6795 told 0.47619  |
враждебен : hostile 0.8 nor 0.7 to 0.7 friendly 0.6 man 0.6  |
всю : all 0.76522 and 0.73043 be 0.73043 die 0.69565 tormented 0.69565  |
вы : you 0.8 die 0.76522 to 0.73043 born 0.69565 should 0.69565  |
где : where 0.70476 world 0.70476 both 0.70476 are 0.70476 the 0.70476  |
даже : even 0.8 the 0.77143 of 0.77143 on 0.74286 an 0.74286  |
два : two 0.8 these 0.64 images 0.64 are 0.48 incompatible 0.32  |
для : for 0.66473 that 0.48618 you 0.47536 a 0.46222 is 0.40348  |
должны : should 0.76522 die 0.73043 and 0.73043 to 0.69565 be 0.69565  |
древних : ancient 0.8 begin 0.74667 a 0.74667 to 0.69333 few 0.69333  |
дружествен : friendly 0.8 is 0.7 neither 0.7 it 0.6 nor 0.6  |
единственно : only 0.8 that 0.74667 world 0.74667 imagine 0.69333 was 0.69333  |
ее : her 0.8 to 0.7619 afraid 0.7619 of 0.72381 it 0.68571  |
если : if 0.8 edge 0.77143 an 0.77143 the 0.74286 abyss 0.74286  |
есть : it 0.68571 world 0.6 the 0.51429 where 0.38095 both 0.3619  |
живем : live 0.8 told 0.77143 we 0.77143 been 0.74286 that 0.74286  |
жизнь : life 0.8 be 0.76522 tormented 0.73043 and 0.69565 by 0.69565  |
значит : means 0.8 afraid 0.7619 it 0.7619 be 0.72381 of 0.72381  |
и : to 0.5631 and 0.47785 it 0.43929 be 0.40497 the 0.35238  |
идем : go 0.8 an 0.77143 but 0.77143 of 0.74286 even 0.74286  |
из : from 0.8 a 0.68571 of 0.68571 be 0.57143 miracles 0.57143  |
```

Рисунок 3.22 – Отриманий російсько-англійський словник

Вхідні словники мали точність 87%(словник представлений на рисунку

3.22) та 92% (словник представлений на рисунку 3.21) слів як основний переклад, та 95% та 100% мали як один із можливих перекладів.

```
він : it 0.73333 is 0.68333 what 0.4 neither 0.3 world 0.26667  |
важко : it 0.8 is 0.74667 difficult 0.69333 to 0.64 imagine 0.58667  |
вам : you 0.78667 have 0.56988 been 0.53975 that 0.4795 the 0.35143  |
вам_вселили : you 0.8 have 0.75983 been 0.71967 that 0.63934 told 0.44762  |
ви : you 0.8 die 0.76522 to 0.73043 born 0.69565 should 0.69565  |
власна : our 0.8 about 0.6 own 0.6 what 0.4  |
вниз : along 0.74286 walk 0.71429 nothing 0.68571 if 0.65714 pushes 0.65714  |
вони : that 0.8 is 0.73846 they 0.73846 death 0.67692 are 0.67692  |
ворожий : hostile 0.8 nor 0.7 to 0.7 friendly 0.6 man 0.6  |
все : should 0.76522 and 0.73043 be 0.73043 die 0.69565 tormented 0.69565  |
вселили : have 0.8 you 0.75983 been 0.75983 that 0.6795 told 0.47619  |
вселили_що : have 0.8 you 0.75983 been 0.75983 that 0.6795 told 0.47619  |
два : two 0.8 these 0.64 images 0.64 are 0.48 incompatible 0.32  |
де : world 0.70476 where 0.70476 both 0.70476 are 0.70476 the 0.70476  |
див : of 0.8 miracle 0.68571 miracles 0.68571 a 0.57143 be 0.45714  |
дивом : a 0.8 be 0.68571 miracle 0.68571 would 0.57143 of 0.57143  |
для : to 0.66473 that 0.48618 you 0.47536 a 0.46222 is 0.40348  |
думкою : by 0.8 tormented 0.76522 this 0.76522 be 0.73043 thought 0.73043  |
живемо : live 0.8 told 0.77143 we 0.77143 been 0.74286 that 0.74286  |
життя : should 0.8 be 0.76522 tormented 0.73043 and 0.69565 by 0.69565  |
з : whis 0.8 a 0.68571 of 0.68571 be 0.57143 miracles 0.57143  |
задоволення : was 0.8 world 0.74667 created 0.74667 the 0.69333 solely 0.69333  |
значить : means 0.8 afraid 0.7619 it 0.7619 be 0.72381 of 0.72381  |
```

Рисунок 3.23 – Еталонний українсько-англійський словник

```
а : and 0.576 about 0.35733 our 0.17649  |
але : the 0.512 even 0.50362 if 0.47924 an 0.42481 edge 0.41077  |
б : be 0.544 would 0.44267 a 0.42667 from 0.34133 a 0.2113  |
близьких : loved 0.57257 of 0.384 death 0.36648 of 0.17168  |
боятися : afraid 0.56685 her 0.48 means 0.42667 it 0.39467  |
брехня : lie 0.512 a 0.15848  |
був : was 0.53333 that 0.46545 only 0.42473 is 0.41732 world 0.41595  |
було : would 0.544 be 0.45867 a 0.35413 a 0.27733 from 0.256  |
в : in 0.44783 that 0.28817 thinking 0.23631 death 0.23631 of 0.21661  |
відійшли : old 0.53333 the 0.46311 values 0.41651 that 0.40533 in 0.4036  |
він : it 0.49866 is 0.2176 friendly 0.18925 it 0.17554 is 0.15602  |
важко : it 0.52267 is 0.47079 is 0.41732 world 0.33276 was 0.32582  |
вам : you 0.58738 have 0.36572 that 0.28708 for 0.17253 the 0.12822  |
вам_вселили : you 0.57212 have 0.50868 that 0.38276 we 0.16203 the 0.16131  |
ви : you 0.544 should 0.46254 all 0.45573 life 0.43947 to 0.36539  |
власна : our 0.52364 about 0.37333 and 0.23467  |
вниз : along 0.56457 along 0.46324 if 0.42436 of 0.34132 is 0.29218  |
вони : that 0.592 meaningless 0.48328 death 0.45457 that 0.43081 in 0.35596  |
ворожий : hostile 0.51911 to 0.44685 to 0.30247 friendly 0.29897 is 0.13616  |
```

Рисунок 3.24 – Результуючий словник

Еталонний словник(рисунок 3.23) мав точність 90% та 100%

відповідно. В результаті після генерації ми отримали словник(рисунок 3.24) з точністю 80% основного варіанта слова та 89% мали можливий переклад серед запропонованих п’яти перекладів..

З цих результатів ми можемо зробити висновок, що після такої генерації ми можемо отримати словник яким можливо користуватися, але його ефективність на порядок нища.

3.3.3 Побудова словників за допомогою словників з низькою точністю

Для перевірки працездатності застосунку використаємо словники з якістю менше ніж 50%. Для цього візьмемо один і той самий текст(рисунки 3.25, 3.26 та 3.27) на українській, російській та мові есперанто, та виконаємо дії аналогічні тим, які були виконані в пункті 4.2. Так як мова епереанто не схожа за структурою з іншими, то якість словника буде досить поганою. Приведемо тексти які використовувалися для генерації словника.

```
Автор есперанто, Лазар Людовік Заменгоф, в 1878 році будучи в останньому класі гімназії закінчив перший проект мову.
І навчаючись в університеті, Заменгоф протягом повних семи років до 1885 допрацьовував свою мову.
Нарешті, в 1887 році у Варшаві з'явився перший підручник під заголовком "Міжнародна мова Доктора Есперанто".
Автор видав підручник не під своїм ім'ям, а під псевдонімом "Д-р Есперанто", тобто "Той хто сподівається".
Тепер і сама мова названа цим псевдонімом.
У 1913 році під керівництвом відомого французького письменника Трістана Бернарда есперанто проекзаменували у спеціальній комісії.
Певний літературний текст було перекладено багатьма мовами, між іншим також і есперанто, а потім той же текст знову було перекладено іншим перекладачем, який н
Результат показав, що найдосконаліший переклад був зроблений за допомогою есперанто, тому що текст, переведений назад з есперанто, був найближчим до оригіналу.
Перший загальний конгрес есперанто був організований у Болоньї (Франція) у 1905 році.
```

Рисунок 3.25 – Український текст

```
Автор эсперанто, Лазарь Людовик Заменгоф, в 1878 будучи в последнем классе гимназии закончил первый проект язык.
И учась в университете, Заменгоф на протяжении полных семи лет до 1885 года дорабатывал свой язык.
Наконец, в 1887 в Варшаве появился первый учебник под заголовком "Международный язык Доктора Эсперанто".
Автор издал учебник не под своим имененем, а под псевдонимом "Д-р Эсперанто", т_е "Тот кто надеется".
Теперь и сам язык назван этим псевдонимом.
В 1913 году под руководством известного французского писателя Тристана Бернард эсперанто проэкзаменовали в специальной комиссии.
