Міністерство освіти і науки України

Харківський національний університет ім. В. Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Кібербезпека»

"Допущено до захисту"

В.о. зав. кафедрою БІСТ

Мелкозьорова О.М.

_____

« » 2024 р.

Пояснювальна записка

на тему: "Розробка програмного забезпечення для автоматизації перевірки та порівняння робіт студентів"

оцінка « » Керівник к.т.н. Громико І.О.

Голова ЕК

Лемешко О.В. _____ Рецензент Краснобаєв В.А.

Виконавець: студент групи КБ-41

Жілєнков Д.В.

Харків – 2024

РЕФЕРАТ

Дипломна робота має обсяг 56 сторінки, містить в собі 25 ілюстрацій, 10 джерел за переліком посилань, 4 додатки.

Мета роботи полягає в створенні ПЗ, яке зможе слугувати викладачам для перевірки великої кількості файлів між собою, буде зручним та простим у використанні, не буде потребувати доступу до інтернету чи інших спеціальних умов, буде виправно і швидко працювати, та формувати результат у вигляді відсоткового представлення збігів. Кінцевою метою є поєднання в рамках одного програмного рішення створеного ПЗ та "ПЗ ДЛЯ ПЕРЕВІРКИ ЯКОСТІ ДИСТАНЦІЙНОГО НАВЧАННЯ" зі створенням візуального інтерфейсу.

Об’єкт дослідження - алгоритми та методи порівняння текстів, методи розробки алгоритмів на програмному рівні.

Предмет дослідження - робота, ефективність, властивості алгоритмів та методів порівняння текстів і деталі методів розробки алгоритмів на програмному рівні.

Основними методами дослідження є аналіз та порівняння алгоритмів перевірки подібності текстів з подальшою реалізацією обраних алгоритмів і розробкою кінцевого ПЗ.

Через велику кількість можливостей, що існують у наш час для плагіату, розробка ПЗ для перевірки на подібність між текстовими файлами у локальній директорії є актуальним завданням, враховуючи відсутність у відкритому доступі засобів, що змогли б задовільнити всі умови та вимоги, поставлені щодо розроблюваного ПЗ.

У роботі досліджено сучасні алгоритми для перевірки текстових файлів на подібність, проведено аналіз переваг та недоліків алгоритмів. Проведено порівняння алгоритмів і подальший вибір найбільш підходящих із них для створення заданого ПЗ

Результатом роботи є ПЗ, яке можуть використовувати викладачі для перевірки локальних файлів між собою на подібність, яке виводить у результаті

список порівняння з відсотковими збігами, є зручним у використанні, має зрозумілий та зручний інтерфейс, є ефективним, не потребує підключення до інтернету.

Результати роботи можуть бути застосовані для порівняння word-файлів з відповідями на контрольні роботи, екзаменаційні білети, будь-які роботи, у яких у студентів є спільні питання і є можливість списати щось один у одного або з лекцій.

В якості продовження та розвитку розробки розглядається розширення функціоналу програми, а саме: створення функції виділення кольором знайдених збігів у найбільш схожих між собою файлах, оптимізація роботи алгоритмів та ПЗ в цілому, застосування штучного інтелекту у процесі порівняння робіт.

Ключові слова: ПОДІБНІСТЬ ФАЙЛІВ, АВТОМАТИЗАЦІЯ ПРОЦЕСІВ, ЯКІСТЬ ЗНАНЬ, ДОПОМОГА ВИКЛАДАЧАМ, ЗРУЧНІСТЬ, ЕФЕКТИВНІСТЬ.

The thesis is 56 pages long, contains 25 illustrations, 10 references, and 4 appendices.

The purpose of the work is to create software that can serve teachers to check a large number of files against each other, be convenient and easy to use, not require Internet access or other special conditions, work correctly and quickly, and generate the result in the form of a percentage representation of matches. The ultimate goal is to combine the developed software and the "DISTANCE LEARNING QUALITY CHECKER" with the creation of a visual interface within a single software solution.

Object of research - algorithms and methods of text comparison, methods of developing algorithms at the program level.

The subject of the study is the operation, efficiency, properties of algorithms and methods of text comparison and details of methods of developing algorithms at the program level.

The main research methods are the analysis and comparison of text similarity checking algorithms, followed by the implementation of the selected algorithms and the development of the final software.

Due to the large number of opportunities for plagiarism that exist today, the development of software for checking for similarity between text files in a local directory is an urgent task, given the lack of open access tools that could satisfy all the conditions and requirements for the software being developed.

The paper investigates modern algorithms for checking text files for similarity, analyzes the advantages and disadvantages of the algorithms. The algorithms were compared and the most suitable ones were selected for the creation of the software.

The result of the work is software that can be used by teachers to check local files for similarity, which displays a comparison list with percentage matches, is easy to use, has a clear and user-friendly interface, is efficient, does not require an Internet connection.

The results of the work can be used to compare word files with answers to quizzes, exam papers, any papers in which students have common questions and have the opportunity to copy something from each other or from lectures.

As a continuation and development of the development, we are considering expanding the functionality of the program, namely: creating a function of color highlighting the found matches in the most similar files, optimizing the work of algorithms and software in general, and using artificial intelligence in the process of comparing works.

Keywords: FILE SIMILARITY, PROCESS AUTOMATION, QUALITY OF KNOWLEDGE, ASSISTANCE TO TEACHERS, CONVENIENCE, EFFICIENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ СКОРОЧЕНЬ І ТЕРМІНІВ

NLP -Natural Language Processing

TF -Term Frequency

IDF -Inverse Document Frequency

ДНК -Дезоксирибонуклеїнова кислота

ПЗ -Програмне забезпечення

POI -point of interest

IDE -Integrated Development Environment

ВСТУП

У наш час дуже гострою для України є проблема дистанційного навчання в університетах. На протязі вже п’яти років величезна кількість студентів не має можливості очно відвідувати університети, в яких вони навчались до цього. Через це більшість закладів вищої освіти були вимушені повністю або частково перейти на форму дистанційного навчання.

Дистанційне навчання вимагає від студентів достатньо високого рівня відповідальності і самодисципліни, таке навчання викликає складнощі з підтримкою концентрації і організації робочого часу, складнощі щодо сприйняття інформації таким способом, а також збільшує навантаження на студентів, і на викладачів, які мають не тільки давати змогу отримати знання, а й перевіряти чи студент їх дійсно отримав, перевіряти їх якість, що є складнішим у контексті онлайн навчання, порівнюючи з очним.

Якість знань оцінюється не тільки правильністю відповіді студента на задане питання, але й унікальністю його відповіді. Саме перевірка унікальності робіт і є основною тематикою моєї роботи. Онлайн навчання – це найсприятливіші з усіх можливих умов для плагіату та списування, для швидкого пошуку інформації та копіювання цієї інформації всього за декілька натискань, не потрібно навіть нічого переписувати або придумувати, якщо мова йде про електронний вид завдань та відповідей. Це є великим недоліком робіт в електронному вигляді, але в той же час такий вид робіт є дуже зручним у плані виконання та економним у плані часу, і більшість студентів, надають перевагу саме електронному виду робіт.

На плечі викладача під час перевірки знань покладаються доволі нелегкі та трудомісткі зобов’язання, особливо якщо перевірка йде у письмовому вигляді:

-перевірити правильність наданих відповідей (що вже може викликати складнощі через погану чіткість зображення написаної відповіді, незрозумілий почерк, а також займає багато часу)

-запам’ятати відповіді студентів, для того, щоб потім наткнувшись на схожі відповіді згадати в якій саме роботі вони були знайдені (це також дуже складно, а запам’ятати все практично неможливо, враховуючи величезну кількість студентів та питань).

-порівняти ці, на перший погляд, «збіжності», зрозуміти чи насправді вони сплагіачені, чи це просто схожі формулювання відповідей (це також лишнє навантаження і може сильно виснажити викладача).

Отже, перевірка знань в електронному вигляді є більш зручною не тільки для студентів, а й для викладачів: її легше перевіряти, не потрібно розбиратися в почерках, в нечітких зображеннях, залишається тільки питання з приводу порівняння робіт студентів між собою на плагіат. Виходячи з цього, ціллю моєї роботи буде вирішення проблеми плагіату в роботах студентів задля того, щоб більшість перевірок знань проводилися саме в такому, найбільш зручному для студентів і викладачів виді, і щоб викладачам не потрібно було виконувати ту частину роботи, яку можливо автоматизувати.

Зараз є багато засобів, які використовуються для перевірки на плагіат (усілякі сайти, утиліти, навіть влаштовані в систему можливості), але вони мають недоліки та характеристики, які не відповідають вимогам, що потрібні для цілей, описаних у даній роботі, та потрібні викладачам для використання:

- По-перше, область перевірки. Мається на увазі те, що від ПЗ не вимагається і не є потрібним порівняння з інтернет ресурсами, на відміну від більшості засобів, які розраховані саме на такий варіант антиплагіату (наприклад Turnitin, Grammarly, Copyscape й інші). Зазвичай відповіді на питання з навчальних дисциплін є в лекціях, і саме вони є найбільш достовірними, такими, які влаштовують викладачів та дають змогу зрозуміти, що студент читав та вчив матеріал, який йому давали на парах і в процесі навчання. Тому ПЗ орієнтоване саме на порівняння між локальними файлами, а не з інтернет-ресурсами.
- По-друге, кількість перевіряємих файлів, їх розмір та швидкість перевірки. Більшість ресурсів, які є у вільному доступі не підтримують

порівняння більш ніж двох файлів між собою (Diffchecker), мають обмежену кількість файлів, обмежений розмір цих файлів (Text Compare) і виконують перевірку доволі повільно у масштабі великої кількості перевіряємих файлів (Meld).

- По-третє, складність використання та спеціальні умови. Ресурси, які задовольняють вимоги, описані вище, в більшості випадків є доволі складними у використанні (Beyond Compar), зі складним інтерфейсом (Meld), можуть не підтримувати певні формати файлів, потребують підключення до інтернету (Draftable), що в наш час з постійними відключеннями світла є доволі актуальною проблемою.
- По-четверте, результат перевірки та користь. Результатом засобів для порівнянь файлів зазвичай не є відсоток співпадінь, як в антиплагіаті, їх результатом є візуалізація збігів і різниць між файлами шляхом виділення(Diffchecker, Text Compare, WinMerge, Meld, Beyond Compare, KDiff3), відокремлення(Draftable), або виведення різниць/співпадінь в окремий файл. Для заданого ПЗ важливо вивести відсоткове відношення подібності, через те, що перевіряється багато робіт, а отже, потрібно звести до мінімуму непотрібні дії для викладача, такі як відкриття та перевірка на подібність файлів, в яких немає плагіату. Тому буде доцільно щоб ПЗ виводила список з відсотками співпадінь між парами файлів, щоб викладач міг одразу побачити які файли найбільш схожі та які потрібно відкрити і перевірити, не витрачаючи час та сили на роботи без плагіату.

Отже, у вільному доступі відсутні засоби, утиліти, програми, що могли б задовільнити всі вищевказані потреби. У даній роботі я розроблю своє ПЗ, яке зможе слугувати викладачам для перевірки великої кількості файлів між собою, буде зручним та простим у використанні, не буде потребувати доступу до інтернету чи інших спеціальних умов, буде виправно і швидко працювати, та формувати результат у вигляді відсоткового представлення збігів.

Це ПЗ можна буде застосовувати для порівняння word-файлів з відповідями на контрольні роботи, екзаменаційні білети, будь-які роботи, в яких у студентів є спільні питання і є можливість списати щось один у одного. Також можна добавити файли лекцій до папки, в якій будуть міститися файли для перевірки, таким чином можна виявити хто і з яких лекцій скопіював матеріал.

Моя робота тісно пов’язана з роботою магістра КБ Коршенко Владислава, який є автором дипломної роботи на тему «ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПЕРЕВІРКИ ЯКОСТІ ДИСТАНЦІЙНОГО НАВЧАННЯ», та є продовженням його розробки шляхом доповнення своїм ПЗ, результатом роботи буде єдине ПЗ для допомоги викладачеві у перевірці робіт.

1. МЕТОДИ ПОРІВНЯННЯ ТЕКСТОВИХ ФАЙЛІВ

1.1 Існуючі методи порівняння текстових файлів.

1.1.1 Алгоритм Левенштейна (Levenshtein Distance).

Опис:

Цей алгоритм динамічного програмування знаходить відстань редагування (найменшу кількість операцій вставки, видалення або заміни символів) між двома рядками. Він будує матрицю розмірністю $(m+1)\times(n+1)$, де m і n - довжини двох рядків. Кожен елемент матриці містить мінімальну відстань редагування між префіксами рядків до цієї позиції. Обчислення відбувається ітеративно, заповнюючи матрицю знизу-вгору та зліва-направо, використовуючи такі правила:

Вставка: dist[i][j] = dist[i][j-1] + 1

Видалення: dist[i][j] = dist[i-1][j] + 1

Заміна: dist[i][j] = dist[i-1][j-1] + (s1[i]!=s2[j])

Складність часу - O(mn), а складність пам'яті - O(min(m,n)).

Використання:

-Простий у реалізації.

-Використовується для перевірки орфографії, пошуку схожих рядків, порівняння ДНК-послідовностей.

1.1.2 Алгоритм Дамерау-Левенштейна.

Опис:

Цей алгоритм є розширенням Левенштейна і додатково дозволяє операцію транспозиції - перестановку сусідніх символів. Він використовує ту саму базову ідею динамічного програмування, що й Левенштейн, але додає ще один крок у обчислення матриці дистанцій. Якщо символи в двох рядках на одній і тій же позиції не збігаються, але два попередні символи утворюють транспозицію, то

dist[i][j] = dist[i-2][j-2] + 1. Цей додатковий крок збільшує складність алгоритму до O(mn).

Використання:

-Корисний для виявлення типових помилок введення, таких як перестановки символів.

1.1.3 Алгоритм Jaccard Similarity.

Опис:

Ця міра схожості визначається як відношення розміру перетину двох множин до розміру їх об'єднання: J(A, B) = |A ∩ B| / |A ∪ B|. У контексті обробки текстів, кожен документ представляється як множина його термінів (слів або n-грам). Перетин - це множина термінів, що містяться в обох документах, а об'єднання - множина всіх унікальних термінів з обох документів. Значення Jaccard Similarity варіюється від 0 (множини не перетинаються) до 1 (множини ідентичні).

Використання:

-Використовується для порівняння наборів даних, таких як ключові слова або шингли (підрядки фіксованої довжини).

1.1.4 Алгоритм Cosine Similarity.

Опис:

Ця міра схожості базується на косинусній відстані між векторами. Текстові документи представляються як вектори в векторному просторі, де кожен вимір відповідає одному терміну (слову або n-грамі). Значення кожного виміру може бути обчислено різними способами, наприклад, за допомогою TF-IDF. Косинусна подібність двох векторів обчислюється як косинус кута між ними і варіюється від 0 (вектори ортогональні) до 1 (вектори паралельні та ідентичні за напрямком).

Використання:

-Використовується в обробці природної мови (NLP) для порівняння документів на основі частотності термінів (TF-IDF).

1.1.5 TF-IDF (Term Frequency-Inverse Document Frequency).

Опис:

TF-IDF - це статистична міра, яка використовується для оцінки важливості слова в документі та колекції документів. TF (Term Frequency) - це частота появи слова в документі. Чим більше разів слово зустрічається в документі, тим вищим є його TF. IDF (Inverse Document Frequency) визначається як логарифм від відношення загальної кількості документів до кількості документів, що містять це слово. IDF зменшує вагу широко вживаних слів. TF-IDF є добутком TF та IDF, і чим вищим є значення TF-IDF для слова в документі, тим більш релевантним і важливим воно вважається для цього документа.

Використання:

-Використовується для порівняння та кластеризації документів.

1.1.6 Алгоритм Smith-Waterman.

Опис:

Це алгоритм локального вирівнювання для пошуку найбільш схожих локальних підпослідовностей між двома послідовностями (наприклад, нуклеотидними або амінокислотними). На відміну від глобального вирівнювання, локальне вирівнювання намагається знайти найдовші ділянки високої схожості без вирівнювання всієї послідовності. Алгоритм будує матрицю подібності, заповнену значеннями схожості, використовуючи систему зважування відповідностей і невідповідностей. Максимальне значення в матриці вказує найкращу локальну підпослідовність. Час роботи - O(mn), де m і n - довжини двох послідовностей.

Використання:

-Широко використовується в біоінформатиці для порівняння ДНК або білкових послідовностей.

1.1.7 Шинглінг (Shingling).

Опис:

Це техніка для подання документа у вигляді множини чисел (хешів), яка використовується для визначення подібності або знаходження дублікатів документів. Документ розбивається на перекриваючі n-грами (шингли) фіксованої довжини, наприклад, 4-грами символів або 5-грами слів. Потім кожна n-грама хешується за допомогою хеш-функції, а множина всіх хешів становить компактне представлення документа.

Два документи вважаються схожими, якщо їхні множини шинглів мають високий ступінь перекриття (наприклад, високе значення Jaccard Similarity). Шинглінг корисний у системах виявлення плагіату або для видалення дублікатів.

Використання:

-Використовується для виявлення схожих документів в інтернеті.

1.2 Аналіз переваг та недоліків алгоритмів.

1.2.1 Алгоритм Левенштейна.

Переваги:

- Простий та ефективний для обчислення відстані редагування між рядками
- Забезпечує інтуїтивну міру подібності рядків
- Має ряд варіацій та розширень для задоволення різних потреб

Недоліки:

- Не враховує контекст або семантику рядків
- Обчислювальна складність квадратична відносно довжини рядків
- Всі операції редагування (вставка, видалення, заміна) мають однакову вагу

1.2.2 Алгоритм Дамерау-Левенштейна.

Переваги:

- Покращена міра редакційної відстані порівняно з Левенштейном
- Враховує типові помилки введення, такі як транспозиції символів

Недоліки:

- Більш складний у реалізації порівняно з Левенштейном
- Все ще не враховує семантику або структури даних

1.2.3 Алгоритм Jaccard Similarity.

Переваги:

- Забезпечує просту та інтуїтивну міру подібності множин
- Ефективний для порівняння текстових документів
- Не залежить від порядку або дублікатів елементів всередині множин

Недоліки:

- Не враховує частоту елементів всередині множин
- Може бути чутливим до розміру множин
- Не підходить для порівняння впорядкованих послідовностей

1.2.4 Алгоритм Cosine Similarity.

Переваги:

- Корисний для виявлення семантичної подібності текстів
- Враховує вагу та важливість термінів у векторному представленні
- Може бути використаний з різними техніками зважування термінів (TF-IDF, Word2Vec тощо)

Недоліки:

- Чутливий до розміру документів та нормалізації даних
- Не враховує порядок слів у тексті

- Вимагає попередньої обробки та векторизації текстів

1.2.5 TF-IDF.

Переваги:

- Забезпечує корисну статистичну міру важливості терміна
- Допомагає виділити найбільш релевантні терміни в документі
- Може бути використаний у поєднанні з іншими алгоритмами (наприклад, косинусною подібністю)

Недоліки:

- Не враховує семантику або контекст термінів
- Може надавати надмірну вагу рідкісним термінам
- Не підходить для коротких текстів через недостатню статистику

1.2.6 Алгоритм Smith-Waterman.

Переваги:

- Ефективний для знаходження локальних схожих регіонів у послідовностях
- Широко використовується в біоінформатиці для вирівнювання послідовностей
- Гнучкий завдяки можливості налаштування системи зважування відповідностей

Недоліки:

- Не підходить для глобального вирівнювання повних послідовностей
- Вимагає ретельного налаштування ваг підрахунку
- Може бути обчислювально витратним для дуже довгих послідовностей

1.2.7 Шинглінг (Shingling).

Переваги:

- Забезпечує ефективний спосіб пошуку подібних/дублікатних документів
- Дозволяє працювати з документами довільного формату
- Стійкий до невеликих змін у документі

Недоліки:

- Не враховує семантичний зміст або структуру документа
- Чутливий до вибору розміру шинглів
- Втрачає інформацію про порядок термінів у документі

2. ВИБІР І ДОСЛІДЖЕННЯ ОБРАНИХ АЛГОРИТМІВ, ПІДГОТОВКА ДО РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Порівняння алгоритмів на основі попереднього аналізу та розгляду.

Порівняння на практиці:

- Алгоритм Левенштейна: Підходить для задач перевірки орфографії, автозаповнення та виправлення опечаток. Він ефективний для коротких текстів, але не враховує семантичну близькість.
- Алгоритм Дамерау-Левенштейна: Краще підходить для випадків з частими перестановками символів. Використовується в тих самих областях, що й алгоритм Левенштейна, але забезпечує більшу точність.
- Алгоритм Джаккара: Підходить для порівняння великих текстових масивів та задач інформаційного пошуку. Простий у реалізації, але не враховує порядок слів.
- Косинусна схожість: Використовується для порівняння довгих текстів і документів, враховує частоту слів. Потрібна попередня обробка текстів.
- Алгоритм Джаро-Вінклера: Ефективний для порівняння коротких рядків, таких як імена. Враховує порядок символів, але менш ефективний для довгих текстів.
- Н-грамми: Підходять для задач кластеризації та розпізнавання тексту. Простий у реалізації, але чутливий до вибору параметра n.

Кожен з цих алгоритмів має свої переваги і недоліки, і вибір алгоритму залежить від конкретного завдання та характеру текстів, які потрібно порівняти.

2.2 Вибір алгоритмів та його обґрунтування.

Після розгляду та аналізу алгоритмів порівняння текстових документів було розпочато вибір алгоритмів, використання яких буде найефективнішим у контексті створюваного ПЗ.

