

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ ЛОКАЛЬНО-ЧУТТЄВОГО
ГЕШУВАННЯ. SOFTWARE IMPLEMENTATION OF THE METHODS FOR
LOCALITY SENSITIVE HASHING

Виконав: студент 4 курсу, групи КНТ-137
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Василенко Д.С.

(прізвище та ініціали)

Керівник Субботін С.О.

(прізвище та ініціали)

Рецензент Гофман Є.О.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
 “ ” 2021 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТА

Василенка Дмитра Сергійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Програмна реалізація методів локально-чутливого
гешування. Software Implementation of the Methods for Locality Sensitive Hashing
 Керівник проекту Субботін Сергій Олександрович, д.т.н., професор,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проекту (роботи) 17 травня 2021 року

3. Вихідні дані до проекту (роботи) рекомендована література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Тестування та експлуатація програми. 5. Текст програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Субботін С.О., професор		
Нормоконтроль	Андрєєв М.О., асистент		

7. Дата видачі завдання 15 березня 2021 р. _____

КАЛЕНДАРНИЙ ПЛАН

[illegible]

Студент _____ Василенко Д.С.
(підпис) (прізвище та ініціали)

Керівник проекту _____ **Субботін С.О.**
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
86 с., 4 табл., 31 рис., 2 дод., 16 джерел.

ГЕШУВАННЯ, ГЕШ, ГЕШ-ЗНАЧЕННЯ, ДАЙДЖЕСТ, ЛОКАЛЬНО-ЧУТТЄВЕ ГЕШУВАННЯ.

Об'єкт дослідження – процес локально-чуттєвого гешування.

Предмет дослідження – програмне забезпечення для реалізації методів локально-чуттєвого гешування.

Мета роботи – розроблення програмного забезпечення, яке реалізує методи локально-чуттєвого гешування.

Матеріали, методи та технічні засоби: строга типізована об'єктно-орієнтована мова програмування Java, бібліотека графічного інтерфейсу Swing, персональний комп'ютер з процесором AMD Ryzen 5 1600 Six-Core Processor під управлінням операційної системи Microsoft Windows 7.

Результати. Розроблено програмне забезпечення, яке оцінює подібність двох текстів на основі методів локально-чуттєвого гешування.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 86 pages, 31 figures, 4 tables, 2 appendixes, 16 sources.

HASHING, HASH, HASH-VALUE, DIGEST, LOCALITY-SENSITIVE HASHING.

Object of study is the process of locality sensitive hashing.

Subject of study is the software for the implementation of the methods for locality sensitive hashing.

The purpose of the work is development of software that implements methods of local-sensitive hashing.

Materials, methods, and tools: strict typed object-oriented Java programming language, Swing GUI library, personal computer with AMD Ryzen 5 1600 Six-Core Processor running Microsoft Windows 7 operating system.

Results. Developed software that evaluates the similarity of two texts based on local-sensitive hashing methods.

ЗМІСТ

ВСТУП	7
1 ОГЛЯД МЕТОДІВ ЛОКАЛЬНО-ЧУТТЄВОГО ГЕШУВАННЯ	8
1.1 Загальні відомості про локально-чуттєве гешування	8
1.2 Огляд методів локально-чуттєвого гешування	11
1.2.1 Метод Minhash	11
1.2.2 Метод Simhash	14
1.2.3 Метод TLSH	18
1.2.4 Метод Nilsimsa	23
2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ	27
2.1 Архітектура програми	27
2.2 Вибір мови програмування	30
2.3 Вибір середовища розробки	32
2.4 Вибір бібліотеки для графічного інтерфейсу програми	33
3 РОЗРОБКА ПРОГРАМИ.....	34
3.1 Опис елементів структури програми	34
3.1.1 Опис класу Context пакету context	34
3.1.2 Опис класів пакету methodsLSH	34
3.1.3 Опис класу Main пакету main.....	37
3.2 Опис графічних компонентів програми	38
3.2.1 Опис головної форми програми	38
3.2.2 Опис дочірньої форми програми.....	40
4 ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ ПРОГРАМИ	41
4.1 Тестування програми.....	41
4.2 Інструкція по експлуатації програми.....	42
ВИСНОВКИ.....	45
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ.....	47
ДОДАТОК А Текст програми.....	49
ДОДАТОК Б Слайди презентації	80

ВСТУП

Існують багато алгоритмів для виявлення подібності даних в багатовимірних просторах. Серед них це локально-чуттєве гешування. Локально-чуттєве гешування є загальною назвою методів, котрі дозволяють отримувати схожі сигнатури для схожих даних. Кожен метод має свої переваги та недоліки.

Локально-чуттєве гешування застосовується в багатьох задачах. Цей алгоритм використовується для пошуку ближнього сусіда, пошуку дублікатів елементів та кластеризації даних. Тому локально-чуттєве гешування використовується в багатьох відомих компаніях та сервісах для створення рекомендаційних систем, антиплагіату, виявлення спаму, пошуку дублікатів заголовків веб-сторінок та ін. Локально-чуттєве гешування є швидким алгоритмом на відміну від інших, тому для багатьох задач він є актуальним підходом.

Об'єктом дослідження дипломної роботи є процес локально-чуттєвого гешування.

Предметом дослідження є програмне забезпечення для реалізації методів локально-чуттєвого гешування.

Метою дипломної роботи є розробка програмного забезпечення, яке реалізує методи локально-чуттєвого гешування.

Задачами дипломної роботи є огляд та аналіз методів локально-чуттєвого гешування, розробка архітектури програмного забезпечення, яке реалізує ці методи, вибір технологій для її розробки, розробка програмного забезпечення, тестування та її експлуатація.

1 ОГЛЯД МЕТОДІВ ЛОКАЛЬНО-ЧУТТЄВОГО ГЕШУВАННЯ

1.1 Загальні відомості про локально-чуттєве гешування

Локально-чуттєве гешування – це алгоритм, який застосовується для зниження розмірності багатовимірних даних. Ідея алгоритму полягає в тому, щоб з високою ймовірністю гешувати схожі дані в одні й ті ж самі контейнери. Гешування – це процес створення певного виводу з вихідних даних різного розміру за допомогою математичних формул, які називаються геш-функціями [1]. Отже гешування – перетворення даних, вироблене геш-функцією.

Опираючись з назви, ціллю ЛЧГ є локальна чутливість, тобто при зміні якоїсь частини об'єкта, що гешується, сам геш не повинен абсолютно змінюватись. Методи, що відносяться до локально-чуттєвого гешування на відміну від простих методів гешування не мінімізують конфлікти гешування, а навпаки намагаються максимізувати ці конфлікти.

Сімейство ЛЧГ-функцій F задовольняють двом умовам для двох точок x_1 та x_2 , використовуючи функцію $h \in F$:

$$\begin{aligned} P(h(x_1) = h(x_2)) &\geq p_1, \text{ якщо } d(x_1, x_2) \leq r, \\ P(h(x_1) = h(x_2)) &\leq p_2, \text{ якщо } d(x_1, x_2) \geq cr \quad [2], \end{aligned}$$

де d – міра подібності;

r – порогове значення;

c – коефіцієнт наближення;

P – ймовірність.

Якщо $p_1 > p_2$, тоді сімейство F називають чуттєвою.

У випадку якщо дані знаходяться близько один від одного відносно функції, то ймовірність того, що вони будуть гешуватися в один і той же контейнер є високою. В іншому випадку дані з низькою ймовірністю будуть

гешуватися в один і той же контейнер. Для більш кращого розуміння на рисунку 1.1 представлено реалізацію локально-чуттєвого гешування.

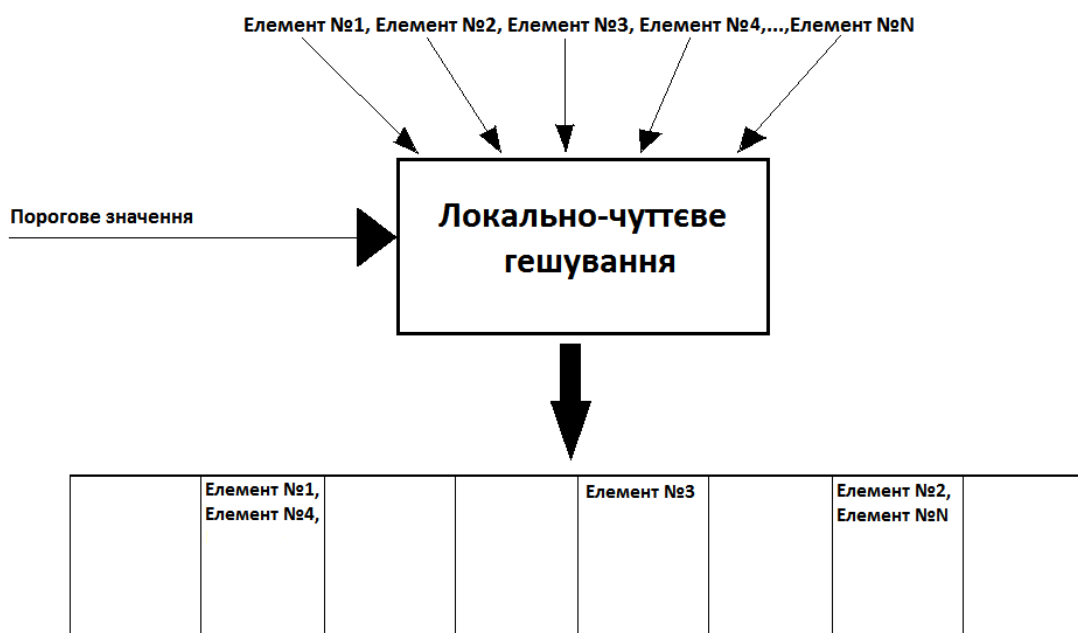


Рисунок 1.1 – Реалізація локально-чуттєвого гешування

На рисунку вхідними даними є набір елементів, а вихідними – таблиці(у даному випадку одна), записи яких мають елементи. Елементи, які подібні між собою мають значення більше або дорівнює пороговому значенню. У протилежному випадку – елементи не мають подібності.

Отже, ЛЧГ є алгоритмом, який відображає подібні елементи з однаковим значенням, а різні елементи відображаються з різними значеннями. Подібність між залежить від встановлення порогового значення.

Локально-чуттєве гешування вважається найбільш багатообіцяючим підходом для пошуку подібності в багатовимірних просторах. Його головними перевагами є простота у застосуванні та висока швидкість алгоритму при порівнянні з іншими.

Однак, незважаючи на переваги ЛЧГ не є досить точним. Алгоритм може повертати помилково позитивні та помилково негативні результати.

Помилково позитивні спрацьовування трапляються коли пари елементів мають подібність між собою, але між ними низький поріг. Помилково негативні спрацьовування трапляються у випадку коли елементи є різними, але між ними є високе порогове значення [3].

Локально-чуттєве гешування має широку область застосувань. Воно використовується в таких областях науки про дані як пошук ближнього сусіда(ЛЧГ є одним з найефективніших способів ближнього сусіда тому, що схожі елементи мають майже однакові геші), пошук дублікатів(однакові елементи можна ефективно виявити без порівняння с кожним елементом в наборі даних, бо ЛЧГ забезпечує попадання однакових елементів в один контейнер), зниження розмірності(геш-значення для елемента є представлення елемента в більш меншій розмірності), кластеризація(елементи в однаковому геші схожі і можуть належати кластеру).

Нижче наведений перелік, де локально-чуттєве гешування може застосовуватися:

- комп'ютерна безпека;
- антиплагіат;
- розпізнання подібності аудіо, відео, зображень і т.д.;
- розпізнання схожості експресії генів;
- навчання нейронних мереж;
- ієрархічна кластеризація;
- пошук ближнього сусіда.

Локально-чуттєве гешування використовується в таких відомих сервісах і компаніях як Google, Uber, Spotify, Eventbrite і т.д. Google використовує цей алгоритм для виявлення дублікатів веб-сторінок та рекомендацій новин. Uber застосовує ЛЧГ для знаходження схожих маршрутів. Spotify використовує локально-чуттєве гешування для рекомендацій. За допомогою ЛЧГ сервіс Eventbrite виявляє спам та шкідливі повідомлення для організаторів подій. Отже, локально-чуттєве гешування застосовується в багатьох сервісах, науках

та в деяких задачах. Воно є дуже корисним засобом для вирішення багатьох задач через його швидкість та простоту алгоритму.

1.2 Огляд методів локально-чуттєвого гешування

Локальне-чуттєве гешування представляє ряд методів, що дозволяють отримати схожі сигнатури для схожих наборів даних. В основному методи, що входять до ЛЧГ використовують поділ даних на частини та гешування цих частин з наступним випадковим(але згідно певного правила чи ознаки) відбиранням елементів з отриманої колекції. Відібрані елементи одного масиву даних можуть порівнюватися з елементами іншого за допомогою коефіцієнта Жаккара або конкатенуватися у текстовий рядок і порівнюватися між собою за допомогою інших метрик(наприклад, за відстанню Хеммінга) [4]. У наступних пунктах буде проводитися детальний огляд кожного із методів ЛЧГ.

1.2.1 Метод Minhash

Метод Minhash – є одним із методів локально-чуттєвого гешування. Він є технікою швидкого оцінювання того, наскільки схожі два набори. Метод був винайдений Андрієм Бродером і спочатку використовувався в пошуковій системі AltaVista для виявлення дублікатів веб-сторінок та усунення їх із результатів пошуку [5]-[6].

Ідеєю методу Minhash є отримання сигнатури з кожного елементу, щоб надалі можна було оцінити схожість цих сигнатур за допомогою коефіцієнта Жаккара.

Коефіцієнт Жаккара – це міра подібності двох наборів. Коефіцієнт Жаккара визначається як відношення розміру перетину двох множин A та B та розміру їх поєднання:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

Графічне представлення визначення коефіцієнта Жаккара наведено на рисунку 1.2.

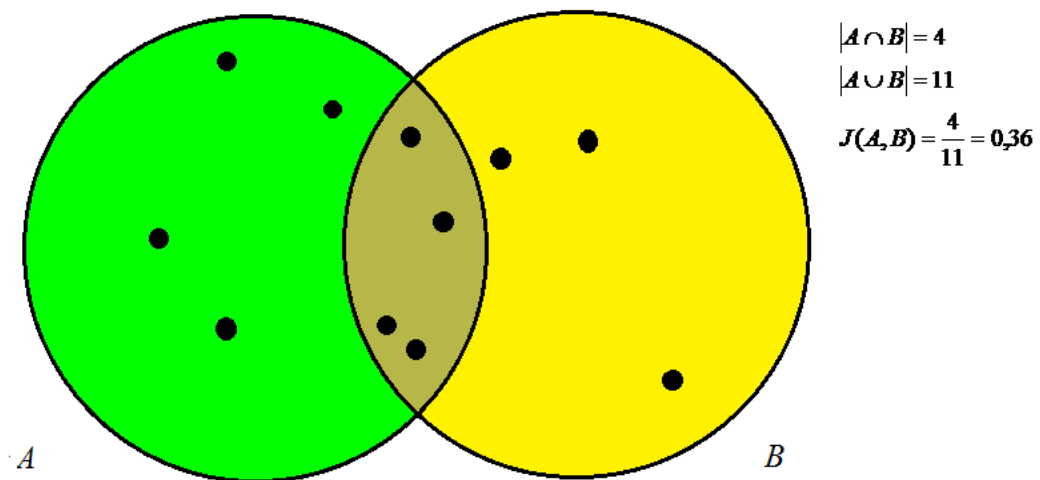


Рисунок 1.2 – Графічне представлення визначення коефіцієнта Жаккара

Коефіцієнт Жаккара має результат як число від нуля до одиниці. Чим більш подібні два елементи, тим ближче результат до 1, якщо елементи менш схожі між собою, тоді результат буде ближчий до нуля. Отже, для ідентичних наборів даних коефіцієнт Жаккара буде дорівнювати одиниці, абсолютно різні – нулю.

Метод Minhash має тісний зв'язок з коефіцієнтом Жаккара: ймовірність того, що геш-функція для випадкової перестановки рядків отримує одне і теж значення для двох наборів дорівнює коефіцієнту Жаккара цих наборів [7]:

$$P(h(A) = h(B)) = J(A, B),$$

де P – ймовірність;

h – геш-функція;

A та B – набори;

J – коефіцієнт Жаккара.