Определённый литературный текст был переведён на многие языки, между прочим также и на эсперанто, а затем тот же текст снова был переведён другим переводчиком, незнающим
Результат показал, что самый совершенный перевод был сделан с помощью эсперанто, потому что текст, переведённый обратно с эсперанто был наиболее близок к оригиналу.
Первый всеобщий конгресс эсперанто был организован в Болонье (Франция) в 1905 году.
```

Рисунок 3.26 – Російський текст

```
La autoro de esperanto Lazary Ludovik Zamenhof en 1878 esan nel laste klaso de gimnazio endi et une projekto del lingo.
Trastudan nel universiteto, Zamenhof dum plene sep yaros ad la yaro 1885 prosesari e sue lingo.
Enden, nel yaro 1887 en Varshava aperi la une manuelo sub la titulo "Internacia Lingvo de Doktoro Esperanto".
La autoro no eldoni e manuelo sub sue namo, bat sub la pseudonimo "D-ro Esperanto", s_e "Kvele espera".
Prezenten la lingo same es namate segun eze pseudonimo.
Nel yaro 1913 sub la gvido de populare franse skritero Tristan Bernard os exzamini et esperanto en spesiale comiseo.
Definite literature texto esi tradukite al multe lingos, inter otren gam al esperanto, ve posten la same texto esi retradukite per otre tradukistos, inkonante et originale texto,
La rezulto montri, ke la plu perfekte tradukos esari farite per helpo de esperanto, kar la retradukite texto from esperanto esi la plu proxime al originalo.
La une universale kongreso de esperanto esari organizate en Bolongne-sur-Mer (Fransia) nel yaro 1905.
```

Рисунок 3.27 – Текст на мові есперанто

Після генерації отримаємо наступні словники.

```
1878 : 1878 0.87381 zamenhof 0.8 ludovik 0.7619 en 0.7619 lazary 0.72381  |
1885 : 1885 0.9 yaro 0.8 la 0.75 ad 0.7 prosesari 0.7  |
1887 : 1887 0.90556 yaro 0.8 nel 0.75556 enden 0.71111 en 0.71111  |
1905 : fransia 0.8 1905 0.77857 bolongnesurmer 0.74286 nel 0.74286 en 0.68571  |
1913 : 1913 0.90789 yaro 0.8 nel 0.75789 sub 0.71579 la 0.67368  |
і : la 0.60699 same 0.42246 lingo 0.36553 trastudan 0.26667 gam 0.26667  |
ім'ям : sub 0.8 manuelo 0.75556 sue 0.75556 e 0.71111 namo 0.71111  |
іншим : otren 0.64516 gam 0.64516 esperanto 0.64516 ve 0.64516 la 0.64516  |
а : esperanto 0.64444 la 0.60573 sue 0.4 ve 0.3871 sub 0.37778  |
автор : la 0.8 autoro 0.75873 esperanto 0.43175 de 0.3619 no 0.35556  |
багатьма : al 0.8 tradukite 0.77419 multe 0.77419 esi 0.74839 lingos 0.74839  |
бернарда : skritero 0.8 franse 0.75789 tristan 0.75789 populare 0.71579 bernard 0.71579  |
болоньї : organizate 0.8 esari 0.74286 en 0.74286 esperanto 0.68571 bolongnesurmer 0.68571  |
був : de 0.67692 esperanto 0.65788 esari 0.63883 la 0.49817 per 0.41026  |
будучи : 1878 0.8 en 0.7619 esan 0.7619 zamenhof 0.72381 nel 0.72381  |
було : esi 0.60645 tradukite 0.60645 al 0.60645 multe 0.60645 lingos 0.60645  |
в : nel 0.73036 zamenhof 0.5494 en 0.52857 la 0.5125 1878 0.3619  |
відомого : gvido 0.8 la 0.75789 de 0.75789 sub 0.71579 populare 0.71579  |
варшаві : varshava 0.8 en 0.75556 aperi 0.75556 1887 0.71111 la 0.71111  |
видав : autoro 0.8 la 0.75556 no 0.75556 eldoni 0.71111 e 0.66667  |
гімназії : klaso 0.8 laste 0.7619 nel 0.72381 gimnazio 0.72381 esan 0.68571  |
до : la 0.50769 from 0.38462 esi 0.38462 ad 0.375 yaro 0.375  |
доктора : internacia 0.8 titulo 0.75556 lingvo 0.75556 de 0.71111 sub 0.66667  |
допомогою : esari 0.8 tradukos 0.76923 farite 0.76923 perfekte 0.73846 per 0.73846  |
допрацьовував : 1885 0.8 yaro 0.75 prosesari 0.75 la 0.7 e 0.7  |
др : bat 0.75556 namo 0.71111 pseudonimo 0.71111 sue 0.66667 dro 0.66667  |
есперанто : esperanto 0.68639 de 0.53456 la 0.53317 en 0.24858 retradukite 0.24727  |
же : la 0.8 posten 0.77419 same 0.77419 ve 0.74839 esperanto 0.72258  |
з : kar 0.76923 retradukite 0.76923 esperanto 0.73846 texto 0.73846 de 0.70769  |
з'явився : aperi 0.8 varshava 0.75556 la 0.75556 en 0.71111 une 0.71111  |
за : tradukos 0.8 perfekte 0.76923 esari 0.76923 plu 0.73846 farite 0.73846  |
загальний : une 0.8 la 0.74286 universale 0.74286 kongreso 0.68571 de 0.62857  |
заголовком : sub 0.8 manuelo 0.75556 une 0.71111 titulo 0.71111 la 0.66667  |
закінчив : klaso 0.7619 gimnazio 0.7619 laste 0.72381 endi 0.72381 nel 0.68571  |
заменгоф : zamenhof 0.7369 nel 0.61071 la 0.59881 lazary 0.4 dum 0.4  |
знає : et 0.8 inkonante 0.77419 originale 0.77419 tradukistos 0.74839 otre 0.72258  |
знову : same 0.77419 la 0.74839 retradukite 0.74839 posten 0.72258 per 0.72258  |
зроблений : perfekte 0.8 plu 0.76923 tradukos 0.76923 esari 0.73846 ke 0.70769  |
керівництвом : la 0.8 sub 0.75789 gvido 0.75789 1913 0.71579 de 0.71579  |
класі : laste 0.8 nel 0.7619 klaso 0.7619 esan 0.72381 1878 0.68571  |
комісії : et 0.8 exzamini 0.75789 esperanto 0.75789 os 0.71579 en 0.71579  |
конгрес : universale 0.8 une 0.74286 kongreso 0.74286 la 0.68571 de 0.68571  |
```

Рисунок 3.28 – Словник між українською мовою та мовою есперанто

```
1878 : 1878 0.85 будучи 0.75 последнем 0.75 классе 0.7 в 0.65  |
1885 : 1885 0.95 до 0.75 года 0.75 лет 0.7 дорабатывал 0.7  |
1887 : 1887 0.89286 варшаве 0.74286 в 0.68571 появился 0.68571 наконец 0.62857  |
1905 : 1905 0.75 году 0.66667 франция 0.46667 болонье 0.4 в 0.33333  |
1913 : 1913 0.89667 году 0.8 под 0.74667 в 0.69333 руководством 0.69333  |
ad : семи 0.8 полных 0.75 лет 0.75 протяжении 0.7 до 0.7  |
al : эсперанто 0.51826 текст 0.45768 и 0.45333 также 0.44667 прочим 0.44  |
aperi : первый 0.8 появился 0.74286 учебник 0.74286 варшаве 0.68571 под 0.68571  |
autoro : автор 0.75 эсперанто 0.55 издал 0.4 лазарь 0.375 учебник 0.375  |
bat : псевдонимом 0.8 др 0.75 а 0.7 эсперанто 0.7 имененем 0.65  |
bernard : проэкзаменовали 0.8 эсперанто 0.74667 бернард 0.69333 специальной 0.69333 тристана 0.64  |
bolongnesurmer : франция 0.73333 1905 0.73333 болонье 0.66667 году 0.66667 в 0.6  |
comiseo : комиссии 0.58667 специальной 0.53333 проэкзаменовали 0.42667 эсперанто 0.37333 бернард 0.32  |
de : эсперанто 0.61436 в 0.36611 первый 0.31151 язык 0.2369 был 0.23188  |
de_esperanto : эсперанто 0.73792 был 0.46377 в 0.43889 лазарь 0.26667 первый 0.26111  |
definite : определённый 0.8 литературный 0.77333 текст 0.74667 был 0.72 переведён 0.69333  |
del : язык 0.68667 оригинала 0.4 оригинальный 0.34667 незнающим 0.33333 переводчиком 0.32  |
doktoro : эсперанто 0.62857 доктора 0.57143 язык 0.51429 международный 0.45714 заголовком 0.4  |
dro : тот 0.8 т_е 0.75 кто 0.75 эсперанто 0.7 надеется 0.7  |
dum : заменгоф 0.8 университете 0.75 на 0.75 в 0.7 протяжении 0.7  |
e : дорабатывал 0.4 под 0.4 года 0.375 свой 0.375 не 0.375  |
eldoni : не 0.8 учебник 0.75 под 0.75 издал 0.7 своим 0.7  |
en : в 0.49881 эсперанто 0.4331 первый 0.3631 будучи 0.2 язык 0.2  |
enden : наконец 0.8 в 0.74286 1887 0.68571 варшаве 0.57143 появился 0.51429  |
endi : язык 0.8 проект 0.75 первый 0.7 закончил 0.65 гимназии 0.6  |
es : назван 0.8 язык 0.68571 этим 0.68571 сам 0.57143 псевдонимом 0.57143  |
esan : последнем 0.8 классе 0.75 будучи 0.7 гимназии 0.7 1878 0.65  |
esari : был 0.69855 эсперанто 0.66522 с 0.4 в 0.4 сделан 0.38261  |
esi : текст 0.56773 эсперанто 0.55072 был 0.50435 переведён 0.4 на 0.4  |
espera : надеется 0.7 кто 0.65 тот 0.6 т_е 0.55 эсперанто 0.5  |
esperanto : эсперанто 0.63467 был 0.27471 текст 0.24058 тот 0.18042 в 0.17917  |
et : язык 0.48111 эсперанто 0.38111 комиссии 0.26667 незнающим 0.26667 переводчиком 0.25778  |
exzamini : специальной 0.8 комиссии 0.74667 проэкзаменовали 0.69333 эсперанто 0.64 бернард 0.58667  |
eze : псевдонимом 0.68571 этим 0.57143 назван 0.45714 язык 0.34286 сам 0.22857  |
farite : помощью 0.8 с 0.76522 эсперанто 0.76522 сделан 0.73043 потому 0.73043  |
franse : тристана 0.8 писателя 0.74667 бернард 0.74667 французского 0.69333 эсперанто 0.69333  |
fransia : 1905 0.8 году 0.73333 франция 0.66667 болонье 0.6 в 0.53333  |
from : наиболее 0.76522 близок 0.73043 обратно 0.69565 к 0.69565 переведённый 0.66087  |
gam : также 0.8 прочим 0.77333 и 0.77333 между 0.74667 языки 0.72  |
gimnazio : проект 0.8 первый 0.75 язык 0.75 закончил 0.7 гимназии 0.65  |
gvido : известного 0.8 руководством 0.74667 французского 0.74667 под 0.69333 писателя 0.69333  |
helpo : потому 0.8 эсперанто 0.76522 помощью 0.73043 текст 0.73043 с 0.69565  |
```

Рисунок 3.29 – Словник між мовою есперанто та російською мовою

Після генерації цих словників, використаємо нашу програму для генерації українсько-російського словника. Отримані словники мають точність 43% (рисунок 3.28) та 50% (рисунок 3.29).Крім того порівняємо його зі словником згенерованом на основі цих самих текстів.