Алгоритм Левенштейна є ефективним методом знаходження навіть незначних змін у тексті, такі як зміни букв у словах, заміна слова, зміна закінчення, його легко реалізувати і він добре працює з невеликими текстами. Але разом з цим, в нього є і недоліки, такі як низька ефективність при перевірці великих текстів, пряма залежність ефективності від впорядкованості слів та речень, при перестановці або видаленні речень алгоритм втрачає свій потенціал, оскільки він працює на основі послідовного порівняння символів. Обчислювальна складність алгоритму теж залежить від розміру тексту, чим більший обсяг, тим більше часу та ресурсів потрібно для обчислення відстані Левенштейна між двома текстами. Це може стати проблемою у випадках, коли потрібно обробити велику кількість документів або текстів. Сам по собі алгоритм не є хорошим вибором для створення заданого ПЗ.

Метод Шинглінгу – ефективний метод розподілу тексту на частини і їх перевірки між собою. Метод є стійким до невеликих змін у документах, таких як перефразування, додавання/видалення окремих слів або речень, адже виконується пошук найбільш схожих між собою частин тексту, що дозволяє знайти найбільш точні співпадіння. Оскільки шингли - це перекриваючі n-грами, навіть якщо частина документа змінилася, значна частина шинглів залишиться незмінною. Метод дає змогу дозволяє прискорити процес порівняння файлів, адже перевіряються не самі тексти, а їх хеш-представлення(шингли). Використання шинглів забезпечує більш точне виявлення локальних подібностей у текстах, навіть якщо вони мають різну структуру або порядок слів. Це важливо для текстів, що можуть бути переписані або змінені. У цього методу є такі недоліки: що він погано працює з невеликими файлами, не є самостійним (потрібен алгоритм для обраховування подібності шинглів), не запам’ятовує структуру тексту. Сам по собі, окремо від інших, цей метод, як і Алгоритм Левенштейна не буде корисним у контексті заданого ПЗ.

Отже, окремо від інших алгоритмів, алгоритм Левенштейна та Шинглінгл не зможуть задовільнити вимоги створюваного ПЗ. Але їх недоліки, через які

вони не можуть бути корисними по одинці, можна нівелювати. І зробити це можна просто поєднавши ці два алгоритми в одному ПЗ.

Так як алгоритм Левенштейна краще працює з короткими рядками, він ідеально поєднується з Шинглінгом, тому що може перевіряти короткі шингли окремо, а не весь текст цілком. Один з головних недоліків Шинглінгу – низька ефективність при роботі з текстами малого обсягу, і шляхом поєднання з алгоритмом Левенштейна, який навпаки має високу ефективність при такій умові, цей недолік нівелюється. Ці алгоритми доповнюють один одного, не заважають один одному функціонувати, не створюють зайвого навантаження, а найголовніше – підвищують ефективність роботи. Ця комбінація алгоритмів є найбільш ефективним рішенням для використання при роботі з текстами різних розмірів та їх різної кількості.

Вибір алгоритмів шинглів та Левенштейна для порівняння текстових файлів є обґрунтованим завдяки їхній здатності ефективно обробляти великі обсяги даних, точності виявлення подібностей, гнучкості та простоті реалізації. Ці алгоритми дозволяють створити надійне програмне забезпечення для оцінки подібності файлів, що є основною метою даного дипломного проекту, а також задовольняють всі вимоги, які поставлені перед ПЗ:

-велика швидкість обчислення та порівняння великої кількості файлів, що буде досягнута за рахунок синергії двох алгоритмів, що нівелюють головні недоліки один одного.

-простота у використанні та обробці отриманих даних, які будуть подаватися у вигляді відсоткового співвідношення кожної пари файлів.

-доволі висока точність знаходження плагіату за рахунок поєднання двох алгоритмів, які безпосередньо допомагають один одному в обчисленнях.

2.3 Опис обраних алгоритмів.

У цьому розділі буде детально розглянуто два алгоритми, що використовуються для порівняння текстових файлів: алгоритм шинглів та алгоритм Левенштейна. Ці алгоритми є важливими інструментами для

визначення схожості між текстами, і кожен з них має свої особливості та області застосування.

2.3.1 Алгоритм шинглів.

Опис алгоритму: Алгоритм шинглів, також відомий як метод n-грам, розбиває текст на підрядки фіксованої довжини (шингли). Кожен шингл представляє собою послідовність із n символів або слів, що дозволяє порівнювати тексти на основі частково збігаючихся підрядків.

Основні кроки алгоритму:

1) Розбиття тексту на шингли:
    - Для заданого тексту створюються всі можливі підрядки довжини n.
    - Наприклад, для тексту "example" при n=3 отримаємо шингли: "exa", "xam", "amp", "mpl", "ple".
2) Побудова множин шинглів:
    - Для кожного тексту будується множина шинглів.
3) Порівняння множин шинглів:
    - Вимірюється схожість між множинами шинглів двох текстів за допомогою коефіцієнта Жаккара:

$J(A,B)=|A\cap B||A\cup B|J(A,B)=|A\cup B||A\cap B|$

- Де AA і BB - множини шинглів двох текстів.

Переваги:

- Простота реалізації.
- Ефективність для великих текстових масивів.
- Висока точність у виявленні дублікатів або схожих документів.

2.3.2 Алгоритм Левенштейна.

Опис алгоритму: Алгоритм Левенштейна (Levenshtein Distance), також відомий як відстань редагування, визначає мінімальну кількість операцій,

необхідних для перетворення одного рядка в інший. Ці операції включають вставку, видалення та заміну символів.

Основні кроки алгоритму:

1) Ініціалізація матриці:
    - Створюється матриця розміром $(m+1)\times(n+1)(m+1)\times(n+1)$, де mm та nn - довжини рядків.
    - Початкові значення встановлюються так, щоб представляти кроки для перетворення порожнього рядка в рядок з іншим текстом.
2) Заповнення матриці:
    - Для кожного символу в першому рядку порівнюється з кожним символом у другому рядку.
    - Визначається вартість операцій (вставка, видалення, заміна) і обирається мінімальна вартість.
3) Обчислення відстані:
    - Значення в нижньому правому куті матриці представляє відстань Левенштейна між двома рядками.

Переваги:

- Точне вимірювання відмінностей між текстами.
- Можливість визначення не тільки кількості, але й типу необхідних операцій для перетворення одного тексту в інший.

Приклад обчислення відстані Левенштейна: Розглянемо два рядки: "kitten" і "sitting". Мінімальна кількість операцій для перетворення "kitten" в "sitting" дорівнює 3 (заміна 'k' на 's', заміна 'e' на 'i', вставка 'g').

2.4 Розробка теоретичних моделей.

У цьому підрозділі буде розглянуто теоретичні моделі для порівняння текстових файлів, що поєднують алгоритм шинглів та алгоритм Левенштейна, а також описано концепцію, математичні основи та процес порівняння текстів з використанням цих алгоритмів.

2.4.1 Концепція моделі.

Основна ідея розробленої моделі полягає у створенні шинглів (підрядків фіксованої довжини) для кожного текстового файлу, після чого ці шингли порівнюються між собою за допомогою алгоритму Левенштейна для визначення їхньої схожості. Результати порівняння шинглів обробляються, щоб отримати загальний відсоток подібності між текстовими файлами.

2.4.2 Математичні основи.

Розбиття на шингли:

- Кожен текст TT розбивається на множину шинглів $S(T)S(T)$ певної довжини nn. Наприклад, текст "приклад тексту" при n=3n=3 буде представлений шинглами ["при", "рик", "икл", "кла", "лад", "ад ", "д т", " те", "тек", "екс", "кст", "сту"].

Відстань Левенштейна:

- Для двох шинглів AA і BB відстань Левенштейна $L(A,B)L(A,B)$ визначається як мінімальна кількість операцій (вставка, видалення, заміна), необхідних для перетворення одного шингла в інший.

Схожість між шинглами:

- Відстань Левенштейна між шинглами використовується для обчислення їхньої схожості. Чим менша відстань, тим більша схожість між шинглами.

Обчислення загальної схожості текстів:

- Загальна схожість текстів $T1T1$ і $T2T2$ обчислюється шляхом знаходження середньої схожості між усіма можливими парами шинглів з $S(T1)S(T1)$ і $S(T2)S(T2)$. Для цього використовуються такі кроки:
    1) Створення множини шинглів $S(T1)S(T1)$ і $S(T2)S(T2)$.
    2) Обчислення відстані Левенштейна для кожної пари шинглів.
    3) Визначення найбільш схожих шинглів між множинами $S(T1)S(T1)$ і $S(T2)S(T2)$.

4) Обчислення середнього значення схожості для всіх шинглів і виведення результату у відсотках.

2.5 Аналіз точності та продуктивності.

У цьому розділі буде розглянуто теоретичний аналіз точності та продуктивності алгоритмів шинглів та Левенштейна. Будуть охарактеризовані основні фактори, що впливають на якість та швидкість роботи алгоритмів, а також наведені приклади їхнього застосування.

2.5.1 Теоретична точність.

Алгоритм шинглів:

- Переваги:

Алгоритм шинглів дозволяє розділити текст на підрядки фіксованої довжини, що забезпечує стійкість до дрібних змін у тексті (додавання, видалення, заміна символів). Він ефективний для виявлення загальної схожості між текстами на рівні фраз або речень.

- Обмеження:

Основним недоліком є чутливість до перестановок і структурних змін, оскільки шингли не враховують порядок появи слів чи підрядків. Точність алгоритму залежить від вибраної довжини шинглів: занадто короткі шингли можуть давати багато хибних спрацьовувань, а занадто довгі можуть пропустити дрібні зміни. Тому велику роль грає правильно підібрана довжина шинглів, при якій результати будуть максимально точними.

Алгоритм Левенштейна:

- Переваги:

Алгоритм Левенштейна визначає мінімальну кількість операцій (вставка, видалення, заміна), необхідних для перетворення одного рядка в інший, що дозволяє точно вимірювати схожість між текстами на рівні символів. Він ефективний для виявлення точних змін і може використовуватися для порівняння шинглів.

- Обмеження:

Висока обчислювальна складність для великих текстів або великих наборів шинглів, що може призвести до зниження продуктивності. Алгоритм не враховує семантичну схожість текстів, зосереджуючись лише на синтаксичних змінах.

2.5.2 Теоретична продуктивність.

Алгоритм шинглів:

- Складність:

Створення шинглів з тексту має лінійну складність $O(n)O(n)$, де nn – довжина тексту. Це означає, що час обробки зростає лінійно з довжиною тексту, що є ефективним для великих обсягів даних.

- Використання пам'яті:

Кількість створених шинглів пропорційна довжині тексту та вибраній довжині шингла. Це може призвести до значного споживання пам'яті при роботі з великими текстами або малими шинглами.

Алгоритм Левенштейна:

- Складність:

Обчислювальна складність алгоритму Левенштейна для двох рядківдовжини mm і nn становить $O(m×n)O(m×n)$. Для порівняння великих текстів це може бути обчислювально витратним, особливо при великій кількості порівнянь шинглів.

- Використання пам'яті:

Алгоритм вимагає зберігання матриці розміру $m×nm×n$, що може призвести до значного споживання пам'яті при порівнянні довгих рядків або великої кількості шинглів.

2.5.3 Теоретичні фактори, що впливають на точність та продуктивність.

Довжина шинглів:

- Вплив на точність:

Вибір довжини шинглів критично впливає на точність порівняння. Короткі шингли можуть виявляти більше схожостей, включаючи випадкові збіги, тоді як довші шингли більш стійкі до випадкових змін, але можуть пропустити дрібні редагування.

- Вплив на продуктивність:

Довші шингли зменшують кількість підрядків для порівняння, що підвищує продуктивність, але можуть збільшувати час створення шинглів.

Розмір текстів:

- Вплив на точність:

Для довгих текстів точність алгоритму шинглів може знижуватися через збільшення кількості випадкових збігів. Алгоритм Левенштейна зберігає високу точність незалежно від розміру тексту.

- Вплив на продуктивність:

Продуктивність обох алгоритмів знижується з ростом розміру текстів, але алгоритм Левенштейна є більш чутливим до зростання обсягу даних через квадратичну складність. Але за рахунок того, що алгоритм працює лише з шинглами, його продуктивність не буде погіршуватися зі збільшенням розміру тексту.

Алгоритмічні оптимізації:

- Паралельне обчислення:

Використання паралельних обчислень для порівняння шинглів дозволяє значно підвищити продуктивність, особливо при великих наборах даних.

- Фільтрація та індексація:

Попереднє фільтрування та індексація шинглів може зменшити кількість порівнянь, підвищуючи продуктивність без значної втрати точності.

2.5.4 Висновок.

Теоретичний аналіз точності та продуктивності алгоритмів шинглів та Левенштейна показує, що їх поєднання дозволяє досягти балансу між точністю та продуктивністю при порівнянні текстових файлів. Алгоритм шинглів

забезпечує стійкість до дрібних змін, тоді як алгоритм Левенштейна дозволяє точно вимірювати схожість на рівні символів.

2.6 Визначення вимог та постановка задачі для подальшої розробки програмного забезпечення.

Вимоги:

- ПЗ повинне експортувати текст з WORD-файлів (без врахування форматування, тільки текст, щоб нічого не заважало подальшим крокам) у програму для подальших обчислень.
- ПЗ повинне мати можливість обробляти декілька файлів .docx одночасно.
- ПЗ повинне за допомогою метода Шинглінга розподіляти експортований текст з файлів на шингли
- ПЗ повинне реалізувати алгоритм Shingle-Levenstein для обчислення подібності між шинглами текстових файлів.
- ПЗ повинне знаходити середнє значення подібності шинглів текстових файлів для подальшого переведення у відсоткову систему подібності.
- ПЗ повинне обчислювати відсоток плагіату між кожною парою файлів (між двома списками шинглів).
- Програма повинна виводити результати у зручному для аналізу форматі.

Задача:

Розроблення програми для виявлення плагіату серед великої кількості текстових файлів формату .docx., яка буде задовольняти всі вищевказані вимоги.

2.7 Проєктування архітектури обраного алгоритму.

Були обрані алгоритми – шинглінг та алгоритм левенштейна. Спроєктуємо архітектуру алгоритму, що буде поєднувати ці два методи:

Вхідний рівень:

Вхідними даними є набір текстових файлів у форматі .docx, розташованих у вказаній директорії(папці).

Програма використовує клас File та інтерфейс FilenameFilter з стандартної бібліотеки Java для знаходження всіх файлів .docx у директорії

Рівень обробки тексту:

-Вилучення тексту: Для кожного знайденого файлу викликається метод extractText(), який використовує бібліотеку Apache POI для вилучення тексту з файлів .docx.

-Токенізація тексту: Метод tokenize() розбиває вилучений текст на токени (слова) за допомогою правил токенізації, які враховують літери та цифри.

Рівень обчислення шинглів:

-Метод computeShingles() отримує токенізований текст і обчислює множину шинглів для цього тексту.

-Шингли формуються як послідовності фіксованого розміру SHINGLE_SIZE (у вашому випадку 5) послідовних токенів.

-Цей компонент відповідає за створення основних одиниць порівняння - шинглів.

Рівень обчислення подібності:

-Обчислення подібності множин шинглів: Метод computeSimilarity() отримує дві множини шинглів, що відповідають двом текстовим файлам, і обчислює значення подібності між цими множинами, використовуючи метод computeStringDistance().Кінцева подібність обчислюється на основі середнього значення подібності пар шинглів і кількості шинглів у меншому з текстів.

-Обчислення подібності пар шинглів: Метод computeStringDistance() реалізує алгоритм Левенштейна для обчислення відстані редагування між двома рядками (шинглами).

2.8 Теоретична побудова структури і архітектури програми.

Структура програми:

Вхідні дані та обробка файлів:

1) Програма отримує вхідними даними набір текстових файлів формату .docx, розміщених у вказаній директорії.

2) Клас ShingleLevenstain містить головний метод main(), який виконує наступні операції:

-Знаходить всі файли .docx у вказаній директорії за допомогою класу File та інтерфейсу FilenameFilter.

-Для кожного знайденого файлу викликає метод extractText() для вилучення тексту з файлу.

-Зберігає вилучений текст та імена файлів у відповідних списках.

Обробка тексту:

1) Для вилучення тексту з файлів .docx використовується бібліотека Apache POI.
2) Метод extractText() створює об'єкт XWPFDocument з вмістом файлу та використовує клас XWPFWordExtractor для вилучення тексту.
3) Метод tokenize() розбиває вилучений текст на токени (слова) за допомогою правил токенізації, які враховують літери та цифри.

Обчислення шинглів:

1) Метод computeShingles() отримує токенізований текст і обчислює множину шинглів для цього тексту.
2) Шингли формуються як послідовності фіксованого розміру SHINGLE_SIZE (обирається довільно) послідовних токенів.

Обчислення подібності:

1) Метод computeSimilarity() отримує дві множини шинглів, що відповідають двом текстовим файлам, і обчислює значення подібності між цими множинами.
2) Метод computeStringDistance() реалізує алгоритм Левенштейна для обчислення відстані редагування між двома рядками (шинглами).
3) Під час обчислення подібності у методі computeSimilarity() використовується алгоритм Левенштейна для знаходження найбільш схожих шинглів між двома множинами.

4) Кінцева подібність обчислюється на основі середнього значення подібності пар шинглів і кількості шинглів у меншому з текстів.

Вихідні дані:

1) У головному методі main() викликаються всі попередні компоненти для обробки пар файлів.
2) Для кожної пари файлів виводиться відсоток плагіату між ними, який обчислюється на основі значення подібності між їхніми множинами шинглів.

Архітектура програми складається з декількох рівнів абстракції:

1)Рівень введення-виведення (main()):

відповідає за взаємодію з користувачем, читання вхідних файлів і виведення результатів.

2)Рівень підготовки даних (extractText(), tokenize()):

відповідає за підготовку текстових даних для подальшої обробки.

3)Рівень обчислень (computeShingles(), computeSimilarity(), computeStringDistance()):

реалізує основну логіку алгоритму Shingle-Levenstein та обчислення подібності між текстовими файлами.

2.9 Технології та інструменти, що будуть використані для подальшого створення програмного забезпечення.

У даній програмі будуть використовуватися такі технології та інструменти:

1) Java: Основна мова програмування, що використовується для реалізації алгоритму. Це дозволяє забезпечити кросплатформенність та використовувати стандартну бібліотеку Java (Java Standard Edition).
2) Стандартна бібліотека Java (Java Standard Edition)

   У програмі використовуються класи та інтерфейси з різних пакетів стандартної бібліотеки Java, зокрема:

   -java.io для роботи з файлами та потоками введення-виведення.

-java.util для роботи зі структурами даних, такими як списки (ArrayList) та множини (HashSet).

3) Apache POI (Poor Obfuscation Implementation)

   -Програма використовує бібліотеку Apache POI для роботи з файлами Microsoft Office, зокрема для вилучення тексту з файлів .docx.

   -У коді імпортуються та використовуються класи XWPFWordExtractor та XWPFDocument з пакету org.apache.poi.xwpf.usermodel та org.apache.poi.xwpf.extractor

4) Інтегроване середовище розробки (IDE)

   Для розробки та компіляції програми, ймовірно, використовувалось одне з популярних IDE для Java, а саме IntelliJ IDEA.

3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Програмна реалізація алгоритмів.

3.1.1 Алгоритм токенізації тексту tokenize().

Метод tokenize() (рисунок 3.1) відповідає за розбиття тексту на окремі токени (слова).

```
    private static List<String> tokenize(String text) {
        List<String> tokens = new ArrayList<>();
        StringBuilder token = new StringBuilder();

        for (int i = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (Character.isLetterOrDigit(c)) {
                token.append(Character.toLowerCase(c));
            } else {
                if (token.length() > 0) {
                    tokens.add(token.toString());
                    token = new StringBuilder();
                }
            }
        }

        if (token.length() > 0) {
            tokens.add(token.toString());
        }

        return tokens;
    }
}
```

Рисунок 3.1 - Реалізація алгоритму токенізації

Правила токенізації:

-Метод tokenize() проходить циклом по кожному символу в тексті.

-Якщо символ є літерою або цифрою (перевіряється за допомогою Character.isLetterOrDigit(c)), він додається до поточного токена в нижньому регістрі (за допомогою Character.toLowerCase(c)).

-Якщо символ не є літерою або цифрою, то поточний токен вважається завершеним. Якщо він не порожній, він додається до списку токенів tokens.

-Після циклу, якщо останній токен не порожній, він також додається до списку токенів.

Використання структур даних:

1) List<String> tokens:

-Список tokens типу ArrayList використовується для зберігання всіх токенів, що були згенеровані з тексту.

-ArrayList є динамічним масивом, що дозволяє ефективно додавати та видаляти елементи з будь-якої позиції.

2) StringBuilder token:

-StringBuilder використовується для побудови поточного токена, оскільки він є більш ефективним, ніж конкатенація рядків.

-Клас StringBuilder призначений для ефективної маніпуляції рядками, особливо при часто виконуваних операціях додавання, видалення або заміни символів.

Використання ArrayList для зберігання токенів забезпечує ефективне додавання та видалення елементів, а також динамічне розширення розміру списку за потреби. Використання StringBuilder для формування токенів забезпечує ефективне поєднання символів в рядок, уникаючи надмірних витрат пам'яті та часу на операції конкатенації рядків.

Завдяки цим структурам даних метод tokenize() може ефективно розбивати текст на токени, забезпечуючи хорошу продуктивність та масштабованість для обробки великих обсягів тексту.

3.1.2 Алгоритм обчислення шинглів computeShingles().

Метод computeShingles (рисунок 3.2) у програмі відповідає за обчислення множини шинглів для заданого тексту.

```
private static Set<String> computeShingles(String text) {
    Set<String> shingles = new HashSet<>();
    List<String> tokens = tokenize(text);

    for (int i = 0; i <= tokens.size() - SHINGLE_SIZE; i++) {
        StringBuilder shingle = new StringBuilder();
        for (int j = 0; j < SHINGLE_SIZE; j++) {
            shingle.append(tokens.get(i + j)).append(" ");
        }
        shingles.add(shingle.toString().trim());
    }

    return shingles;
```

Рисунок 3.2 - Реалізація алгоритму обчислення шинглів

Деталі формування шинглів як послідовностей SHINGLE_SIZE послідовних токенів:

1) Спочатку метод викликає метод tokenize() для розбиття вхідного тексту на список токенів (слів).
2) Потім він ітеративно проходить через список токенів, використовуючи вкладений цикл.
3) Для кожної можливої позиції в списку токенів створюється новий рядок shingle (за допомогою StringBuilder).
4) До рядка shingle додаються наступні SHINGLE_SIZE токенів із поточної позиції, розділені пробілами.
5) Після створення рядка shingle він додається до множини shingles (без початкових і кінцевих пробілів за допомогою trim()).
6) Цикл продовжується до тих пір, поки не будуть згенеровані всі можливі шингли заданого розміру SHINGLE_SIZE.