Метод Minhash представляє елементи як колекцію множин. Ця колекція потім використовується для побудови характеристичної матриці. Далі необхідно визначити геш для кожного стовпця характеристичної матриці, значення геш стовпця – це значення індексу першого рядка в перестановці якій стовпець має 1. Для визначення гешу для усієї колекції необхідно обрати кількість перестановок рядків. Однак для великої кількості характеристичної матриці необхідно переставляти мільйони та навіть мільярди рядків, такий підхід займає багато часу. Для вирішення цієї проблеми застосовуються геш-функції, які моделюють ефект випадкових перестановок. Тобто замість вибору кількості випадкових перестановок рядків, достатньо обрати кількість випадкових геш-функцій. Нижче наведено рисунок 1.3 [8], який показує алгоритм методу з випадковими перестановками рядків:

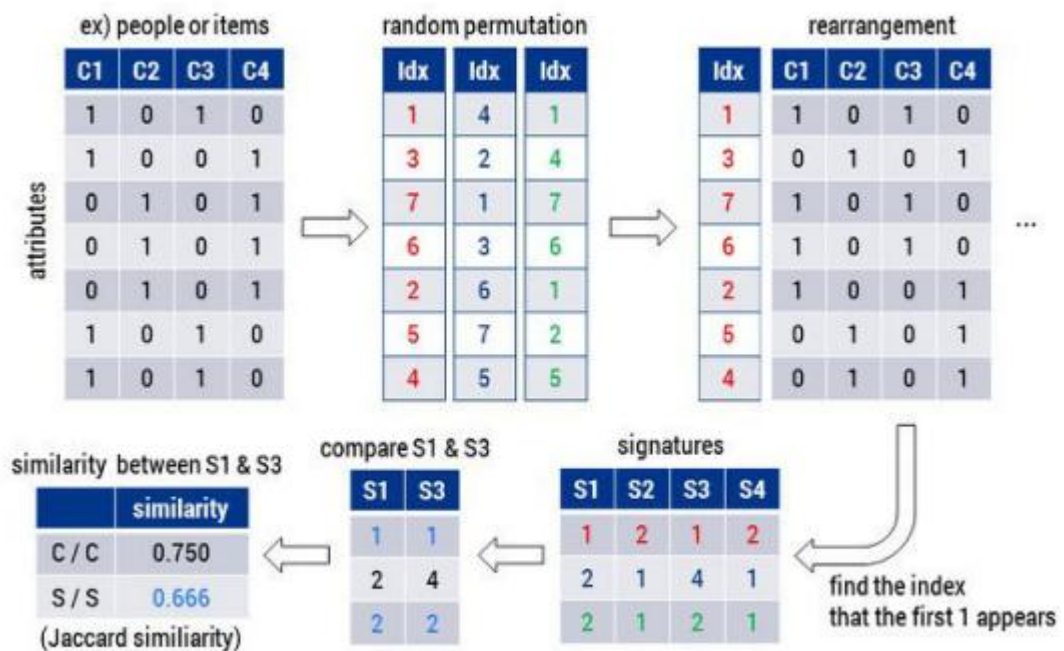


Figure 3. Minwise Hashing

Рисунок 1.3 – Алгоритм методу Minhash [8]

Покроковий алгоритм наведений нижче.

Крок 1. Обрати кількість геш-функцій $h_1, h_2, \dots, h_n$.

Крок 2. Для кожної множини із S .

Крок 3. Побудувати характеристичну матрицю C .

Крок 4. Побудувати матрицю сигнатур Sig , де $Sig(i, j)$ відповідає i -тої геш-функції і стовпцю j .

Крок 5. Для кожного рядка r :

- відносити $h_1(r), h_2(r), \dots, h_n(r)$;
- для кожного стовпця j : якщо $C(r, j) = 1$, тоді $Sig(i, j) = \min(Sig(i, j), h_i(r))$.

Крок 6. Оцінити подібність між множинами використовуючи фінальну версію матриці сигнатур.

Точність методу Minhash буде залежати від вибору кількості геш-функцій. Чим більше геш-функцій було обрано, тим точніше буде результат. Однак, при використанні великої кількості геш-функцій необхідно буде використати великий об'єм пам'яті, що є недоліком цього методу. Ще одним недоліком методу є похибка результатів подібності між множинами, який видає коефіцієнт Жаккара.

Метод Minhash застосовується в багатьох задачах. Цей метод підходить для вирішення задач кластеризації та виникнення дублікатів серед елементів. Метод також має застосування в біоінформатиці для порівняння послідовностей геномів. Minhash використовується в пошукових системах, системах машинного навчання та в відповідності змісту.

1.2.2 Метод Simhash

Метод, названий Simhash є одним із методів локально-чутливого гешування. Цей метод використовується для порівняння великих наборів даних.

Засновником методу Simhash являється один із співробітників Google Мозес Чарикар [9].

Ідея методу полягає в тому, щоб зменшити розмірність елементу(наприклад тексту) для генерування значення геш-функції, щоб потім оцінити подібність цих геш-значень за відстанню Хеммінга або інших метрик подібності(наприклад косинусного коефіцієнта) на відміну від методу Minhash.

Відстань Хеммінга – це число позицій, у яких відповідні цифри двох двійкових слів однакової довжини різні [10]. Нехай дані два двійкові рядки 1011 та 1101, і надамо їм назви змінних x та y відповідно. Визначимо відстань Хеммінга між x та y :

$$d(x, y) = 2.$$

Як бачимо відстань між x та y дорівнює двом тому, що біти, які знаходяться на другій позиції після крайніх лівих бітів та біти, котрі знаходяться на другій позиції після крайніх правих бітів цих рядків відрізняються, тому результат має значення два.

Для відстані Хеммінга існують такі характерні властивості:

- _ $d(x, y) \geq 0$;
- _ $d(x, x) = 0$;
- _ $d(x, y) = d(y, x)$;
- _ $d(x, z) \leq d(x, y) + d(y, z)$.

Відстань Хеммінга використовується як метрика для елементів однакової розмірності.

Результатом алгоритму Simhash є сигнатура. Сигнатура є списком бітів який може надалі порівнюватися із другими списками бітів. Для отримання сигнатури необхідно виконати такі кроки:

Крок 1. Обрати розмір сигнатури.

Крок 2. Створити масив V такого ж розміру, як і розмір сигнатури і заповнити цей масив нулями.

Крок 3. Обрати любую геш-функцію, що дає результати розміру аналогічно обраному.

Крок 4. Для кожного набору, для кожного елементу в наборі розрахувати значення геш-функції.

Крок 5. Для кожного отриманого гешу в наборі, для кожного біта i :

- якщо біт = 1, додаємо один до $V[i]$;
- якщо біт = 0, віднімаємо один від $V[i]$.

Крок 6. Для кожного елементу i масиву V :

- якщо $V[i] > 0$, $V[i] = 1$;
- якщо $V[i] \leq 0$, $V[i] = 0$ [4].

Після усіх виконаних кроків буде отримана сигнатура у вигляді двійкового коду. Нижче наведено рисунок 1.4 який описує алгоритм Simhash.

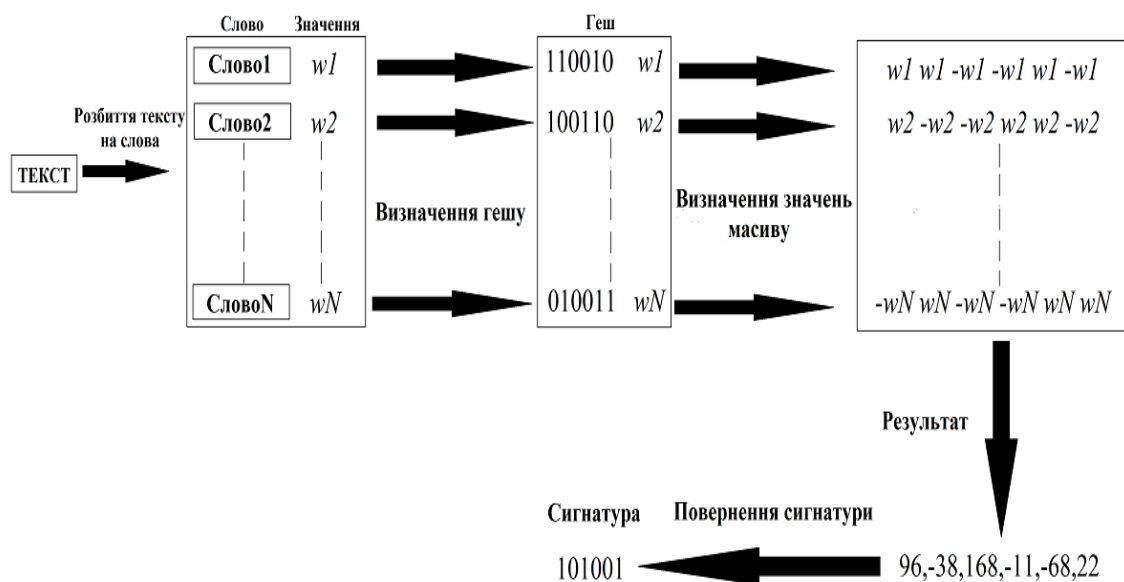


Рисунок 1.4 – Опис алгоритму Simhash

За допомогою сигнатур можна порівнювати подібність даних. Наприклад є два текстових рядка, їх сигнатури 10111001 та 10110100 визначимо їх подібність за допомогою відстані Хеммінга:

$$d(10111001, 10110100) = 2.$$

Як бачимо, відстань дорівнює двом тому, що сигнатури розрізняються між собою у двох позиціях. Далі можна оцінити відсоткову подібність цих двох сигнатур таким чином:

$$\left(1 - \frac{2}{8}\right) \times 100\% = 75\%.$$

Отже подібність між двома сигнатурами дорівнює 75%, звідки випливає, що подібність елементів теж дорівнює 75%.

Після опису алгоритму для методу Simhash можна визначити такі основні властивості:

- чим більш схожі елементи, тим меншою буде відстань Хеммінга їх геш-значень;
- схожі елементи мають схожі сигнатури;
- геш-значення елементу – це геш його частин.

На відміну від методу Minhash метод Simhash повертає результат у список бітів. У випадку з Minhash значення в масиві представляє мінімальне геш-значення всіх елементів набору щодо даної геш-функції. Головною відмінністю між цими двома методами є ймовірність геш-колізій. Геш-колізія – це коли два різних вхідних даних x та y для геш-функції h таких, що $h(x) = h(y)$ [11]. Тому, залежно від того, як визначається подібність між елементами, той чи інший метод може бути доцільним.

Метод Simhash є швидким алгоритмом та має легшу реалізацію на відміну від інших методів локально-чутливого гешування. Відомо, що Google запатентовав цей алгоритм для пошуку дублікатів веб-сторінок [12].

1.2.3 Метод TLSH

Метод Trend Micro Locality Sensitive Hashing(TLSH) – це метод гешування, який належить до локально-чуттєвого гешування. TLSH був розроблений Джоном Олівером та Джейсоном Прайдом із компанії Trend Micro для поєднання з машинним навчанням з ціллю виявлення зловмисних файлів. TLSH генерує геш-значення, які потім можна проаналізувати на наявність подібностей, завдяки цьому можна визначити чи безпечно запустити файл на основі його подібності з відомими файлами [13].

Ідея TLSH полягає в створенні дайджестів для пошуку подібності між даними. Дайджести – це гешований рядок фіксованої довжини, отриманий у результаті перетворення повідомлення довільної довжини. Метод обробляє вхідний об'єкт, використовуючи розсувне вікно розміром 5 байтів, перелічуючи байт за байтом, де на кожному кроці обирається шість триграм(комбінацій символів вікна) для заповнення 128-бітового масиву підрахунків контейнерів за геш-функції Пірсона. Геш-функція Пірсона – це така геш-функція, яка приймає на вході будь-яку кількість байтів та видає на виході один байт від 0-255. Після заповнення масиву обчислюються квартильні точки масиву. Дайджест з фіксованою довжиною в 35 байт – це вихід, згенерований методом TLSH, де перші три байти складають заголовок дайджесту, а решта – його тіло. Заголовок побудований на основі квартильних точок розміру об'єкта та контрольної суми. Тіло дайджесту побудоване таким чином, що кожне положення масиву порівнюється з квартильними точками. Результатом між цими точками є бітова пара, визначена відповідно до величини квартиля, на якій вона знаходиться.

Нижче приведений покроковий алгоритм для отримання дайджесту.

Крок 1. Обробка байтового рядка за допомогою розсунутого вікна.

Крок 2. Обчислення квартильних точок q_1 , q_2 та q_3 .

Крок 3. Побудова заголовку дайджесту.

Крок 4. Побудова тіла дайджесту.

Крок 5. Складання заголовку та тіла дайджесту.

На першому кроці за допомогою розсувного вікна обробляється рядок для заповнення масиву підрахунків контейнерів. На рисунку 1.5 [14] наведений цей процес.

```

For ew = 4 to len-1 {
    // sw is the start of the window
    sw = ew - 4
    For tri = 1 to 6 {
        (c1,c2,c3)=Triplet(tri,Byte[sw..ew])
        bi = b_mapping(c1, c2, c3)
        bucket[ bi ] ++
    }
}

```

Рисунок 1.5 – Процес обробки рядка за допомогою розсувного вікна [14]

Розсувне вікно розміром 5 байт використовується для вилучення триграм. Нехай розсувне вікно має байти {А, Б, В, Г, Д}. З цих байтів можливо отримати такі триграми:

- АБВ;
- АБГ;
- АБД;
- АВГ;
- АВД;
- АГД;
- БВГ;
- БВД;
- БГД;
- ВГД.

Останні три триграми пропускаються тому, що обробляються рухом розсувного вікна, отже загалом виходить шість триграм.

На рисунку 1.3 змінні $c1$, $c2$ та $c3$ є символами триграми. Функція `b_mapping` є функцією відображення, яка називається геш-функцією Пірсона.

Після виконання першого кроку здійснюється перехід до другого, на котрому здійснюється обчислення кuartильних точок q_1 , q_2 та q_3 таким чином:

- 75% кількості контейнерів $\geq q_1$;
- 50% кількості контейнерів $\geq q_2$;
- 25% кількості контейнерів $\geq q_3$ [14].

На третьому кроці відбувається побудова заголовку дайджесту. Заголовок дайджесту – це перші три байти гешу. Перший байт є контрольною суммою рядка байтів за модулем 256. Другий байт є значенням логарифму довжини рядка байтів за модулем 256. Третій байт є результатом побудови двох 16-бітових величин, які визначаються за формулами (1.1) та (1.2):

$$v_1 = \left(\frac{q_1 \times 100}{q_3} \right) \bmod 16, \quad (1.1)$$

$$v_2 = \left(\frac{q_2 \times 100}{q_3} \right) \bmod 16, \quad (1.2)$$

де v_1 та v_2 - 16-бітові величини;

q_1 , q_2 та q_3 - кuartильні точки.

На четвертому кроці будується тіло дайджесту. Побудова дайджест-тіла будується за допомогою масиву контейнерів, користуючись такою процедурою, яка зображена на рисунку 1.6 [14].

```

For bi = 0 to 127 {
    if bucket[bi] <= q1      Emit(00)
    else if bucket[bi] <= q2 Emit(01)
    else if bucket[bi] <= q3 Emit(10)
    else                    Emit(11)
}

```

Рисунок 1.6 – Процедура побудови тіла дайджесту [14]

У цьому процесі виконується порівняння всіх позицій масиву лічильників контейнерів з кватильними точками. Результатом порівняння є пара бітів, після усіх порівнянь пари бітів об'єднуються для отримання тіла дайджесту.

На п'ятому кроці заголовок та тіло дайджесту об'єднуються між собою. Результатом об'єднання є остаточний дайджест, побудований із байтового рядка, який може використовуватися для обчислення різниці між іншими дайджестами для пошуку подібності.

Для розрахунку різниці між дайджестами метод TLSH застосовує оцінку відстані. Оцінка відстані між дайджестами розраховується як сума відстані заголовків та тіл дайджестів. Алгоритми функцій для розрахунку відстані між заголовками та тіл дайджестів представлені на рисунку 1.7 [14] та на рисунку 1.8 [14].

```

Function distance_headers(tX, tY)
int diff=0
ldiff = mod_diff(tX.lvalue, tY.lvalue, 256);
If ldiff <= 1
    diff = diff + ldiff
else
    diff = diff + ldiff * 12;
qldiff = mod_diff(tX.q1ratio, tY.q1ratio, 16);
If qldiff <= 1
    diff = diff + qldiff
else
    diff = diff + (qldiff-1) * 12;
q2diff = mod_diff(tX.q2ratio, tY.q2ratio, 16);
If q2diff <= 1
    diff = diff + q2diff
else
    diff = diff + (q2diff-1) * 12;
If tX.checksum <> tY.checksum
    diff = diff + 1
return(diff)

```

Рисунок 1.7 – Процедура визначення відстані заголовків дайджестів [14]

```

Function distance_bodies(tX, tY)
int diff=0
For I = 1 to 64 {
    x1 = tX.hex[i+5] / 4
    x2 = tX.hex[i+5] % 4
    y1 = tY.hex[i+5] / 4
    y2 = tY.hex[i+5] % 4
    d1 = abs(x1 - y1)
    d2 = abs(x2 - y2)
    if (d1 == 3)
        diff = diff + 6
    else
        diff = diff + d1
    if (d2 == 3)
        diff = diff + 6
    else
        diff = diff + d2
}
return(diff)

```

Рисунок 1.8 – Процедура визначення відстані тіл дайджестів [14]

При розрахунку відстані заголовків тіл дайджестів використовується функція `mod_diff` для оцінки відстані між різними частинами заголовків. Ця функція обчислює мінімальну кількість кроків між двома значеннями у круговій черзі.

