Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Безпека інформаційних систем і технологій

«Допущено до захисту»

Зав.кафедрою БІСТ

Сватовський І.І._____

« » червня 2023р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

спеціальність: 125 Кібербезпека

на тему: «Математичний алгоритм верифікації відбитків пальців на основі циліндр-кодів з розробкою методу порівняння циліндрів та шаблонів»

оцінка « »

Голова ЕК

Лемешко О.В._____

Керівник проф. Кузнецов О.О.
(прізвище та ініціали/підпис)

Рецензент проф. Краснобаєв В.А.
(прізвище та ініціали/підпис)

Виконавець студент групи КБ-41

Стоянов М.О
(прізвище та ініціали/підпис)

Харків – 2023

РЕФЕРАТ

Пояснювальна записка містить 42 сторінок, 26 рисунків, 4 таблиць, 3 додатків, 11 джерел.

Метою дипломної роботи є дослідження та розробка методу порівняння методу циліндр-кодів із методом шаблонів.

Об'єктом дослідження дипломної роботи є процесс порівняння верифікації людини за відбитком пальця, а також самі відбитки пальців та їх шаблони, що використовуються для верифікації та порівняння методів циліндр-коду та шаблонів.

Предметом розробки є математичний алгороитм верифікації відбитків пальців на основі циліндр-кодів, а також розробка ефективного методу порівняння цього методу та методу шаблонів для покращення ефективності працездатності даних методів.

Методи дослідження: дослідження існуючих рішень верифікації за відбитком пальця на основі методу циліндр-кодів та метду шаблонів із розробкою методу що ефективно вимірює витрачені на верифікацію час, пам'ять, а також отримує кількість похибок при роботі кожного з методів.

Результатами проведеної роботи є метод порівняння методів циліндр-кодів та шаблонів, проведення експеременту на базі даних відбитків пальців та отримання даних щодо швидкості, витраченої пам'яті та можливої похибки при роботі кожного з методів.

Ключові слова: ВІДБИТКІ ПАЛЬЦІВ, МЕТОД ЦИЛІНДР-КОДІВ, МЕТОД ШАБЛОНІВ, МЕТОД ПОРІВНЯННЯ ЦИЛІНДРІВ ТА ШАБЛОНІВ, БІОМЕТРИЧНА АУТЕНТИФІКАЦІЯ.

ABSTRACT

The explanatory note contains 42 pages, 26 figures, 4 tables, 3 annexes, 11 sources.

The aim of the thesis is research and development of a method of comparing the cylinder-code method with the template method.

The subject matter is the process of comparing a person's fingerprint verification, as well as the fingerprints themselves and their patterns used to verify and compare cylinder code and pattern methods.

The scope of the study is a mathematical algorithm for the verification of fingerprints based on cylinder codes, as well as the development of an effective method for comparing this method and the template method to improve the performance of these methods.

Research methods: research of existing fingerprint verification solutions based on the cylinder-code method and the template method with the development of a method that effectively measures the time spent on verification, memory, and also obtains the number of errors during the operation of each of the methods.

The results are method of comparing the methods of cylinder codes and templates, conducting an experiment on the database of fingerprints and obtaining data on the speed, memory consumption and possible error in the operation of each of the methods.

Keywords: FINGERPRINTS, CYLINDER CODE METHOD, PATTERN METHOD, CYLINDER AND PATTERN COMPARISON METHOD, BIOMETRIC AUTHENTICATION.

ЗМІСТ

УМОВНІ ПОЗНАЧЕННЯ

AFIS – бібліотека Automated Fingerprint Identification System;
MCC – алгоритм Minutae Cylinder-Code.

CUDA – Мова програмування від компанії Nvidea, що походить від C.