Використання структур даних для ефективного зберігання шинглів:

У методі computeShingles() використовується HashSet (рисунок 3.3) для зберігання множини шинглів:

```
Set<String> shingles = new HashSet<>();
```

Рисунок 3.3 - Реалізація збереження множини шинглів з використанням хеш-таблиці

HashSet є реалізацією інтерфейсу Set з використанням хеш-таблиці для зберігання елементів. Це забезпечує такі переваги:

1) Відсутність дублікатів: HashSet гарантує, що кожен елемент (у цьому випадку шингл) присутній у множині лише один раз.

2) Швидкий пошук та вставка: Операції пошуку та вставки елементів у HashSet мають середню складність O(1), що робить їх дуже ефективними для великих наборів даних.

3) Динамічне розширення: Як і інші колекції Java, HashSet автоматично розширюється при додаванні нових елементів, уникаючи необхідності явного виділення пам'яті.

Використання HashSet для зберігання шинглів забезпечує ефективне усунення дублікатів та швидкий доступ до множини шинглів, що є важливим для подальших операцій порівняння множин у алгоритмі виявлення плагіату.

Загалом, метод computeShingles() ефективно генерує множину шинглів із заданого тексту, використовуючи токенізацію та ітеративно формуючи шингли як послідовності SHINGLE_SIZE послідовних токенів. Використання HashSet забезпечує ефективне зберігання та обробку множини шинглів, уникаючи дублікатів та забезпечуючи швидкий доступ.

3.1.3 Алгоритм Левенштейна для обчислення відстані редагування. computeStringDistance()

Метод computeStringDistance() у програмі реалізує алгоритм Левенштейна (Рисунок 3.4) для обчислення відстані редагування між двома рядками (шинглами):

```
private static double computeStringDistance(String s1, String s2) {
    int[] costs = new int[s2.length() + 1];
    for (int j = 0; j < costs.length; j++) {
        costs[j] = j;
    }

    for (int i = 1; i <= s1.length(); i++) {
        costs[0] = i;
        int nw = i - 1;
        for (int j = 1; j <= s2.length(); j++) {
            int cj = Math.min(1 + Math.min(costs[j], costs[j - 1]), s1.charAt(i - 1) == s2.charAt(j - 1) ? nw : nw + 1);
            nw = costs[j];
            costs[j] = cj;
        }
    }

    double distance = costs[s2.length()];
    double maxLength = Math.max(s1.length(), s2.length());
    return 1.0 - distance / maxLength;
}
```

Рисунок 3.4 - Реалізація алгоритму Левенштейна для обчислення відстані редагування

Деталі обчислення відстані редагування між двома рядками (шинглами):

Алгоритм Левенштейна обчислює найменшу кількість операцій редагування (вставок, видалень або заміни символів), необхідних для перетворення одного рядка в інший. У цьому методі реалізовано динамічне програмування для ефективного обчислення відстані редагування між двома рядками s1 та s2.

1) Спочатку створюється масив costs розміром s2.length() + 1, який зберігатиме проміжні значення відстаней редагування.
2) Перший рядок масиву ініціалізується значеннями 0, 1, 2, ..., s2.length(), що відповідає відстані редагування між порожнім рядком та s2.
3) Потім, за допомогою двох вкладених циклів, обчислюється відстань редагування для решти комбінацій префіксів рядків s1 та s2.

4) На кожному кроці циклу обчислюється мінімальна вартість з трьох можливих операцій: вставки, видалення або заміни символу.
5) Кінцева відстань редагування зберігається в costs[s2.length()].

Використання динамічного програмування та матриці для ефективного обчислення відстані:

Метод computeStringDistance() використовує техніку динамічного програмування для ефективного обчислення відстані редагування між рядками. Це досягається за допомогою матриці costs, в якій зберігаються проміжні результати обчислень.

Динамічне програмування дозволяє уникнути повторних обчислень і значно підвищити ефективність алгоритму. Замість того, щоб обчислювати відстані редагування для всіх можливих підрядків, метод використовує попередньо обчислені значення, зберігаючи їх у матриці costs.

Використання матриці costs забезпечує ефективний доступ до проміжних результатів та їх збереження. Це дозволяє уникнути повторних обчислень і значно прискорити алгоритм, особливо для довгих рядків.

Загалом, метод computeStringDistance() реалізує класичний алгоритм Левенштейна з використанням динамічного програмування та матриці для ефективного обчислення відстані редагування між рядками. Це є ключовим компонентом для обчислення подібності між шинглами в алгоритмі виявлення плагіату.

3.1.4 Алгоритм обчислення подібності між множинами шинглів.

Метод computeSimilarity(Set<String> set1, Set<String> set2) обчислює міру подібності між двома множинами шинглів set1 і set2 (Рисунок 3.5).

Він працює наступним чином:

1) Визначається менша і більша множини за розміром, присвоюючи їх змінним smallerSet і largerSet відповідно.

2) Ініціалізується змінна similarity, яка зберігатиме підсумкову міру подібності.

3) Кожен шингл з меншої множини smallerSet порівнюється з шинглами більшої множини largerSet:

- Якщо шингл зі smallerSet наявний у largerSet, то значення similarity збільшується на 1.

```
private static double computeSimilarity(Set<String> set1, Set<String> set2) {

    Set<String> smallerSet, largerSet;
    if (set1.size() < set2.size()) {
        smallerSet = set1;
        largerSet = set2;
    } else {
        smallerSet = set2;
        largerSet = set1;
    }

    double similarity = 0.0;

    for (String shingle : smallerSet) {
        if (largerSet.contains(shingle)) {
            similarity++;
        } else {
            double maxSimilarity = 0.0;
            for (String other : largerSet) {
                double shingleSimilarity = computeStringDistance(shingle, other);
                if (shingleSimilarity > maxSimilarity) {
                    maxSimilarity = shingleSimilarity;
                }
            }
            similarity += maxSimilarity;
        }
    }

    return similarity / smallerSet.size();
}
```

Рисунок 3.5 – Реалізація алгоритму обчислення подібності між множинами шинглів

- Якщо шингл зі smallerSet відсутній у largerSet, то обчислюється найбільш схожий шингл з largerSet за допомогою методу computeStringDistance(), який використовує алгоритм Левенштейна для обчислення відстані редагування між двома рядками. Найбільша знайдена подібність додається до similarity.

4) Після перебору всіх шинглів зі smallerSet, кінцева міра подібності обчислюється як similarity / smallerSet.size().

Формулу обчислення кінцевої подібності можна записати наступним чином:

similarity = (count_of_identical_shingles + sum_of_max_similarities) / size_of_smaller_set

Де:

count_of_identical_shingles - кількість абсолютно ідентичних шинглів між множинами

sum_of_max_similarities - сума найбільших подібностей для неідентичних шинглів, обчислених за допомогою алгоритму Левенштейна

size_of_smaller_set - розмір меншої з двох порівнюваних множин шинглів

Таким чином, метод computeSimilarity() враховує як ідентичні шингли, так і найбільш подібні шингли між множинами, використовуючи алгоритм Левенштейна для знаходження останніх.

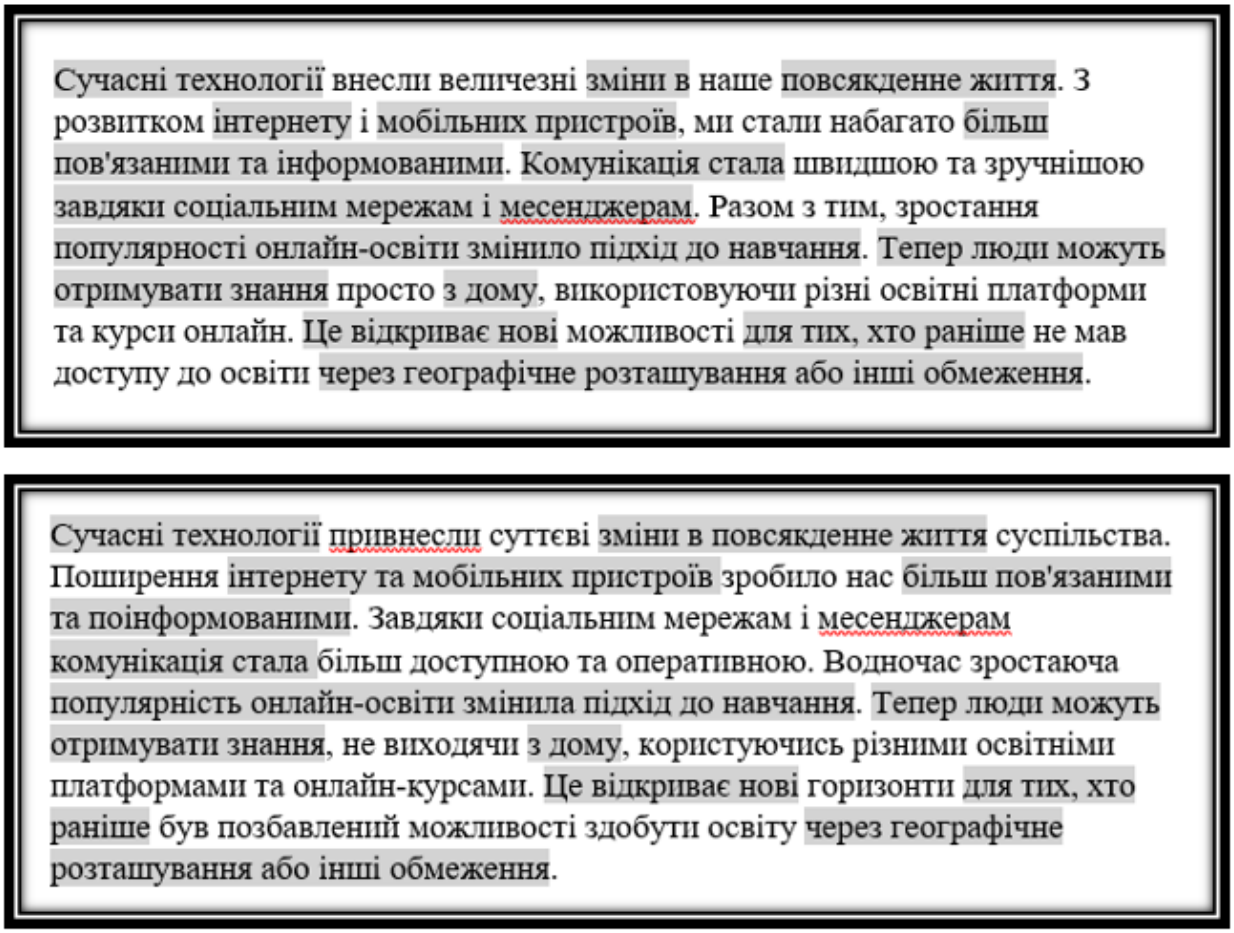
3.2 Тестування та валідація.

Тестування самого алгоритму «Shingle-Levenstain» на правильність результату в порівнянні з декількома іншими алгоритмами:

Було обрано декілька алгоритмів для того, щоб протестувати їх та обрати найкращій і найстабільніший

1) Перевірка між двома файлами(filek1 та filek2), які мають такий зміст (Рисунок 3.6)

Як видно з рисунку, файли дуже схожі, більша частина тексту співпадає, тож очікуваним результатом від алгоритмів буде відсоток подібності приблизно 60-70%, оскільки різниця між реченнями також присутня. Перевіримо на практиці як алгоритми оцінюють цю подібність (Рисунок 3.7).



Сучасні технології внесли величезні зміни в наше повсякденне життя. З розвитком інтернету і мобільних пристроїв, ми стали набагато більш пов'язаними та інформованими. Комунікація стала швидшою та зручнішою завдяки соціальним мережам і месенджерам. Разом з тим, зростання популярності онлайн-освіти змінило підхід до навчання. Тепер люди можуть отримувати знання просто з дому, використовуючи різні освітні платформи та курси онлайн. Це відкриває нові можливості для тих, хто раніше не мав доступу до освіти через географічне розташування або інші обмеження.

Сучасні технології привнесли суттєві зміни в повсякденне життя суспільства. Поширення інтернету та мобільних пристроїв зробило нас більш пов'язаними та поінформованими. Завдяки соціальним мережам і месенджерам комунікація стала більш доступною та оперативною. Водночас зростаюча популярність онлайн-освіти змінила підхід до навчання. Тепер люди можуть отримувати знання, не виходячи з дому, користуючись різними освітніми платформами та онлайн-курсами. Це відкриває нові горизонти для тих, хто раніше був позбавлений можливості здобути освіту через географічне розташування або інші обмеження.

Рисунок 3.6 – Зміст «filek1»(верхній текст) і «filek2»(нижній текст), збіжності виділені сірим кольором(надалі цей колір буде використовуватися з ціллю виділення збігів у тексті).

Рисунок 3.7 - Результати порівняння файлів «filek1» та «filek2» між собою за допомогою різних алгоритмів.

У лівій верхній частині видно результат порівняння за допомогою алгоритма Левенштейна (67% подібності), його результат підпадає під мої очікування (60-70%), тож з цією подібністю можна вважати він впорався непогано, що не дивно, враховуючи впорядкованість рядків та слів.

Справа від алгоритму Левенштейна ми бачимо результат роботи алгоритму пошуку найдовшого спільного рядка (76% подібності). Його результат також близький до моїх очікувань, що теж не дивно, враховуючи специфіку текстів, їх подібність та впорядкованість речень і слів.

Зліва знизу зображено результати перевірки алгоритму шинглів на основі порівняння Jaro Winkler'a (85% подібності), як видно, значення подібності є зависоким для цих файлів, тож тут цей алгоритм проявив себе погано.

Справа знизу бачимо алгоритм, що був обраний для створення ПЗ, з цими файлами він впорався добре та виправдав очікування (70% подібності), отже з впорядкованими текстами, що мають багато спільних слів його результати є коректними.

2) Перевірка між файлами «filek3» та «filek4», які мають такий зміст відповідно (Рисунок 3.8)

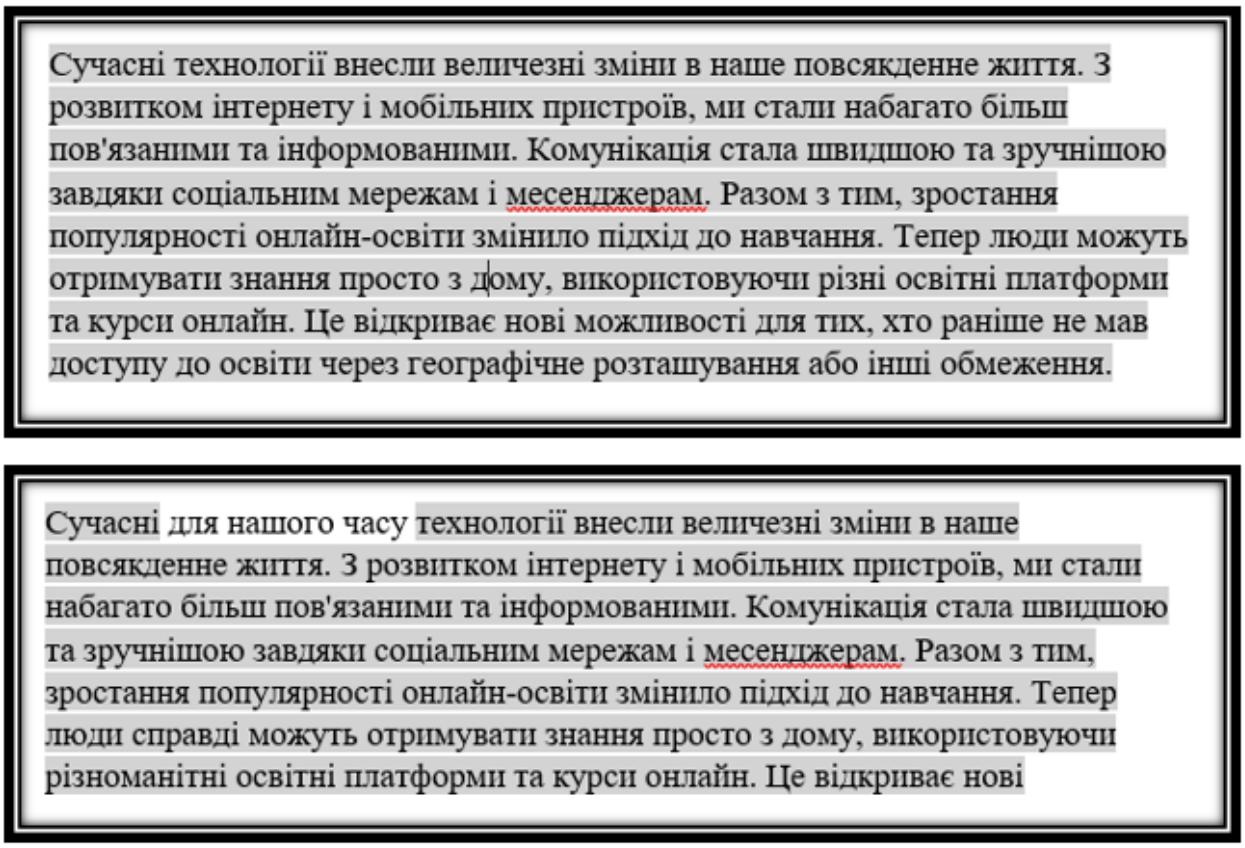


Рисунок 3.8 – зміст файлів «filek3» та «filek4»

Як видно з рисунка, файли майже ідентичні один з одним, різниця тільки в невеликому словосполученні. Очікуваний результат від алгоритмів приблизно 95% подібності, зважаючи на розмір тексту.

Розглянемо результати алгоритмів (Рисунок 3.9). У лівій верхній частині видно результат порівняння за допомогою алгоритма Левенштейна, його результат (94% подібності) знову підпадає під мої приблизні очікування (95% подібності).

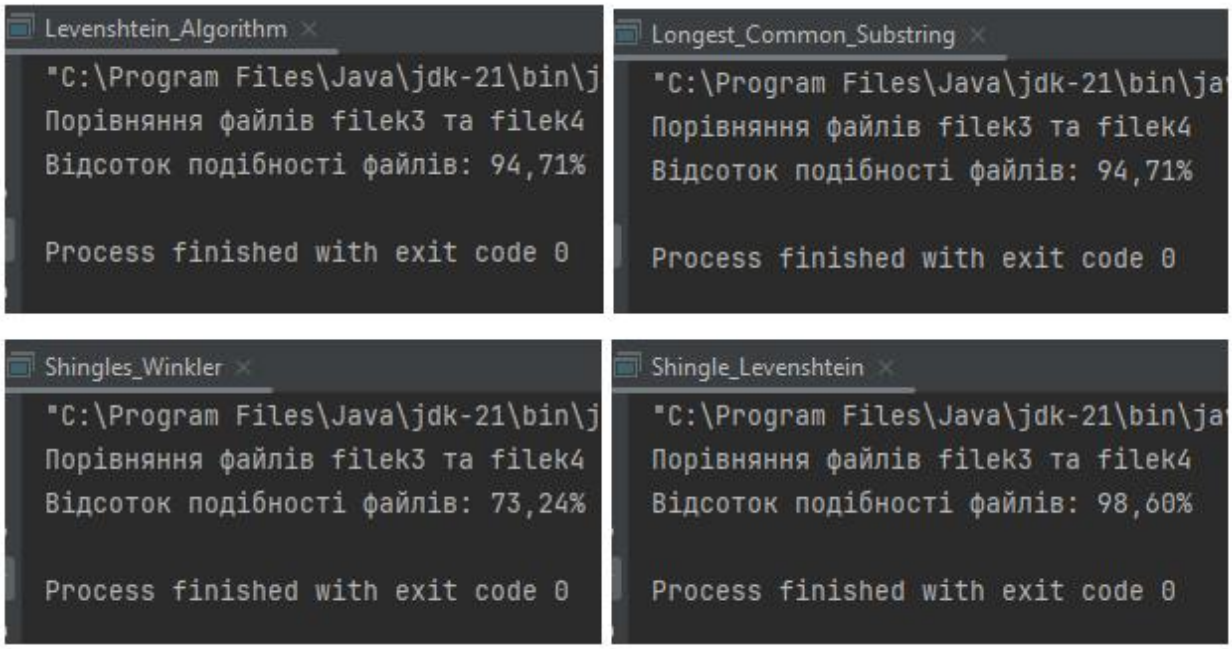


Рисунок 3.9 - Результати порівняння файлів «filek3» та «filek4» між собою за допомогою різних алгоритмів

Результат алгоритму пошуку найдовшого спільного рядка, який знаходиться справа від алгоритму Левенштейна, має ідентичні з ним результати (94% подібності), тож також добре впорався з перевіркою.

Алгоритм шинглів на основі порівняння Jaro Winkler'a, як і на минулій перевірці виявився зовсім не ефективним (73% подібності) в порівнянні з іншими алгоритмами.

Алгоритм «Шингла-Левенштейна», який було обрано для створення ПЗ, впорався з перевіркою добре (98%), та виправдав очікування щодо результатів (95%).