Функція розрахунку відстані між тілами дайджестів обчислює апроксимацію відстані Хеммінга. Різниця з відстанню Хеммінга отримується шляхом додавання значення шість до відстані, коли лічильники контейнерів першого та другої дайджестів розташовані відповідно один нижньому кварталі, а інший у верхньому кварталі.

Після розрахунку відстаней заголовків та тіл між дайджестами можна визначити оцінку подібності склавши їх. Оцінка подібності повертає значення, яке визначає різницю між дайджестами.

- якщо значення оцінки дорівнює нулю, тоді дайджести є ідентичними;
- якщо значення оцінки більше нуля, тоді дайджести різні і чим більше значення оцінки, тим більше різниця між ними.

Отже, якщо дайджести є ідентичними, тоді дані теж є ідентичними, у протилежному випадку: якщо дайджести різні, тоді і дані теж різні.

Метод TLSH є ефективним методом локально-чуттєвого гешування. Він використовується в багатьох задачах комп'ютерної безпеки та цифрових технологій. Сервіси Malware Bazaar та VirusTotal використовують цей метод для пошуку зразків за назвою сімейств зловмисних програмних забезпечень. TLSH може бути корисним підприємствам для визначення безпечних програм та файлів їхнього середовища [15].

1.2.4 Метод Nilsimsa

Алгоритм названий Nilsimsa є одним із методів локально-чуттєвого гешування, який в основному використовується для боротьби зі спамом. Метод був запропонований оператором Spamhaus у 2001 році, і потім був розглянутий

професором комп'ютерних наук Ернесто Даміані у своїй статті «Метод виявлення спаму на основі відкритого дайджесту» [13].

Метод Nilsimsa має задовольняти таким трьома вимогам:

- дайджест, що ідентифікує повідомлення не повинен суттєво змінюватися для змін, які можуть бути зроблені;
- кодування має бути надійним проти зловмисних атак;
- кодування повинно підтримувати низький ризик помилкових спрацьовувань [13].

Існують геш-функції, які генерують дайджести повідомлень (наприклад MD5 та SHA-1) для того, щоб можна було перевірити їх на схожість. Спамери використовують способи вставлення в повідомлення декілька символів, які обрані випадковим чином тим самим обходити засоби захисту. Звідки і впливає перша вимога до методу Nilsimsa. Для запобігання таких випадків необхідна така ЛЧГ-функція, яка використовується в системах пошуку інформації.

Методики, які призначені для пошуку інформації не адаптовані для запобігання зловмисних атак. Спамери застосовують деякі технології (наприклад кодування MIME) з метою навмисно приховувати зміст повідомлення під час передачі, завдяки цьому спам може обійти фільтрування на основі змісту. Іноді спамери використовують HTML-коди для того, щоб можна було замаскувати в тілі HTML-повідомлення обрані символи. Для рішення цієї проблеми метод Nilsimsa має припускати, що спамери можуть витратити ресурси на розробку технологій, які спроможні автоматично виробляти спам, які мають здатність обходити перевірки.

Нарешті, кодування, придатне для використання в анти-спам фільтрі не повинно піддавати користувачеві ризик випадкового видалення можливих повідомлень. Тому, звідки і впливає третя вимога [13].

Ідея методу Nilsimsa дуже схожа з ідеєю метода TLSH. Nilsimsa створює дайджест повідомлення для того, щоб дайджести схожих повідомлень були схожі між собою. Метод створює вікно з 5 символів, яке ковзає по тексту

повідомлення по одному символу за кожний раз. Коли символ потрапляє у вікно генеруються триграми. Далі геш-функція приймає на вході кожну триграму і повертає значення i від 0 до 255. Це значення є i -тим лічильником у масиві цілих чисел розміром 255, потім використовується накопичувач і значення збільшується на одиницю. Після текстового аналізу накопичувач представляє в i -й комірці кількість триграм, які були знайдені у тексті шляхом застосування геш-функції. Після того порівнюється відносна частота кожного контейнера із середньою частотою контейнера, що спостерігається для великої колекції повідомлень і значенням, яке представляє співвідношення, котре асоціюється с контейнером. Наступним кроком здійснюється розрахунок співвідношення кожного контейнеру, і якщо i -те співвідношення більше ніж медіана, тоді i -й біт дорівнює одиниці, в іншому випадку – 0. Далі в тексті результату повертається 32-байтовий код. На рисунку 1.8 [13] представлено принцип дії методу Nilsimsa.

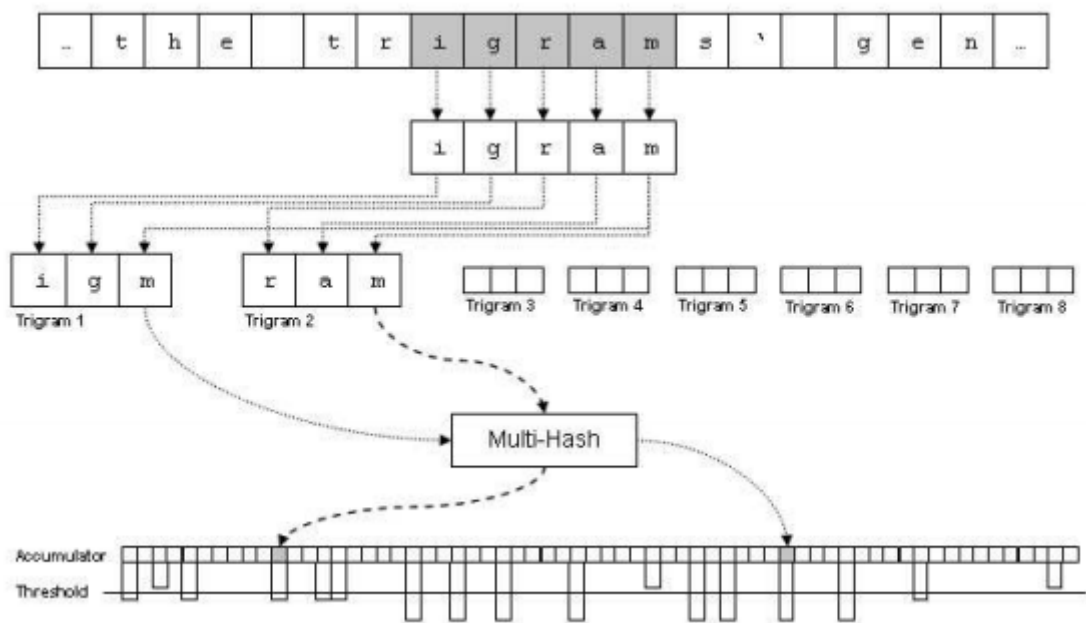


Рисунок 1.9 – Принцип дії методу Nilsimsa [13]

Для того, щоб визначити подібність між двома повідомленнями необхідно порівняти їх дайджести. Порівняння здійснюється шляхом перевірки

кількості бітів в одному положенні, що мають однакові значення в двох обраних 256-біт послідовностей. Значення подібності дорівнює кількості бітів, які ідентичні у двох дайджестах віднявши від цієї кількості 128. Отже метод Nilsimsa повертає значення від 128-256, це значення є оцінкою подібності e двох дайджестів. Чи ідентичними є дайджести або ні залежить від значення оцінки e :

- якщо $e = 128$, тоді дайджести абсолютно різні;
- якщо $e = 256$, тоді дайджести є ідентичними.

Метод Nilsimsa є схожим з методом TLSH, але Nilsimsa на відміну від TLSH не створює заголовки та тіла дайджестів. Також метод Nilsimsa видає значно більше помилково-позитивних результатів ніж метод TLSH, тому при виборі між двома методами більш доцільно обрати метод TLSH.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Архітектура програми

Після теоретичного аналізу та огляду методів локально-чуттєвого гешування можна спроектувати архітектуру програми. Завдяки грамотно спроектованій архітектурі можна значно підвищити ефективність, безпечність, легкість в експлуатації програми та її зручність. Тому при проектуванні архітектури програмного забезпечення не можна ігнорувати дані критерії.

Програма складатиметься з чотирьох пакетів, кожен з яких відповідає за певну частину функціоналу. Діаграма цих пакетів наведена на рисунку 2.1.

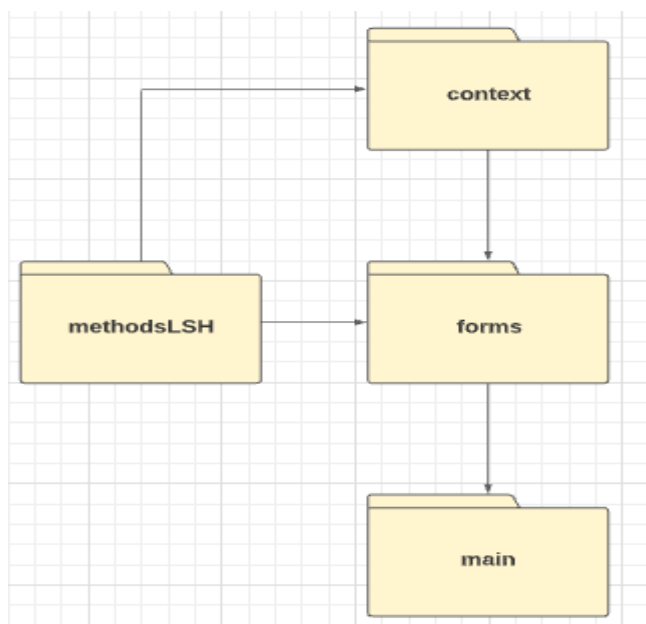


Рисунок 2.1 – Діаграма пакетів програмного забезпечення

Отже, програма містить такі пакети:

- methodsLSH;
- context;
- forms;
- main.

Пакет methodsLSH відповідає за програмну реалізацію методів локально-чуттєвого гешування. Він містить класи для реалізації цих методів.

Пакет context містить в собі один клас, який має посилання на один із класів пакету methodsLSH.

Пакет forms відповідає за графічний інтерфейс користувача. У цьому пакеті містяться два класи, які відповідають за відображення форм.

Останній пакет main містить клас, який запускає програму.

Так як було обрано об'єктно-орієнтований підхід для проектування архітектури програмного забезпечення, то можна визначити діаграму класів, яка наведена нижче на рисунку 2.2.

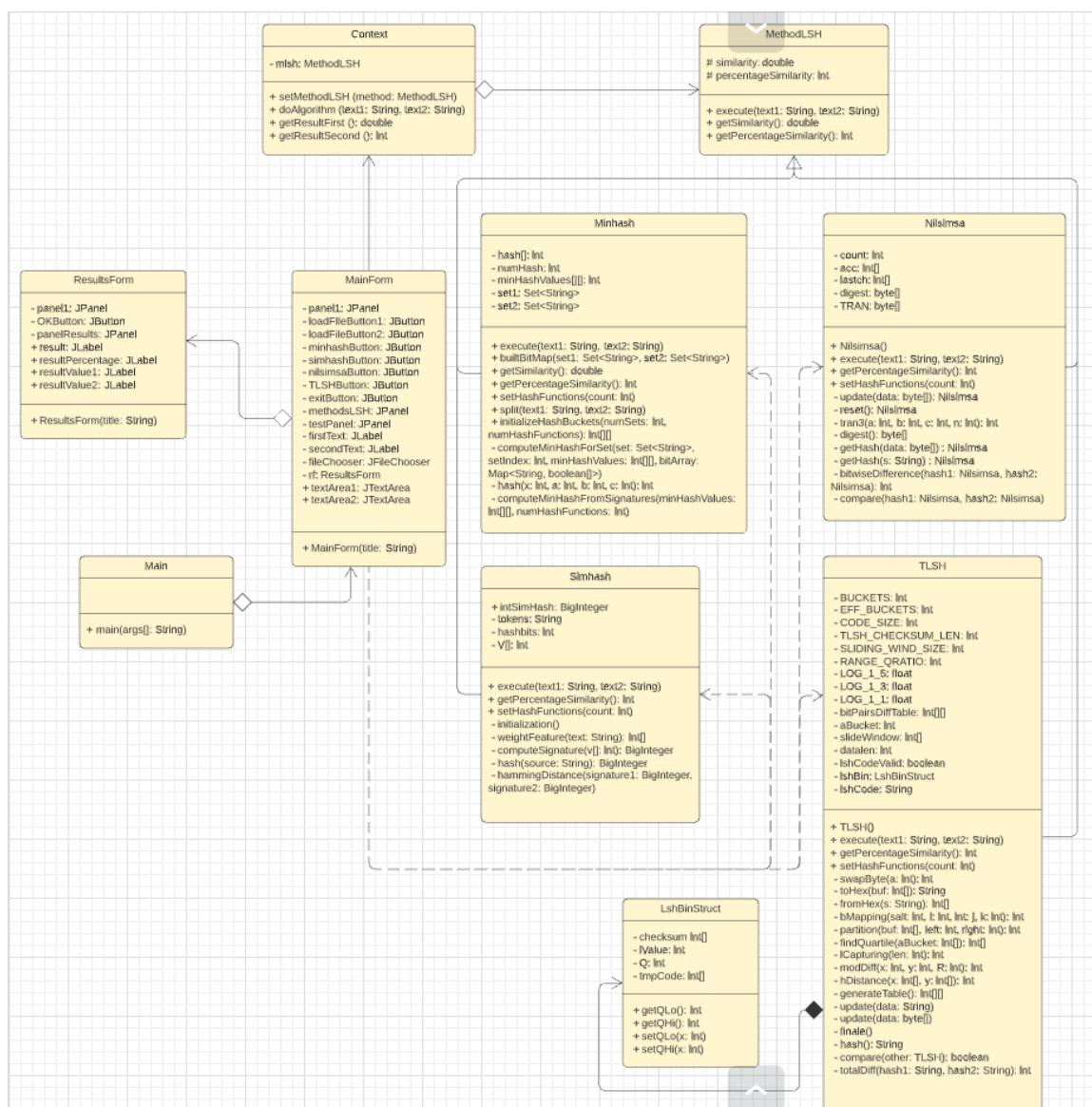


Рисунок 2.2 – Діаграма класів

MethodsLSH є базовим класом для класів Minhash, Simhash, Nilsimsa та TLSH. Цей клас задає загальний набір властивостей для класів-нащадків. Також клас TLSH має вкладений клас LshBinStruct. Усі ці класи містяться в пакеті methodsLSH.

Клас Context, який знаходиться в пакеті context має посилання на об'єкт MethodsLSH та використовується в класі MainForm.

Класи MainForm та ResultsForm містяться в пакеті forms. Клас MainForm відповідає за відображення головної форми програми, він містить посилання на об'єкт ResultsForm. Клас ResultsForm відповідальний за відображення форми результатів методів ЛЧГ.

Клас Main забезпечує запуск програми, він використовує клас MainForm для відображення головної форми програми через виклик конструктору цього класу.

Класи MainForm, Context, MethodLSH, Minhash, Simhash, Nilsimsa, TLSH реалізують паттерн «Стратегія». Паттерн «Стратегія» - це поведінковий шаблон проектування, який визначає сімейство подібних алгоритмів та розміщує кожен з них у власному класі [16]. Структура паттерну наведена нижче на рисунку 2.3 [16]:

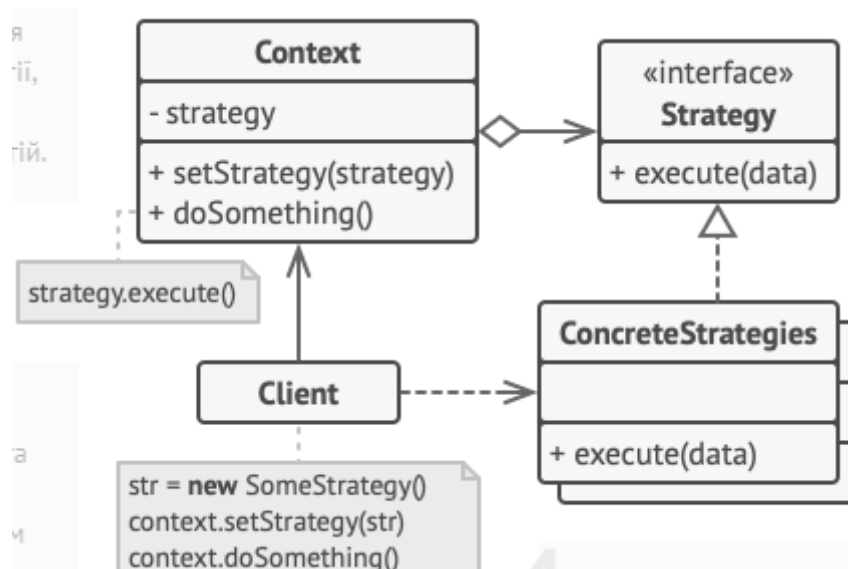


Рисунок 2.3 – Структура паттерну «Стратегія» [16]

В нашому випадку клас `MethodLSH` є стратегією. Він визначає інтерфейс, який є спільним для варіантів алгоритму.