```
1878 : 1878 0.95 в 0.75 будучи 0.75 заменгоф 0.7 людовик 0.65  |
1885 : 1885 0.9 до 0.8 лет 0.75 семи 0.7 года 0.7  |
1887 : 1887 0.95 в 0.74286 наконец 0.68571 варшаве 0.68571 появился 0.62857  |
1905 : 1905 0.95 году 0.73333 франция 0.66667 болонье 0.6 в 0.53333  |
1913 : 1913 0.95 в 0.74667 году 0.74667 под 0.69333 руководством 0.64  |
і : и 0.79111 на 0.40556 язык 0.31381 также 0.26667 прочим 0.25778  |
ім'ям : имеменем 0.8125 своим 0.77 а 0.75 под 0.7 псевдонимом 0.65909  |
іншим : и 0.74 прочим 0.67333 снова 0.66 между 0.66 затем 0.66  |
а : а 0.87333 эсперанто 0.69778 тот 0.59667 также 0.39333 и 0.38667  |
автор : автор 0.96 эсперанто 0.52222 издал 0.395 в 0.375 лазарь 0.36667  |
багатьма : на 0.85 многие 0.79 был 0.78 переведён 0.77333 языки 0.74667  |
бернарда : бернард 0.9625 тристана 0.77167 эсперанто 0.76889 писателя 0.70583 проэкзаменовали 0.7  |
болоньї : болонье 0.94286 франция 0.74762 в 0.73333 организован 0.67576 был 0.63333  |
був : был 0.6942 эсперанто 0.63865 потому 0.41087 обратно 0.40849 перевод 0.40849  |
будучи : будучи 0.91667 последнем 0.76111 1878 0.7 классе 0.7 гимназии 0.6625  |
було : был 0.63333 тот 0.63333 между 0.62 многие 0.61667 прочим 0.61667  |
в : в 0.85 заменгоф 0.5 первый 0.39524 будучи 0.35 последнем 0.35  |
відомого : известного 0.8425 французского 0.78417 руководством 0.76333 под 0.72667 писателя 0.69333  |
варшаві : варшаве 0.88571 появился 0.8125 первый 0.75952 учебник 0.68571 в 0.67143  |
видав : издал 0.84 автор 0.79 учебник 0.76429 не 0.7 под 0.68333  |
гімназії : гимназии 0.825 закончил 0.8125 первый 0.75 классе 0.71667 проект 0.7  |
```

Рисунок 3.30 – Згенерований словник звичайним методом

```
1878 : 1878 0.59419 в 0.48762 в 0.48 заменгоф 0.46324 в 0.30403  |
1885 : 1885 0.684 семи 0.448 года 0.448 в 0.36632 эсперанто 0.29385  |
1887 : 1887 0.64683 наконец 0.45511 в 0.40699 в 0.36632 в 0.28377  |
1905 : 1905 0.512 1905 0.46714 франция 0.43581 в 0.40015 в 0.27363  |
1913 : 1913 0.65126 в 0.40825 под 0.38448 в 0.36632 эсперанто 0.26395  |
і : эсперанто 0.23782 язык 0.2163 язык 0.20731 и 0.17067 также 0.17067  |
ім'ям : под 0.45118 под 0.42972 а 0.42667 свой 0.24178 дорабатывал 0.22756  |
іншим : прочим 0.4129 также 0.4129 эсперанто 0.4129 эсперанто 0.32757 эсперанто 0.25277  |
а : эсперанто 0.32721 эсперанто 0.24774 эсперанто 0.23733 под 0.20292 свой 0.128  |
автор : автор 0.45524 эсперанто 0.31344 учебник 0.22756 эсперанто 0.21922 эсперанто 0.17787  |
багатьма : переведён 0.49548 многие 0.49548 языки 0.47897 текст 0.33991 эсперанто 0.33169  |
бернарда : бернард 0.512 тристана 0.48505 эсперанто 0.48505 писателя 0.45811 проэкзаменовали 0.4581
болоньї : болонье 0.512 был 0.41514 франция 0.40228 эсперанто 0.34816 в 0.29644  |
був : был 0.357 эсперанто 0.33403 эсперанто 0.3327 эсперанто 0.23193 эсперанто 0.19518  |
будучи : 1878 0.544 последнем 0.48762 в 0.43429 в 0.38989 в 0.30403  |
було : переведён 0.38813 многие 0.38813 языки 0.38813 текст 0.27544 эсперанто 0.25144  |
в : в 0.39342 в 0.32964 1878 0.24609 в 0.21092 эсперанто 0.2008  |
відомого : известного 0.512 писателя 0.45811 под 0.38448 эсперанто 0.37249 эсперанто 0.29694  |
варшаві : появился 0.512 1887 0.50794 первый 0.48356 в 0.3015 эсперанто 0.27861  |
видав : учебник 0.48356 автор 0.48 не 0.45511 эсперанто 0.29603 дорабатывал 0.21333  |
гімназії : закончил 0.512 гимназии 0.48762 проект 0.46324 последнем 0.43885 в 0.38989  |
до : семи 0.24 наиболее 0.23546 эсперанто 0.19891 текст 0.17469 в 0.17171  |
доктора : эсперанто 0.512 доктора 0.48356 эсперанто 0.44902 под 0.3581 эсперанто 0.3495  |
допомогоч : сделан 0.49231 помощью 0.49231 был 0.47261 был 0.44707 эсперанто 0.41748  |
```

Рисунок 3.31 – Згенерований словник методом аугментації

В результаті генерування методом аугментації був отриманий словник, з якістю близькою до 40%(рисунок 3.31). Найбільшу кількість коректних результатів складають ті слова, які були вірно перекладені коректно в обох словниках. Так як генерувалося на однакових текстах, то відсоток слів які співпадають досить високий. Якщо генерувати словник такої якості на основі різних текстів, то його якість буде близькою до добутку якості цих словників, тобто при наших віхідних даних очікувана якість 25%.

Словник, який було згенеровано звичайним методом (рисунок 3.30) маж точність 92%. Ці результати показують, що при можливості використання звичайного алгоритма, потрібно користуватися ним.

3.3.4 Побудова словників за допомогою побудованих словників

Для побудови словника за допомогою уже побудованого, використаємо уже згенерований в пункті 4.2. словник. Для того щоб його використати, згенеруємо словник між англійською мовою, та мовою есперанто, після чого за допомогою нашої програми згенеруємо словник між українською мовою, та мовою есперанто.

```
La mondo estas tıa, kıa gı estas.
Estas malfacile imagi, ke la mondo estis kreita nur por kontentigi niajn bezonojn.