3) Перевірка між файлами «filek6» та «filek7» (Рисунок 3.10)

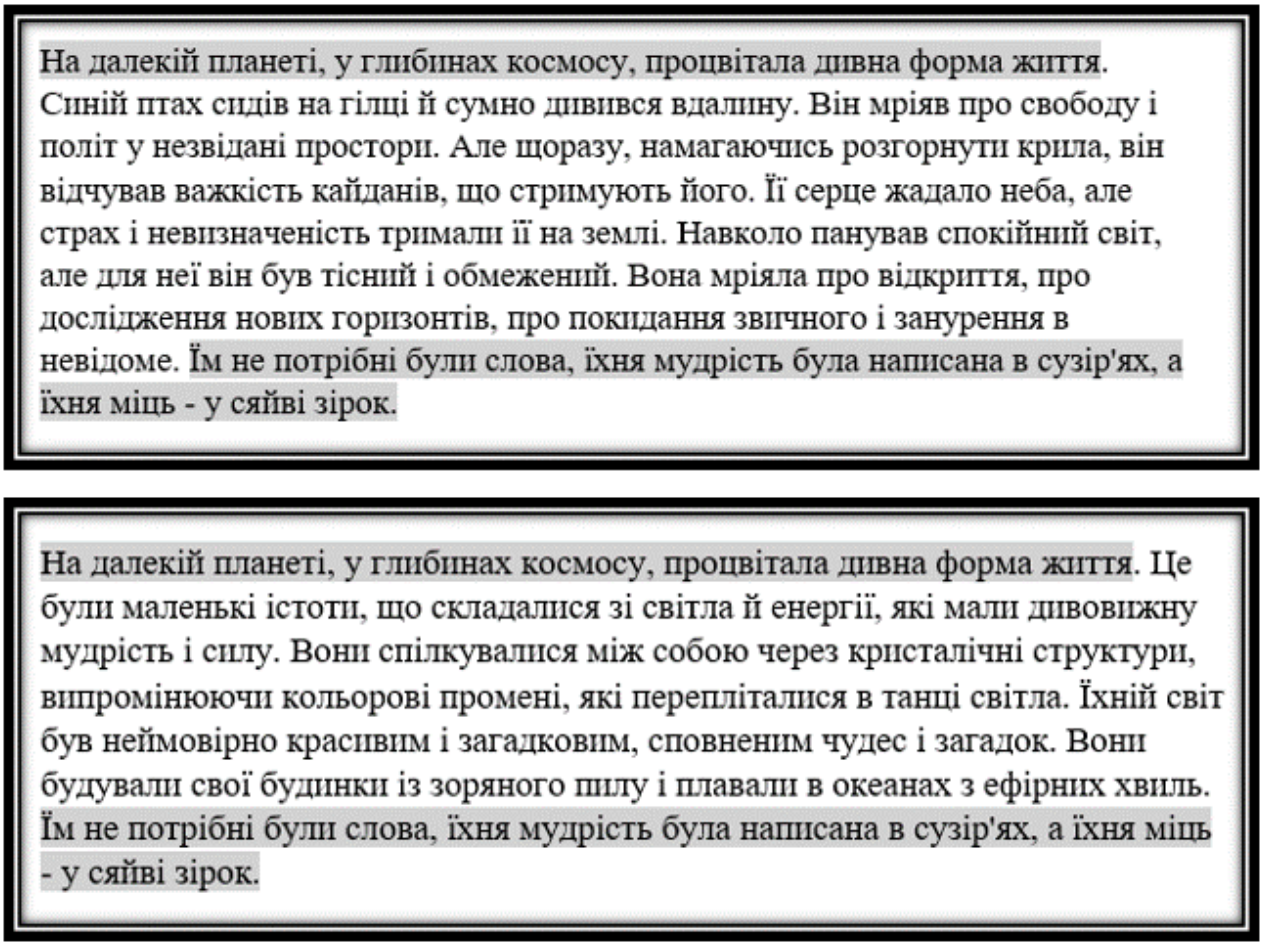


Рисунок 3.10 - зміст файлів «filek6» та «filek7»

Як видно на рисунку, тексти мають різний зміст, але однакові перше та останнє речення. Очкуваний результат від алгоритмів – приблизно 40-50%. Подивимось на практичні результати (Рисунок 3.11).

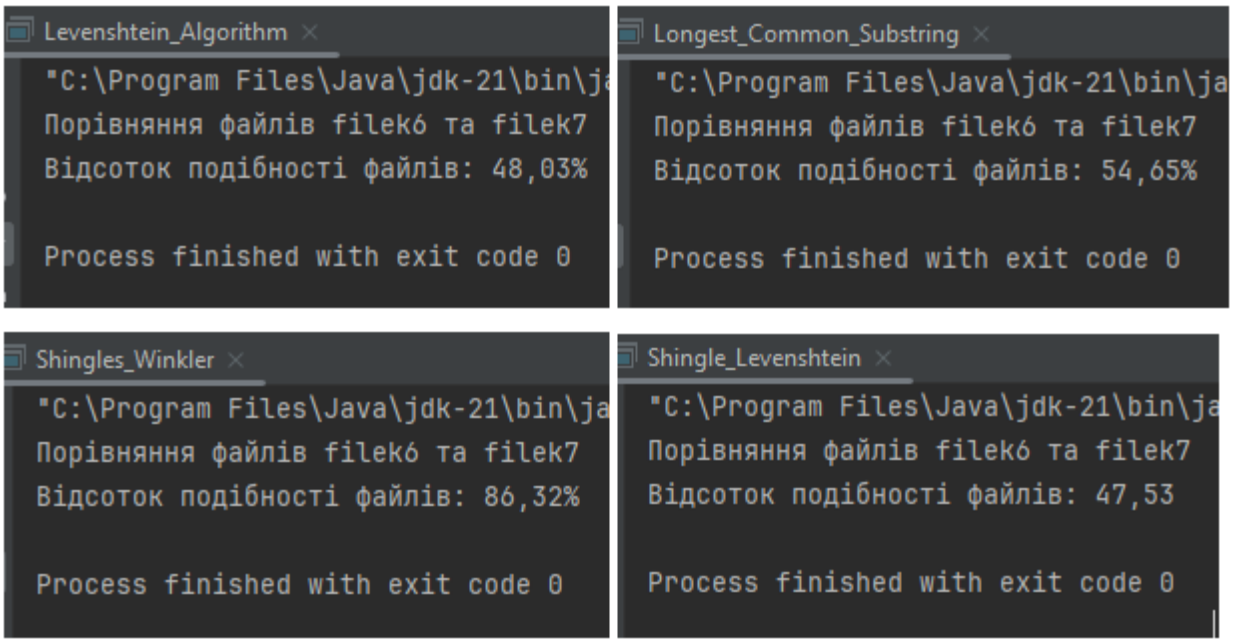


Рисунок 3.11 - Результати порівняння файлів «filek6» та «filek7» між собою за допомогою різних алгоритмів

Алгоритм Левенштейна отримав доволі близький до очікуваного відсоток подібності (48%), отже добре впорався з роботою.

Результат алгоритму пошуку найдовшого спільного рядка, який знаходиться справа від алгоритму Левенштейна, має вищий відсоток (54%), який перевищує очікуваний результат, тож цей алгоритм не є ефективним в даному випадку.

Алгоритм шинглів на основі порівняння Jaro Winkler'a, знову виявився зовсім не ефективним в порівнянні з іншими алгоритмами, отримавши абсурдні результати (86%).

Алгоритм «Шингла-Левенштейна» знову впорався з перевіркою добре (47%), та виправдав очікування щодо результатів.

Як бачимо, алгоритм «Shingle-Levenstein» показує доволі точний результат для файлів з різною схожістю, приблизний до очікуваного, його результат стабільніший та коректніший в порівнянні з іншими алгоритмами(алгоритм левенштейна теж непогано себе проявив, але він також є частиною створеного алгоритму).

Але при перевірці абсолютно різних файлів подбіність обчислена даним алгоритмом може досягати 30% (Рисунок 3.12), і навіть на декілька відсотків більше.

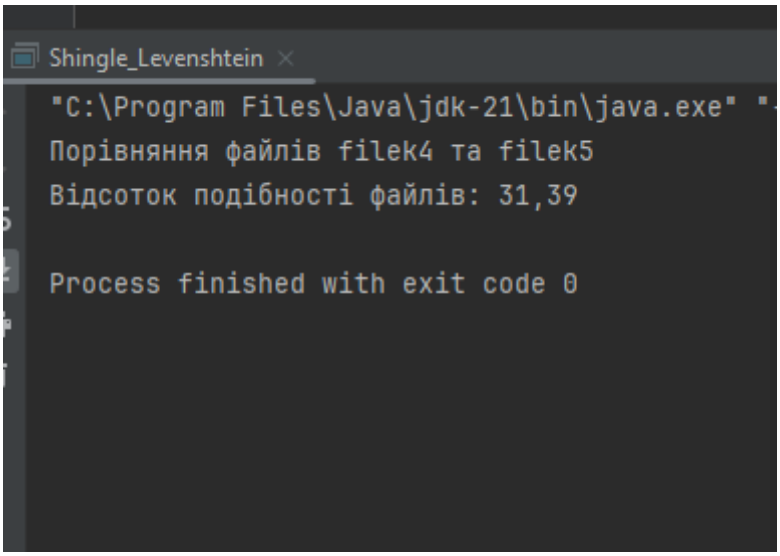


Рисунок 3.12 – Результат перевірки абсолютно різних між собою файлів

Такий результат є наслідком того, що при перевірці подібності алгоритм шукає найбільш схожі один на одного шингли, тому навіть у різних текстах результат такого алгоритму може бути до 40% (зазвичай не вище 35%), потрібно враховувати це при користуванні програмою. Через це відсоток нижче 40 можна вважати мінімальним збігом у контексті такої перевірки.

Тестування алгоритму Shingle-Levenstein для перевірки директорії(папки) з великою кількістю файлів (Рисунок 3.13).

```
Відсоток плагіату між файлами 'filek1' и 'filek9': 90,78%
Відсоток плагіату між файлами 'filek10' и 'filek2': 100,00%
Відсоток плагіату між файлами 'filek10' и 'filek3': 70,37%
Відсоток плагіату між файлами 'filek10' и 'filek4': 70,37%
Відсоток плагіату між файлами 'filek10' и 'filek5': 32,00%
Відсоток плагіату між файлами 'filek10' и 'filek6': 33,71%
Відсоток плагіату між файлами 'filek10' и 'filek7': 31,79%
Відсоток плагіату між файлами 'filek10' и 'filek8': 27,69%
Відсоток плагіату між файлами 'filek10' и 'filek9': 70,37%
Відсоток плагіату між файлами 'filek2' и 'filek3': 70,85%
Відсоток плагіату між файлами 'filek2' и 'filek4': 68,07%
Відсоток плагіату між файлами 'filek2' и 'filek5': 31,23%
Відсоток плагіату між файлами 'filek2' и 'filek6': 33,21%
Відсоток плагіату між файлами 'filek2' и 'filek7': 31,45%
Відсоток плагіату між файлами 'filek2' и 'filek8': 31,55%
Відсоток плагіату між файлами 'filek2' и 'filek9': 67,52%
Відсоток плагіату між файлами 'filek3' и 'filek4': 98,60%
Відсоток плагіату між файлами 'filek3' и 'filek5': 31,25%
Відсоток плагіату між файлами 'filek3' и 'filek6': 32,43%
Відсоток плагіату між файлами 'filek3' и 'filek7': 29,97%
Відсоток плагіату між файлами 'filek3' и 'filek8': 32,43%
Відсоток плагіату між файлами 'filek3' и 'filek9': 90,78%
Відсоток плагіату між файлами 'filek4' и 'filek5': 31,39%
Відсоток плагіату між файлами 'filek4' и 'filek6': 32,40%
Відсоток плагіату між файлами 'filek4' и 'filek7': 30,24%
Відсоток плагіату між файлами 'filek4' и 'filek8': 32,43%
Відсоток плагіату між файлами 'filek4' и 'filek9': 89,30%
Відсоток плагіату між файлами 'filek5' и 'filek6': 30,71%
Відсоток плагіату між файлами 'filek5' и 'filek7': 30,43%
Відсоток плагіату між файлами 'filek5' и 'filek8': 35,23%
Відсоток плагіату між файлами 'filek5' и 'filek9': 31,38%
Відсоток плагіату між файлами 'filek6' и 'filek7': 47,53%
Відсоток плагіату між файлами 'filek6' и 'filek8': 36,50%
Відсоток плагіату між файлами 'filek6' и 'filek9': 32,88%
Відсоток плагіату між файлами 'filek7' и 'filek8': 35,73%
Відсоток плагіату між файлами 'filek7' и 'filek9': 30,23%
Відсоток плагіату між файлами 'filek8' и 'filek9': 32,43%
```

Рисунок 3.13 – Результати перевірки алгоритма «Шингла-Левенштейна» між директорією з 10 файлів, які також були застосовані в попередніх перевірках

Вихідні дані повністю співпадають з даними, отриманими при окремій перевірці файлів даним алгоритмом, отже він працює виправно та доволі точно для змісту наданих текстових файлів, виправдовує всі теоретичні очікування. Як видно з рисунку 3.13, найнижчі показники приблизно 30%, цей результат отримано при повністю або майже повністю різних текстових файлах, отже цей показник можна брати за мінімальний рівень плагіату.

3.3 Поєднання ПЗ і створення інтерфейсу.

Отже, основний алгоритм для ПЗ обрано та реалізовано, доведено до стабільного результату. Наступним кроком буде поєднання створеного ПЗ та ПЗ "ПЗ ДЛЯ ПЕРЕВІРКИ ЯКОСТІ ДИСТАНЦІЙНОГО НАВЧАННЯ" для формування єдиного ПЗ, ціллю якого є автоматизація різних видів взаємодії викладачів з файлами(перевірка якості виділення лекцій, порівняння текстових файлів між собою), а також створення візуального інтерфейсу для цього програмного засобу.

Для того щоб поєднати дві програми в одну зі спільним візуальним інтерфейсом та вибором між тим, яку функцію застосувати, створимо новий проєкт для графічного інтерфейсу, використовуючи бібліотеку «Swing»(Рисунок 3.14). У цьому новому проекті створимо клас, що містить основний метод main і код для ініціалізації графічного інтерфейсу (Рисунок 3.15). У цьому ж класі створимо дві окремі функції (Рисунок 3.16) для запуску кожної з програм (сканування файлів і їх порівняння). У графічному інтерфейсі додамо два елементи, а саме пункти меню, для вибору між двома функціями (Рисунок 3.17). При натисканні на відповідну кнопку або пункт меню буде викликана відповідна функцію для запуску потрібної програми.

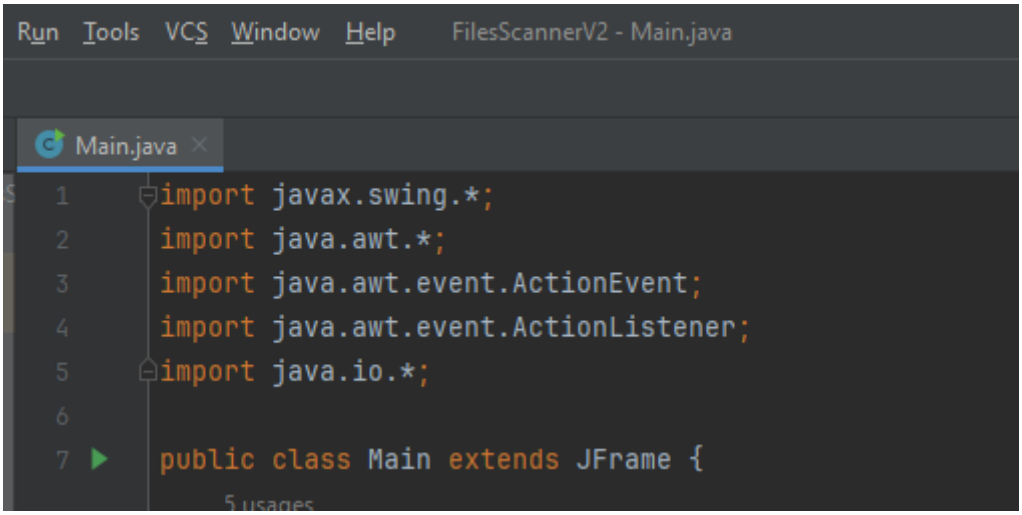


Рисунок 3.14 – Створення проєкту для графічного інтерфейсу з використанням бібліотеки «swing»

```
public class Main extends JFrame {
    5 usages
    private JTextArea outputArea;
    2 usages
    private JScrollPane scrollPane;
    1 usage
    public Main() {
        setTitle("FileScanner");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize( width: 800,  height: 600);
        setLocationRelativeTo(null);


        JMenuBar menuBar = new JMenuBar();
        JMenu menu = new JMenu( s: "Функції");
        menuBar.add(menu);

        JMenuItem scannerItem = new JMenuItem( text: "Сканування лекцій");
        JMenuItem comparisonItem = new JMenuItem( text: "Порівняння файлів");
```

Рисунок 3.15 - Створення класу, що містить основний метод main і код для ініціалізації графічного інтерфейсу

```
private void runScanner() {
    System.out.println("Початок сканування лекцій...");
    LecturesScannerV1 scanner = new LecturesScannerV1();
    redirectOutput(scanner);

}

1 usage
private void runComparison() {
    System.out.println("Початок порівняння файлів...");
    ShingleLevenstain comparison = new ShingleLevenstain();
    redirectOutput(comparison);

}
```

Рисунок 3.16 – Створення двох окремих функцій для запуску кожної з програм (сканування файлів і їх порівняння)

```
scannerItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        runScanner();
    }
});

comparisonItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        runComparison();
    }
});

menu.add(scannerItem);
menu.add(comparisonItem);
setJMenuBar(menuBar);
```

Рисунок 3.17 - Створення двох елементів(пунктів меню) для вибору між двома функціями

Після виконання вказаних дій і підключення наших ПЗ, отримуємо такий результат при запусканні програми (Рисунок 3.18):

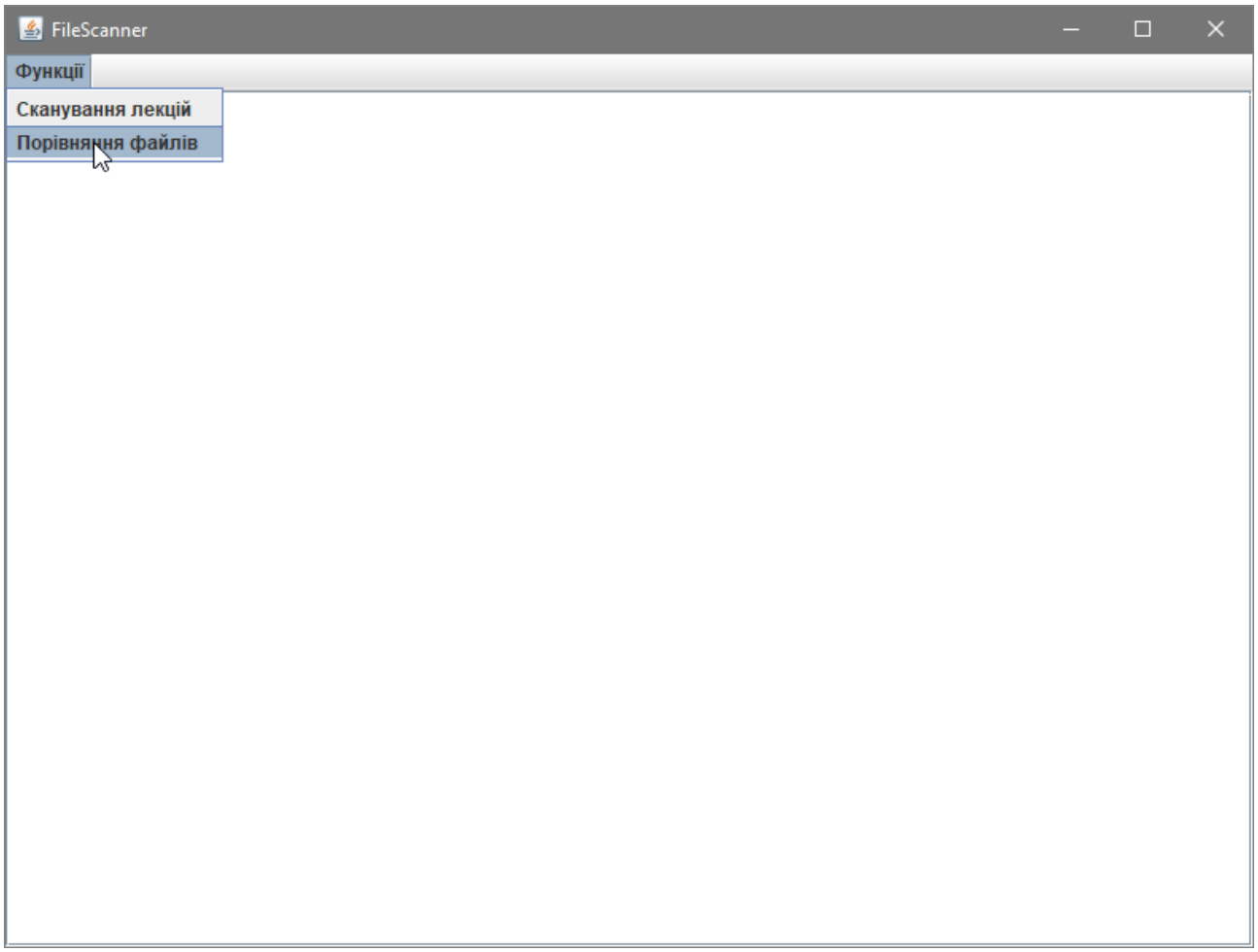


Рисунок 3.18 – Створений графічний інтерфейс для програми, що поєднує в собі ПЗ для порівняння файлів та ПЗ для сканування лекцій, з можливістю вибору функції.

3.4 Інструкція з використання об’єднаного ПЗ та його тестування.

1) Для початку роботи потрібно помістити виконуваний файл ПЗ з назвою «FilesScanner.jar» у директорію з файлами, з якими буде відбуватися робота. Наприклад, мені потрібно порівняти 10 файлів (зміст файлів наведено на рисунках А.1 – А.6 в «Додаток А») між собою. Я створюю папку, куди поміщаю потрібні файли і файл «FilesScanner.jar» (Рисунок 3.19).