Класи `Minhash`, `Simhash`, `Nilsimsa` та `TLSH` є конкретними стратегіями, ці класи реалізують різні варіанти алгоритму.

Клас `Context` містить посилання на об'єкт конкретної стратегії та працює з ним через загальний інтерфейс стратегій. `Context` використовує клас `MethodLSH` для виклику алгоритмів.

Клас `MainForm` є «клієнтом», в ньому створюється об'єкт конкретної стратегії та передає його до контексту. В класі `MainForm` мається можливість швидко замінити стратегію.

Паттерн «Стратегія» застосовується коли існує кілька варіантів одного алгоритму і їх необхідно використовувати в одному класі або якщо є багато класів, які відрізняються лише деякою поведінкою.

Основними перевагами цього паттерну є швидка заміна алгоритмів та ізоляція коду від інших класів.

2.2 Вибір мови програмування

Перш ніж приступити до розробки програмного забезпечення необхідно визначитися з вибором мови програмування. Існують багато мов програмування, які надають унікальні переваги, котрими можна скористатися, серед них це Java, C#, Python та Ruby.

Java – це високорівнева об'єктно-орієнтована мова програмування, яка підтримує сувору типізацію. Суворі типізація – це коли значення змінної або вираз мають конкретний тип і він суворо визначений. Програми, які написані на Java є кросплатформними, тобто вони можуть запускатися без проблем на інших платформах. Це досягається завдяки тому, що Java-програми транслюються в байт-код, який виконується віртуальною машиною Java (JVM). JVM обробляє цей байт-код і передає конструкції обладнанню як інтерпретатор. Також Java забезпечує автоматичне управління пам'яттю.

C# - як і Java є високорівневою об'єктно-орієнтованою мовою програмування. Вона теж підтримує сувору типізацію та автоматичне управління пам'яттю, але в C# не така підтримка кросплатформності як в Java, C# більш підходить для розробки додатків під Windows.

Python – високорівнева об'єктно-орієнтована мова програмування. Як і Java та C#, ця мова програмування забезпечує автоматичне управління пам'яттю, але на відміну від цих мов Python має динамічну типізацію, це означає, що змінні можуть приймати значення різних типів. Також мова програмування Python є кросплатформною.

Ruby – це повністю об'єктно-орієнтована мова програмування з динамічною типізацією. Він як і Java та Python теж є кросплатформною мовою програмування.

Нижче наведена порівняльна таблиця мов програмування:

Таблиця 2.1 – Порівняльна таблиця мов програмування

Мова Критерії	Java	C#	Python	Ruby
Об'єктно-орієнтованість	Так	Так	Так	Так
Кросплатформність	Так	Ні	Так	Так
Автомат. управління пам'яттю	Так	Так	Так	Так
Типізація	Сувора	Сувора	Динамічна	Динамічна
Швидкодія	Висока	Висока	Низька	Низька

Після порівнянь даних мов програмування було вирішено обрати мову Java для розробки програмного забезпечення. Java є кросплатформною мовою програмування, що забезпечує роботу програм на будь-якому пристрої. Java забезпечує автоматичне управління пам'яттю, що дозволяє не слідкувати за

розподілом пам'яті при розробці програмного забезпечення. Програми, які написані на Java є швидкими та безпечними за рахунок суворої типізації.

2.3 Вибір середовища розробки

Після вибору мови програмування необхідно визначитися з середовищем розробки. Для мови програмування Java існують три найпопулярніші інтегровані середовища розробки (IDE):

- NetBeans;
- Eclipse;
- IntelliJ IDEA.

NetBeans – це інтегроване середовище розробки, яка підтримується компанією Oracle. NetBeans є середовищем розробки також для інших мов програмування: Python, PHP, JavaScript, C, C++ та інших. Ця IDE підтримує рефакторінг, виділення синтаксичних конструкцій кольором мовних конструкцій та автодоповнення цих конструкцій. NetBeans підтримує плагіни, що дозволяє розширити можливості середовища.

Eclipse – це вільне інтегроване середовище розробки, ця IDE підтримується некомерційною організацією Eclipse Foundation. Середовище має велику кількість плагинів, що робить цю IDE універсальною та легко налагоджується.

IntelliJ IDEA – це інтегроване середовище розробки, яке було розроблено компанією JetBrains. IntelliJ IDEA надає широкий набір інтегрованих інструментів для рефакторінгу. Ця IDE постачається з такими функціями, як інтелектуальне завершення коду та його аналіз.

Середовищем розробки були вирішено обрати IntelliJ IDEA. На відміну від інших середовищ розробки IntelliJ IDEA надає більше функціональності та інструментів для розробки програмного забезпечення. Обрана IDE дозволяє сконцентруватися на функціональних завданнях завдяки тому, що вона бере на себе виконання рутинних операцій.

2.4 Вибір бібліотеки для графічного інтерфейсу програми

Існують дві бібліотеки графічного інтерфейсу для програм на мові Java. Це AWT та Swing.

AWT (Abstract Window Toolkit) є пакетом, який використовується для графічного інтерфейсу користувача. Цей пакет має базовий набір компонентів елементів управління, діалогів та вікон, що підтримують придатний та простий до використання, але обмежений у можливостях графічний інтерфейс.

Swing – це пакет, який відповідає за створення графічного інтерфейсу користувача. Swing було створено через значні обмеження в використанні AWT. Цей пакет дає можливість працювати з різними операційними системами (ОС) та платформами.

Бібліотеку Swing було вирішено вибрати для розробки графічного інтерфейсу користувача. Swing на відміну від AWT має багатий набір елементів управління та можливість налаштовувати зовнішній вигляд компонентів на різних платформах.

3 РОЗРОБКА ПРОГРАМИ

У цьому розділі буде детально описуватися класи та їх методи кожного пакету, що належать програмі. Також будуть описуватися графічні компоненти програми.

3.1 Опис елементів структури програми

3.1.1 Опис класу Context пакету context

Опис основних методів класу Context наведений в таблиці 3.1.

Таблиця 3.1 – Основні методи класу Context

Метод	Параметри та тип значення, що повертає	Опис
setMethodLSH()	MethodLSH method void	Задає метод локально-чуттєвого гешування.
doAlgorithm()	String text1 String text2 void	Приймає два рядки та виконує алгоритм заданого методу ЛЧГ.
getResultFirst()	double	Повертає оцінку подібності між рядками.
getResultSecond()	int	Повертає відсоткову подібність між рядками.

3.1.2 Опис класів пакету methodsLSH

Опис основних методів класів пакету methodsLSH наведений в таблиці 3.2.

Таблиця 3.2 – Опис основних методів класів пакету methodsLSH

Назва класу	Метод	Параметри та тип значення, що повертає	Опис
MethodLSH	execute()	String text1 String text2 void	Приймає на вході два рядки. Метод буде відповідальним за виконання методів ЛЧГ. Тіло методу є пустим тому, що він є абстрактним.
	getSimilarity()	double	Метод буде повертати оцінку подібності в класах-нащадках.
	getPercentageSimilarity()	double	Метод буде повертати відсоткове значення подібності в класах-нащадках.
Minhash	execute()	String text1 String text2 void	Приймає на вході два рядки. Метод буде відповідальний за виконання алгоритму методу Minhash.
	setHashFunctions()	int count void	Встановлює кількість геш-функцій.
	initializeHashBuckets()	String text1 String text2 int[][]	Метод приймає на вході два рядки та ініціалізує матрицю сигнатур.
	computeMinHashForSet()	Set<String> set int setIndex int[][] minHashValues Map<String, boolean[]> bitArray	Метод розраховує значення матриці сигнатур.
	hash()	int x int a int b int c	Метод відповідальний за генерацію геш-функцій.
	buildBitMap()	Set<String> set1 Set<String> set2	Метод будує характеристичну матрицю.
	computeMinHashFromSignatures()	int[][] minHashValues int numHashFunctions void	Метод розраховує коефіцієнт Жаккара.
	split()	String text1 String text2 void	Метод розбиває рядки на слова та розміщає ці слова в множинах.
	getSimilarity()	double	Метод повертає оцінку подібності, тобто коефіцієнт Жаккара.
	getPercentageSimilarity()	int	Метод повертає відсоткове значення подібності.

Продовження таблиці 3.2

Назва класу	Метод	Параметри та тип значення, що повертає	Опис
Simhash	execute()	String text1 String text2 void	Приймає на вході два рядки. Метод буде відповідальний за виконання алгоритму методу Simhash.
	initialiazation()	void	Ініціалізує кількість геш-бітів.
	weightFeature()	String text int[]	Приймає на вході текст та визначає значення масиву.
	compteSignature()	int[] v BigInteger	Методи приймає масив та визначає сигнатуру.
	hash()	String source BigInteger	Метод приймає слово із рядка та розраховує геш-значення для нього.
	hammingDistance()	BigInteger signature1 BigInteger signature2 int	Метод приймає дві сигнатури та розраховує відстань Геммінга між ними.
	getSimilarity()	double	Метод повертає оцінку подібності, тобто відстань Геммінга.
Nilsimsa	execute()	String text1 String text2 void	Приймає на вході два рядки. Метод буде відповідальний за виконання алгоритму методу Nilsimsa.
	digest()	byte[]	Метод повертає дайджест-повідомлення як масив байтів.
	getHash()	byte[] data Nilsimsa	Метод повертає геш-значення.
	getHash()	String s Nilsimsa	Метод приймає рядок та геш-значення.
	bitwiseDifference()	Nilsimsa hash1 Nilsimsa hash2 int	Метод повертає кількість бітів, які відрізняються між двома геш-значеннями.
	compare()	Nilsimsa hash1 Nilsimsa hash2 int	Метод розраховує оцінку подібності між гешами. Оцінка подібності розраховується як різниця числа 256 та кількістю бітів, які відрізняються між двома геш-значеннями.
	getSimilarity()	double	Метод повертає оцінку подібності.

Продовження таблиці 3.2

Назва класу	Метод	Параметри та тип значення, що повертає	Опис
	getPercentageSimilarity()	int	Метод повертає відсоткове значення подібності.
TLSH	execute()	String text1 String text2 void	Приймає на вході два рядки. Метод буде відповідальний за виконання алгоритму методу TLSH.
	update()	String data void	Метод обробляє рядок за допомогою розсувного вікна та заповнює масив підрахунків контейнерів за геш-функції Пірсона.
	finale()	void	Метод обчислює квартильні точки.
	hash()	String	Метод відповідає за побудову дайджесту.
	totalDiff()	String hash1 String hash2 int	Метод відстань між дайджестами.
	modDiff()	int x int y int R int	Метод оцінює відстані між різними частинами заголовків.
	bMapping()	int salt int i int j int k int	Геш-функція Пірсона.
	getSimilarity()	double	Метод повертає оцінку подібності.
	getPercentageSimilarity()	int	Метод повертає відсоткове значення подібності.

3.1.3 Опис класу Main пакету main

Клас Main відповідальний за запуск програмного забезпечення. В його методі main() запускається головна форма програми.

3.2 Опис графічних компонентів програми

За відображення графічних компонентів програми відповідають класи MainForm та ResultsForm, Ці класи відображають головну та дочірню форми відповідно. Детальний опис цих форм наводиться в пунктах 3.2.1 та 3.2.2.

3.2.1 Опис головної форми програми

Головна форма програми наведена нижче на рисунку 3.1.

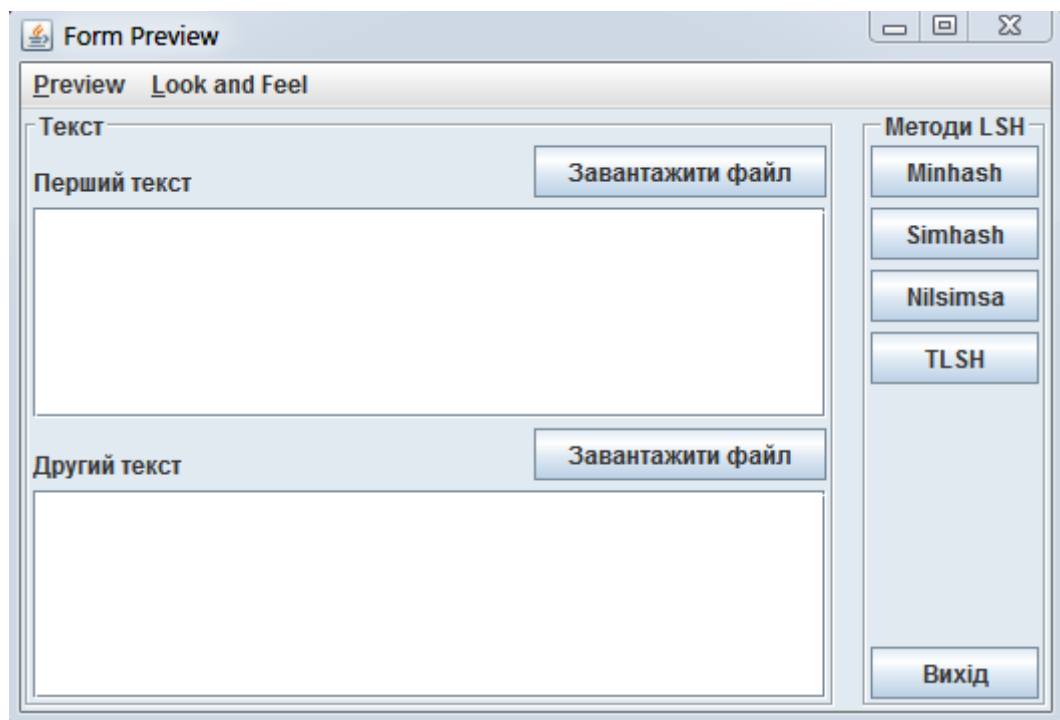


Рисунок 3.1 – Головна форма програми

Головна форма складається з таких елементів керування інтерфейсу:

- JFrame – цей елемент відображає вікно форми програми;
- JPanel – це елемент-контейнер, який зберігає в собі інші елементи керування;
- JButton – цей елемент керування є кнопкою, який при натисканні починає виконувати функції, що написані в програмі;

- JTextArea – це елемент керування графічного інтерфейсу, який є багаторядковим текстовим полем;
- JLabel – це текстова мітка.

Головна форма програми має три панелі, сім кнопок, два текстових областей та дві текстові мітки.

Кнопки «Завантажити файл» дозволяють обрати директорію та завантажити текстові файли, які будуть відображатися в текстових областях.

При натисканні кнопок «Minhash», «Simhash», «Nilsimsa» та «TLSH» відбувається оцінка подібності між двома текстами за допомогою обраних методів локально-чуттєвого гешування, після цього виводиться дочірня форма з результатами. Якщо один з текстових областей є пустим, тоді виводиться повідомлення, в якому повідомляється про це. На рисунку 3.2 наведено діалогове вікно, яке повідомляє про помилку.

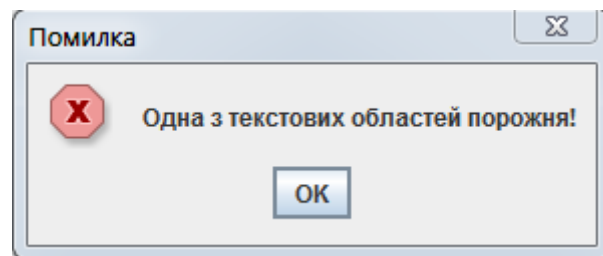


Рисунок 3.2 – Повідомлення про помилку

Кнопка «Вихід» відповідає за вихід з програми. При натисканні на неї головна форма закривається.

3.2.2 Опис дочірньої форми програми

Дочірня форма програми наведена на рисунку 3.3.

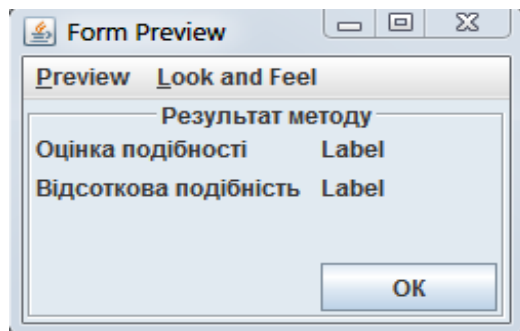


Рисунок 3.3 – Дочірня форма програми

Форма відображає результати вибраного методу локально-чуттєвого гешування. При натисканні на кнопку «ОК» дочірня форма закривається.