ВСТУП

У сучасному світі, разом із розвитком технологій зростає і попит щодо ідентифікацію осіб та їх безпеки. Саме це і спонукає людство до розробок та досліджень нових методів та алгоритмів у сфері ідентифікації осіб за допомогою відбитків пальців, що зробили б це точно, ефективно та швидко. Через те, що відбитки пальців є унікальними та не змінюються протягом усього життя , вони є одними з найнадійніших факторів для ідентифікації особи.

Загалом люди вже не перше сторіччя використовують відбитки пальців задля ідентифікації людини. Ще з давнини даний метод був використаний задля ідентифікації осіб, що були звинувачені в злочинах. Загалом у сучасності «батьком» системи, що ідентифікує осіб за відбитками пальців, вважається Хуан Вучетів, який був аргентинським криміналістом. Дана систему він назвав «дактилоскопією». Вона базувалась на використанні унікальних рис пальців людини, таких як кількість і форма довжинних борозен, що знаходяться на поверхні пальців. Згодом даний метод став використовуватись по всьому світу, де у 20-му сторіччі в США система ідентифікації за відбитками пальців стала частиною судової процедури, а у 1924 було створено першу базу даних відбитків пальців у Федеральному бюро розслідувань США.

На сьогоднішній день звісно що використовування відбитків пальців використовується не лише у криміналістиці, а й у повсякденному житті. Гарним прикладом є не лише використання відбитку пальця у якості перевірки смартфоном чи є ви його власником, а й при відвідуванні лікаря, подачі заяв на отримання паспорту або візи та при працевлаштування. У США їх використовують для ідентифікації пасажирів на міжнародних рейсах, а в Індії та Китаї даний метод використовують для автоматичного входу до будівель та розрахунках за товари у супермаркетах.

Метою цієї дипломної роботи є розробка математичного алгоритму верифікації відбитків пальців на основі циліндр-кодів з розробкою методу порівняння циліндрів та шаблонів. Даний метод надає можливість ефективно і точно ідентифікувати осіб за їхніми відбитками пальців. Задля досягнення поставленої мету було вирішено використовувати циліндр-коди, що використовуються для представлення відбитку пальця у вигляді набору циліндрів. Розроблений метод порівняння циліндрів та шаблонів дозволить ефективно знаходити відповідності між циліндр-кодами та шаблонами, що є важливим етапом верифікації.

Людство досить далеко зайшло у темі дослідження методів та алгоритмів для розпізнавання відбитків пальців, але все ще існуючі способи ідентифікації людини за відбитками пальців не є ідеальними, а отже і виникає потреба у подальшому вивченні даної сфери та розробці нових методів та алгоритмів, що сприяли б покращенню вже отриманих результатів.

Окрім того, було розроблено програмне забезпечення, яке реалізує розроблені методи та алгоритми для верифікації відбитків пальців на основі циліндр-кодів. Використовуючи дане програмне забезпечення було проведено ряд експериментів задля перевірки ефективності та швидкості розроблених методів та алгоритмів на практиці.

Результати експериментальної перевірки надають змогу оцінити ефективність та швидкості запропонованого методу верифікації відбитків пальців на основі циліндр-кодів з розробкою методу порівняння циліндрів та шаблонів.

В даній дипломній роботі будуть описані теоретичні основи розробки методу верифікації відбитків пальців на основі циліндр-кодів з розробкою методу порівняння циліндрів та шаблонів, а також буде надано детальний опис програмної реалізації розробленого методу та алгоритму. Для експериментальної перевірки будуть використані різні набори даних відбитків пальців, які будуть

оброблені за допомогою розробленого програмного забезпечення.

Отже, результати цієї дипломної роботи будуть корисними для розробки нових методів та алгоритмів ідентифікації осіб за їхніми відбитками пальців, а також для покращення існуючих методів та алгоритмів.