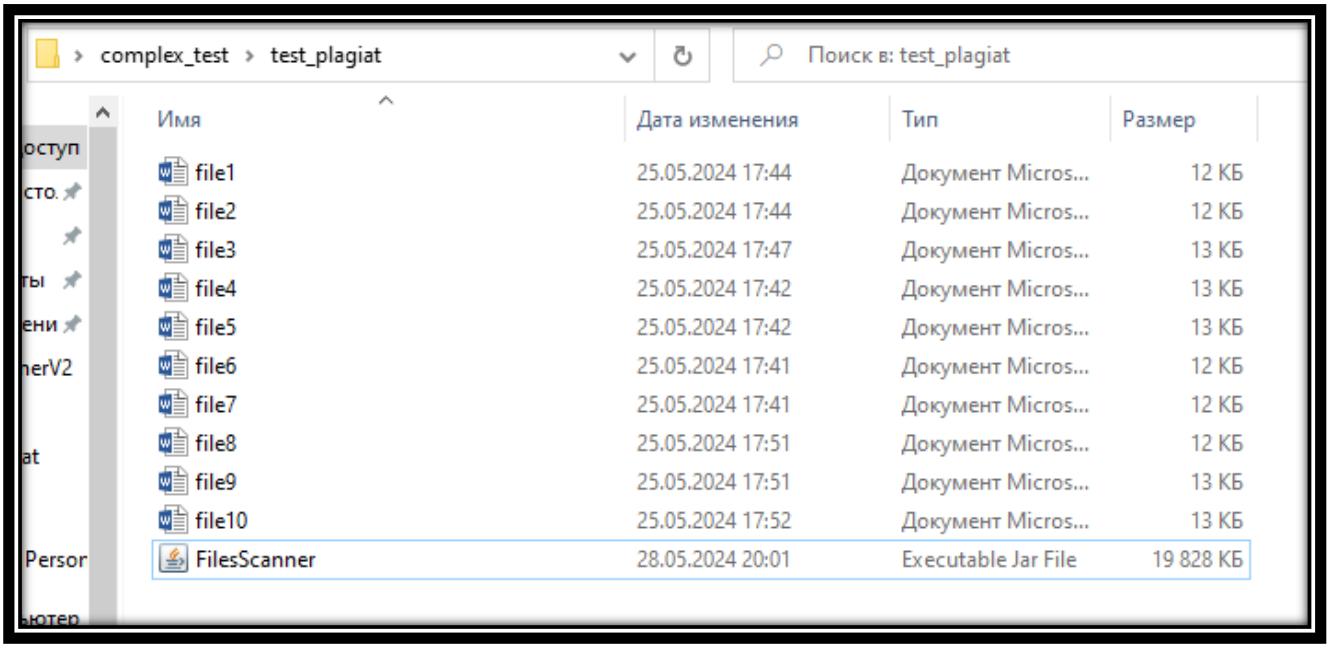


Рисунок 3.19 – Початковий стан директорії

2) Після цього натискаємо два рази на файл «FilesScanner.jar», тим самим запускаємо наш інтерфейс (Рисунок 3.20).

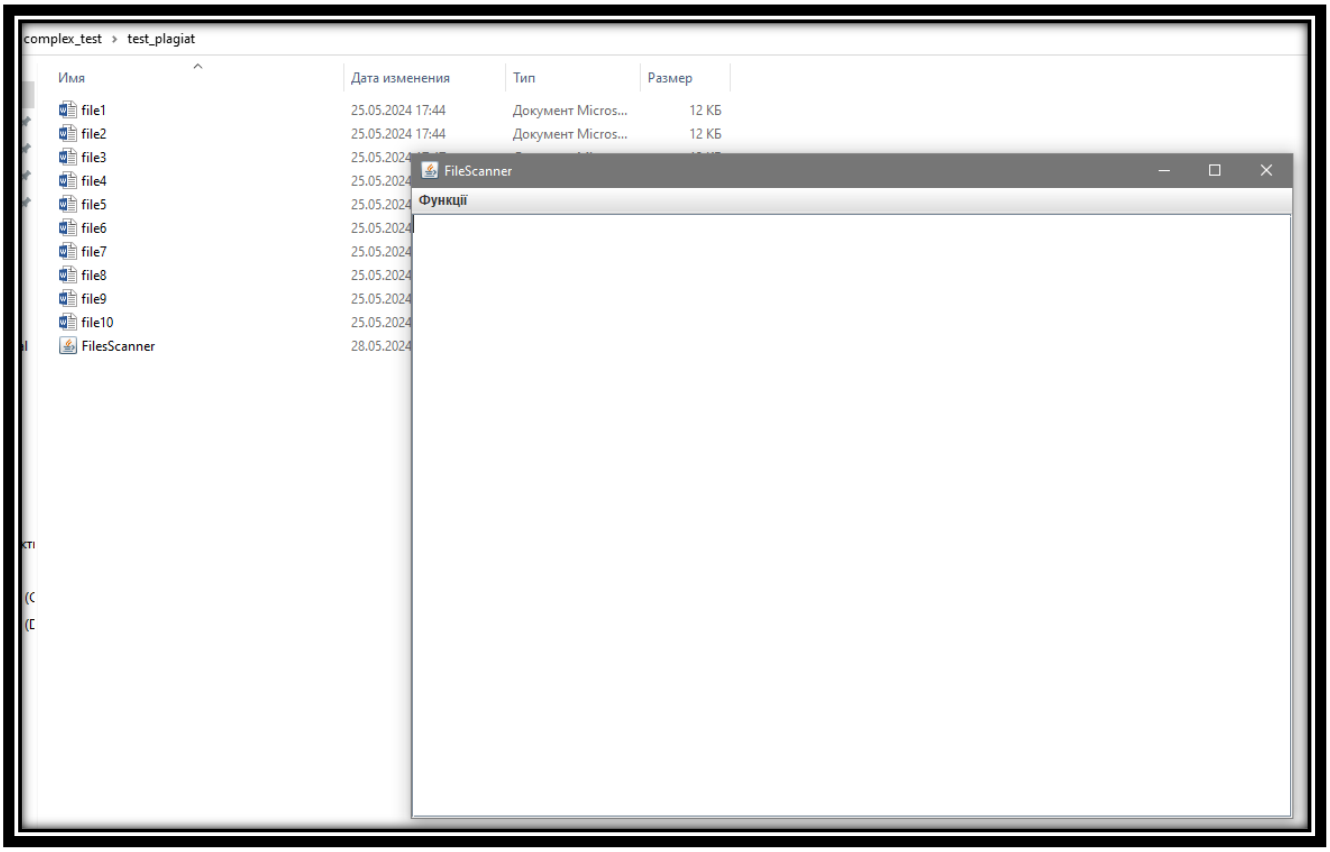


Рисунок 3.20 – Запуск інтерфейсу після активації файлу

3) Після того як з’явився інтерфейс, натискаємо на «Функції» і обираємо потрібну нам функцію (Рисунок 3.21), в цьому випадку перевіримо функцію «порівняння».

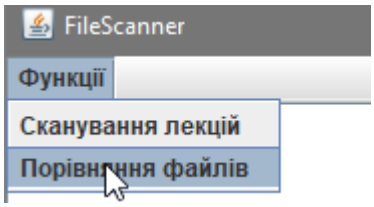


Рисунок 3.21 – Вибір потрібної функції

4) Отримуємо у текстовому вікні інтерфейсу результат після виконання роботи програми (Рисунок 3.22). Згідно змісту файлів (Рисунки А.1-А.6), очікуваний результат від порівняння можна описати так:

Очікувана подібність між файлами 1 і 2 – приблизно 90%, так як більшість тексту файлів збігається.

Очікувана подібність між файлами 3 і 4 – приблизно 70%, так як більшість тексту файлів збігається, але є словосполучення та слова, що відрізняються.

Очікувана подібність між файлами 4 і 5 – для файлу 4 збіг повинен бути приблизно 100%, через те, що його зміст повністю збігається з частиною файлу 5.

Очікувана подібність між файлами 6 і 7 – приблизно 50% для меншого файлу (7)

Очікувана подібність між файлами 8 і 9 – для файлу 8 збіг повинен бути приблизно 100%, через повну збіжність з частиною текста файлу 9.

Очікувана подібність між файлами 9 і 10 – так як для алгоритма умовно мінімальним рівнем є приблизно 30% подібності, очікуваним результатом від нього тут буде теж приблизно 30%.

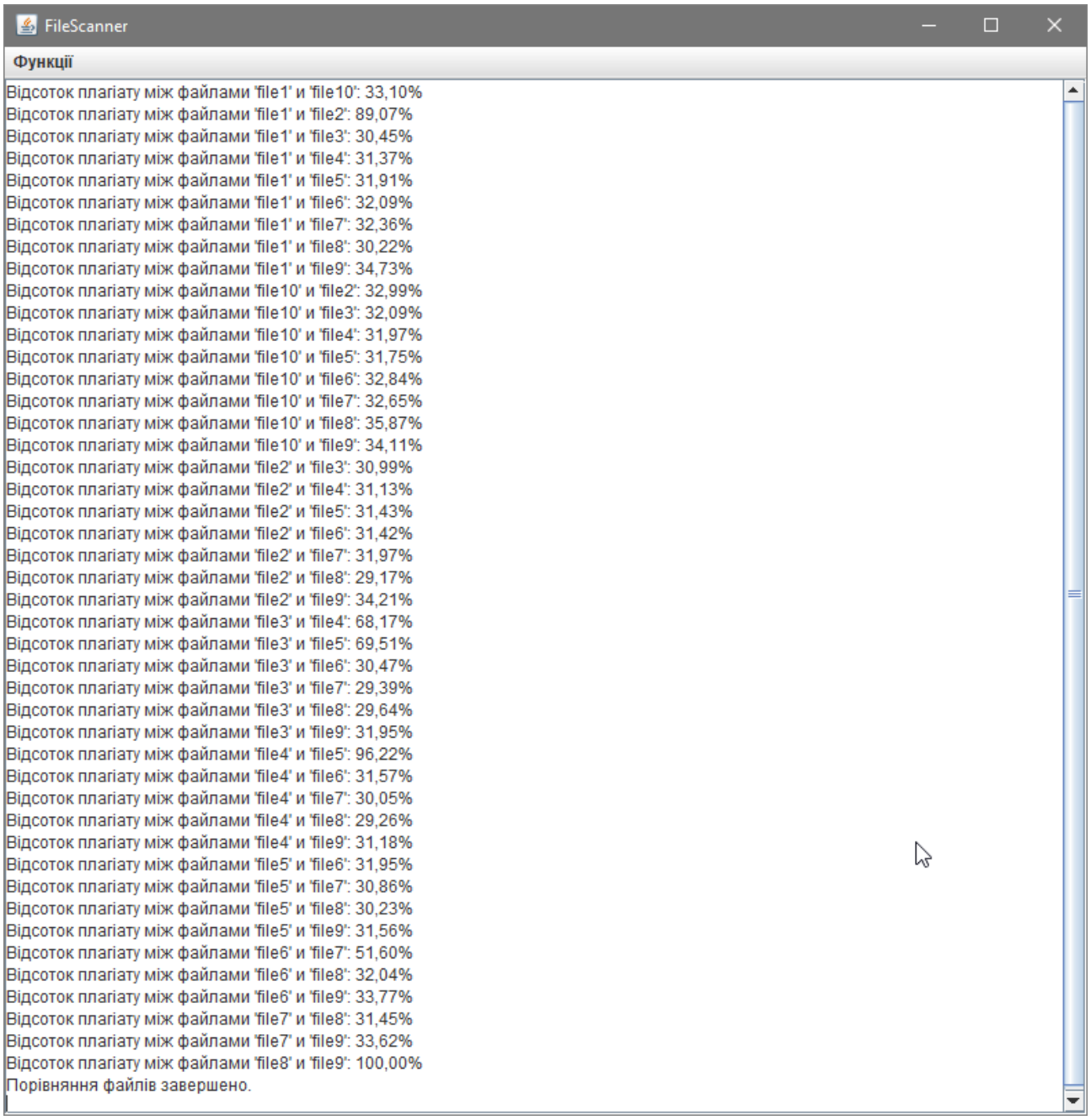


Рисунок 3.22 – Результат порівняння файлів з директорії

Як видно на рисунку 3.22, всі результати збігаються з очікуваними, що є хорошим показником ефективності та стабільності роботи програми.

Перевіримо функціональність другої частини ПЗ, тобто функції «сканування».

1) Розмістимо виконуваний файл «FilesScanner.jar» у директорію з файлами (зміст файлів на Рисунках А.7-А.10 у додатку А) для сканування та еталонним файлом «reference.word» (Рисунок 3.23)

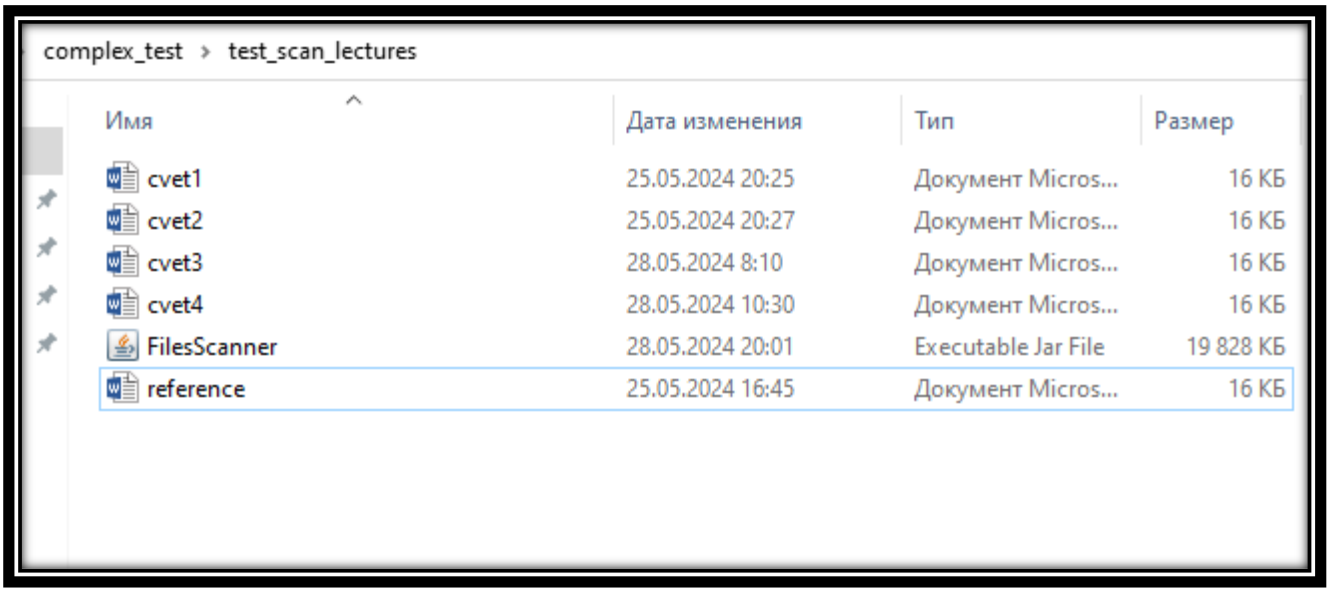


Рисунок 3.23 – Коректна директорія для сканування файлів

2) Натиснемо на виконуваний файл, та у з’явившомуся інтерфейсі оберемо потрібну функцію (сканування файлів) (Рисунок 3.24).

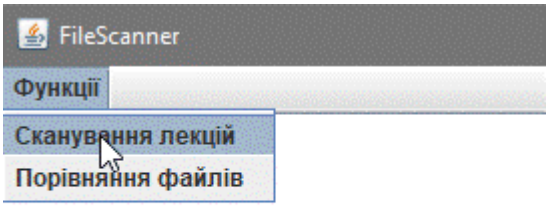


Рисунок 3.24 – Вибір функції сканування лекцій

3) Отримуємо у текстовому вікні інтерфейсу(він також зберігається у текстовий файл на випадок, якщо потрібно буде переглянути ще раз без запуску програми) (Рисунок 3.25). Результати сходяться з перевіркою цього ПЗ окремо, тож всі результати коректні.

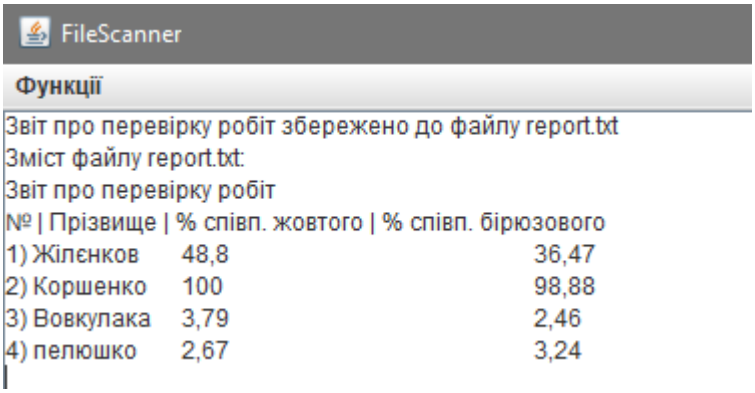


Рисунок 3.25 – Результати сканування файлів

3.5 Аналіз результатів.

В порівнянні з іншими алгоритмами, поєднання алгоритму Левенштейна та методу Шинглів є ефективним та влучним рішенням для створення ПЗ з метою перевірки робіт студентів на антиплагіат між собою. Алгоритм добре показує себе з різними типами файлів, та отримує коректні результати щодо подібності цих файлів.

Специфікою цього алгоритму є те, що мінімальна збіжність становить приблизно 30%. Це є недоліком, але це плата за велику точність виявлення дрібних змін у тексті. Тобто результати потрібно інтерпретувати таким чином, що 30% і нижче - відсутність плагіату взагалі. Приблизно до 36% - допустимий рівень, який свідчить про те, що плагіату серед перевіряємих файлів немає. Як видно з результатів (Рисунок 3.22), майже немає результатів, які знаходились би в межах між 34% та 50%, це означає, що при справжній наявності плагіату алгоритм без проблем фіксує цей факт і показує це у відсотковому відношенні приблизно 50% і більше, а при великій кількості плагіату, це значення збільшується ще значніше. Отже, 30% - нижній поріг, 40-50% - фіксація плагіату у невеликій кількості, >50% - більший рівень плагіату відповідно.

В цілому алгоритм ефективно та коректно, тож може використовуватись у заданих цілях. В поєднанні з "ПЗ ДЛЯ ПЕРЕВІРКИ ЯКОСТІ ДИСТАНЦІЙНОГО НАВЧАННЯ" вони утворюють єдине ПЗ, що може стати у нагоді кожному викладачу і полегшити їх працю під час дистанційного навчання.

ВИСНОВКИ

У результаті виконання дипломної роботи було досліджено декілька алгоритмів порівняння текстових файлів, таких як: Levenshtein Distance, Алгоритм Дамерау-Левенштейна, Jaccard Similarity, Cosine Similarity, TF-IDF, Smith-Waterman, Shingling. З них було обрано два алгоритми, що з теоретичного розрахунку найкраще підійшли в якості інструменту створення ПЗ для перевірки текстових файлів на подібність. Враховуючи недоліки та переваги обраних алгоритмів, були побудовані моделі та архітектура для їх поєднання в одному ПЗ, а потім, за допомогою цієї моделі безпосередньо було створено та протестовано це ПЗ.

Алгоритм шинглінгу використовується для розбиття текстових файлів на підрядки фіксованої довжини (шингли), що забезпечує стійкість до дрібних змін у текстах. Алгоритм Левенштейна застосовується для точного обчислення відстані редагування між шинглами, що дозволяє визначити рівень їхньої подібності.

Розроблено теоретичну модель, яка поєднує ці два алгоритми для ефективного порівняння текстових файлів та виявлення плагіату. Проведено аналіз точності та продуктивності обраних методів, визначено вимоги до програмного забезпечення та сформульовано задачу його розробки.

Спроєктовано архітектуру програми, що включає рівні введення-виведення, підготовки даних та обчислень. Визначено необхідні технології та інструменти, зокрема мову програмування Java, бібліотеку Apache POI для роботи з файлами .docx та стандартну бібліотеку Java.

У розділі з програмної реалізації описано основні алгоритми: токенізації тексту, обчислення шинглів та обчислення відстані Левенштейна між шинглами. Використано ефективні структури даних, такі як ArrayList, HashSet та StringBuilder, для забезпечення високої продуктивності та масштабованості програми.

Розроблене програмне забезпечення дозволяє ефективно виявляти плагіат серед великої кількості текстових файлів формату .docx, порівнюючи їхній вміст на основі комбінації алгоритмів шинглінгу та Левенштейна. Результати представляються у зручному для аналізу вигляді, що полегшує роботу викладачів при перевірці навчальних робіт студентів.

За результатами тестування можна зробити висновок, що ПЗ є ефективним та дієвим рішенням для вирішення проблеми плагіату серед студентів під час виконання робіт з відкритою відповіддю.

Після того як основне ПЗ було доведено до стабільного результату, наступним кроком було поєднано створене ПЗ та ПЗ "ПЗ ДЛЯ ПЕРЕВІРКИ ЯКОСТІ ДИСТАНЦІЙНОГО НАВЧАННЯ" для формування єдиного ПЗ, ціллю якого є автоматизація різних видів взаємодії викладачів з файлами(перевірка якості виділення лекцій, порівняння текстових файлів між собою), а також було створено візуальний інтерфейс для цього програмного засобу. Для того щоб поєднати дві програми в одну зі спільним візуальним інтерфейсом та з меню вибору функції яку користувач хоче застосувати, було створено новий проєкт для графічного інтерфейсу з використанням бібліотеки «Swing». У цьому новому проекті було створено клас, що містить основний метод main і код для ініціалізації графічного інтерфейсу. У цьому ж класі було створено дві окремі функції для запуску кожної з програм. У графічному інтерфейсі було додано два елементи, а саме пункти меню, для вибору між двома функціями. При натисканні на відповідну кнопку або пункт меню буде викликана відповідна функцію для запуску потрібної програми. Таким чином було об’єднано два ПЗ в одному програмному рішенні з графічним інтерфейсом.