4 ТЕСТУВАННЯ ТА ЕКСПЛУАТАЦІЯ ПРОГРАМИ

В цьому розділі буде проводитися тестування програмного забезпечення, а також наводиться інструкція по експлуатації програмного забезпечення.

4.1 Тестування програми

Важливим етапом розробки програмного забезпечення є тестування. Тестування дозволяє зекономити багато часу, ресурсів, можливих втрат даних, перевірити програмне забезпечення на ефективність та доцільність. Існують різні методи тестування, серед них:

- інтеграційне тестування – цей метод тестування, який перевіряє взаємодію між програмними компонентами програми;
- модульне тестування – метод тестування, який перевіряє функціонування окремо взятого елемента системи;
- системне тестування – це метод тестування, який перевіряє функціональні та нефункціональні вимоги до програмного забезпечення;
- ручне тестування – це метод тестування, який проводиться тестувальником-оператором без використання програмних засобів. Метод застосовується для перевірки програмного забезпечення шляхом моделювання дій користувача.

Результати тестування наведені в таблиці 4.1.

Таблиця 4.1. – Результати тестування програми

№	Дія	Очікуваний результат	Результат
1	Запуск програми.	Виведення головної форми програми.	Вірно

Продовження таблиці 4.1.

№	Дія	Очікуваний результат	Результат
2	Завантаження текст. файлу для першого тексту.	Виведення завантаженого тексту на текстову область.	Вірно
3	Завантаження текст. файлу для другого тексту.	Виведення завантаженого тексту на текстову область.	Вірно
5	Натискання кнопки «Minhash».	Відображення дочірньої форми з результатами порівняння двох текстів методом Minhash.	Вірно
6	Натискання кнопки «Simhash».	Відображення дочірньої форми з результатами порівняння двох текстів методом Simhash.	Вірно
7	Натискання кнопки «Nilsimsa».	Відображення дочірньої форми з результатами порівняння двох текстів методом Nilsimsa.	Вірно
8	Натискання кнопки «TLSH».	Відображення дочірньої форми з результатами порівняння двох текстів методом TLSH.	Вірно
9	Натискання кнопки «ОК» дочірньої форми програми.	Закриття дочірньої форми програми.	Вірно
10	Натискання кнопки «Вихід» головної форми програми.	Закриття головної форми програми.	Вірно

В таблиці показано, що при усіх сценаріях роботи із програмою, завжди отримується позитивний результат, а отже програмне забезпечення готове до експлуатації.

4.2 Інструкція по експлуатації програми

Перш ніж застосовувати програму необхідно її запустити. Після запуску виведеться головне вікно програми, яке наведено на рисунку 3.1. Далі потрібно

завантажити два текстових файли або ввести тексти самому в текстових областях головного вікна програми, на рисунку 4.1 це показано.

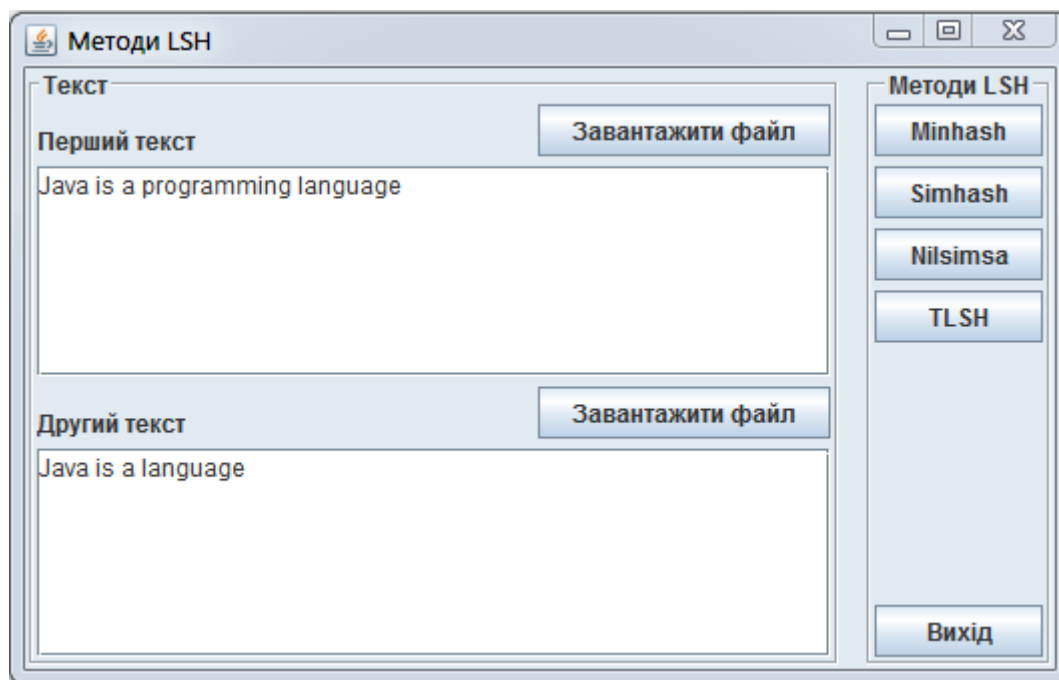


Рисунок 4.1 – Введені тексти в текстових областях головного вікна програми

Після цього можна оцінити подібність між текстами за допомогою одного з методів локально-чуттєвого гешування. Для цього необхідно натиснути відповідну кнопку і після цього буде відображатися дочірня форма з результатами. Нижче на рисунках 4.2 - 4.5 наведені результати методів ЛЧГ.

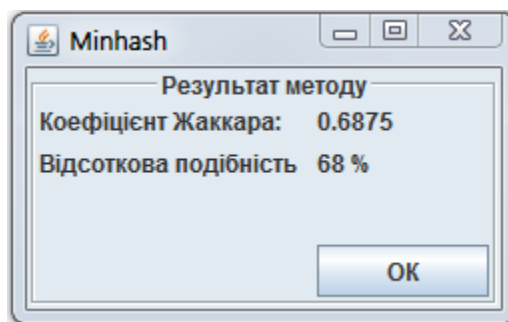


Рисунок 4.2 – Результати методу Minhash

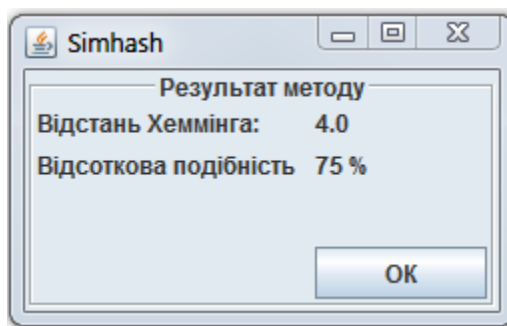


Рисунок 4.3 – Результати методу Simhash

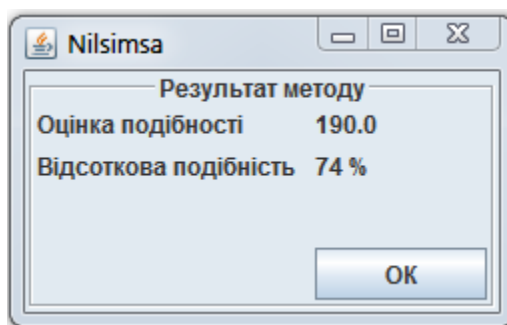


Рисунок 4.4 – Результати методу Nilsimsa

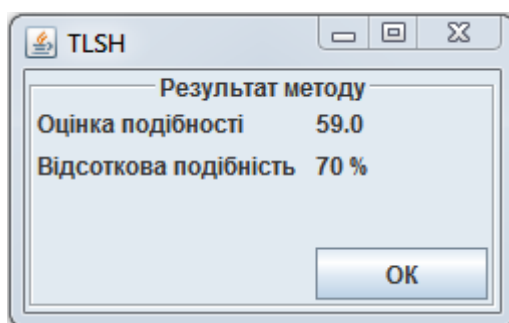


Рисунок 4.5 – Результати методу TLSH

Якщо необхідно закрити дочірню форму програми, тоді можна натиснути кнопку «ОК».

Для того, щоб завершити роботу програми потрібно натиснути кнопку «Вихід» і після цього закриються усі форми програмного забезпечення.

ВИСНОВКИ

В ході виконання дипломної роботи було розроблено програмне забезпечення, що реалізує методи локально-чуттєвого гешування.

Програмне забезпечення було протестовано, налагоджено та випробувано. Це показало коректність роботи програми.

В першому розділі були приведені загальні відомості про локально-чуттєве-гешування та огляд методів ЛЧГ.

В другому розділі була розроблена архітектура програмного забезпечення та наведено обґрунтування вибір мови програмування та інших технологій для розробки програмного забезпечення.

В третьому розділі були описані основні компоненти програми.

В четвертому розділі було проведено тестування програми та наведено інструкцію по її використанню.

Локально-чуттєве гешування є найактуальнішим та багатообіцяючим підходом для вирішення багатьох задач для пошуку подібності в багатовимірних просторах. Через свою швидкість та легкість у використанні його застосовують в багатьох сервісах та компаніях для боротьби зі спамом, створення рекомендаційних систем, ідентифікації схожості між об'єктами і т.п.

Локально-чуттєве гешування забезпечує локальну чутливість об'єкта при зміні його частини тому, що ЛЧГ намагається не змінювати абсолютно його геш-значення в такому випадку.

Методи, що відносяться до ЛЧГ на відміну від простих методів гешування намагаються максимізувати конфлікти гешування. Принцип дії цих методів полягає в тому, щоб поділити дані на частини та гешувати ці частини з наступним випадковим відбиранням елементів з отриманої колекції згідно певного правила чи ознаки.

Методи ЛЧГ відрізняються між собою ймовірністю геш-колізій, легкістю реалізації, точністю результатів, швидкістю алгоритму та кількістю

помилково-позитивних та помилково-негативних результатів. Тому при виборі застосування одного із методів слід орієнтуватися на ці показники.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Что такое хеширование [Электрон. ресурс]. – Режим доступа: <https://academy.binance.com/ru/articles/what-is-hashing>.
2. Gionis A. Similarity Search in High Dimensions via Hashing / A. Gionis, P. Indyk, R. Motwani // The 25th Very Large Database (VLDB) Conference, Stanford, September 1999 : proceedings. – P. 522.
3. Locality Sensitive Hashing (LSH) [Electronic resource]. – Access mode: <https://aerodatablog.wordpress.com/2017/11/29/locality-sensitive-hashing-lsh/>.
4. Білощицький А.О. Оптимізація системи пошуку збігів за допомогою використання алгоритмів локально-чутливого хешування наборів текстових даних / А. О. Білощицький, О. В. Діхтяренко // Управління розвитком складних систем. – 2014. – № 19. – С. 115–116.
5. Broder A.Z. On the resemblance and containment of documents / A. Z. Broder // Compression and Complexity of Sequences, June 11-13, 1997: proceedings. - Salerno: IEEE, 1997. – P. 21–29.
6. Broder A.Z. Min-wise independent permutations / [A. Z. Broder, M. Charikar, A. M. Frieze et al.] // 30th ACM Symposium on Theory of Computing, 1998: proceedings, - New York, 1998. – P. 327–336.
7. Lascovec J. Finding Similar Items / J. Lascovec, A. Rajaraman, J. Ullman. // Mining of Massive Datasets – Cambridge: Cambridge University Press, 2010. – Chapter 3 - 82 p.
8. MinHash - SteemKR [Electronic resource]. – Access mode: <https://steemkr.com/minhash/@verysimple/minhash>.
9. Charikar M.S. Similarity estimation techniques from rounding algorithms / M. S. Charikar // The thirty-fourth annual ACM symposium on Theory of computing, May, 2002: proceedings. - Montreal: ACM, 2002. – P. 380 – 388.
10. Відстань Хеммінга [Електрон. ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Відстань_Геммінга.

11. Коллизия хеш-функции [Электрон. ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Коллизия_хеш-функции.
12. Gurmeet S.M. Detecting near-duplicates for web-crawling / S. M. Gurmeet, A. Jain, A.D. Sarma // The 16th International Conference on World Wide Web, May 8-12, 2007: proceedings. – P. 141.
13. Damiani E. An Open Digest-based Technique for Spam Detection / [E. Damiani, S. De Capitani di Vercati, S. Paraboschi et al.] // The ISCA 17th International Conference on Parallel and Distributed Computing Systems, September 15-17, 2004 : proceedings. – San Francisco, 2004. – P. 559-562.
14. Oliver J. TLSH – A Locality Sensitive Hash / J. Oliver, C. Cheng, Y. Chen // The Fourth Cybercrime and Trustworthy Computing Workshop, November 21-22, 2013 : proceedings. – Sydney, 2013. – P. 7.
15. Smart Whitelisting Using Locality Sensitive Hashing [Electronic resource]. – Access mode: https://www.trendmicro.com/en_us/research/17/c/smart-whitelisting-using-locality-sensitive-hashing.html.
16. Паттерн Стратегія [Електрон. ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/strategy>.

ДОДАТОК А
Текст програми

```

package methodsLSH;

public abstract class MethodLSH {

    protected double similarity;
    protected int percentageSimilarity;

    public abstract void execute(String text1, String text2);
    public abstract double getSimilarity();
    public abstract int getPercentageSimilarity();

}

package methodsLSH;

import java.util.*;

final public class Minhash extends MethodLSH {

    private int hash[];
    private int numHash;
    private int[][] minHashValues;

    private Set<String> set1;
    private Set<String> set2;

    @Override
    public void execute(String text1, String text2){

        split(text1, text2);
        //setHashFunctions(set1.size() + set2.size());
        setHashFunctions(16);
        int numSets = 2;
        Map<String, boolean[]> bitMap = buildBitMap(set1, set2);

        minHashValues = initializeHashBuckets(numSets, numHash);

        computeMinHashForSet(set1, 0, minHashValues, bitMap);
        computeMinHashForSet(set2, 1, minHashValues, bitMap);

        computeMinHashFromSignatures(minHashValues, numHash);
    }
}

```

```

private void setHashFunctions(int count) {
    numHash = count;

    hash = new int[numHash];

    Random r = new Random(11);
    for (int i = 0; i < numHash; i++){
        int a = (int)r.nextInt();
        int b = (int)r.nextInt();
        int c = (int)r.nextInt();
        int x = hash(a*b*c, a, b, c);
        hash[i] = x;
    }
}

private void split(String text1, String text2) {

    String textFeatures1[] = text1.split(" ");
    String textFeatures2[] = text2.split(" ");

    set1 = new HashSet<String>();
    for(String feature : textFeatures1) {
        set1.add(feature);
    }

    set2 = new HashSet<String>();
    for(String feature : textFeatures2) {
        set2.add(feature);
    }
}

private static int[][] initializeHashBuckets(int numSets, int numHashFunctions) {
    int[][] minHashValues = new int[numSets][numHashFunctions];

    for (int i = 0; i < numSets; i++) {
        for (int j = 0; j < numHashFunctions; j++) {
            minHashValues[i][j] = Integer.MAX_VALUE;
        }
    }
    return minHashValues;
}

```

```
private void computeMinHashForSet(Set<String> set, int setIndex, int[][]
minHashValues, Map<String, boolean[]> bitArray){
```

```
    int index=0;
```

```
    for(String element : bitArray.keySet()) { // for every element in the bit array
        for (int i = 0; i < numHash; i++){ // for every hash
            if(set.contains(element)) { // if the set contains the element
                int hindex = hash[index]; // get the hash
                if (hindex < minHashValues[setIndex][index]) {
                    // if current hash is smaller than the existing hash in the slot then
                    replace with the smaller hash value
                    minHashValues[setIndex][i] = hindex;
                }
            }
        }
        index++;
    }
}
```

```
private static int hash(int x, int a, int b, int c) {
    int hashValue = (int)((a * (x >> 4) + b * x + c) & 131071);
    return Math.abs(hashValue);
}
```

```
public Map<String, boolean[]> buildBitMap(Set<String> set1, Set<String> set2){
```

```
    Map<String, boolean[]> bitArray = new HashMap<String,boolean[]>();
```

```
    for(String t : set1){
        bitArray.put(t, new boolean[]{true,false});
    }
```

```
    for(String t : set2){
        if(bitArray.containsKey(t)){
            // item is not present in set1
            bitArray.put(t, new boolean[]{true,true});
        }else if(!bitArray.containsKey(t)){
            // item is not present in set1
            bitArray.put(t, new boolean[]{false,true});
        }
    }
}
```

```

        return bitArray;
    }

    private void computeMinHashFromSignatures(int[][] minHashValues, int
numHashFunctions) {

        int identicalMinHashes = 0;
        for (int i = 0; i < numHashFunctions; i++){
            if (minHashValues[0][i] == minHashValues[1][i]) {
                identicalMinHashes++;
            }
        }
        similarity = (1.0 * identicalMinHashes) / numHashFunctions;
    }

    @Override
    public double getSimilarity() {
        return similarity;
    }

    @Override
    public int getPercentageSimilarity() {
        percentageSimilarity = (int) (similarity * 100);
        return percentageSimilarity;
    }
}

package methodsLSH;

import java.math.BigInteger;
import java.util.StringTokenizer;

final public class Simhash extends MethodLSH {

    private String tokens;
    public BigInteger intSimHash;
    private int hashbits;
    private int[] V;

    @Override
    public void execute(String text1, String text2) {

        initialization();