1 ЗАГАЛЬНІ ВІДОМОСТІ ТА ОГЛЯД ЛІТЕРАТУРИ

1.1 Загальні відомості про метод шаблонів

Метод шаблонів використовується для розпізнавання біометричних даних з високою точністю. Перед початком роботи з методом шаблонів необхідно створити базу даних, де зберігаються шаблони відбитків пальців, що були зібрані раніше. Шаблон - це структурована інформація, яка описує геометричні та топологічні особливості відбитка пальця.

Для створення шаблону відбитка пальця, спочатку необхідно зібрати дані відбитка. Це може бути зроблено за допомогою датчика відбитка пальця, який сканує пальці та отримує зображення. Далі зображення обробляється та аналізується з метою відокремлення особливостей відбитка, наприклад, кінчиків ліній та точок перетину, які називаються мінуціями.

Отримана інформація про особливості відбитка подається на вхід алгоритму вирішення задачі розпізнавання. В ході цього процесу шаблон порівнюється з тими, які містяться в базі даних, та визначається ступінь схожості. Якщо ступінь схожості перевищує певний поріг, визнається, що відбиток пальця належить до особи, чия інформація міститься в базі даних.

Існують різні методи обробки та порівняння шаблонів відбитків пальців. Один з таких методів - це метод динамічного програмування, який застосовується для порівняння двох шаблонів з метою визначення найбільшої кількості однакових точок. Інший метод - це метод головних компонент, який дозволяє зменшити розмірність шаблону та зменшити обчислювальну складність при порівнянні відбитків пальців. Метод головних компонент зводить високовимірні дані до набору незалежних змінних, які називаються головними компонентами.

Крім того, в методі шаблонів використовується технологія захисту від

шахрайства, яка полягає у виявленні спроб обману системи розпізнавання пальців, таких як використання зубочисток або гумових манжет. Для цього система може аналізувати тиснення на датчику відбитка пальця, вимірювати відстань між пальцем та датчиком або вимірювати температуру пальця.

Застосування методу шаблонів у розпізнаванні пальців досить широке. Він використовується в системах контролю доступу до приміщень, системах ідентифікації клієнтів у банківських терміналах, системах збереження медичної інформації та інших застосуваннях, де необхідний високий рівень безпеки.

Незважаючи на свою ефективність та широке застосування, даний метод має певні недоліки. Наприклад, він не здатний дати точний результат, якщо відбиток пальця був пошкоджений або забруднений. Крім того, існує можливість отримання хибних результатів при використанні шаблонів, які не враховують варіації відбитків від певної особи залежно від часу дня, стану здоров'я або стану пальця.

У цілому, метод шаблонів є ефективним та зручним для використання методом розпізнавання пальців.

1.2 Загальні відомості про метод циліндрів

Метод циліндрів є ще одним методом розпізнавання пальців, який використовується для ідентифікації особи на основі відбитків її пальців. Цей метод полягає у використанні тривимірної моделі циліндра, яка відображає форму та структуру відбитка пальця.

Він був запропонований в 2003 році дослідниками з університету Мічигану, які використовували його для розпізнавання пальців у системах безпеки. Основною перевагою цього методу є те, що він може враховувати не тільки форму відбитка пальця, але й його структуру та особливості.

Даний метод використовується для створення тривимірної моделі пальця, яка складається з декількох циліндрів з різним радіусом та довжиною. Ці

циліндри відображають структуру відбитка пальця та дозволяють враховувати особливості його поверхні.

Для розпізнавання пальців за допомогою методу циліндрів використовується алгоритм, який порівнює тривимірну модель пальця з відбитком пальця, який був наданий для ідентифікації. Алгоритм використовує математичні методи для порівняння форми та структури пальця, і визначає ступінь відповідності між тривимірною моделлю та відбитком.

Метод циліндрів має декілька переваг порівняно з іншими методами розпізнавання пальців. Він може враховувати особливості поверхні пальця та структуру його відбитка, що дозволяє отримувати більш точні результати. Крім того, метод циліндрів менше чутливий до різних змін, таких як зміна точки зору або нахилу пальця, що дозволяє йому бути ефективнішим у реальних умовах використання.