Estus miraklo vidi
La mondo estas neŭtrala.
Li ne estas afabla kaj malamika al homo.
Oni instruis al vi, ke la homo estas naskita por morti, kaj ke vi devas suferi la tutan ĉi vivon.
Kial?
Morto ne estas fakto de konscio.
"La punkto de pensi pri morto estas ke ili estas sensencaj," Monterlan skribis.
La morto de amatoj mirigas nin.
Kaj la nia?
Timi ĝin signifas imagi kaj la mondon kie ni estas kaj la mondon kie ni ne estas.
Ĉi tiuj du bildoj estas malkongruaj.
Oni diris al vi, ke ni loĝas sur la rando de abismo, sed eĉ se ni marŝas sur la rando de abismo, nenio puŝas nin malsupren.
Oni instruis al vi, ke malnovaj moralaj valoroj estas afero de la pasinteco.
Ĉi tio estas mensogo.
Mi memorigos vin, ke vi komencu per kelkaj veroj same malnovaj kiel la mondo.
```

Рисунок 3.32 – Вихідний текст мовою есперанто

Після генерації отримаємо наступний словник.

```
difficult : imagi 0.82 malfacile 0.74957 ke 0.73846 la 0.72692 estas 0.69692  |
down : de 0.05 oni 0.033333 afero 0.02 moralaj 0.014286 valoroj 0.014286  |
edge : de 0.51154 afero 0.48154 pasinteco 0.47265 la 0.46154 estas 0.42  |
even : pasinteco 0.68803 la 0.61538 de 0.60385 afero 0.51231 estas 0.45077  |
fact : pri 0.8 morto 0.75846 pensi 0.73846 estas 0.69692 de 0.67692  |
few :  |
for : ne 0.8 morto 0.70667 estas 0.66667 fakto 0.55333 de 0.4  |
friendly : vi 0.85 al 0.81 ke 0.81 la 0.77 instruis 0.7325  |
have : estas 0.34308 al 0.29615 vi 0.27564 instruis 0.26667 tio 0.26667  |
have_been : estas 0.34308 al 0.29615 oni 0.27949 instruis 0.27917 vi 0.27564  |
hostile : la 0.85 ke 0.81 homo 0.81 vi 0.77 estas 0.76  |
```

Рисунок 3.33 – Англійсько-есперанто словник

Після результуючого генерування отримали наступний результат.

```
вниз : pasinteco 0.21269 pasinteco 0.19958 pasinteco 0.16376 estas 0.1171 la 0.10422  |
вони : in 0.35596 la 0.2283 estas 0.12362 nin 0.093725 estas 0.089963  |
ворожий : la 0.35299 vi 0.2033 estas 0.12453 estas 0.084292 estas 0.054572  |
все : estas 0.11304 oni 0.075338 kial 0.021222 kial 0.0096712 kial 0.0092508  |
вселили : estas 0.15056 moralaj 0.10363 estas 0.086591 oni 0.07555 kial 0.0033998  |
вселили_що : estas 0.14981 moralaj 0.10363 estas 0.090477 oni 0.07555 kial 0.0033998  |
два : oni 0.38351 al 0.26838 oni 0.24576 al 0.050658  |
де : malkongruaj 0.22784 estas 0.14947 estas 0.083643 estas 0.072186 estas 0.0066906  |
див : from 0.45867 la 0.16024 miraklo 0.11118 la 0.10407 estas 0.062609  |
дивом : from 0.45867 miraklo 0.23278 la 0.15799 la 0.10813 estas 0.10435  |
для : ne 0.31765 estas 0.094839 la 0.05392 oni 0.042106 estas 0.040553  |
думкою : la 0.15361 la 0.14681 estas 0.11131 kial 0.016053 kial 0.0085334  |
живемо : moralaj 0.39607 de 0.16249 la 0.11157 de 0.10953 estas 0.088257  |
життя : la 0.13845 estas 0.12 kial 0.018502 kial 0.010752 kial 0.0089106  |
з : from 0.544 miraklo 0.16677 la 0.13028 la 0.12539 estas 0.083478  |
задоволення : only 0.44218 nur 0.34018 por 0.31883 kontentigi 0.27491 ne 0.25887  |
значить : her 0.448 malkongruaj 0.3965 du 0.29531 estas 0.23709  |
йдемо : go 0.5728 pasinteco 0.31159 pasinteco 0.25036 pasinteco 0.2402 la 0.10596  |
кілька : ne 0.22916 la 0.15988 la 0.13542 la 0.13292 estas 0.13139  |
краю : la 0.24624 pasinteco 0.23695 de 0.17906 la 0.098364 de 0.094808  |
```

Рисунок 3.34 – Результуючий словник

Так як якість словника була досить низька, результуючий словник також має досить низьку якість, на рівні 20%. Цей експеремент показує, що використання таких словників є можливим, але лише на гарних вхідних данних та лише за крайньою необхідністю.

4 ТЕСТУВАННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1 Алгоритм аугментації тексту методом зворотного перекладу

Для того щоб використати метод зворотого перекладу, потрібно для кожного зі слів вихідної мови, знайти переклад його перекладу потрібної нам мови. Тому нам потрібно вибрати алгоритм який допоможе зробити це ефективно. Розглянемо наступні алгоритми:

-алгоритм бінарного пошуку;

-метод двох вказників;

-корневе дерево(бор);

-хешування;

-використання збалансованого дерева;

-алгоритм за допомогою кореневої декомпозиції;

-алгоритм за квадрат.

В кожному з розглянутих алгоритмів проведемо тестування часу на великому обсязі даних, порівняємо їх, знайдемо недоліки та визначемо область застосування у рамках даної задачі.

Для проведення алгоритму експеременту будемо використовувати наступний метод. Згенеруємо потрібну кількість досить довгих слів (8-12 символів) в алфавітому порядку. Потім будемо генерувати таку саму кількість слів, та для кожного з нових слів будемо знаходити його відповідник за домогою вибраного алгоритму.

4.2 Алгоритм бінарного пошуку

Так як наші словники відсортовані, ми можемо досить просто застосувати метод бінарного пошуку. Цей медод допоможе нам досить швидко знайти потрібну інформацію, але не дасть додаткової, яку можна

буде використати пізніше.

Після проведених експерементів отримали наступні результати:

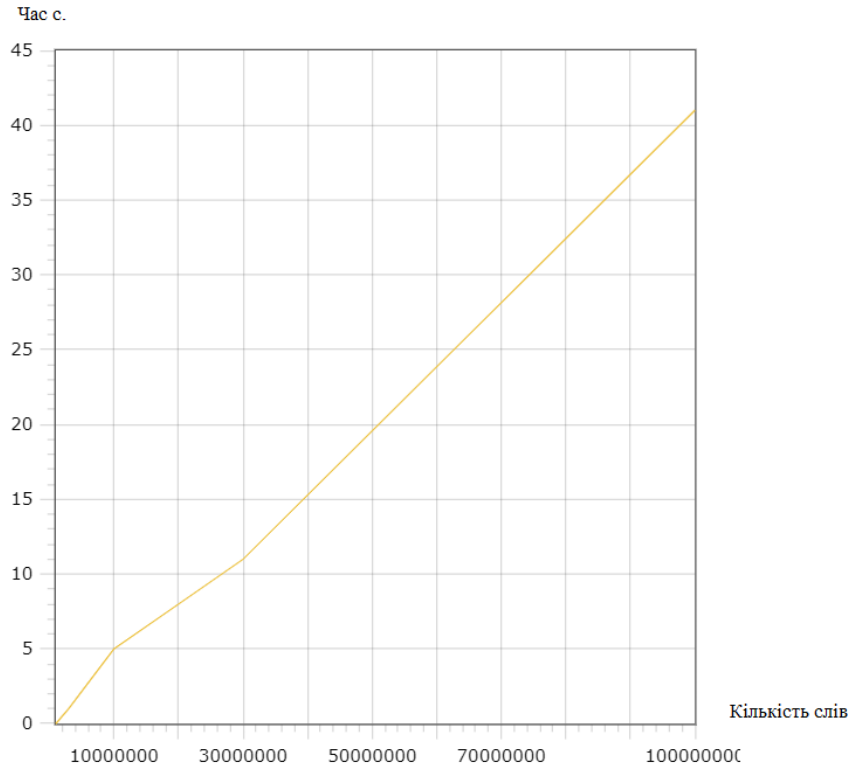

Рисунок 4.1 - Залежність часу роботи від розміру вхідних даних

Таблиця 4.1 – Результати часу роботи алгоритму бінарного пошуку

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	0	1	5	11	41

З даного експеременту ми можемо побачити, що час при великій кількості вхідних даних росте пропорційно, та алгоритм є досить ефективним.

4.3 Метод двох вказівників

Для методу двох вказівників потрібно використати наступну ідею. Для кожної пари слів потрібно зробити запит з другого словника. Так як словник уже відсортований, то для швидкого пошуку слів необхідно відсортувати запити і тоді скористатися наступною ідеєю.