В результаті всієї роботи, враховуючи створення ПЗ для порівняння робіт та його поєднання з ПЗ для сканування лекцій, отримано робоче ПЗ, яке зможе слугувати викладачам для перевірки великої кількості файлів між собою, буде зручним та простим у використанні, не буде потребувати доступу до інтернету чи інших спеціальних умов, буде виправно і швидко працювати, та формувати результат у вигляді відсоткового представлення збігів.

СПИСОК ДЖЕРЕЛ ПОСИЛАННЯ

1. Broder A. Algorithms for duplicate documents [Електронний ресурс] / Andrei Broder. – 2005. – Режим доступу до ресурсу: http://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/Princeton.pdf

2. Valls, E. & Rosso, P., “Detection of near-duplicate user generated contents: the SMS spam collection”, in Proceedings of the 3rd international workshop on search and mining usergenerated contents, 2011, pp. 27–34 [Електронний ресурс] - Режим доступу до ресурсу: https://www.researchgate.net/publication/221615491_Detection_of_near-duplicate_user_generated_contents_The_SMS_spam_collection

3. «Mining of Massive Datasets» 2010, 2011, 2012, 2013, 2014 Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman Режим доступу до ресурсу: http://infolab.stanford.edu/~ullman/mmds/book.pdf?ref=quarksandbits.com

4. «Default methods» Java Documentation [Електронний ресурс] - Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/java/IandI/defaultmethods.html

5. Levenshtein distance, Wikipedia. [Електронний ресурс] - Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Levenshtein_distance

6. Introduction to Levenshtein distance [Електронний ресурс] - Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/

7. Best NLP Algorithms to get Document Similarity, Jair Neto [Електронний ресурс] - Режим доступу до ресурсу: https://medium.com/analytics-vidhya/best-nlp-algorithms-to-get-document-similarity-a5559244b23b

8. George, T. & Caulfield, J. (2022, July 18). How Do Plagiarism Checkers Work?. Scribbr. Retrieved May 27, 2024 [Електронний ресурс] - Режим доступу до ресурсу:

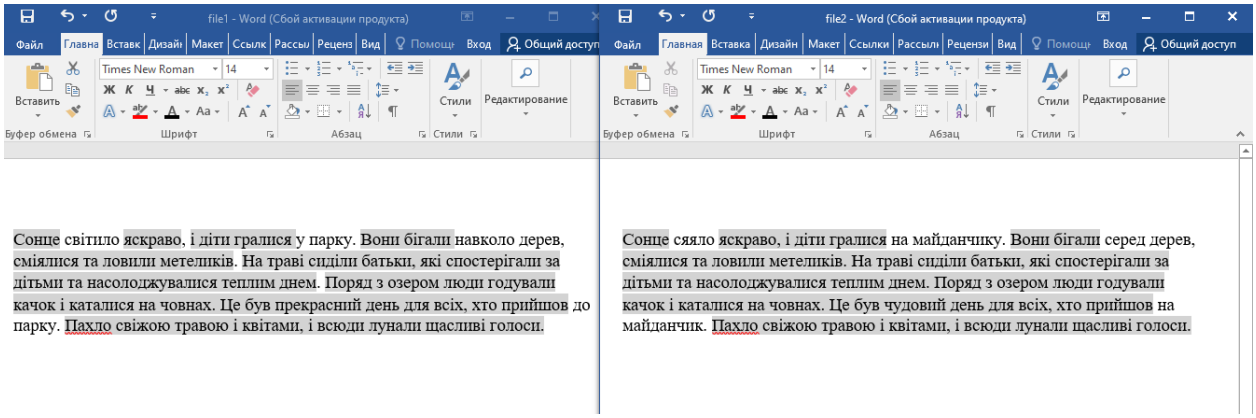
https://www.scribbr.com/plagiarism/how-do-plagiarism-checkers-work/

9. Apache POI - Official Documentation Режим доступу до ресурсу:
https://poi.apache.org/

10. Lesson: Packaging Programs in JAR Files – офіційний посібник - Режим доступу до ресурсу:
https://docs.oracle.com/javase/tutorial/deployment/jar/index.html

ДОДАТОК А

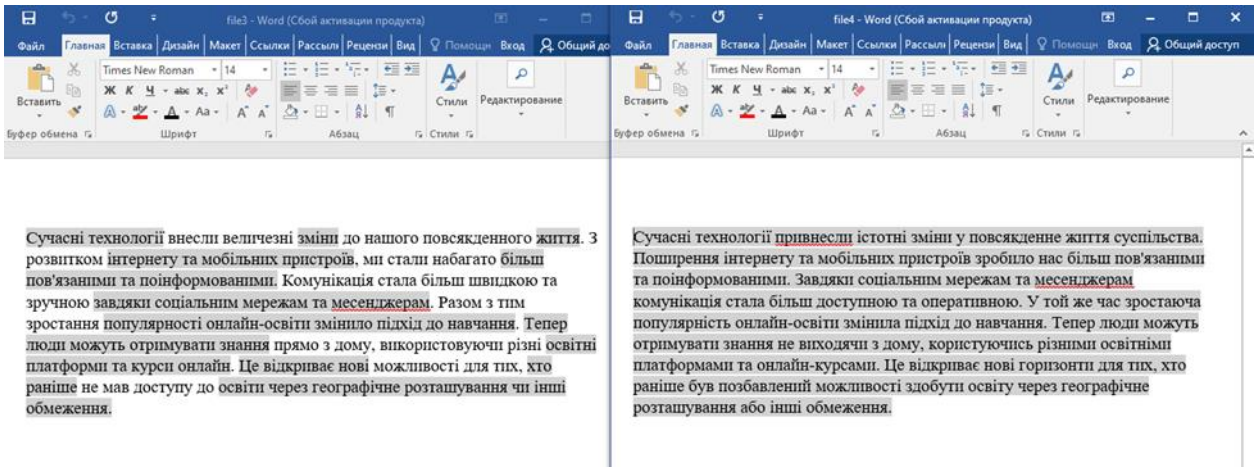
Зміст файлів «file1»,«file2»,…,«file9», «file10»



Сонце світило яскраво, і діти гралися у парку. Вони бігали навколо дерев, сміялися та ловили метеликів. На траві сиділи батьки, які спостерігали за дітьми та насолоджувалися теплим днем. Поряд з озером люди годували качок і каталися на човнах. Це був прекрасний день для всіх, хто прийшов до парку. Пахло свіжою травою і квітами, і всюди лунали щасливі голоси.

Сонце сяяло яскраво, і діти гралися на майданчику. Вони бігали серед дерев, сміялися та ловили метеликів. На траві сиділи батьки, які спостерігали за дітьми та насолоджувалися теплим днем. Поряд з озером люди годували качок і каталися на човнах. Це був чудовий день для всіх, хто прийшов на майданчик. Пахло свіжою травою і квітами, і всюди лунали щасливі голоси.

Рисунок А.1 – Зміст файлів «file1» і «file2», з виділенням збігів у змісті.



Сучасні технології внесли величезні зміни до нашого повсякденного життя. З розвитком інтернету та мобільних пристроїв, ми стали набагато більш пов'язаними та поінформованими. Комунікація стала більш швидкою та зручною завдяки соціальним мережам та месенджерам. Разом з тим зростання популярності онлайн-освіти змінило підхід до навчання. Тепер люди можуть отримувати знання прямо з дому, використовуючи різні освітні платформи та курси онлайн. Це відкриває нові можливості для тих, хто раніше не мав доступу до освіти через географічне розташування чи інші обмеження.

Сучасні технології привнесли істотні зміни у повсякденне життя суспільства. Поширення інтернету та мобільних пристроїв зробило нас більш пов'язаними та поінформованими. Завдяки соціальним мережам та месенджерам комунікація стала більш доступною та оперативною. У той же час зростаюча популярність онлайн-освіти змінила підхід до навчання. Тепер люди можуть отримувати знання не виходячи з дому, користуючись різними освітніми платформами та онлайн-курсами. Це відкриває нові горизонти для тих, хто раніше був позбавлений можливості здобути освіту через географічне розташування або інші обмеження.

Рисунок А.2 – Зміст файлів «file3» і «file4», з виділенням збігів у змісті.

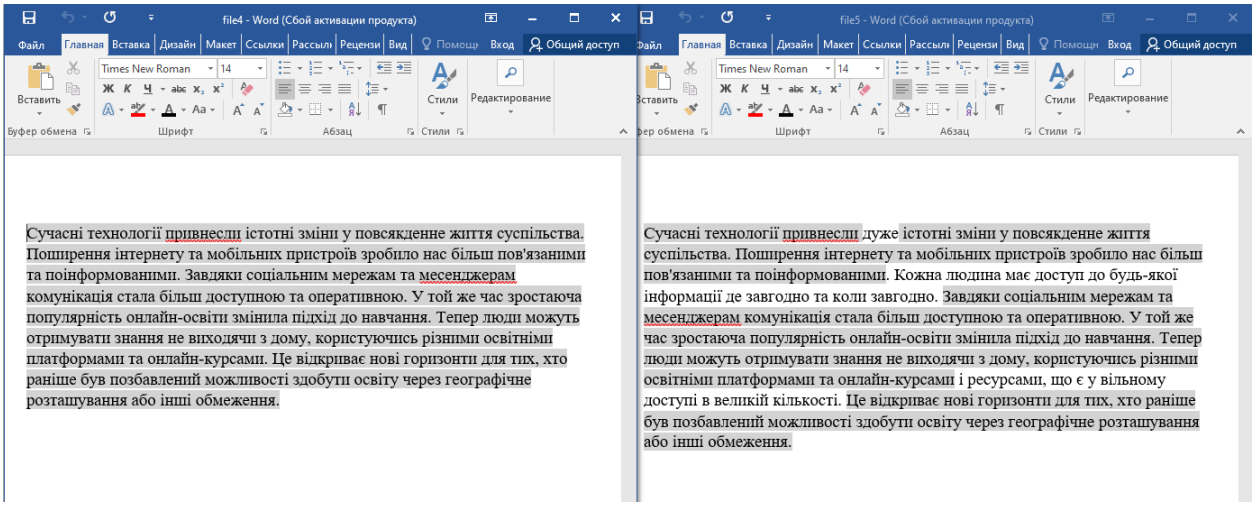


Рисунок А.3 – Зміст файлів «file4» і «file5», з виділенням збігів у змісті.

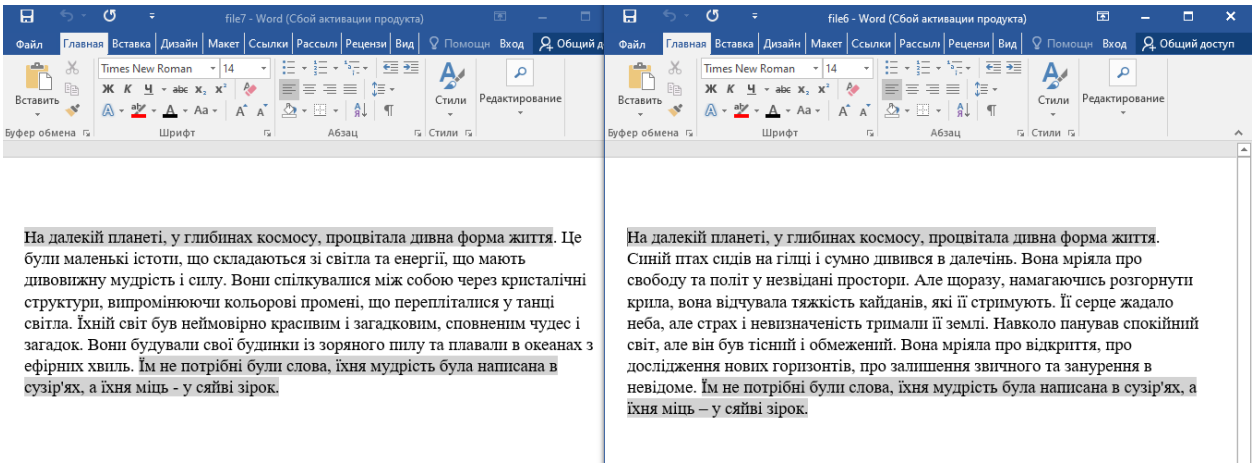


Рисунок А.4 – Зміст файлів «file6» і «file7», з виділенням збігів у змісті.

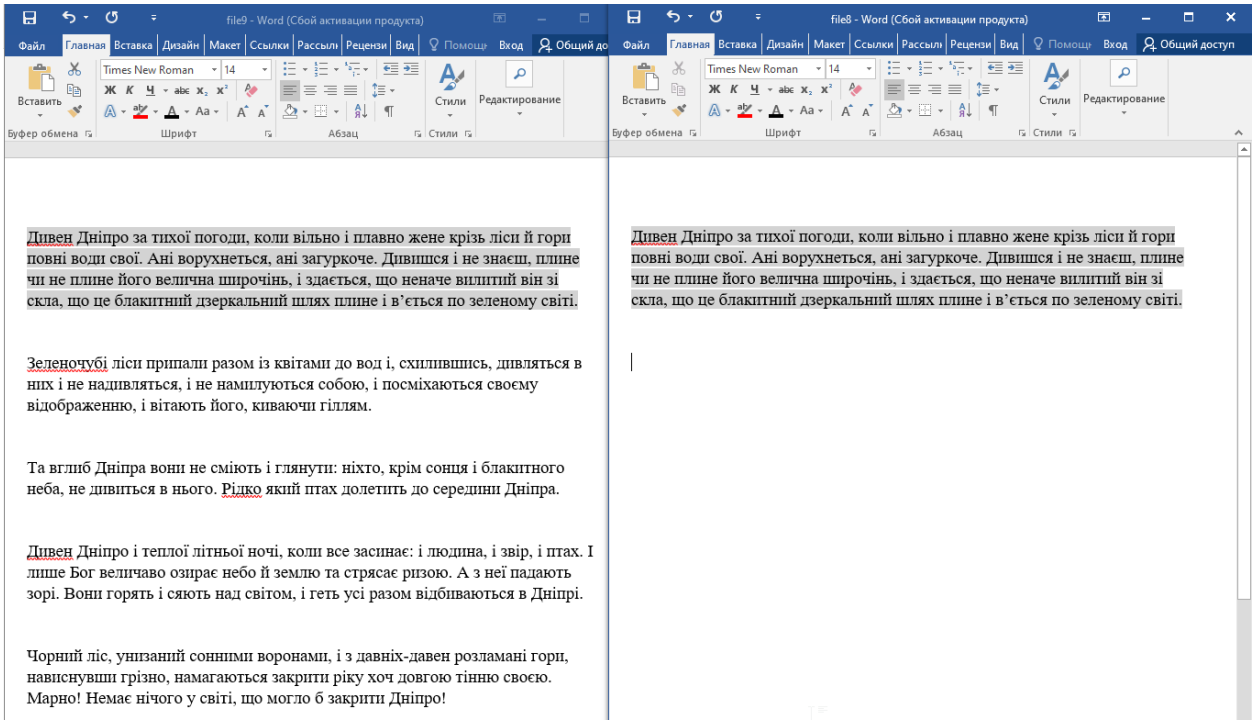


Рисунок А.5 - Зміст файлів «file8» і «file9», з виділенням збігів у змісті.

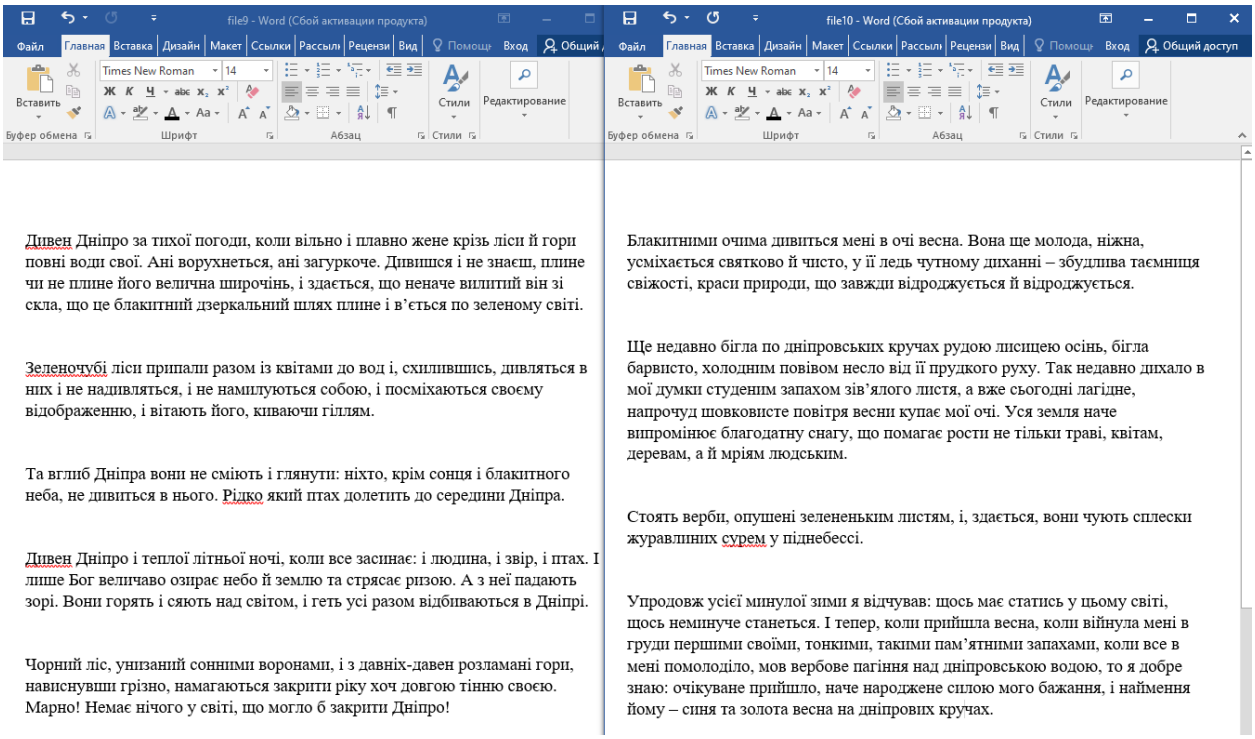


Рисунок А.6 - Зміст файлів «file9» і «file10», з виділенням збігів у змісті(точні збіги відсутні).

Жілєнков

Україна – це велика східноєвропейська країна з багатою історією, культурою та традиціями. Її розташування, на перехресті Європи та Азії, зробило Україну важливим геополітичним і культурним центром протягом століть.

Історія та культура

(2)

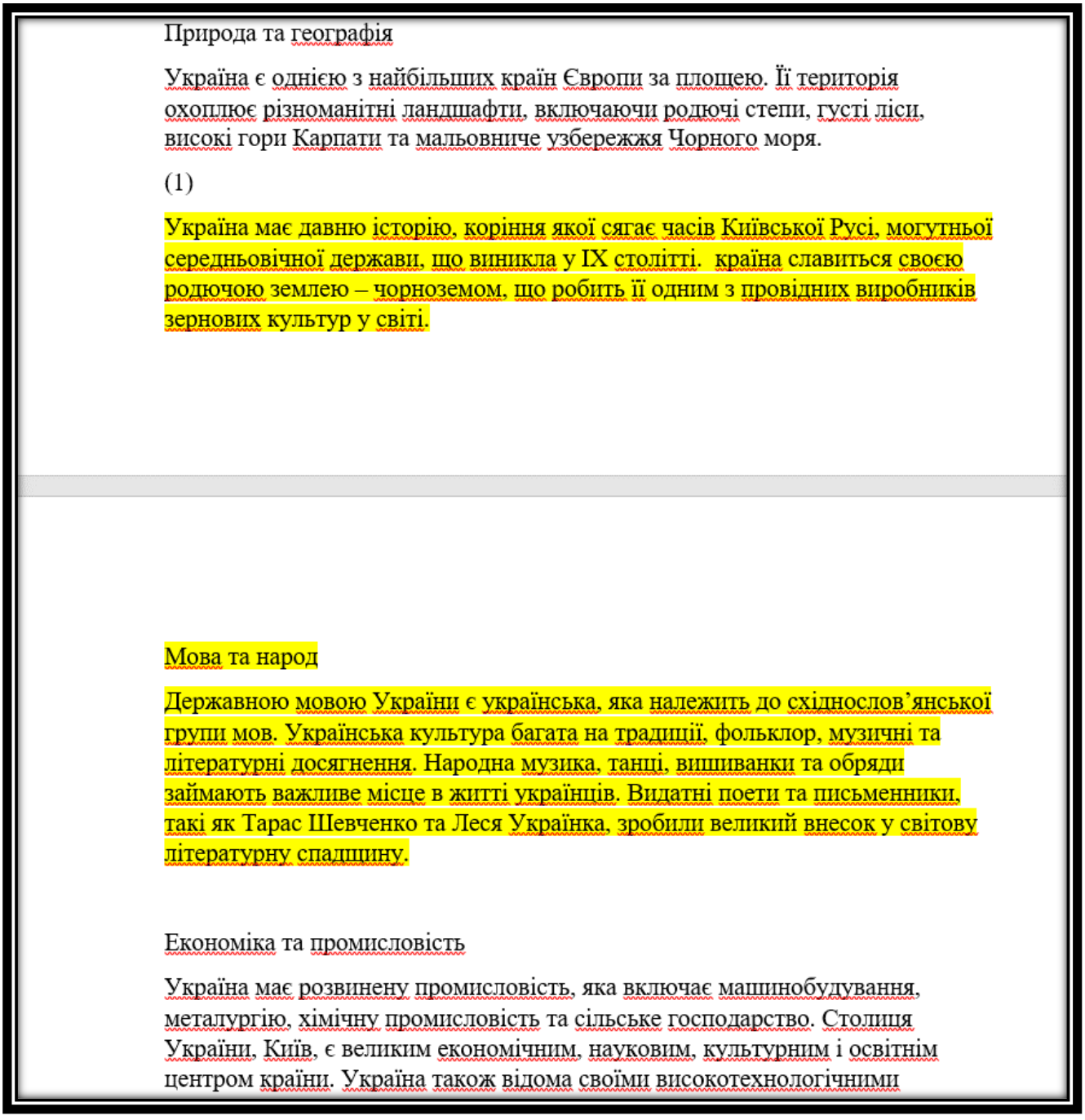
Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Київська Русь стала одним із найбільших і найвпливовіших держав Європи, і Київ був її політичним та культурним центром. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Після розпаду Київської Русі на початку XIII століття, територія сучасної України пережила численні випробування, включаючи монгольські навали, польське та литовське панування, а згодом входження до складу Російської та Австро-Угорської імперій.