Однак, у нього існують і деякі недоліки, такі як складність обробки даних та висока вимогливість до обчислювальних ресурсів. Також, якщо відбиток пальця має складні структури або нерегулярну форму, то цей метод може давати неточні результати.

Незважаючи на ці обмеження, метод циліндрів все ще є популярним і досить ефективним методом розпізнавання пальців. Він використовується в різних системах безпеки, таких як контроль доступу та ідентифікація осіб на кордоні.

Одним з важливих напрямків розвитку методу циліндрів є використання штучного інтелекту, зокрема глибокого навчання, для поліпшення точності розпізнавання та скорочення часу обробки даних. Також, дослідники продовжують вивчати та досліджувати різні модифікації цього методу з метою поліпшення його ефективності та застосовності.

1.3 Різниця між алгоритмами шаблонів та циліндр-кодів

Основна різниця між алгоритмами шаблонів та алгоритмом циліндр-кодів полягає у способі опису відбитку пальця та використанні цієї інформації для верифікації. Алгоритми шаблонів використовують готові шаблони або витяги з відбитку пальця, які порівнюються зі зразком для визначення ідентичності, тоді як алгоритм циліндр-кодів базується на конструюванні унікального коду на основі геометричних параметрів відбитку.

Алгоритми шаблонів можуть бути заснованими на різних ознаках відбитку пальця, таких як контур ліній, відстань між точками, орієнтація ліній тощо. У порівнянні з алгоритмами циліндр-кодів, алгоритми шаблонів зазвичай потребують більшої обробки даних та менш точні, оскільки вони не враховують геометричних характеристик відбитку.

Алгоритм циліндр-кодів використовує відстань між точками та кут між лініями, які з'єднують ці точки, для побудови унікального коду. Цей код може бути збережений у базі даних та порівняння з іншими відбитками пальців для верифікації. Алгоритм циліндр-кодів вважається більш точним у порівнянні з алгоритмами шаблонів, оскільки він враховує геометричні параметри відбитку та їх залежності від інших параметрів.

Обидва методи використовуються в системах біометричної ідентифікації та верифікації відбитків пальців. Вибір конкретного методу залежить від специфікації та потреб клієнта.

Окрім того, алгоритми шаблонів та циліндр-кодів мають різний підхід до обробки даних. У методі шаблонів, після витягування важливих ознак з відбитка пальця, вони зберігаються як шаблон для порівняння з іншими відбитками пальців. У методі циліндр-кодів, важливі ознаки зображення відбитка пальця використовуються безпосередньо для створення циліндричної проекції відбитка пальця, яка тоді порівнюється з іншими циліндричними проекціями. Таким чином, метод циліндр-кодів працює з більш повним набором даних зображення

відбитка пальця, ніж метод шаблонів.

Інша важлива відмінність полягає в тому, що метод циліндр-кодів може бути більш ефективним для розпізнавання відбитків пальців у випадку, коли відбиток пальця має не стандартну форму або зазначені неоднорідності, такі як рубці або подряпини. Метод циліндр-кодів дозволяє враховувати ці неоднорідності та знижувати їх вплив на процес розпізнавання.

У підсумку, хоча обидва методи шаблонів та циліндр-кодів широко використовуються для розпізнавання відбитків пальців, вони мають різні підходи до обробки даних та різні відмінності в тому, як вони працюють. Вибір між цими методами залежить від конкретної ситуації та потреб користувача.

1.4 The Minutia Cylinder-Code

Для початку розглянемо статтю Рафаєля Капеллі, Маттео Феррарі та Девіда Малтоні під назвою Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition. У даній статті описується новий метод представлення та порівняння відбитків пальців та їх ідентифікації. Він базується на 3D-структурах, які називаються циліндрами. Циліндри можуть бути створені з використанням певного набору обов'язкових ознак, таких як положення та напрямок мінуцій, якими користуються стандарти, наприклад ISO/IEC 19794-2 (2005). Одразу потрібно розібратись, що ж означає слово мінуція у контексті даної статі.