Так як слова відсортовані, то якщо одне слово в списку лексикографічно більше за друге слово в іншому списку, то всі слова що йдуть за ним також більші за вибране слово. Тому якщо ми поставимо вказівники на початки обох списків, та будемо переміщувати той, який лексикографічно менший, то ми не пропустимо жодної пари одакових слів. Так як в словнику кожне слово зустрічається лише 1 раз, а в запитах не завжди, то при рівності слів, вказівник буде рухатися в списку запитів.

Після проведених експерементів отримали наступні результати:

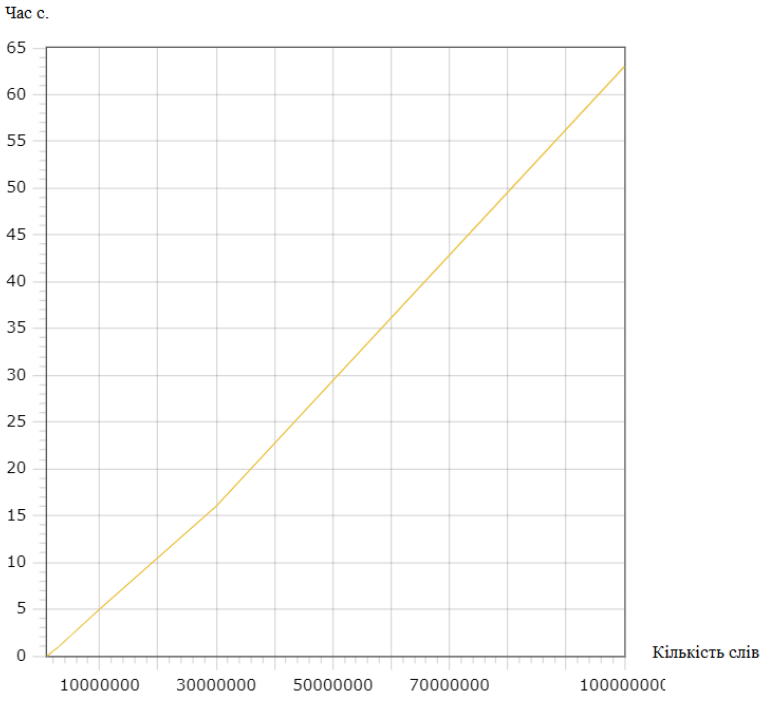

Рисунок 4.2 - Залежність часу роботи від розміру вхідних даних

З даного експеременту ми можемо побачити, що час при великій кількості вхідних даних росте пропорційно, та алгоритм є досить ефективним.

Таблиця 4.2 – Результати часу роботи алгоритму двох вказівників

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	0	1	5	17	63

Проте при великій кількості слів, алгоритм трохи програє бінарному пошуку. Це можливо пов’язано з реалізацією алгоритму, тому що при великих даних навіть невелика константа в складності може збільшувати реальний час роботи алгоритму в рази, хоча складність обох однакова.

4.4 Бор

Для наступного алгоритму використаємо метод бору. Сам метод описаний в пункті 2.2.7. Для даного тестування використаємо метод заснований на завчасно відомому алфавіті розміром в 26 символів.

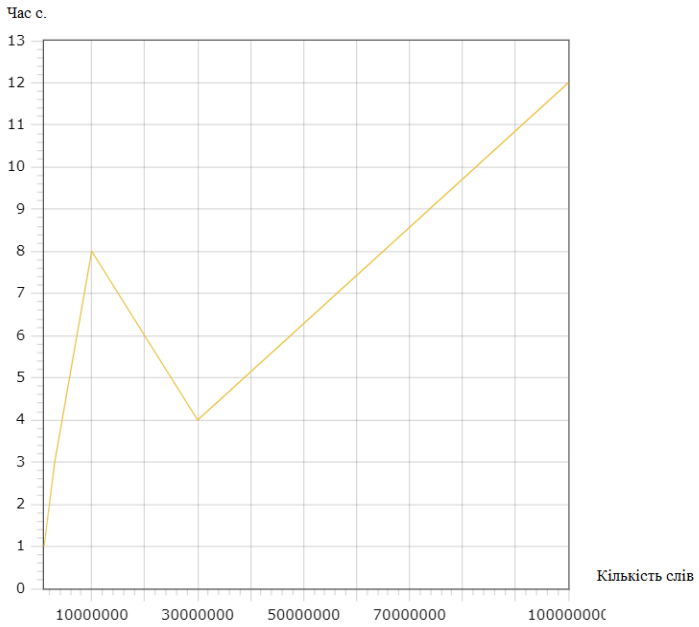

Рисунок 4.3 - Залежність часу роботи від розміру вхідних даних

Після проведених експеременті отримали результати, наведені на рисунку 4.3.

Таблиця 4.3 – Результати часу роботи бору

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	1	3	8	4*	12*

Як стало відомо в ході експеременту, при дуже великому обсязі вхідних даних, незважаючи на те, що алгоритм працює досить швидко, він потребує досить великих обсягів пам’яті. Це унеможливлює його роботу на дереві великих об’ємів. Тому для останніх двох тестів були проведені зміни. Замість дерева побудованому на 3*10^7 та 10^8 слів, ми побудували дерево з перших 10^6 слів повторених 30 або 100 разів відповідно. Та після цього зробили ті самі 3*10^7 або 10^8 разів запити на знаходження слова. Так як при внесенні одного і того самого слова дерево не збільшується, то швидкість роботи в рази зменшилася порівняно з очікуваним. Тому можемо зробити висновок, що дана версія бору має обмеження близьке до 10^7 слів. Що для звичайних мов вистачає.

Спробуємо модифікувати алгоритм так, щоб він працював в необхідних умовах. Так як кожна вершина в борі зберігає у себе посилання на 26 інших вершин, навіть якщо їх не існує, то при великій кількості даних стандартна реалізація методу не задовільняє наші потреби взагалі, або для неї потрібно використовувати комп’ютери, які можуть мати більше пам’яті доступної для роботи. Проте, так як ми не знаємо на якому алфавіті ми будемо застосовувати нашу програму, то все стає набагато гірше. Тому спробуємо внести модифікації для роботи алгоритму, які будут економити пам’ять не

зважаючи на час.

4.5 Модифікований бор з економією пам’яті

Для економії пам’яті змінемо структуру вершини. Якщо раніше ми використовували статичні посилання на кожну літеру, то зараз використаємо розширюваний масив, який дасть нам необхідність витрачати більше часу на доступ до однієї літери, але дозволить зменшити витрати необхідної пам’яті.

Після проведених експерементів отримали наступні результати:

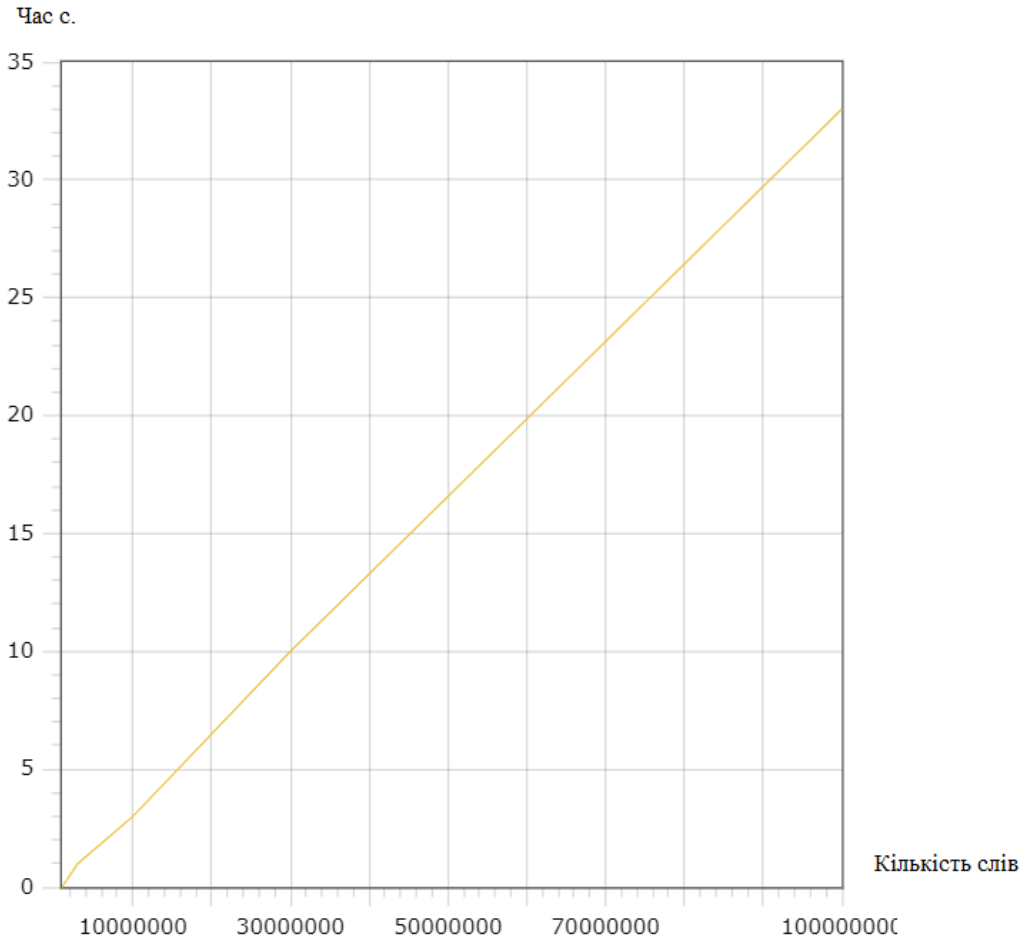

Рисунок 4.4 - Залежність часу роботи від розміру вхідних даних

Таблиця 4.4 – Результати часу роботи покращеного бору

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	0	1	3	10	33

Як ми можемо бачити з результатів, швидкість зросту часу лінійна по відношенню до розміру вхідних данних. Тому на великих даних цей алгоритм є трохи швидшим за бінарний пошук, який має логарифмічну складність. Проте, час який використовується на пошук літери є порівняною до логарифму, та буде зростати зі збільшенням використаного алфавіту, що може призвести до погіршення результатів алгоритму на словах з більшого алфавіту. Тому ми не можемо однозначно сказати що цей алгоритм кращий, потрібно зважати на дані на яких він буде працювати, так як він більше залежить від них порівняно з іншими алгоритмами.