(1)

Україна здобула свою незалежність у 1991 році після розпаду Радянського Союзу. Це стало важливим моментом в історії країни, відкривши нову еру демократичного розвитку та економічних реформ. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті.Сучасна Україна – це незалежна держава з власною Конституцією, багатопартійною політичною системою та ринковою економікою.

Природа та географія

Рисунок А.7 – вміст файлу «cvet1» перша частина



Природа та географія

Україна є однією з найбільших країн Європи за площею. Її територія охоплює різноманітні ландшафти, включаючи родючі степи, густі ліси, високі гори Карпати та мальовниче узбережжя Чорного моря.

(1)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. країна славиться своєю родючою землею – чорноземом, що робить її одним з провідних виробників зернових культур у світі.

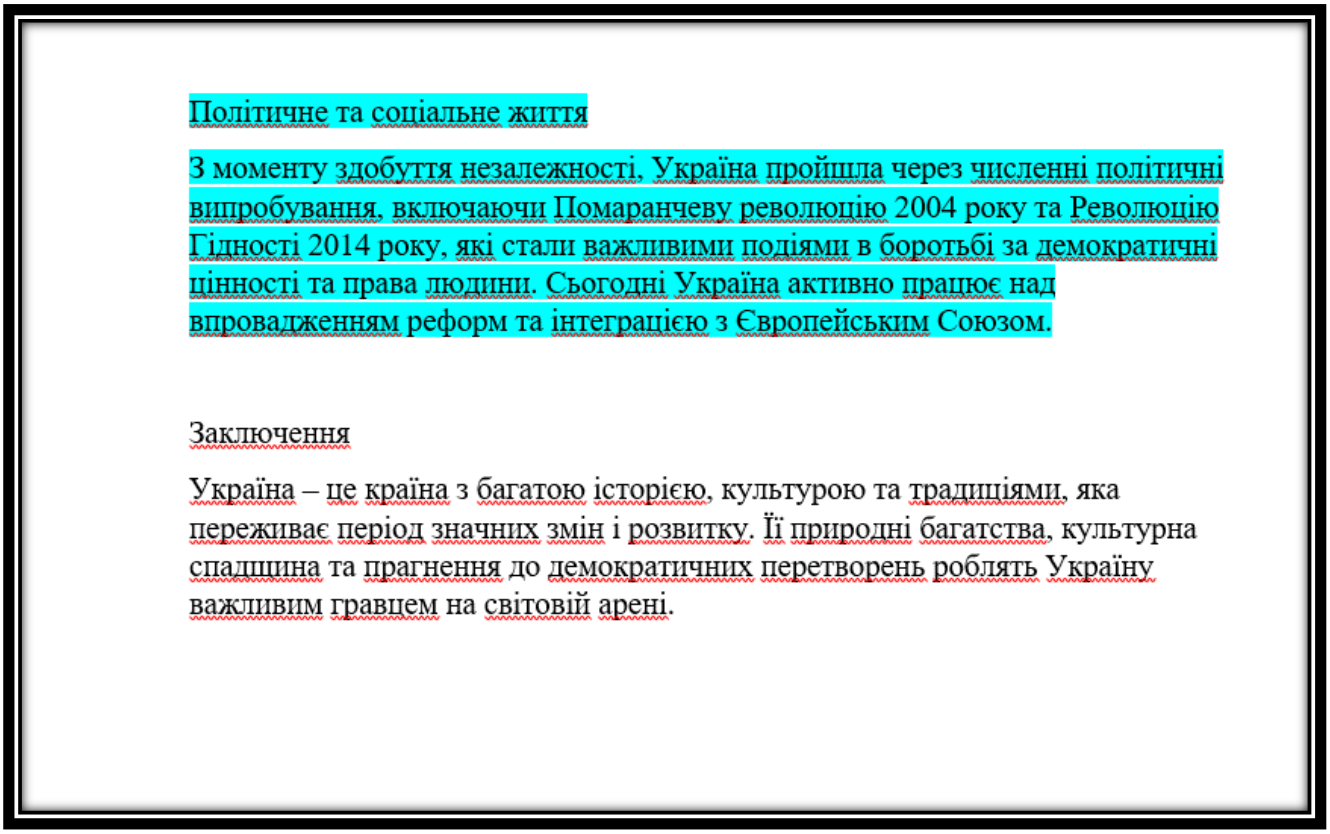
Мова та народ

Державною мовою України є українська, яка належить до східнослов'янської групи мов. Українська культура багата на традиції, фольклор, музичні та літературні досягнення. Народна музика, танці, вишиванки та обряди займають важливе місце в житті українців. Видатні поети та письменники, такі як Тарас Шевченко та Леся Українка, зробили великий внесок у світову літературну спадщину.

Економіка та промисловість

Україна має розвинену промисловість, яка включає машинобудування, металургію, хімічну промисловість та сільське господарство. Столиця України, Київ, є великим економічним, науковим, культурним і освітнім центром країни. Україна також відома своїми високотехнологічними

Рисунок А.8 – вміст файлу «cvet1» друга частина



Політичне та соціальне життя

З моменту здобуття незалежності, Україна пройшла через численні політичні випробування, включаючи Помаранчеву революцію 2004 року та Революцію Гідності 2014 року, які стали важливими подіями в боротьбі за демократичні цінності та права людини. Сьогодні Україна активно працює над впровадженням реформ та інтеграцією з Європейським Союзом.

Заключення

Україна – це країна з багатою історією, культурою та традиціями, яка переживає період значних змін і розвитку. Її природні багатства, культурна спадщина та прагнення до демократичних перетворень роблять Україну важливим гравцем на світовій арені.

Рисунок А.9 – Вміст файлу «cvet1» третя частина

Коршенко

Україна – це велика східноєвропейська країна з багатою історією, культурою та традиціями. Її розташування, на перехресті Європи та Азії, зробило Україну важливим геополітичним і культурним центром протягом століть.

Історія та культура

(2)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Київська Русь стала одним із найбільших і найвпливовіших держав Європи, і Київ був її політичним та культурним центром. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Після розпаду Київської Русі на початку XIII століття, територія сучасної України пережила численні випробування, включаючи монгольські навали, польське та литовське панування, а згодом входження до складу Російської та Австро-Угорської імперій.

(1)

Україна здобула свою незалежність у 1991 році після розпаду Радянського Союзу. Це стало важливим моментом в історії країни, відкривши нову еру демократичного розвитку та економічних реформ. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті.Сучасна Україна – це незалежна держава з власною Конституцією, багатопартійною політичною системою та ринковою економікою.

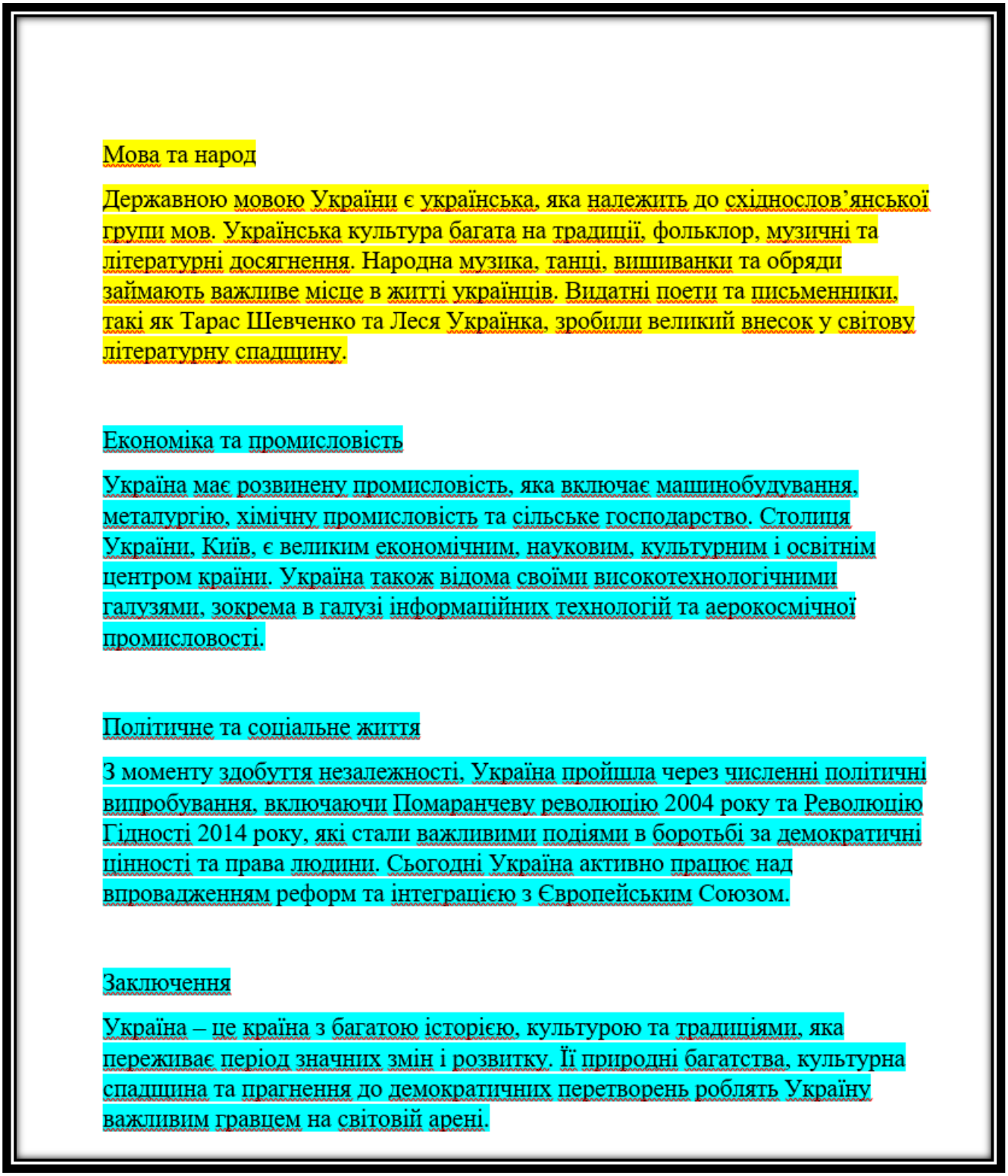
Природа та географія

Україна є однією з найбільших країн Європи за площею. Її територія охоплює різноманітні ландшафти, включаючи родючі степи, густі ліси, високі гори Карпати та мальовниче узбережжя Чорного моря.

(1)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. країна славиться своєю

Рисунок А.10 – вміст файлу «cvet2» перша частина



Мова та народ

Державною мовою України є українська, яка належить до східнослов'янської групи мов. Українська культура багата на традиції, фольклор, музичні та літературні досягнення. Народна музика, танці, вишиванки та обряди займають важливе місце в житті українців. Видатні поети та письменники, такі як Тарас Шевченко та Леся Українка, зробили великий внесок у світову літературну спадщину.

Економіка та промисловість

Україна має розвинену промисловість, яка включає машинобудування, металургію, хімічну промисловість та сільське господарство. Столиця України, Київ, є великим економічним, науковим, культурним і освітнім центром країни. Україна також відома своїми високотехнологічними галузями, зокрема в галузі інформаційних технологій та аерокосмічної промисловості.

Політичне та соціальне життя

З моменту здобуття незалежності, Україна пройшла через численні політичні випробування, включаючи Помаранчеву революцію 2004 року та Революцію Гідності 2014 року, які стали важливими подіями в боротьбі за демократичні цінності та права людини. Сьогодні Україна активно працює над впровадженням реформ та інтеграцією з Європейським Союзом.

Заключення

Україна – це країна з багатою історією, культурою та традиціями, яка переживає період значних змін і розвитку. Її природні багатства, культурна спадщина та прагнення до демократичних перетворень роблять Україну важливим гравцем на світовій арені.

Рисунок А.11 – вміст файлу «cvet2» друга частина

Вовкулака

Україна – це велика східноєвропейська країна з багатою історією, культурою та традиціями. Її розташування, на перехресті Європи та Азії, зробило Україну важливим геополітичним і культурним центром протягом століть.

Історія та культура

(2)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Київська Русь стала одним із найбільших і найвпливовіших держав Європи, і Київ був її політичним та культурним центром. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Після розпаду Київської Русі на початку XIII століття, територія сучасної України пережила численні випробування, включаючи монгольські навали, польське та литовське панування, а згодом входження до складу Російської та Австро-Угорської імперій.

(1)

Україна здобула свою незалежність у 1991 році після розпаду Радянського Союзу. Це стало важливим моментом в історії країни, відкривши нову еру демократичного розвитку та економічних реформ. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті.Сучасна Україна – це незалежна держава з власною Конституцією, багатопартійною політичною системою та ринковою економікою.

Природа та географія

Україна є однією з найбільших країн Європи за площею. Її територія охоплює різноманітні ландшафти, включаючи родючі степи, густі ліси, високі гори Карпати та мальовниче узбережжя Чорного моря.

(1)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. країна славиться своєю

Рисунок А.12 – вміст файлу «cvet3» перша частина

Мова та народ

Державною мовою України є українська, яка належить до східнослов'янської групи мов. Українська культура багата на традиції, фольклор, музичні та літературні досягнення. Народна музика, танці, вишиванки та обряди займають важливе місце в житті українців. Видатні поети та письменники, такі як Тарас Шевченко та Леся Українка, зробили великий внесок у світову літературну спадщину.

Економіка та промисловість

Україна має розвинену промисловість, яка включає машинобудування, металургію, хімічну промисловість та сільське господарство. Столиця України, Київ, є великим економічним, науковим, культурним і освітнім центром країни. Україна також відома своїми високотехнологічними галузями, зокрема в галузі інформаційних технологій та аерокосмічної промисловості.

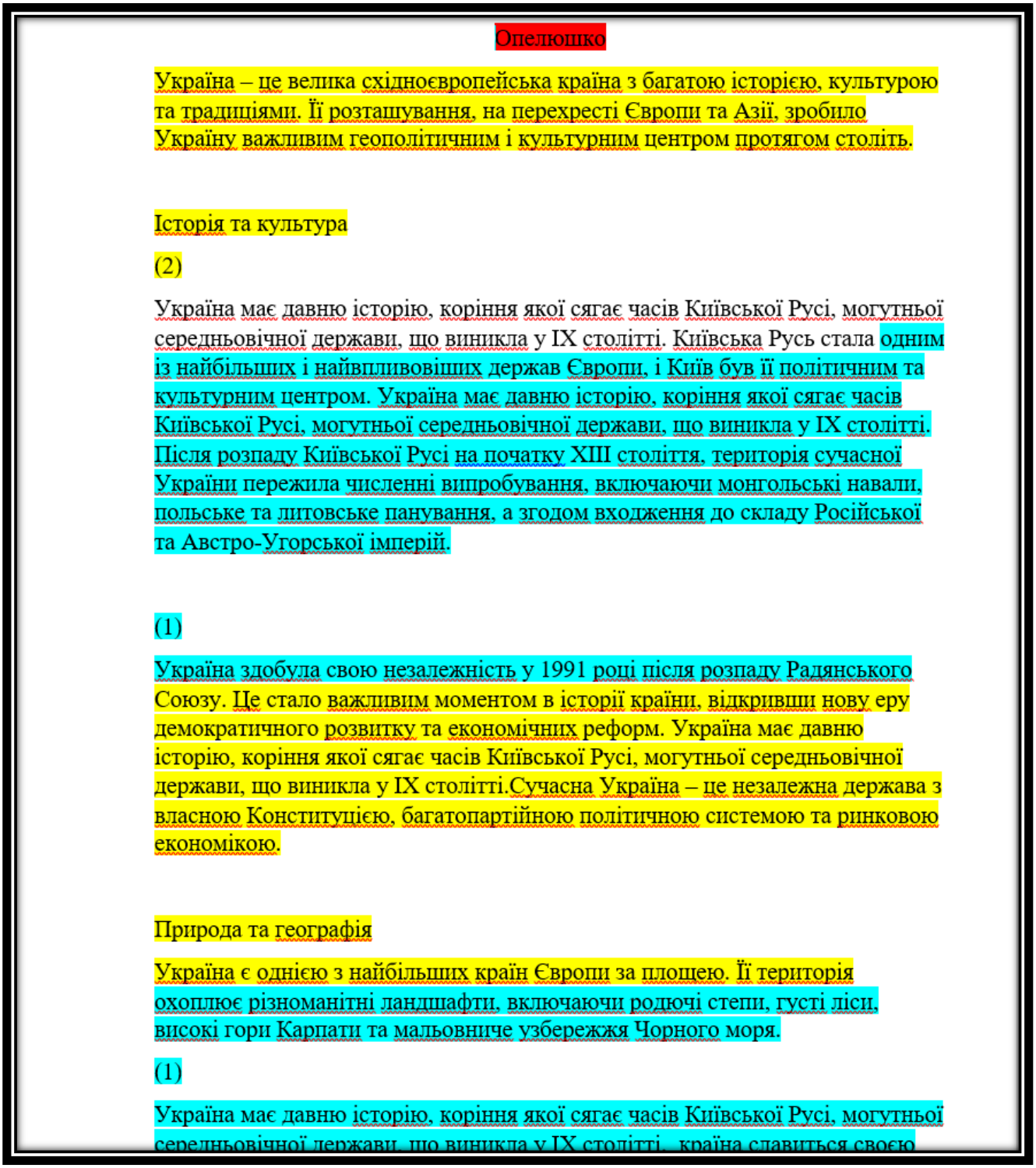
Політичне та соціальне життя

З моменту здобуття незалежності, Україна пройшла через численні політичні випробування, включаючи Помаранчеву революцію 2004 року та Революцію Гідності 2014 року, які стали важливими подіями в боротьбі за демократичні цінності та права людини. Сьогодні Україна активно працює над впровадженням реформ та інтеграцією з Європейським Союзом.

Заключення

Україна – це країна з багатою історією, культурою та традиціями, яка переживає період значних змін і розвитку. Її природні багатства, культурна спадщина та прагнення до демократичних перетворень роблять Україну важливим гравцем на світовій арені.

Рисунок А.13 – вміст файлу «cvet3» друга частина



Опелюшко

Україна – це велика східноєвропейська країна з багатою історією, культурою та традиціями. Її розташування, на перехресті Європи та Азії, зробило Україну важливим геополітичним і культурним центром протягом століть.

Історія та культура

(2)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Київська Русь стала одним із найбільших і найвпливовіших держав Європи, і Київ був її політичним та культурним центром. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Після розпаду Київської Русі на початку XIII століття, територія сучасної України пережила численні випробування, включаючи монгольські навали, польське та литовське панування, а згодом входження до складу Російської та Австро-Угорської імперій.

(1)

Україна здобула свою незалежність у 1991 році після розпаду Радянського Союзу. Це стало важливим моментом в історії країни, відкривши нову еру демократичного розвитку та економічних реформ. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті.Сучасна Україна – це незалежна держава з власною Конституцією, багатопартійною політичною системою та ринковою економікою.

Природа та географія

Україна є однією з найбільших країн Європи за площею. Її територія охоплює різноманітні ландшафти, включаючи родючі степи, густі ліси, високі гори Карпати та мальовниче узбережжя Чорного моря.

(1)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. країна славиться своєю

Рисунок А.14 – вміст файлу «cvet4» перша частина

Мова та народ

Державною мовою України є українська, яка належить до східнослов'янської групи мов. Українська культура багата на традиції, фольклор, музичні та літературні досягнення. Народна музика, танці, вишиванки та обряди займають важливе місце в житті українців. Видатні поети та письменники, такі як Тарас Шевченко та Леся Українка, зробили великий внесок у світову літературну спадщину.

Економіка та промисловість

Україна має розвинену промисловість, яка включає машинобудування, металургію, хімічну промисловість та сільське господарство. Столиця України, Київ, є великим економічним, науковим, культурним і освітнім центром країни. Україна також відома своїми високотехнологічними галузями, зокрема в галузі інформаційних технологій та аерокосмічної промисловості.

Політичне та соціальне життя

З моменту здобуття незалежності, Україна пройшла через численні політичні випробування, включаючи Помаранчеву революцію 2004 року та Революцію Гідності 2014 року, які стали важливими подіями в боротьбі за демократичні цінності та права людини. Сьогодні Україна активно працює над впровадженням реформ та інтеграцією з Європейським Союзом.

Заключення

Україна – це країна з багатою історією, культурою та традиціями, яка переживає період значних змін і розвитку. Її природні багатства, культурна спадщина та прагнення до демократичних перетворень роблять Україну важливим гравцем на світовій арені.

Рисунок А.15 – вміст файлу «cvet4» друга частина

Україна – це велика східноєвропейська країна з багатою історією, культурою та традиціями. Її розташування, на перехресті Європи та Азії, зробило Україну важливим геополітичним і культурним центром протягом століть.

Історія та культура

(2)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Київська Русь стала одним із найбільших і найвпливовіших держав Європи, і Київ був її політичним та культурним центром. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. Після розпаду Київської Русі на початку XIII століття, територія сучасної України пережила численні випробування, включаючи монгольські навали, польське та литовське панування, а згодом входження до складу Російської та Австро-Угорської імперій.

(1)

Україна здобула свою незалежність у 1991 році після розпаду Радянського Союзу. Це стало важливим моментом в історії країни, відкривши нову еру демократичного розвитку та економічних реформ. Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті.Сучасна Україна – це незалежна держава з власною Конституцією, багатопартійною політичною системою та ринковою економікою.