Мінуції – це особливі точки на відбитку пальця, що можуть бути використані для ідентифікації людини. Вони зазвичай представлені як дуже маленькі виступи або поглиблення на поверхні пальця, також можуть мати різний напрямок. Зазвичай, для ідентифікації використовуються три основних типи мінуцій: довга мінуція (розташована паралельно до краю пальця), коротка мінуція (розташована перпендикулярно до краю пальця) та розгалужена мінуція (материнська мінуція, що розгалужується на двох або багатьох дочок). За

допомогою циліндрів вдається зберігати цю важливу інформацію, а даний алгоритм, Minutia Cylinder-Code, успішно використовується для ідентифікації людини.

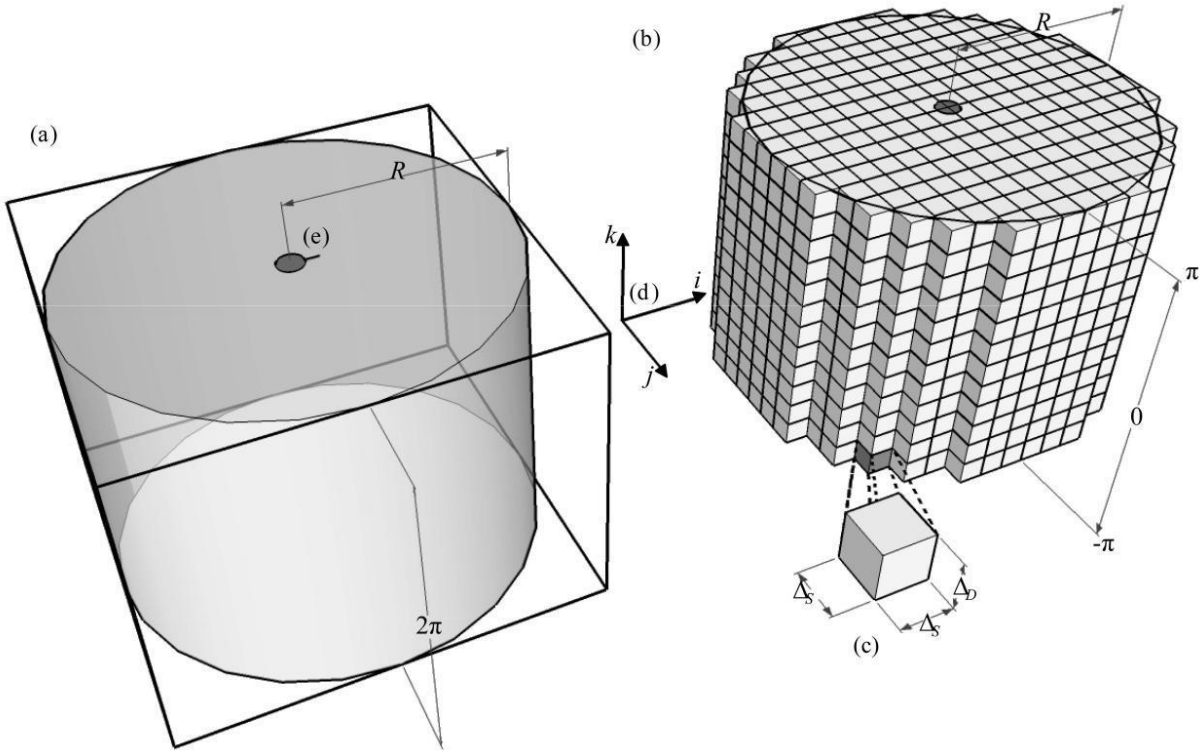

Рисунок 1.1 – Графічне представлення локальної структури, пов'язаної з заданою дрібницею: (a) циліндр із охоплюючим кубоїдом (б) дискретизація кубоїда на клітинки (c) розміром $\Delta_s \times \Delta_s \times \Delta_D$, де показано лише клітинки, центр яких знаходиться всередині циліндра. Також треба зауважити, що кубоїд повернути так, що що (d) вісь *i* суміщена до (e) напрямком відповідної контрольної точки.