```

```

int[] v1 = weightFeature(text1);
int[] v2 = weightFeature(text2);

BigInteger signature1 = computeSignature(v1);
BigInteger signature2 = computeSignature(v2);

similarity = hammingDistance(signature1, signature2);

}

private void initialization() {

    hashbits = 16;

}

private int[] weightFeature(String text) {
    // Define eigenvectors / arrays
    // 1. After removing the format of the text, word segmentation
    V = new int[hashbits];
    StringTokenizer stringTokens = new StringTokenizer(text);
    while (stringTokens.hasMoreTokens()) {
        String temp = stringTokens.nextToken();
        // 2. hash each participle into a set of fixed length sequence, such as a 64 bit
integer
        BigInteger t = this.hash(temp);
        for (int i = 0; i < this.hashbits; i++) {
            BigInteger bitmask = new BigInteger("1").shiftLeft(i);
            // 3. Create a 64 bit integer array (assuming that you want to generate a 64
bit digital fingerprint, or other numbers),
            // Judge the sequence after each participle hash, if it is 1000... 1, then add 1
to the first and last bit of the array,
            // In other words, every one plus one, every zero minus one. Until all the
participle hash sequences are judged
            if (t.and(bitmask).signum() != 0) {
                // Here is the vector sum of all the features of the whole document
                // In practice, we need + - weight instead of simple + 1 / - 1,
                V[i] += 1;
            } else {
                V[i] -= 1;
            }
        }
    }
}

```

```

    return V;
}

private BigInteger computeSignature(int[] v) {

    BigInteger fingerprint = new BigInteger("0");
    for (int i = 0; i < this.hashbits; i++) {
        // 4. Finally, the array is judged. If it is greater than 0, it is 1. If it is less than
or equal to 0, it is 0. A 64 bit digital fingerprint / signature is obtained
        if (v[i] >= 0) {
            fingerprint = fingerprint.add(new BigInteger("1").shiftLeft(i));
        }
    }
    return fingerprint;
}

private BigInteger hash(String source) {
    if (source == null || source.length() == 0) {
        return new BigInteger("0");
    } else {
        char[] sourceArray = source.toCharArray();
        BigInteger x = BigInteger.valueOf(((long) sourceArray[0]) << 7);
        BigInteger m = new BigInteger("1000003");
        BigInteger mask = new BigInteger("2").pow(this.hashbits).subtract(new
BigInteger("1"));
        for (char item : sourceArray) {
            BigInteger temp = BigInteger.valueOf((long) item);
            x = x.multiply(m).xor(temp).and(mask);
        }
        x = x.xor(new BigInteger(String.valueOf(source.length())));
        if (x.equals(new BigInteger("-1"))) {
            x = new BigInteger("-2");
        }
        return x;
    }
}

private int hammingDistance(BigInteger signature1, BigInteger signature2) {

    BigInteger x = signature1.xor(signature2);
    int tot = 0;

    // Count the number of binary digit 1 in x

```

// Let's think about it. If we subtract 1 from a binary number, then all the numbers after the last 1 (including that 1) are reversed, right? Then, $n \& 1$ is equivalent to clearing 0,

// Let's see how many times n can do this.

```

while (x.signum() != 0) {
    tot += 1;
    x = x.and(x.subtract(new BigInteger("1")));
}
return tot;
}

@Override
public double getSimilarity() {
    return similarity;
}

@Override
public int getPercentageSimilarity() {
    return (int) ((1 - similarity / hashbits) * 100);
}
}

package methodsLSH;

import java.util.*;

final public class Nilsimsa extends MethodLSH {

    private int count = 0; // num characters seen
    private int[] acc = new int[256]; // accumulators for computing the digest
    private int[] lastch = new int[4]; // the last four seen characters
    private byte[] digest = null; // the Nilsimsa digest

    // pre-defined transformation array
    private static final byte[] TRAN = {(byte) 0x02, (byte) 0xD6, (byte) 0x9E, (byte)
0x6F, (byte) 0xF9, (byte) 0x1D,
        (byte) 0x04, (byte) 0xAB, (byte) 0xD0, (byte) 0x22, (byte) 0x16, (byte) 0x1F,
        (byte) 0xD8, (byte) 0x73,
        (byte) 0xA1, (byte) 0xAC, (byte) 0x3B, (byte) 0x70, (byte) 0x62, (byte) 0x96,
        (byte) 0x1E, (byte) 0x6E,
        (byte) 0x8F, (byte) 0x39, (byte) 0x9D, (byte) 0x05, (byte) 0x14, (byte) 0x4A,
        (byte) 0xA6, (byte) 0xBE,

```

(byte) 0xAE, (byte) 0x0E, (byte) 0xCF, (byte) 0xB9, (byte) 0x9C, (byte) 0x9A, (byte) 0xC7, (byte) 0x68,
 (byte) 0x13, (byte) 0xE1, (byte) 0x2D, (byte) 0xA4, (byte) 0xEB, (byte) 0x51, (byte) 0x8D, (byte) 0x64,
 (byte) 0x6B, (byte) 0x50, (byte) 0x23, (byte) 0x80, (byte) 0x03, (byte) 0x41, (byte) 0xEC, (byte) 0xBB,
 (byte) 0x71, (byte) 0xCC, (byte) 0x7A, (byte) 0x86, (byte) 0x7F, (byte) 0x98, (byte) 0xF2, (byte) 0x36,
 (byte) 0x5E, (byte) 0xEE, (byte) 0x8E, (byte) 0xCE, (byte) 0x4F, (byte) 0xB8, (byte) 0x32, (byte) 0xB6,
 (byte) 0x5F, (byte) 0x59, (byte) 0xDC, (byte) 0x1B, (byte) 0x31, (byte) 0x4C, (byte) 0x7B, (byte) 0xF0,
 (byte) 0x63, (byte) 0x01, (byte) 0x6C, (byte) 0xBA, (byte) 0x07, (byte) 0xE8, (byte) 0x12, (byte) 0x77,
 (byte) 0x49, (byte) 0x3C, (byte) 0xDA, (byte) 0x46, (byte) 0xFE, (byte) 0x2F, (byte) 0x79, (byte) 0x1C,
 (byte) 0x9B, (byte) 0x30, (byte) 0xE3, (byte) 0x00, (byte) 0x06, (byte) 0x7E, (byte) 0x2E, (byte) 0x0F,
 (byte) 0x38, (byte) 0x33, (byte) 0x21, (byte) 0xAD, (byte) 0xA5, (byte) 0x54, (byte) 0xCA, (byte) 0xA7,
 (byte) 0x29, (byte) 0xFC, (byte) 0x5A, (byte) 0x47, (byte) 0x69, (byte) 0x7D, (byte) 0xC5, (byte) 0x95,
 (byte) 0xB5, (byte) 0xF4, (byte) 0x0B, (byte) 0x90, (byte) 0xA3, (byte) 0x81, (byte) 0x6D, (byte) 0x25,
 (byte) 0x55, (byte) 0x35, (byte) 0xF5, (byte) 0x75, (byte) 0x74, (byte) 0x0A, (byte) 0x26, (byte) 0xBF,
 (byte) 0x19, (byte) 0x5C, (byte) 0x1A, (byte) 0xC6, (byte) 0xFF, (byte) 0x99, (byte) 0x5D, (byte) 0x84,
 (byte) 0xAA, (byte) 0x66, (byte) 0x3E, (byte) 0xAF, (byte) 0x78, (byte) 0xB3, (byte) 0x20, (byte) 0x43,
 (byte) 0xC1, (byte) 0xED, (byte) 0x24, (byte) 0xEA, (byte) 0xE6, (byte) 0x3F, (byte) 0x18, (byte) 0xF3,
 (byte) 0xA0, (byte) 0x42, (byte) 0x57, (byte) 0x08, (byte) 0x53, (byte) 0x60, (byte) 0xC3, (byte) 0xC0,
 (byte) 0x83, (byte) 0x40, (byte) 0x82, (byte) 0xD7, (byte) 0x09, (byte) 0xBD, (byte) 0x44, (byte) 0x2A,
 (byte) 0x67, (byte) 0xA8, (byte) 0x93, (byte) 0xE0, (byte) 0xC2, (byte) 0x56, (byte) 0x9F, (byte) 0xD9,
 (byte) 0xDD, (byte) 0x85, (byte) 0x15, (byte) 0xB4, (byte) 0x8A, (byte) 0x27, (byte) 0x28, (byte) 0x92,
 (byte) 0x76, (byte) 0xDE, (byte) 0xEF, (byte) 0xF8, (byte) 0xB2, (byte) 0xB7, (byte) 0xC9, (byte) 0x3D,
 (byte) 0x45, (byte) 0x94, (byte) 0x4B, (byte) 0x11, (byte) 0x0D, (byte) 0x65, (byte) 0xD5, (byte) 0x34,

```

        (byte) 0x8B, (byte) 0x91, (byte) 0x0C, (byte) 0xFA, (byte) 0x87, (byte) 0xE9,
        (byte) 0x7C, (byte) 0x5B,
        (byte) 0xB1, (byte) 0x4D, (byte) 0xE5, (byte) 0xD4, (byte) 0xCB, (byte)
        0x10, (byte) 0xA2, (byte) 0x17,
        (byte) 0x89, (byte) 0xBC, (byte) 0xDB, (byte) 0xB0, (byte) 0xE2, (byte)
        0x97, (byte) 0x88, (byte) 0x52,
        (byte) 0xF7, (byte) 0x48, (byte) 0xD3, (byte) 0x61, (byte) 0x2C, (byte) 0x3A,
        (byte) 0x2B, (byte) 0xD1,
        (byte) 0x8C, (byte) 0xFB, (byte) 0xF1, (byte) 0xCD, (byte) 0xE4, (byte)
        0x6A, (byte) 0xE7, (byte) 0xA9,
        (byte) 0xFD, (byte) 0xC4, (byte) 0x37, (byte) 0xC8, (byte) 0xD2, (byte)
        0xF6, (byte) 0xDF, (byte) 0x58,
        (byte) 0x72, (byte) 0x4E};

```

```

public Nilsimsa() {
    reset();
}

```

```

@Override
public void execute(String text1, String text2) {

    Nilsimsa firstValue = Nilsimsa.getHash(text1);
    Nilsimsa secondValue = Nilsimsa.getHash(text2);

    similarity = Nilsimsa.compare(firstValue, secondValue);

}

```

```

private Nilsimsa update(byte[] data) {
    for (int ch : data) {
        ch = ch & 0xff;
        count++;

        // incr accumulators for triplets
        if (lastch[1] > -1) {
            acc[tran3(ch, lastch[0], lastch[1], 0)]++;
        }
        if (lastch[2] > -1) {
            acc[tran3(ch, lastch[0], lastch[2], 1)]++;
            acc[tran3(ch, lastch[1], lastch[2], 2)]++;
        }
        if (lastch[3] > -1) {
            acc[tran3(ch, lastch[0], lastch[3], 3)]++;
        }
    }
}

```

```

        acc[tran3(ch, lastch[1], lastch[3], 4)]++;
        acc[tran3(ch, lastch[2], lastch[3], 5)]++;
        acc[tran3(lastch[3], lastch[0], ch, 6)]++;
        acc[tran3(lastch[3], lastch[2], ch, 7)]++;
    }

    // adjust lastch
    for (byte i = 3; i > 0; i--) {
        lastch[i] = lastch[i - 1];
    }
    lastch[0] = ch;
}
digest = null;
return this;
}

private Nilsimsa reset() {
    count = 0;
    Arrays.fill(acc, (byte) 0);
    Arrays.fill(lastch, -1);
    this.digest = null;
    return this;
}

private static int tran3(int a, int b, int c, int n) {
    int i = (c) ^ TRAN[n];
    return (((TRAN[(a + n) & 255] ^ TRAN[b & 0xff] * (n + n + 1)) + TRAN[i &
0xff]) & 255);
}

private byte[] digest() {
    if (digest != null) {
        return digest;
    }
    int total = 0;
    digest = new byte[32];
    Arrays.fill(digest, (byte) 0);

    if (count == 3) {
        total = 1;
    } else if (count == 4) {
        total = 4;
    } else if (count > 4) {
        total = 8 * count - 28;
    }
}

```

```

    }

    final int threshold = total / 256;
    for (int i = 0; i < 256; i++) {
        if (acc[i] > threshold) {
            digest[31 - (i >> 3)] += 1 << (i & 7);
        }
    }
    return digest;
}

private static Nilsimsa getHash(byte[] data) {
    return new Nilsimsa().update(data);
}

private static Nilsimsa getHash(String s) {
    return getHash(s.getBytes());
}

private static int bitwiseDifference(Nilsimsa hash1, Nilsimsa hash2) {
    int distance = 0;
    int h1;
    int h2;

    byte[] n1 = hash1.digest();
    byte[] n2 = hash2.digest();

    for (int i = 0; i < 32; i += 4) {
        h1 = (n1[i] & 0xFF) | (n1[i + 1] & 0xFF) << 8 | (n1[i + 2] & 0xFF) << 16 |
        (n1[i + 3] & 0xFF) << 24;
        h2 = (n2[i] & 0xFF) | (n2[i + 1] & 0xFF) << 8 | (n2[i + 2] & 0xFF) << 16 |
        (n2[i + 3] & 0xFF) << 24;
        distance += Integer.bitCount(h1 ^ h2);
    }
    return distance;
}

private static int compare(Nilsimsa hash1, Nilsimsa hash2) {
    return 256 - bitwiseDifference(hash1, hash2);
}

@Override
public double getSimilarity() {
    return similarity;
}

```

```

    }

    @Override
    public int getPercentageSimilarity() {
        return (int) ((similarity / 256) * 100);
    }
}

package methodsLSH;

import java.util.Arrays;

final public class TLSH extends MethodLSH {

    static final private int  BUCKETS          = 256;
    static final private int  EFF_BUCKETS      = 128;
    static final private int  CODE_SIZE        = 32;
    static final private int  TLSH_CHECKSUM_LEN = 1;
    static final private int  SLIDING_WND_SIZE = 5;
    static final private int  RANGE_QRATIO     = 16;
    static final private float LOG_1_5 = 0.4054651f;
    static final private float LOG_1_3 = 0.26236426f;
    static final private float LOG_1_1 = 0.095310180f;

    static final private int [] vTable = {
        (int)1, (int)87, (int)49, (int)12, (int)176, (int)178, (int)102, (int)166, (int)121,
        (int)193, (int)6, (int)84, (int)249, (int)230, (int)44, (int)163,
        (int)14, (int)197, (int)213, (int)181, (int)161, (int)85, (int)218, (int)80, (int)64,
        (int)239, (int)24, (int)226, (int)236, (int)142, (int)38, (int)200,
        (int)110, (int)177, (int)104, (int)103, (int)141, (int)253, (int)255, (int)50,
        (int)77, (int)101, (int)81, (int)18, (int)45, (int)96, (int)31, (int)222,
        (int)25, (int)107, (int)190, (int)70, (int)86, (int)237, (int)240, (int)34, (int)72,
        (int)242, (int)20, (int)214, (int)244, (int)227, (int)149, (int)235,
        (int)97, (int)234, (int)57, (int)22, (int)60, (int)250, (int)82, (int)175, (int)208,
        (int)5, (int)127, (int)199, (int)111, (int)62, (int)135, (int)248,
        (int)174, (int)169, (int)211, (int)58, (int)66, (int)154, (int)106, (int)195,
        (int)245, (int)171, (int)17, (int)187, (int)182, (int)179, (int)0, (int)243,
        (int)132, (int)56, (int)148, (int)75, (int)128, (int)133, (int)158, (int)100,
        (int)130, (int)126, (int)91, (int)13, (int)153, (int)246, (int)216, (int)219,
        (int)119, (int)68, (int)223, (int)78, (int)83, (int)88, (int)201, (int)99, (int)122,
        (int)11, (int)92, (int)32, (int)136, (int)114, (int)52, (int)10,
        (int)138, (int)30, (int)48, (int)183, (int)156, (int)35, (int)61, (int)26, (int)143,
        (int)74, (int)251, (int)94, (int)129, (int)162, (int)63, (int)152,