Природа та географія

Україна є однією з найбільших країн Європи за площею. Її територія охоплює різноманітні ландшафти, включаючи родючі степи, густі ліси, високі гори Карпати та мальовниче узбережжя Чорного моря.

(1)

Україна має давню історію, коріння якої сягає часів Київської Русі, могутньої середньовічної держави, що виникла у IX столітті. країна славиться своєю

Рисунок А.16 – вміст файлу «reference» перша частина

Мова та народ

Державною мовою України є українська, яка належить до східнослов'янської групи мов. Українська культура багата на традиції, фольклор, музичні та літературні досягнення. Народна музика, танці, вишиванки та обряди займають важливе місце в житті українців. Видатні поети та письменники, такі як Тарас Шевченко та Леся Українка, зробили великий внесок у світову літературну спадщину.

Економіка та промисловість

Україна має розвинену промисловість, яка включає машинобудування, металургію, хімічну промисловість та сільське господарство. Столиця України, Київ, є великим економічним, науковим, культурним і освітнім центром країни. Україна також відома своїми високотехнологічними галузями, зокрема в галузі інформаційних технологій та аерокосмічної промисловості.

Політичне та соціальне життя

З моменту здобуття незалежності, Україна пройшла через численні політичні випробування, включаючи Помаранчеву революцію 2004 року та Революцію Гідності 2014 року, які стали важливими подіями в боротьбі за демократичні цінності та права людини. Сьогодні Україна активно працює над впровадженням реформ та інтеграцією з Європейським Союзом.

Заключення

Україна – це країна з багатою історією, культурою та традиціями, яка переживає період значних змін і розвитку. Її природні багатства, культурна спадщина та прагнення до демократичних перетворень роблять Україну важливим гравцем на світовій арені.

Рисунок А.17 – вміст файлу «reference» друга частина

ДОДАТОК Б

Клас Main.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;

public class Main extends JFrame {
    private JTextArea outputArea;
    private JScrollPane scrollPane;
    public Main() {
        setTitle("FileScanner");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(800, 600);
        setLocationRelativeTo(null);


        JMenuBar menuBar = new JMenuBar();
        JMenu menu = new JMenu("Функції");
        menuBar.add(menu);

        JMenuItem scannerItem = new JMenuItem("Сканування лекцій");
        JMenuItem comparisonItem = new JMenuItem("Порівняння файлів");

        scannerItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                runScanner();
            }
        });

        comparisonItem.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                runComparison();
            }
        });

        menu.add(scannerItem);
        menu.add(comparisonItem);
        setJMenuBar(menuBar);

        outputArea = new JTextArea();
        outputArea.setEditable(false);
        scrollPane = new JScrollPane(outputArea); // Ініціалізуємо scrollPane
        JScrollPane scrollPane = new JScrollPane(outputArea);
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }

    private void runScanner() {
        System.out.println("Початок сканування лекцій...");
        LecturesScannerV1 scanner = new LecturesScannerV1();
        redirectOutput(scanner);

    }

    private void runComparison() {
        System.out.println("Початок порівняння файлів...");
        ShingleLevenstain comparison = new ShingleLevenstain();
        redirectOutput(comparison);
```

```
    }


    private void redirectOutput(Object obj) {
        PrintStream printStream = new PrintStream(new
CustomOutputStream(outputArea), true);
        System.setOut(printStream);
        System.setErr(printStream);

        if (obj instanceof LecturesScannerV1) {
            LecturesScannerV1 scanner = (LecturesScannerV1) obj;
            // Виклик метода mainFunction() чи інших методів для запуску сканера
            scanner.mainFunction();
            printStream.flush();
        } else if (obj instanceof ShingleLevenstain) {
            ShingleLevenstain comparison = (ShingleLevenstain) obj;
            // Виклик метода main() чи інших методів для запуску порівняння
            comparison.runShingleLevenstain();
            printStream.flush();
        }else {
            System.out.println("Некоректний вибір функції.");
        }

        System.setOut(System.out);
        System.setErr(System.err);
        // Прокрутка вікна вниз після завершення роботи
        SwingUtilities.invokeLater(() -> {
            JScrollBar verticalScrollBar = scrollPane.getVerticalScrollBar();
            verticalScrollBar.setValue(verticalScrollBar.getMaximum());
        });
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new Main().setVisible(true);

            }
        });
    }

    private static class CustomOutputStream extends OutputStream {
        private JTextArea textArea;
        private ByteArrayOutputStream buffer;

        public CustomOutputStream(JTextArea textArea) {
            this.textArea = textArea;
            this.buffer = new ByteArrayOutputStream();
        }

        @Override
        public void write(int b) throws IOException {
            buffer.write(b);
        }

        @Override
        public void flush() throws IOException {
            String output = buffer.toString("UTF-8");
            textArea.append(output);
            textArea.setCaretPosition(textArea.getDocument().getLength());
            buffer.reset();
```

```
            }

        }
}
```

ДОДАТОК В

Клас LecturesScannerV1.java

```
// -*- coding: utf-8 -*-
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import java.io.*;
import java.net.URISyntaxException;
import java.nio.file.Path;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;

public class LecturesScannerV1 {

    //Reference file name
    final String REFERENCE_FILE_NAME = "reference.docx";
    //Path to the directory where the files to be scanned are located
    private Path reference_path;
    //Reference file path
    private File reference_file;
    //Docx scan files quantity
    private int scan_files_quantity;

    ///Scan stuff
    private List<String> referenceListYellow = new ArrayList<>();
    private List<String> referenceListCyan = new ArrayList<>();

    private ArrayList<String> listUnderTestYellow = new ArrayList<>();
    private ArrayList<String> listUnderTestCyan = new ArrayList<>();
    private String studentName = "####";

    //SET isRefListParsingNow = false WHEN PARSING scanDocument.xml !!!!!!!
    private boolean isRefListParsingNow = false;

    //REPORT STRINGS ARRAY
    private ArrayList<String> report;

    public LecturesScannerV1() {
        getParentDirectoryPath();


        report = new ArrayList<>();
        report.add("Звіт про перевірку робіт");
        report.add("№ | Прізвище | % співп. жовтого | % співп. бірюзового");
    }

    public void mainFunction() {
        File referenceDirectory = new File(String.valueOf(reference_path));

        //1# Get files list in reference directory
        File[] filesList = referenceDirectory.listFiles();

        //2# If reference file was not found, terminate program
```

```
        if (!refFileCheck(filesList)) {
            return;
        }

        //3# Extracting referenceDocument.xml
        reference_file = new File(reference_path + File.separator +
REFERENCE_FILE_NAME);
        extractDocXml(reference_file, "reference");

        scan_files_quantity = filesList.length;

        //4# Files list optimization
        for (int i = 0; i < filesList.length; i++) {
            //if file is file and file isn`t .docx
            if (!isDocx(filesList[i]) ||
filesList[i].getName().equals(REFERENCE_FILE_NAME)) {
                filesList[i] = null;
                scan_files_quantity--;
            }

        }

        //5# If there are no scan files, terminate program
        if (scan_files_quantity == 0) {
            File file = new File(reference_path.toString() +
"\\SCAN_FILES_NOT_FOUND.txt");
            try {
                file.createNewFile();
            } catch (IOException e) {
                e.printStackTrace();
            }

            cleanTrashXML();
            return;
        }

        //6# Container for students marks
        ArrayList<String> studentsMarks = new ArrayList<>();

        //7# Scanning referenceDocument.xml
        isRefListParsingNow = true;
        parseXML();
        System.out.println("Reference data:");
        System.out.println("Yellow: " + referenceListYellow);
        System.out.println("Cyan: " + referenceListCyan);

        //8# Scanning scanDocument.xml
        isRefListParsingNow = false;

        int report_lines_counter = 1;
        for (int i = 0; i < filesList.length; i++) {
            if (filesList[i] != null) {
                extractDocXml(filesList[i], "scan");

                //SET isRefListParsingNow = false WHEN PARSING scanDocument.xml
!!!!!!!
                parseXML();

                //Creating new line of report
                StringBuilder report_line = new StringBuilder();
                report_line.append(report_lines_counter++).append(") ");

                report_line.append(studentName).append("\t");
```

```
                studentName = "####";

                //A pattern for rounding to two decimal places
                DecimalFormat decimalFormat = new DecimalFormat("#.##");

                //Calculation the percentage of matches of highlighted YELLOW
text
                report_line

.append(decimalFormat.format(compareHighLightedYellowText()))
                        .append("\t\t");

                //Calculation the percentage of matches of highlighted CYAN text
                report_line

.append(decimalFormat.format(compareHighLightedCyanText()))
                        .append("\t\t");

                printReferenceYellow();
                printScanYellow();

                //Adding new line to report
                report.add(report_line.toString());

                //Deleting path of docx that was scanned
                filesList[i] = null;

                //Renewing lists
                listUnderTestYellow = new ArrayList<>();
                listUnderTestCyan = new ArrayList<>();
            }
        }

        //9# Printing report in report.txt
        printReport();
        System.out.println("Звіт про перевірку робіт збережено до файлу
report.txt");
        System.out.println("Зміст файлу report.txt:");
        try (BufferedReader reader = new BufferedReader(new
FileReader(reference_path + "\\report.txt"))) {
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        //10# DELETER TRASH XML FILES
        cleanTrashXML();

    }

    private void getParentDirectoryPath() {
        try {
            String jarPath = getClass()
                    .getProtectionDomain()
                    .getCodeSource()
                    .getLocation()
                    .toURI()
                    .getPath();

            jarPath = jarPath.substring(1);
```

```
            reference_path = Path.of(jarPath).getParent();

        } catch (URISyntaxException e) {
            e.printStackTrace();
        }
    }

    private boolean refFileCheck(File[] filesList) {
        boolean result = false;
        for (File file : filesList) {
            if (file.getName().equals(REFERENCE_FILE_NAME)) result = true;
        }

        if (result) {
            return result;
        } else {
            File file = new File(reference_path.toString() +
"\\REFERENCE_FILE_NOT_FOUND.txt");
            System.out.println("Еталонний файл не знайдено");

            try {
                file.createNewFile();
            } catch (IOException e) {
                e.printStackTrace();
            }

            return result;
        }
    }

    private boolean isDocx(File file) {
        String name = file.getName();
        try {
            name = name.substring(file.getName().length() - 5);
        } catch (StringIndexOutOfBoundsException e) {
        }
        if (name.equals(".docx")) return true;
        return false;
    }

    private void extractDocXml(File file, String additionalString) {

        try (ZipInputStream zin = new ZipInputStream(new
FileInputStream(file.getPath()))) {
            ZipEntry entry;
            String name;
            while ((entry = zin.getNextEntry()) != null) {
                name = entry.getName(); // получим название файла
                // распаковка
                if (name.equals("word/document.xml")) {
                    FileOutputStream fout = new FileOutputStream
                            (file.getParent() + "\\" + additionalString +
"Document.xml");
                    for (int c = zin.read(); c != -1; c = zin.read()) {
                        fout.write(c);
                    }
                    fout.flush();
                    zin.closeEntry();
                    fout.close();
                }
            }
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
```

```
        }
    }

    private void printReport() {
        try (FileWriter writer = new FileWriter(reference_path + "\\report.txt",
false)) {
            // запись всей строки
            for (String text : report) {
                writer.write(text + "\n");
            }
            writer.flush();
        } catch (IOException ex) {

            System.out.println(ex.getMessage());
        }
    }

    private void cleanTrashXML() {
        File file = new File(reference_path + "\\referenceDocument.xml");
        file.delete();
        file = new File(reference_path + "\\scanDocument.xml");
        file.delete();

    }

    ////////////////////////////////////////////////////////////////
    /// SCANNER STUFF///
    ////////////////////////////////////////////////////////////////

    class XMLHandler extends DefaultHandler {

        private boolean yellowHighlighterFlag = false, cyanHighlighterFlag =
false,
                redHighlighterFlag = false;

        private String lastElementName;

        @Override
        public void startElement(String uri, String localName, String qName,
Attributes attributes) throws SAXException {
            lastElementName = qName;

            if (qName.equals("w:highlight") &&
attributes.getValue("w:val").equals("yellow")) {
                yellowHighlighterFlag = true;
            }
            if (qName.equals("w:highlight") &&
attributes.getValue("w:val").equals("cyan")) {
                cyanHighlighterFlag = true;
            }
            if (qName.equals("w:highlight") &&
attributes.getValue("w:val").equals("red")) {
                redHighlighterFlag = true;
            }
        }

        @Override
        public void characters(char[] ch, int start, int length) throws
SAXException {
            String information = new String(ch, start, length);
            information = information.replace("\n", " ").trim();

            //INSERT HIGHLIGHTED YELLOW TEXT IN CERTAIN LIST
```

```
            if (!information.isEmpty() && lastElementName.equals("w:t") &&
yellowHighlighterFlag) {
                if (isRefListParsingNow) {
                    referenceListYellow.add(information);
                } else {
                    listUnderTestYellow.add(information);
                }
                yellowHighlighterFlag = false;
            }

            //INSERT HIGHLIGHTED CYAN TEXT IN CERTAIN LIST
            if (!information.isEmpty() && lastElementName.equals("w:t") &&
cyanHighlighterFlag) {
                if (isRefListParsingNow) {
                    referenceListCyan.add(information);
                } else {
                    listUnderTestCyan.add(information);
                }
                cyanHighlighterFlag = false;
            }

            //INSERT HIGHLIGHTED RED TEXT IN STUDENT NAME VARIABLE
            if (!information.isEmpty() && lastElementName.equals("w:t") &&
redHighlighterFlag) {
                studentName = information;
                redHighlighterFlag = false;
            }
        }
    }

    private void parseXML() {
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser parser = null;
        try {
            parser = factory.newSAXParser();
        } catch (ParserConfigurationException | SAXException e) {
        }


        XMLHandler handler = new XMLHandler();
        try {
            if (isRefListParsingNow) {
                parser.parse(new File(reference_path +
"\\referenceDocument.xml"), handler);
            } else {
                parser.parse(new File(reference_path + "\\scanDocument.xml"),
handler);
            }

        } catch (SAXException | IOException ex) {
        }
    }

    private double compareHighLightedYellowText() {
        //YELLOW
        int referenceListYellowWeight = 0;
        for (String string : referenceListYellow) {
            referenceListYellowWeight += string.length();
        }

        int scanListYellowWeight = 0;
        int scan_init_position = 0;
        for (int i = 0; i < listUnderTestYellow.size(); i++) {
```

```
            for (int j = scan_init_position; j < referenceListYellow.size();
j++) {
                if
(listUnderTestYellow.get(i).equals(referenceListYellow.get(j))) {

                    System.out.println(listUnderTestYellow.get(i));
                    System.out.println("Weight = " +
listUnderTestYellow.get(i).length());
                    System.out.println();

                    scan_init_position = j;
                    scanListYellowWeight += listUnderTestYellow.get(i).length();
                    break;
                }
            }
        }

        System.out.println();
        System.out.println("Reference Yellow Weight = " +
referenceListYellowWeight);
        System.out.println("Scan Yellow Weight = " + scanListYellowWeight);

        return (double) scanListYellowWeight * 100.0 / (double)
referenceListYellowWeight;
    }

    private double compareHighLightedCyanText() {
        //CYAN
        int referenceListCyanWeight = 0;
        for (String string : referenceListCyan) {
            referenceListCyanWeight += string.length();
        }
        int scanListCyanWeight = 0;
        int scan_init_position = 0;
        for (int i = 0; i < listUnderTestCyan.size(); i++) {
            for (int j = scan_init_position; j < referenceListCyan.size(); j++)
{
                if (listUnderTestCyan.get(i).equals(referenceListCyan.get(j))) {
                    scan_init_position = j;
                    scanListCyanWeight += listUnderTestCyan.get(i).length();
                    break;
                }
            }
        }
        return (double) scanListCyanWeight * 100.0 / (double)
referenceListCyanWeight;
    }

    private void printReferenceYellow() {
        try (FileWriter writer = new FileWriter(reference_path +
"\\referenceYellow.txt", false)) {
            // запись всей строки
            for (String text : referenceListYellow) {
                writer.write(text + "\n");
            }
            writer.flush();
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }

    private void printScanYellow() {
        try (FileWriter writer = new FileWriter(reference_path +
```

```
"\\scanYellow.txt", false)) {
            // запись всей строки
            for (String text : listUnderTestYellow) {
                writer.write(text + "\n");
            }
            writer.flush();
        } catch (IOException ex) {
            System.out.println(ex.getMessage());
        }
    }


}
```

ДОДАТОК Г

Клас ShingleLevenstain

```
// -*- coding: utf-8 -*-
import java.io.FileInputStream;
import java.io.IOException;
import java.io.File;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import java.net.URISyntaxException;
import java.nio.file.Path;


import org.apache.poi.xwpf.extractor.XWPFWordExtractor;
import org.apache.poi.xwpf.usermodel.XWPFDocument;

public class ShingleLevenstain {
    private Path jarDirectory;
    private static final int SHINGLE_SIZE = 5; // Рорзмір шинглів (к-сть слів)

    public ShingleLevenstain() {
        getParentDirectoryPath();
    }

    public void runShingleLevenstain(){
        List<String> fileContents = new ArrayList<>();
        List<String> fileNames = new ArrayList<>();
        File directory = jarDirectory.toFile();
        File[] files = directory.listFiles((dir, name) ->
name.endsWith(".docx"));


        if (files == null || files.length < 2) {
            System.out.println("Недостатньо файлів для порівняння.");
            return;
        }

        for (File file : files) {
            String fileName = file.getName();
            String fileContent = extractText(file.getAbsolutePath());
            if (fileContent != null) {
                fileContents.add(fileContent);
                String nameWithoutExtension = fileName.substring(0,
fileName.lastIndexOf('.'));
                fileNames.add(nameWithoutExtension);
            } else {
                System.out.println("Помилка зчитування файлу: " +
file.getAbsolutePath());
            }
        }

        for (int i = 0; i < fileContents.size(); i++) {
            for (int j = i + 1; j < fileContents.size(); j++) {
                Set<String> shingles1 = computeShingles(fileContents.get(i));
                Set<String> shingles2 = computeShingles(fileContents.get(j));

                double similarity = computeSimilarity(shingles1, shingles2);
                double plagiarismPercentage = similarity * 100;
```

```
                System.out.printf("Відсоток плагіату між файлами '%s' и '%s':
%.2f%%\n", fileNames.get(i), fileNames.get(j), plagiarismPercentage);
            }
        }
        System.out.println("Порівняння файлів завершено.");
    }
        private void getParentDirectoryPath() {
            try {
                String jarPath = getClass()
                        .getProtectionDomain()
                        .getCodeSource()
                        .getLocation()
                        .toURI()
                        .getPath();

                jarDirectory = new File(jarPath).getParentFile().toPath();

            } catch (URISyntaxException e) {
                e.printStackTrace();
            }
        }
    private static double computeSimilarity(Set<String> set1, Set<String> set2)
{

        Set<String> smallerSet, largerSet;
        if (set1.size() < set2.size()) {
            smallerSet = set1;
            largerSet = set2;
        } else {
            smallerSet = set2;
            largerSet = set1;
        }

        double similarity = 0.0;

        for (String shingle : smallerSet) {
            if (largerSet.contains(shingle)) {
                similarity++;
            } else {
                double maxSimilarity = 0.0;
                for (String other : largerSet) {
                    double shingleSimilarity = computeStringDistance(shingle,
other);

                    if (shingleSimilarity > maxSimilarity) {
                        maxSimilarity = shingleSimilarity;
                    }
                }
                similarity += maxSimilarity;
            }
        }

        return similarity / smallerSet.size();
    }

    private static double computeStringDistance(String s1, String s2) {
        int[] costs = new int[s2.length() + 1];
        for (int j = 0; j < costs.length; j++) {
            costs[j] = j;
        }

        for (int i = 1; i <= s1.length(); i++) {
            costs[0] = i;
```

```
            int nw = i - 1;
            for (int j = 1; j <= s2.length(); j++) {
                int cj = Math.min(1 + Math.min(costs[j], costs[j - 1]),
s1.charAt(i - 1) == s2.charAt(j - 1) ? nw : nw + 1);
                nw = costs[j];
                costs[j] = cj;
            }
        }

        double distance = costs[s2.length()];
        double maxLength = Math.max(s1.length(), s2.length());
        return 1.0 - distance / maxLength;
    }

    private static String extractText(String relativePath) {
        try {
            Path currentPath = Paths.get(System.getProperty("user.dir"));
            Path filePath = currentPath.resolve(relativePath).normalize();
            File file = filePath.toFile();

            if (!file.exists() || !file.isFile()) {
                System.err.println("Файл не найден: " + filePath);
                return null;
            }

            try (FileInputStream fis = new FileInputStream(file);
                 XWPFDocument doc = new XWPFDocument(fis)) {
                XWPFWordExtractor extractor = new XWPFWordExtractor(doc);
                return extractor.getText();
            }
        } catch (IOException e) {
            System.err.println("Помилка читання файлу " + relativePath + ": " +
e.getMessage());
            return null;
        }
    }

    private static Set<String> computeShingles(String text) {
        Set<String> shingles = new HashSet<>();
        List<String> tokens = tokenize(text);

        for (int i = 0; i <= tokens.size() - SHINGLE_SIZE; i++) {
            StringBuilder shingle = new StringBuilder();
            for (int j = 0; j < SHINGLE_SIZE; j++) {
                shingle.append(tokens.get(i + j)).append(" ");
            }
            shingles.add(shingle.toString().trim());
        }

        return shingles;
    }

    private static List<String> tokenize(String text) {
        List<String> tokens = new ArrayList<>();
        StringBuilder token = new StringBuilder();

        for (int i = 0; i < text.length(); i++) {
            char c = text.charAt(i);
            if (Character.isLetterOrDigit(c)) {
                token.append(Character.toLowerCase(c));
            } else {
                if (token.length() > 0) {
                    tokens.add(token.toString());
```

```
                    token = new StringBuilder();
                }
            }
        }

        if (token.length() > 0) {
            tokens.add(token.toString());
        }

        return tokens;
    }
}
```