Усі структури пов'язані із заданою мінуцією $m = \{x_m, y_m, \theta_m\}$ зображена циліндром радіусу *R* та висотою 2π, основа якої зосереджена на місці контрольної точки (x_m, y_m) (Рисунок 1.1a). Кожен циліндр укладено в середину прямокутного паралелепіпеда, де основа вирівняна відповідно до напряму мінуції θ_m, а кубоїд

є дисрктизованим у $N_C = N_S \times N_S \times N_D$ комірок. А кожна комірка, як можна вже було зрозуміти з Рисунок 1.1 є малим кубоїдом з основою $\Delta_S \times \Delta_S$ та висотою Δ_D. Δ_S та Δ_D можна обчислити так, як вказано у (1.1).

$$\Delta_S = \frac{2R}{N_S};\ \Delta_{D1} = \frac{2*\pi}{N_D} \tag{1.1}$$

Також важливо зазначити, що у кожної комірки є три унікальні індекси, за якими вона може бути ідентифікована (i, j, k). Вони визначають її позиції у кубі, який охоплює циліндр, де:

$$i, j \in I_S = \{n \in IN, 1 \leq n \leq N_s\} \tag{1.2}$$

$$k \in I_D = \{n \in IN, 1 \leq n \leq N_D\} \tag{1.3}$$

На Рисунку 1.2 можна бачити різні C_m *(i, j, k)* значення представлені різними рівнями сірого (чим світліше, тим більше).

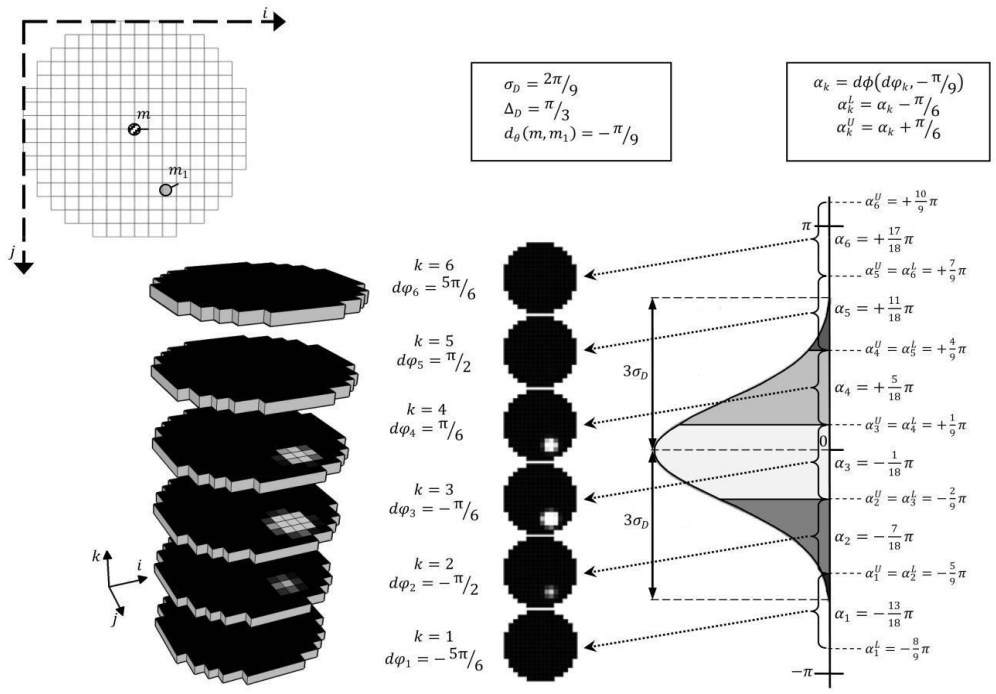

Рисунок 1.2 - Спрощений випадок, коли лише одна мінуція (m_1) вносить свій внесок у циліндр, пов'язаний із мінуцією *m*. [1]