Також можна знайти дивним те, що алгоритм який жертвує часом заради пам’яті почав швидше працювати ніж звичайний алгоритм. Цьому є дві причини. Перша – так як ми робимо лише один запит по кожному слову, перевага звичайного алгоритму, який швидше відповідає на запити знівельована. Друга – так як ми повинні виділити місце під усі 26 посилань, це займає певний час. І тому час побудови дерева починає відігравати більшу роль у загальному часі роботи програми. Так як ми використовуємо згенеровані слова, то одна із важливих величин, така як потужність префіксів – зростає максимально швидко, що може не відбутися в звичайному тексті.

Розберемося що таке потужність префіксів. Так як за структурою бору (рисунок 2.2) ми бачимо, що два слова можуть містити спільні вершини, то зрозуміло що чим більший їх спільний префікс (в даному випадку префіксом ми вважаємо не морфологічний префікс, а підрядок який починається з

початку слова), тим менше вершин буде мати бор.

В звичайній мові ми маємо досить багато спільнокорених слів, та слів які починаються з одного і того ж префіксу, так як мови утворювалися природньо. Тому на відміну від випадково згенерованих послідовностей літер які ми вважаємо словом, на звичайних словах певної мови бор буде себе показувати набагато краще. Це так, тому що кількість різних префіксів у згенерованій послідовності літер буде набагато більшою за кількість префіксів у звичайній мові. А так як кожна вершина дерева містить в собі один з префіксів, то зі збільшенням потужності префіксів, збільшиться і об’єм дерева.

Порівняно зі звичайним алгоритмом, даний алгоритм дозволив скоротити використання пам’яті майже в 13 разів, що дозволяє використовувати його на набагато більших тестових даних.

4.6 Алгоритм хешування

Для цього алгоритму ми будемо генерувати хеш фунцію від кожного рядка, та розподіляти усі слова за нею, після чого отримані результати шукати бінарним пошуком. Проте для нашого тестування усунемо другу частину, та подивимося лише на швидкість підрахунку хешів для наших рядків.

Після проведених експерементів отримали результати, наведені на рисунку 4.5.

З даного експеременту ми можемо побачити, що хешування займає досить невеликий час, але не може бути єдиним методом пошуку слів, так як можуть бути однакові хеші у двох різних рядків, та будуть фальшиві спрацювання. Цей алгоритм дозволить змешити область пошуку, але збільшить необхідні обсяги пам’яті, що може бути критичним при великій кількості вхідних даних, тому його потрібно застосовувати лише за необхідністю.

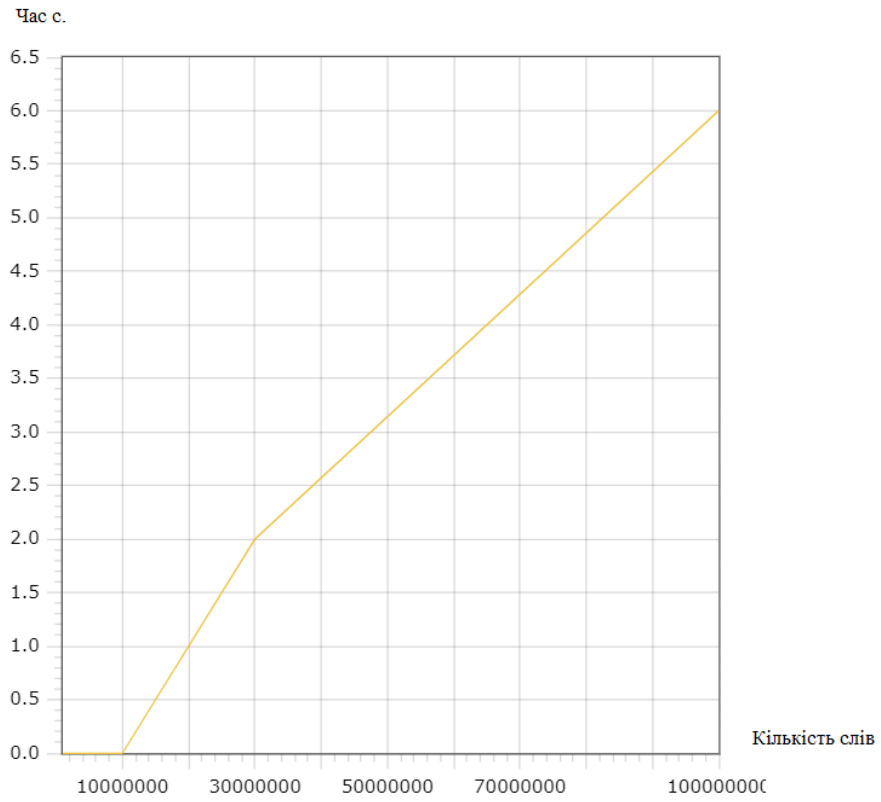

Рисунок 4.5 - Залежність часу роботи від розміру вхідних даних

Таблиця 4.5 – Результати часу роботи алгоритму хешування

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	0	0	0	2	6

4.7 Алгоритм збалансованого дерева

Для даного алгоритму використаємо модифікацію звичайного мепу, де ключами будуть наші рядки.

Після проведених експерементів отримали наступні результати:

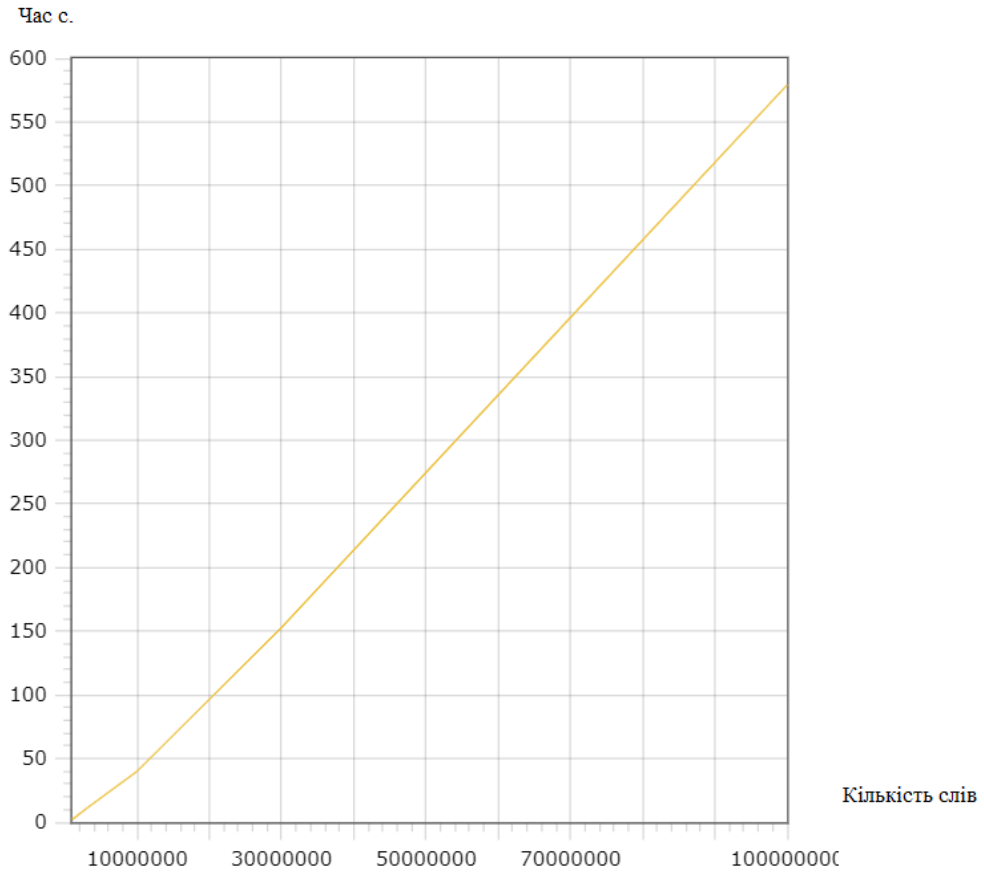

Рисунок 4.6 - Залежність часу роботи від розміру вхідних даних

Таблиця 4.6 – Результати часу роботи алгоритму збалансованого дерева

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	2	11	40	153	579

З даного експеременту ми можемо побачити, що час при великій кількості вхідних даних є дуже високим порівняно з іншими алгоритмами. Так як алгоритм має однакову з іншими складність O(N*logN), це є досить несподіваним результатом. Так як в якості ключа ми використовуємо рядок, їх порівняння займає досить велику кількість часу що призводить до таких затрат часу. Проте чи можемо ми сказати що алгоритм взагалі не робочий?