```

```

        (int)170, (int)7, (int)115, (int)167, (int)241, (int)206, (int)3, (int)150, (int)55,
        (int)59, (int)151, (int)220, (int)90, (int)53, (int)23, (int)131,
        (int)125, (int)173, (int)15, (int)238, (int)79, (int)95, (int)89, (int)16, (int)105,
        (int)137, (int)225, (int)224, (int)217, (int)160, (int)37, (int)123,
        (int)118, (int)73, (int)2, (int)157, (int)46, (int)116, (int)9, (int)145, (int)134,
        (int)228, (int)207, (int)212, (int)202, (int)215, (int)69, (int)229,
        (int)27, (int)188, (int)67, (int)124, (int)168, (int)252, (int)42, (int)4, (int)29,
        (int)108, (int)21, (int)247, (int)19, (int)205, (int)39, (int)203,
        (int)233, (int)40, (int)186, (int)147, (int)198, (int)192, (int)155, (int)33,
        (int)164, (int)191, (int)98, (int)204, (int)165, (int)180, (int)117, (int)76,
        (int)140, (int)36, (int)210, (int)172, (int)41, (int)54, (int)159, (int)8, (int)185,
        (int)232, (int)113, (int)196, (int)231, (int)47, (int)146, (int)120,
        (int)51, (int)65, (int)28, (int)144, (int)254, (int)221, (int)93, (int)189, (int)194,
        (int)139, (int)112, (int)43, (int)71, (int)109, (int)184, (int)209
    };

```

```
static private int [][] bitPairsDiffTable;
```

```
@Override
```

```
public void execute(String text1, String text2) {
```

```
    TLSH ti1 = new TLSH();
```

```
    TLSH ti2 = new TLSH();
```

```
    while (text1.length() < 1023) {
```

```
        text1 += (char)('A' + text1.length() % 26);
```

```
    }
```

```
    text1 += (char)0;
```

```
    ti2.update(text1);
```

```
    ti2.finale();
```

```
    while (text2.length() < 1023) {
```

```
        text2 += (char)('A' + text2.length() % 26);
```

```
    }
```

```
    text2 += (char)0;
```

```
    ti1.update(text2);
```

```
    ti1.finale();
```

```
    similarity = TLSH.totalDiff(ti1.hash(), ti2.hash());
```

```
}
```

```
private class LshBinStruct {
```

```
    public int [] checksum = new int [TLSH_CHECKSUM_LEN];
```

```

public int    IValue = 0;
public int    Q = 0;
public int [] tmpCode = new int [CODE_SIZE];

public int getQLo() {
    return (int)(Q & 0x0F);
}

public int getQHi() {
    return (int)((Q & 0xF0) >> 4);
}

public void setQLo(int x) {
    Q = (int)((Q & 0xF0) | (x & 0x0F));
}

public void setQHi(int x) {
    Q = (int)((Q & 0x0F) | ((x & 0x0F) << 4));
}
}

private int []      aBucket = null;
final private int [] slideWindow = new int [SLIDING_WND_SIZE];
private int         dataLen = 0;
private boolean     lshCodeValid = false;
LshBinStruct lshBin = new LshBinStruct();
private String      lshCode = null;

private int swapByte(int a) {
    int ret = (int) (((a & 0xF0) >> 4) & 0x0F);
    ret |= ((a & 0x0F) << 4) & 0xF0;
    return ret;
}

private String toHex(int [] buf) {
    String ret = "";
    for (int i = 0; i < buf.length; i++) {
        if (buf[i] < 16)
            ret += "0";
        ret += Integer.toHexString(buf[i]);
    }
    return ret.toUpperCase();
}
}

```

```

static private int [] fromHex(String s) {
    int [] ret = new int [s.length() / 2];
    for (int i = 0; i < s.length(); i += 2) {
        ret[i / 2] = Integer.parseInt(s.substring(i, i + 2), 16);
    }
    return ret;
}

```

```

private int bMapping(int salt, int i, int j, int k) {
    int h = 0;
    h = vTable[h ^ salt];
    h = vTable[h ^ i];
    h = vTable[h ^ j];
    h = vTable[h ^ k];
    return h;
}

```

```

private int partition(int [] buf, int left, int right) {
    if (left == right) {
        return left;
    }
    if (left + 1 == right) {
        if (buf[left] > buf[right]) {
            int tmp = buf[left];
            buf[left] = buf[right];
            buf[right] = tmp;
        }
        return left;
    }
}

```

```

int ret = left, pivot = (left + right) >> 1;

```

```

int val = buf[pivot];

```

```

buf[pivot] = buf[right];
buf[right] = val;

```

```

for (int i = left; i < right; i++) {
    if (buf[i] < val) {
        int tmp = buf[ret];
        buf[ret] = buf[i];
        buf[i] = tmp;
        ret++;
    }
}

```

```

    }
    buf[right] = buf[ret];
    buf[ret] = val;

    return ret;
}

private int [] findQuartile(int [] aBucket) {
    int [] bucketCopy = new int [EFF_BUCKETS];
    int [] shortCutLeft = new int [EFF_BUCKETS];
    int [] shortCutRight = new int [EFF_BUCKETS];
    int spl = 0, spr = 0;
    int q1 = 0, q2 = 0, q3 = 0;

    int p1 = EFF_BUCKETS / 4 - 1;
    int p2 = EFF_BUCKETS / 2 - 1;
    int p3 = EFF_BUCKETS - EFF_BUCKETS / 4 - 1;
    int end = EFF_BUCKETS - 1;

    System.arraycopy(aBucket, 0, bucketCopy, 0, EFF_BUCKETS);

    for (int l = 0, r = end; ;) {
        int ret = partition(bucketCopy, l, r);
        if (ret > p2) {
            r = ret - 1;
            shortCutRight[spr] = ret;
            spr++;
        } else if (ret < p2) {
            l = ret + 1;
            shortCutLeft[spl] = ret;
            spl++;
        } else {
            q2 = bucketCopy[p2];
            break;
        }
    }

    shortCutLeft[spl] = p2 - 1;
    shortCutRight[spr] = p2 + 1;

    for (int i = 0, l = 0; i <= spl; i++) {
        int r = shortCutLeft[i];
        if (r > p1) {
            while (true) {

```

```

        int ret = partition(bucketCopy, l, r);
        if (ret > p1) {
            r = ret - 1;
        } else if (ret < p1) {
            l = ret + 1;
        } else {
            q1 = bucketCopy[p1];
            break;
        }
    }
    break;
} else if (r < p1) {
    l = r;
} else {
    q1 = bucketCopy[p1];
    break;
}
}

for (int i = 0, r = end; i <= spr; i++) {
    int l = shortCutRight[i];
    if (l < p3) {
        while (true) {
            int ret = partition(bucketCopy, l, r);
            if (ret > p3) {
                r = ret - 1;
            } else if (ret < p3) {
                l = ret + 1;
            } else {
                q3 = bucketCopy[p3];
                break;
            }
        }
        break;
    } else if (l > p3) {
        r = l;
    } else {
        q3 = bucketCopy[p3];
        break;
    }
}
return new int [] {q1, q2, q3};
}

```

```

private int lCapturing(int len) {
    int i;
    if (len <= 656) {
        i = (int)Math.floor(Math.log((float) len) / LOG_1_5);
    } else if( len <= 3199 ) {
        i = (int)Math.floor(Math.log((float) len) / LOG_1_3 - 8.72777);
    } else {
        i = (int)Math.floor(Math.log((float) len) / LOG_1_1 - 62.5472 );
    }

    return (int) (i & 0xFF);
}

static private int modDiff(int x, int y, int R){
    int dl;
    int dr;
    if (y > x){
        dl = (y - x);
        dr = (x + R - y);
    } else {
        dl = (x - y);
        dr = (y + R - x);
    }
    if (dl > dr)
        return dr;
    return dl;
}

static private int hDistance(int [] x, int [] y)
{
    int diff = 0;
    for (int i = 0; i < x.length; i++) {
        diff += bitPairsDiffTable[x[i]][y[i]];
    }
    return diff;
}

public static int[][] generateTable(){
    int[][] result = new int [256][256];
    for (int i = 0; i < 256; i++) {
        for (int j = 0; j < 256; j++) {
            int x = i, y = j, d, diff = 0;
            d = Math.abs(x % 4 - y % 4); diff += (d == 3 ? 6 : d);
            x /= 4; y /= 4;
        }
    }
}

```

```

        d = Math.abs(x % 4 - y % 4); diff += (d == 3 ? 6 : d);
        x /= 4; y /= 4;
        d = Math.abs(x % 4 - y % 4); diff += (d == 3 ? 6 : d);
        x /= 4; y /= 4;
        d = Math.abs(x % 4 - y % 4); diff += (d == 3 ? 6 : d);
        result[i][j] = diff;
    }
}
return result;
}

public TLSH() {
    bitPairsDiffTable = generateTable();
}

public TLSH(final int[][] assignedBitPairsDiffTable) {
    bitPairsDiffTable = assignedBitPairsDiffTable;
}

public void update(final String data) {
    int j = this.dataLen % SLIDING_WND_SIZE;
    int fedLen = this.dataLen;
    if (this.aBucket == null) {
        this.aBucket = new int [BUCKETS];
    }
    for (int i = 0; i < data.length(); i++) {
        slideWindow[j] = (int)(data.charAt(i) & 0xff);
        if (fedLen >= 4) {
            int j1 = (j + SLIDING_WND_SIZE - 1) % SLIDING_WND_SIZE;
            int j2 = (j + SLIDING_WND_SIZE - 2) % SLIDING_WND_SIZE;
            int j3 = (j + SLIDING_WND_SIZE - 3) % SLIDING_WND_SIZE;
            int j4 = (j + SLIDING_WND_SIZE - 4) % SLIDING_WND_SIZE;
            for (int k = 0; k < TLSH_CHECKSUM_LEN; k++) {
                if (k == 0) {
                    this.lshBin.checksum[k] = bMapping((int) 0, slideWindow[j],
slideWindow[j1], this.lshBin.checksum[k]);
                } else {
                    this.lshBin.checksum[k] = bMapping(this.lshBin.checksum[k - 1],
slideWindow[j], slideWindow[j1], this.lshBin.checksum[k]);
                }
            }
            int r;
            r = bMapping((int)2, slideWindow[j], slideWindow[j1], slideWindow[j2]);
            this.aBucket[r]++;
        }
    }
}

```

```

        r = bMapping((int)3, slideWindow[j], slideWindow[j1], slideWindow[j3]);
        this.aBucket[r]++;
        r = bMapping((int)5, slideWindow[j], slideWindow[j2], slideWindow[j3]);
        this.aBucket[r]++;
        r = bMapping((int) 7, slideWindow[j], slideWindow[j2], slideWindow[j4]);
        this.aBucket[r]++;
        r = bMapping((int) 11, slideWindow[j], slideWindow[j1],
slideWindow[j4]);
        this.aBucket[r]++;
        r = bMapping((int) 13, slideWindow[j], slideWindow[j3],
slideWindow[j4]);
        this.aBucket[r]++;
    }

    fedLen++;
    j = (j + SLIDING_WND_SIZE + 1) % SLIDING_WND_SIZE;
}
this.dataLen += data.length();
}

public void update(byte [] data) {
    int j = this.dataLen % SLIDING_WND_SIZE;
    int fedLen = this.dataLen;
    if (this.aBucket == null) {
        this.aBucket = new int [BUCKETS];
    }
    for (int i = 0; i < data.length; i++) {
        slideWindow[j] = (int)data[i] & 0xff;
        if (fedLen >= 4) {
            int j1 = (j + SLIDING_WND_SIZE - 1) % SLIDING_WND_SIZE;
            int j2 = (j + SLIDING_WND_SIZE - 2) % SLIDING_WND_SIZE;
            int j3 = (j + SLIDING_WND_SIZE - 3) % SLIDING_WND_SIZE;
            int j4 = (j + SLIDING_WND_SIZE - 4) % SLIDING_WND_SIZE;
            for (int k = 0; k < TLSH_CHECKSUM_LEN; k++) {
                if (k == 0) {
                    this.lshBin.checksum[k] = bMapping(0, slideWindow[j],
slideWindow[j1], this.lshBin.checksum[k]);
                } else {
                    this.lshBin.checksum[k] = bMapping(this.lshBin.checksum[k - 1],
slideWindow[j], slideWindow[j1], this.lshBin.checksum[k]);
                }
            }
        }
        int r;
        r = bMapping((int)2, slideWindow[j], slideWindow[j1], slideWindow[j2]);

```

```

        this.aBucket[r]++;
        r = bMapping((int)3, slideWindow[j], slideWindow[j1], slideWindow[j3]);
        this.aBucket[r]++;
        r = bMapping((int)5, slideWindow[j], slideWindow[j2], slideWindow[j3]);
        this.aBucket[r]++;
        r = bMapping((int) 7, slideWindow[j], slideWindow[j2], slideWindow[j4]);
        this.aBucket[r]++;
        r = bMapping((int) 11, slideWindow[j], slideWindow[j1],
slideWindow[j4]);
        this.aBucket[r]++;
        r = bMapping((int) 13, slideWindow[j], slideWindow[j3],
slideWindow[j4]);
        this.aBucket[r]++;
    }

    fedLen++;
    j = (j + SLIDING_WND_SIZE + 1) % SLIDING_WND_SIZE;
}
this.dataLen += data.length;
}

public void finale() { /* word final is reserved */
    if (this.dataLen < 512) {
        this.aBucket = null;
        return;
    }

    int q1, q2, q3;
    int [] tmp = findQuartile(this.aBucket);
    q1 = tmp[0];
    q2 = tmp[1];
    q3 = tmp[2];

    int nonzero = 0;
    for (int i = 0; i < CODE_SIZE; i++) {
        for (int j = 0; j < 4; j++) {
            if (this.aBucket[4*i + j] > 0) {
                nonzero++;
            }
        }
    }
    if (nonzero <= 2 * CODE_SIZE) {
        aBucket = null;
        return;
    }
}

```

```

    }

    for (int i = 0; i < CODE_SIZE; i++) {
        int h = 0;
        for (int j = 0; j < 4; j++) {
            int k = this.aBucket[4 * i + j];
            if (q3 < k) {
                h += 3 << (j * 2);
            } else if (q2 < k) {
                h += 2 << (j * 2);
            } else if (q1 < k) {
                h += 1 << (j * 2);
            }
        }
        this.lshBin.tmpCode[i] = h;
    }

    this.aBucket = null;

    this.lshBin.lValue = lCapturing(this.dataLen);
    this.lshBin.setQLo((int) (((float)(q1 * 100.f) / (float) q3) % 16));
    this.lshBin.setQHi((int) (((float)(q2 * 100.f) / (float) q3) % 16));
    this.lshCodeValid = true;
}

public String hash() {
    if (this.lshCode != null) {
        return this.lshCode;
    }
    if (!this.lshCodeValid) {
        return null;
    }
    LshBinStruct tmp = new LshBinStruct();
    for (int k = 0; k < TLSH_CHECKSUM_LEN; k++) {
        tmp.checksum[k] = swapByte(this.lshBin.checksum[k]);
    }
    tmp.lValue = swapByte(this.lshBin.lValue);
    tmp.Q = swapByte(this.lshBin.Q);
    for (int i = 0; i < CODE_SIZE; i++) {
        tmp.tmpCode[i] = this.lshBin.tmpCode[CODE_SIZE - 1 - i];
    }
    this.lshCode = toHex(tmp.checksum);
    this.lshCode += toHex(new int [] {tmp.lValue});
    this.lshCode += toHex(new int [] {tmp.Q});
}

```

```

        this.lshCode += toHex(tmp.tmpCode);
        return this.lshCode;
    }

    public boolean compare(TLSH other) {
        return this.lshCode.equals(other.lshCode);
    }

    static final public int totalDiff(final String hash1, final String hash2) {
        int diff = 0;
        int [] iHash1 = fromHex(hash1);
        int [] iHash2 = fromHex(hash2);

        int q1diff = modDiff(iHash1[TLSH_CHECKSUM_LEN + 1] & 0xf,
iHash2[TLSH_CHECKSUM_LEN + 1] & 0xf, RANGE_QRATIO);
        if (q1diff <= 1)
            diff += q1diff;
        else
            diff += (q1diff - 1) * 12;

        int q2diff = modDiff(iHash1[TLSH_CHECKSUM_LEN + 1] >> 4,
iHash2[TLSH_CHECKSUM_LEN + 1] >> 4, RANGE_QRATIO);
        if (q2diff <= 1)
            diff += q2diff;
        else
            diff += (q2diff - 1) * 12;

        for (int k = 0; k < TLSH_CHECKSUM_LEN; k++) {
            if (iHash1[k] != iHash2[k]) {
                diff++;
                break;
            }
        }

        diff += hDistance(Arrays.copyOfRange(iHash1, TLSH_CHECKSUM_LEN + 2,
iHash1.length), Arrays.copyOfRange(iHash2, TLSH_CHECKSUM_LEN + 2,
iHash2.length));

        return diff;
    }

    @Override
    public double getSimilarity() {
        return similarity;
    }