Для цього порівняємо його з тими, які будуть мати гіршу складність, та порівняємо його з ними.

4.8 Коренева декомпозиція

Для наступного алгоритму використаємо той, який на відміну від попередніх буде мати складніть на порядок гіршу. Використаємо алгоритм корневої декомпозиції. Для цього у відсортованому масиві виберемо слова, які стоять на позиціях $0, \sqrt{n}, 2\sqrt{n}, .., n$ що містять приблизно корінь від n слів, де n – кількість слів у списку.

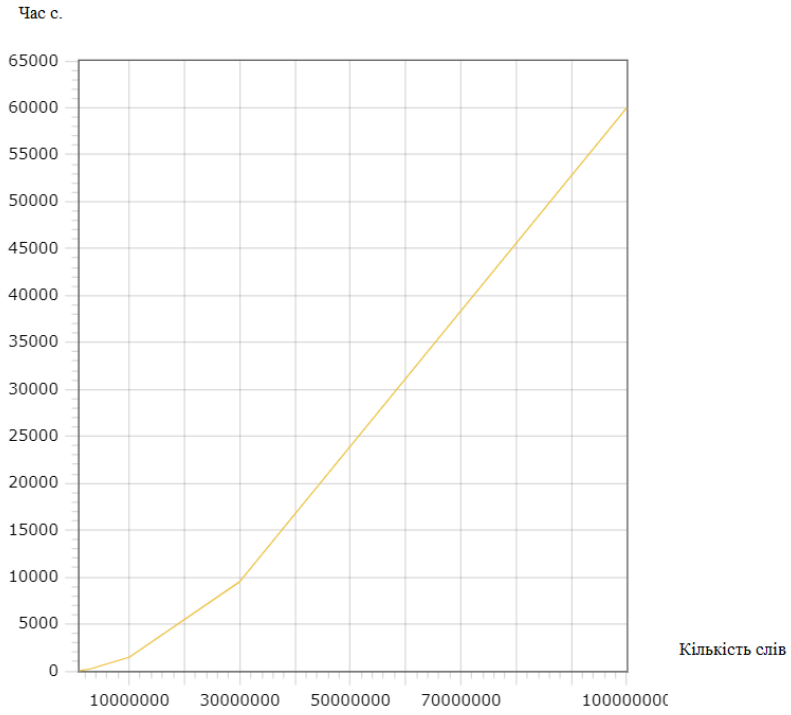

Рисунок 4.7 - Залежність часу роботи від розміру вхідних даних

Так як весь список був відсортований, то обрний таким чином список буде відсортований. Тоді порівнявши вхідне слово з кожним зі слів списку, ми можемо за корінь від довжини списку порівнянь знайти відрізок, на якому може знаходитися шукане слово. Так як ми вибирали слова з різнею в корінь між ними, то довжина знайденого підвідрізка буде рівна кореню від n. Таким

чином за 2 кореня від довжини списку порівнянь ми можемо знайти наше слово.

Після проведених експерементів отримали результати, наведені на рисунку 4.7.

Таблиця 4.7 – Результати часу роботи алгоритму корневої декомпозиції

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (секунди)	45	225	1464	9516*	60000*

Так як уже при розмірі вхідних даних 10^7 слів час роботи наближається до 25 хвилин, було прийнято рішення не проводити експеримент для більших даних, а знайти приблизний час роботи алгоритму. Як бачимо для останнього експеременту знадобиться більше 16 годин часу, що звичайно не є можливим в рамках проведення даного дослідження.

4.9 Квадратичний алгоритм

Зрозуміло, що в рамках даної задачі, перше що можна зробити, це просто виконати умову, та для кожного зі слів перебрати кожне інше слово. Так як це буде досить довго, то замість проведення експеременту підрахуємо очікуваний час роботи алгоритму. За основу візьмемо час роботи алгоритму кореневої декомпозиції, так як він найбільш наближений до нього, та домножимо його на корінь від розміру вхідних даних.

Це можливо, тому що реальний час роботи алгоритму рівний кількості команд яку виконає алгортим, розділену на частоту процесора, або домножену на час виконання однієї команди. Так як ми рахуємо для одного і

того ж процесора, при розділені одного на інше, ця величина скоротиться, а відношення кількості команд буде пропроційне до складності. Так як $n^2 / n\sqrt{n} = \sqrt{n}$, то ми можемо домножити результат роботи корневої декомпозиції на корінь від вхідних даних.

Після проведених підрахунків отримали наступні результати:

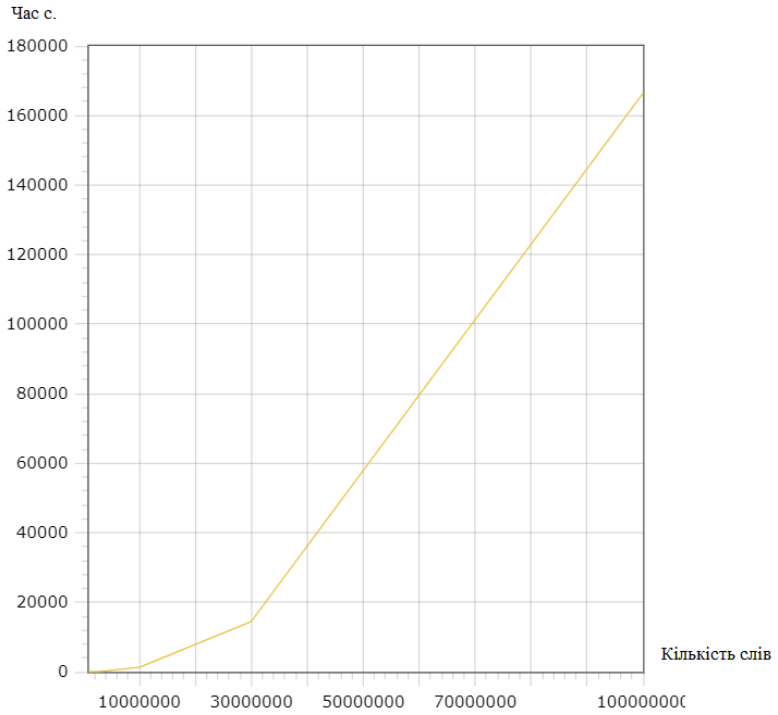

Рисунок 4.8 - Залежність часу роботи від розміру вхідних даних

Таблиця 4.8 – Результати часу роботи алгоритму квадратичного пошуку

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Час роботи (години)	12.5	108	1286	14478	19 років

З даного експеременту ми можемо побачити, що для обробки великої кількості даних, нам необхідно використовувати хоч якісь алгоритми, які будуть покащувати звичайний вибір відповідності для кожного зі слів.

4.10 Порівняння алгоритмів

Для порівняння алгоритмів зведемо усі отримані в ході експерементів дані до спільної таблиці.

Таблиця 4.9 – Порівняння роботи алгоритмів

Розмір вхідних даних (кількість слів)	10^6	3*10^6	10^7	3*10^7	10^8
Бінарний пошук	0	1	5	11	41
Два вказіники	0	1	5	17	63
Бор	1	3	8	4*	12*
Покращений бор	0	1	3	10	33
Хешування	0	0	0	2	6
Збалансоване дерево	2	11	40	153	579
Корнева декомпозиція	45	225	1464	9516**	60000**
Квадратичний пошук	12.5год**	108год**	1286год**	14478год**	19 років**

*-Дані виконувалися на дереві побудованому на 10^6 слів замість необхідних