```

```

    }

    @Override
    public int getPercentageSimilarity() {
        return (int) ((1 - (similarity / 200)) * 100);
    }
}

package context;

import methodsLSH.*;

public class Context {

    static private MethodLSH mlsh;

    static public void setMethodLSH(MethodLSH method) {

        mlsh = method;
    }

    static public void doAlgorithm(String text1, String text2) {

        mlsh.execute(text1, text2);
    }

    static public double getResultFirst() {

        return mlsh.getSimilarity();
    }

    static public int getResultSecond() {

        return mlsh.getPercentageSimilarity();
    }

}

package forms;

```

```

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.*;
import java.nio.charset.Charset;

import context.Context;
import methodsLSH.*;

public class MainForm extends JFrame {
    private JPanel panel1;
    private JButton loadFileButton1;
    private JButton loadFileButton2;
    public JTextArea textArea1;
    public JTextArea textArea2;
    private JButton minhashButton;
    private JButton simhashButton;
    private JButton nilsimsaButton;
    private JButton TLSHButton;
    private JButton exitButton;
    private JPanel methodsLSH;
    private JPanel testPanel;
    private JLabel firstText;
    private JLabel secondText;
    private final JFileChooser fileChooser;
    private ResultsForm rf;

    public MainForm(String title) {

        super(title);

        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setContentPane(panel1);
        this.pack();

        fileChooser = new JFileChooser();

        minhashButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                if (textArea1.getText().isEmpty() || textArea2.getText().isEmpty())
                    JOptionPane.showMessageDialog(MainForm.this, "Одна из текстовых
областей порожня!",

```

```

        "Помилка", JOptionPane.ERROR_MESSAGE);
    else {

        Context.setMethodLSH(new Minhash());
        Context.doAlgorithm(textArea1.getText(), textArea2.getText());

        rf = new ResultsForm("Minhash");
        rf.setVisible(true);

        rf.result.setText("Коефіцієнт Жаккара:");
        rf.resultValue1.setText(String.valueOf(Context.getResultFirst()));
        rf.resultValue2.setText(Context.getResultSecond() + " %");

    }
}
});

simhashButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        if (textArea1.getText().isEmpty() || textArea2.getText().isEmpty())
            JOptionPane.showMessageDialog(MainForm.this, "Одна з текстових
областей порожня!",
                "Помилка", JOptionPane.ERROR_MESSAGE);
        else {

            Context.setMethodLSH(new Simhash());
            Context.doAlgorithm(textArea1.getText(), textArea2.getText());

            rf = new ResultsForm("Simhash");
            rf.setVisible(true);

            rf.result.setText("Відстань Хеммінга:");
            rf.resultValue1.setText(String.valueOf(Context.getResultFirst()));
            rf.resultValue2.setText(Context.getResultSecond() + " %");

        }
    }
});

nilsimsaButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

```

```

        if (textArea1.getText().isEmpty() || textArea2.getText().isEmpty())
            JOptionPane.showMessageDialog(MainForm.this, "Одна з текстових
областей порожня!",
                "Помилка", JOptionPane.ERROR_MESSAGE);
        else {

            Context.setMethodLSH(new Nilsimsa());
            Context.doAlgorithm(textArea1.getText(), textArea2.getText());

            rf = new ResultsForm("Nilsimsa");
            rf.setVisible(true);

            rf.resultValue1.setText(String.valueOf(Context.getResultFirst()));
            rf.resultValue2.setText(Context.getResultSecond() + " %");
        }
    }
});

```

```

TLSHButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        if (textArea1.getText().isEmpty() || textArea2.getText().isEmpty())
            JOptionPane.showMessageDialog(MainForm.this, "Одна з текстових
областей порожня!",
                "Помилка", JOptionPane.ERROR_MESSAGE);
        else {

            Context.setMethodLSH(new TLSH());
            Context.doAlgorithm(textArea1.getText(), textArea2.getText());

            rf = new ResultsForm("TLSH");
            rf.setVisible(true);

            rf.resultValue1.setText(String.valueOf(Context.getResultFirst()));
            rf.resultValue2.setText(Context.getResultSecond() + " %");
        }
    }
});

```

```

loadFileButton1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

```

```

int returnVal = fileChooser.showOpenDialog(MainForm.this);
if (returnVal == JFileChooser.APPROVE_OPTION) {
    File file = fileChooser.getSelectedFile();
    try {
        Charset charset = Charset.forName("UTF-8");
        textArea1.read(new InputStreamReader( new FileInputStream(file),
"windows-1251"), null);
    }
    catch (IOException ex) {
        JOptionPane.showMessageDialog(MainForm.this, "Немає доступу
до файлу" +
        file.getAbsolutePath(), "Помилка",
JOptionPane.ERROR_MESSAGE);
    }
}
});

loadFileButton2.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        int returnVal = fileChooser.showOpenDialog(MainForm.this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            try {
                textArea2.read( new FileReader(file.getAbsolutePath()), null );
            }
            catch (IOException ex) {
                JOptionPane.showMessageDialog(MainForm.this, "Немає доступу
до файлу" +
                file.getAbsolutePath(), "Помилка",
JOptionPane.ERROR_MESSAGE);
            }
        }
    }
});

exitButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        MainForm.this.dispose();
    }
});

```

```

        }
    });
}
}

package forms;

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

public class ResultsForm extends JFrame {
    private JPanel panel1;
    private JButton OKButton;
    public JLabel result;
    public JLabel resultPercentage;
    public JLabel resultValue1;
    public JLabel resultValue2;
    private JPanel panelResults;

    public ResultsForm(String title) {

        super(title);

        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        this.setContentPane(panel1);
        this.pack();

        OKButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {

                ResultsForm.this.dispose();

            }
        });
    }
}

package main;

import forms.*;

import javax.swing.*;

```

```
public class Main {  
  
    public static void main(String[] args) {  
  
        JFrame form = new MainForm("Методи LSH");  
        form.setVisible(true);  
  
    }  
}
```

ДОДАТОК Б
Слайди презентації

Дипломна кваліфікаційна робота бакалавра

Програмна реалізація методів локально-чуттєвого гешування

Виконав:

студент гр. КНТ-137

Василенко Д.С.

Керівник:

зав. каф. ПЗ, д.т.н., професор

Субботін С.О.

1

Рисунок Б.1 – Слайд 1

Мета роботи, об'єкт, предмет та задачі дослідження

Мета роботи - розроблення програмного забезпечення, яке реалізує методи локально чуттєвого гешування.

Об'єкт дослідження – процес локально-чуттєвого гешування.

Предмет дослідження – програмне забезпечення для реалізації методів локально-чуттєвого гешування.

Задачі дослідження:

- огляд та аналіз методів локально-чуттєвого гешування;
- розробка архітектури програмного забезпечення;
- вибір технологій для її розробки;
- розробка програмного забезпечення;
- тестування та її експлуатація.

2

Рисунок Б.2 – Слайд 2

Огляд та аналіз методів локально-чуттєвого
гешування
Метод Minhash

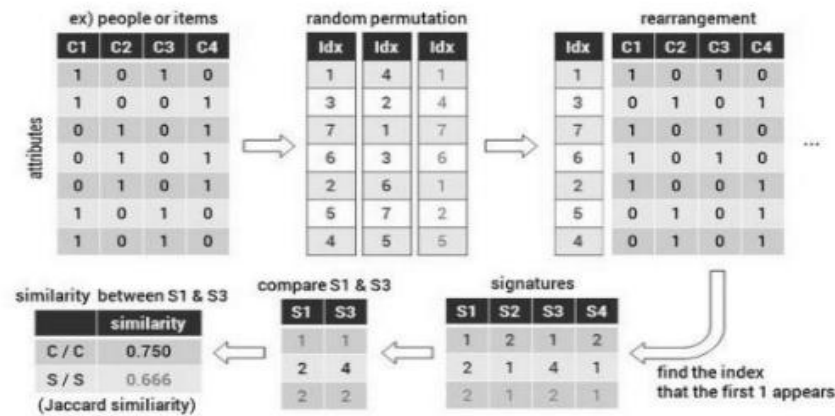


Рисунок 1 – Алгоритм методу Minhash [1]

1. MinHash - SteemKR [Electronic resource]. - Access mode: <https://steemkr.com/minhash/@verysimple/minhash>.

3

Рисунок Б.3 – Слайд 3

Огляд та аналіз методів локально-чуттєвого
гешування
Метод Simhash

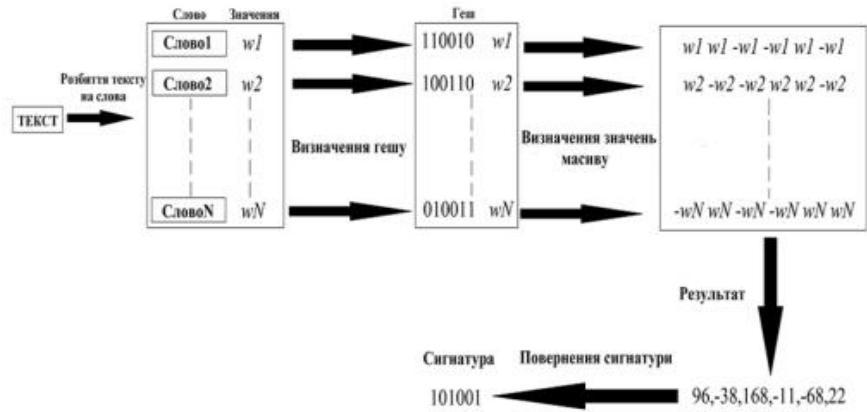


Рисунок 2 – Алгоритм методу Simhash

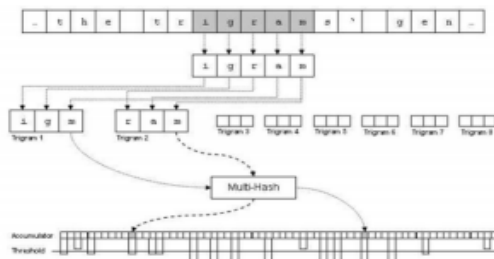
4

Рисунок Б.4 – Слайд 4

Огляд та аналіз методів локально-чуттєвого гешування

Методи Nilsimsa та TLSH

Метод Nilsimsa



Метод TLSH

Метод TLSH діє за таким же принципом як і **метод Nilsimsa**, але на відміну від нього **метод TLSH** будує ще заголовок та тіло дайджесту.

Рисунок 3 – Алгоритм методу Nilsimsa [2]

2. Damiani E. An Open Digest-based Technique for Spam Detection / [E. Damiani, S. De Capitani di Vercati, S. Paraboschi et al.] // The ISCA 17th International Conference on Parallel and Distributed Computing Systems, September 15-17, 2004 : proceedings. – San Francisco, 2004. – P. 559-562.

5

Рисунок Б.5 – Слайд 5

Діаграма пакетів

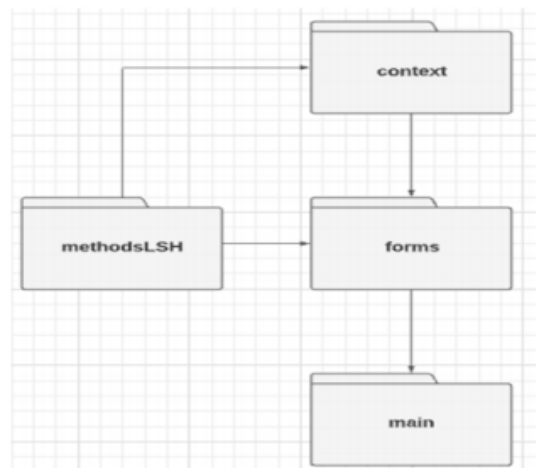


Рисунок 4 – Діаграма пакетів

6

Рисунок Б.6 – Слайд 6

Діаграма класів

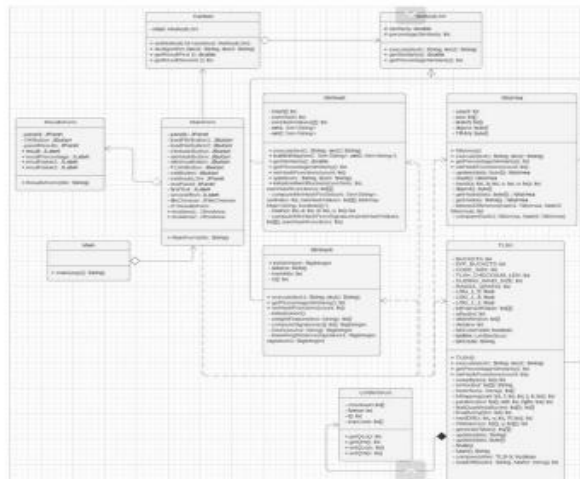


Рисунок 5 – Діаграма класів

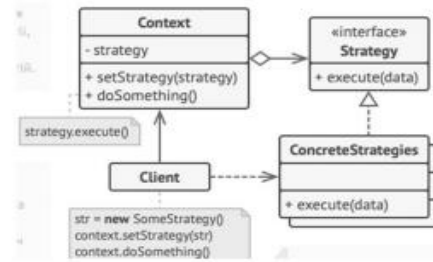


Рисунок 6 – Структура паттерну "Стратегія" [3]

3. Паттерн Стратегія [Електрон. ресурс]. – Режим доступу: <https://refactoring.guru/uk/design-patterns/strategy>.

7

Рисунок Б.7 – Слайд 7

Вибір мови програмування

Мова	Java	C#	Python	Ruby
Критерії				
Об'єктно-орієнтованість	Так	Так	Так	Так
Кросплатформність	Так	Ні	Так	Так
Автомат. управління пам'яттю	Так	Так	Так	Так
Типізація	Суворі	Суворі	Динамічна	Динамічна
Швидкодія	Висока	Висока	Низька	Низька

Таблиця 1 – Порівняльна таблиця мов програмування

8

Рисунок Б.8 – Слайд 8

Форми програми

Головна форма програми

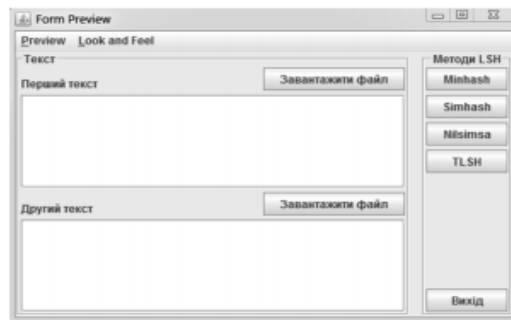


Рисунок 7 – Головна форма програми

Дочірня форма програми

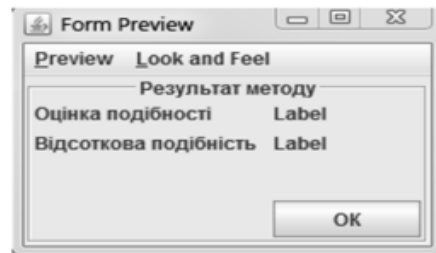


Рисунок 8 – Дочірня форма програми

9

Рисунок Б.9 – Слайд 9

Застосування програми

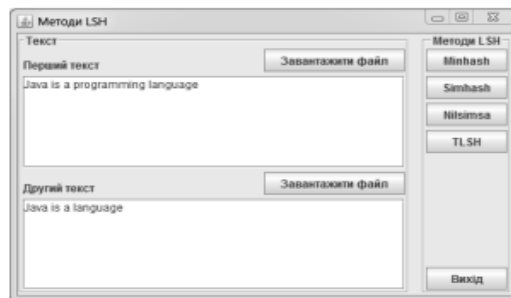


Рисунок 9 – Введені тексти в текстових областях головної форми програми

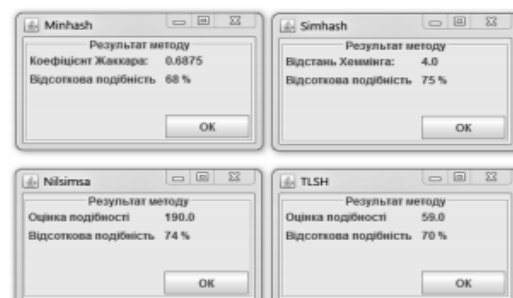


Рисунок 10 – Дочірні форми програми з результатами методів локально-чуттєвого гешування

10

Рисунок Б.10 – Слайд 10

Висновки

- В ході виконання дипломної роботи було розроблено програмне забезпечення, що реалізує методи локально-чуттєвого гешування.
- Програмне забезпечення може бути корисне для користувачів, яким необхідно визначити подібність двох текстів.
- Локально-чуттєве гешування є найактуальнішим та багатообіцяючим підходом для вирішення багатьох задач для пошуку подібності в багатовимірних просторах. Через свою швидкість та легкість у використанні його застосовують в багатьох сервісах та компаніях для боротьби зі спамом, створення рекомендаційних систем, ідентифікації схожості між об'єктами і т.п.

11

Рисунок Б.11 – Слайд 11