**-Підрахований очікуваний час роботи.

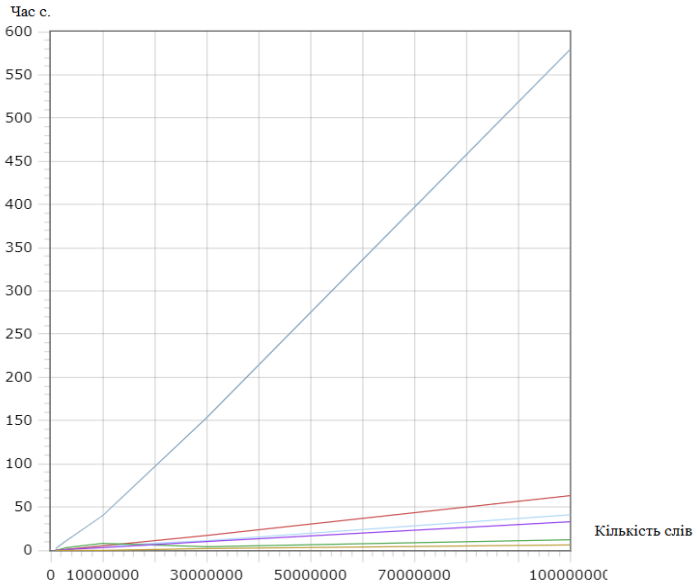

Рисунок 4.9 – Порівняльний графік логарифмічних алгоритмів

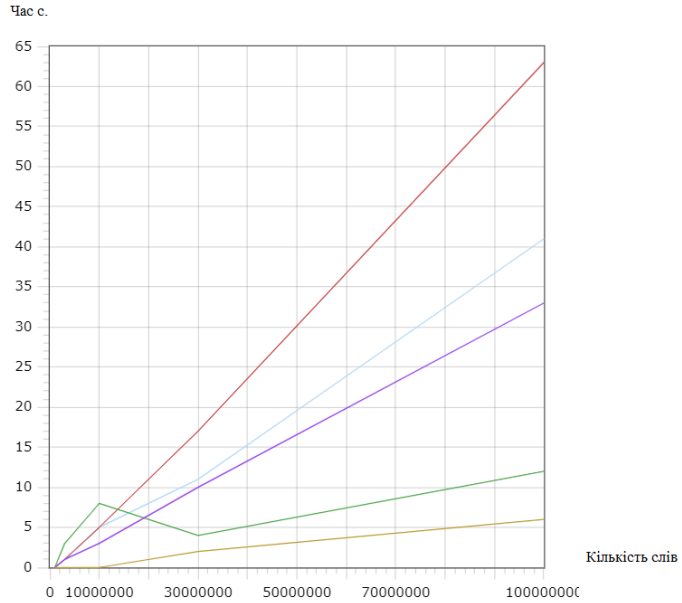

Рисунок 4.10 – Порівняння усіх логарифмічних алгоритмів окрім збалансованого дерева

Після проведених експерементів зрозуміло, що для великих даних обов’язково потрібно використовувати швидкі алгоритми. Найкращим вибором буде або бор із покращенням, який краще працює з невеликими алфавітами, або розподіл хешуванням та використанням бінарного пошуку після цього.

ВИСНОВКИ

В даній роботі була розроблена комп’ютерна система для автоматичного перекладу тексту, та проведені експеременти по її роботі та покращенню.

В основі системи використовуються авторські методи для перекладу та обробки тексту.

Завдання було виконано наступними кроками:

- проаналізовані аналогічні перекладачі;
- проаналізовані та порівняні алгоритми для знаходження слів в тексті та в словнику;
- розроблено та протестовано програму для генерації словників, та проведені експеременти для її прискорення;
- розроблено та протестовано програму на основі методів аугментації тексту, проведено її тестування та вибрані найефективніші методи для роботи з нею.

За допомогою тестуваня вдалось з’ясувати, що для генерації словників можливо використовувати алгоритми для розпаралелювання програми на CPU та на GPU. В ході тестування з’ясувалося, що алгоритм розпаралелювання на CPU дає можливість прискорити програму в певну кількість разів, яка залежить відь кількості задіяних для цього ядер процесора.

В ході тестування алогоритма на GPU з’ясувалося що він дає невелике прискорення на певних вхідних данних, але на великих данних він працює гірше звичайного алгоритму. Тому цей алгоритм не є доречним використовувати в данній програмі.

В ході тестування вибору алгоритму для вибору слова із словника, який використовується в програмі на основі одного з методів аугментації тексту, було протестовано вісім різних алгоритмів, які відрізняються як за

складністю так і за метою застосування. Гірше всього себе показали алгоритми квадратичного пошуку та корневої декомпозиції, які витрачають в рази більше часу зі збільшенням обсягу вхідних даних, який може доходити навіть до декількох років. Застосування цих алгоритмів не є досцільним.

В ході тесування алгоритму бору з'ясувалося, що при великих вхідних даних він витрачає багато пам'яті, тому алгоритм було змінено на його модифікацію яка займає менше пам'яті. Оновлений алгоритм продемонстрував гарний час роботи, та може бути застосований в данній програмі.

Серед простованих решти протевоних алгоритмів, алгоритм збалансованого дерева показав себе найгірше, алгоритми двох вказівників та бінарного пошку майже однаково, але гірше за бор. Накращим виявилося поєднання бінарного пошуку з хешуванням. Для використання рекомендується використовувати бор або хешування в залежності від типу вхідних данних.

В ході тестування програми на основі агументації було з'ясовано, що програма може демонструвати заміну генерування словників звичайною програмою, але якість цього словника менша за звичайний метод. Крім того якість отриманого словника прямо пропорційна якості вхідних словників, та при використанні словників низької якості результат може бути незадовільним.

Програма має досить високий потенціал для використання в випадках, коли не існує звичайних перекладачів. При зростаючій кількості мов, ця проблема може погіршуватися, так як кількість перекладачів не може зростати з аналогічною швидкістю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. COMPUTER AND INFORMATION SYSTEMS AND TECHNOLOGIES April 22 – 23, 2020, Hybrid language processing approach, Anton Havrashenko, Olesia Barkovska

2. Дмитро Казаков. Штучна природність (рус.) // Наука і життя. – 2017. – № 10. – С. 100–107.

3. SLANG DICTIONARY [Елекронний ресурс] URL: https://www.dictionary.com/e/slang/

4. The Limits of Machine Translation [Елекронний ресурс] URL: "Madsen, Mathias: The Limits of Machine Translation (2010)"

5. History of machine translation [Елекронний ресурс] URL: https://web.archive.org/web/20110607092437/http://www.traduceme.org/profiles/blogs/history-of-machine-translation

6. Found in translation: More accurate, fluent sentences in Google Translate [Елекронний ресурс] URL: https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate

7. Zero-Shot Translation with Google’s Multilingual Neural Machine Translation System [Елекронний ресурс] URL: https://ai.googleblog.com/2016/11/zero-shot-translation-with-googles.html

8. Statistical Machine Translation – [Елекронний ресурс] URL: http://www.statmt.org/

9. Statistical machine translation [Елекронний ресурс] https://en.wikipedia.org/wiki/Statistical_machine_translation#Word-based_translation

10. Міщенко А.Л. Машинний переклад у контексті сучасного науково технічного перекладу // Вісник ХНУ ім. В.Н. Каразіна. – Серія «Романо-германська філологія. Методика викладання іноземних мов». – № 1051. – С.172–180, 2013.

11. Tripathi S., Sarkhel J. K. Approaches to Machine Translation // Ann. of Library and Inform. Studies., Vol. 57. – P. 388–393, 2010.

12. Sanyal S., Borgohain R. Machine Translation Systems in India // Computing Research Repository, 2013 [Електронний ресурс]. – Режим доступу: http://arxiv.org/ftp/arxiv/papers/1304/1304.7728.pdf

13. В. Яковина, В. Масюкевич Національний університет «Львівська політехніка», ОГЛЯД ТА АНАЛІЗ МЕТРИК ОЦІНЮВАННЯ ЯКОСТІ МАШИННОГО ПЕРЕКЛАДУ

14. МАШИННИЙ ПЕРЕКЛАД [Елекронний ресурс] URL: http://www.logrus.ru/pages/ua-mashinnij_pereklad.aspx

15. Karen S. Jones. Natural language processing: a historical review //Cambridge: Computer Laboratory, University of Cambridge, 2001. – P.2

16. Hull, DA. Stemming algorithms: a case study for detailed evaluation [Text] / D. A. Hull // Journal of the American Society for Information Science. – 1996. -Vol. 47, №1.-P. 70-84.

17. Алгоритмы предварительной обработки текста: декомпозиция, аннотирование, морфологический анализ [Текст] / В.А. Яцко, М.С. Стариков, Е.В. Ларченко [и др.] // Научно-техническая информация. Сер.2. - 2009. -№ 11. - С. 8-18.

18. Самовчитель Visual C++ . – СПб .: БХВ-Петербург, 2002.

19. Ліппман С. Б., Лажойе Ж. Мова програмування C ++. Вступний курс: Пер. з англ. – 3-е вид. – М .: ДМК, 2001. – 1104 с.

20. Офіційний сайт sql[Елекронний ресурс] URL:https://www.mysql.com/

21. Префикс-функция. Алгоритм Кнута-Морриса-Пратта [Елекронний ресурс] URL:https://e-maxx.ru/algo/prefix_function

22. N-грам [Елекронний ресурс] URL: https://uk.wikipedia.org/wiki/N-грам

23. Про есперанто[Елекронний ресурс] URL: https://lernu.net/ru/esperanto

24. Про есперанто [Елекронний ресурс] URL: https://lingvo.info/en/lingvopedia/esperanto

25. Д-р Эсперанто. Международный язык. Предисловие и полный учебник. – 1-е изд. – Варшава, 1887.

26. Github [Елекронний ресурс] URL: https://github.com/

27. Офіційний сайт Visual Studio [Елекронний ресурс] URL: https://visualstudio.microsoft.com/ru/