Таким чином і створюються циліндри-набори із отриманих шаблонів

мінуцій, де C_m це циліндр, пов'язаний з мінуцією m, що містить значення C_m (i, j, k). Циліндри C_m вважаються недійсними у таких випадках:

- ті, що менше ніж min_{VC} $valid$ комірок в циліндрі
- ті, що менші ніж min_M мінуції, що сприяють циліндру, тобто ті, що min_M мінуції m_t такі як (1.4)

$$d_s(m_t, m) \leq R + 3\sigma_S, \text{ де } m_t \neq m \quad (1.4)$$

Використовуючи накопичення внесків з кожної мінуції m_t її приналежності до сусідів $N_{p_{i,j}^m}$ від $p_{i,j}^m$, для кожної комірки d (i, j, k) розраховується числове значення C_m (i, j, k) , де

$$N_{p_{i,j}^m} = \{m_t \in T;\ m_t \neq m, d_s(m_t, p_{i,j}^m) \leq 3\sigma_S\} \quad (1.5)$$

$3\sigma_S$ є радіусом околиці (Рисунок 1.3).

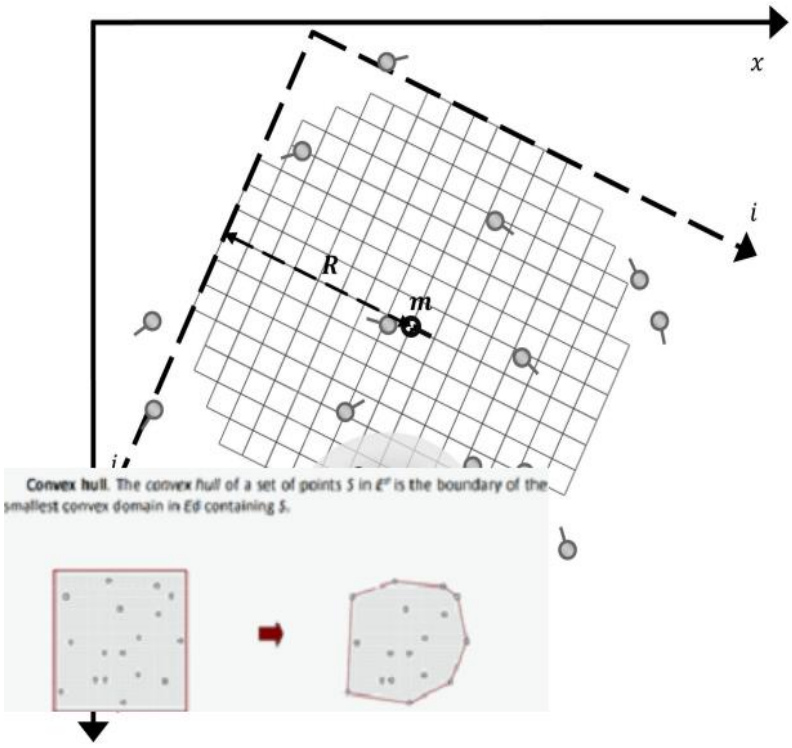

Рисунок 1.3 - Розріз циліндра, пов’язаний з контрольною точкою m. Показані всі мінуції, залучені до конструкції циліндра. [1]

Потрібно звернути увагу, що вони не обов’язково лежать всередині основи циліндра, оскільки зсув $3\sigma_S$ дозволено. $G_S(t)$ значення в околицях даної комірки

(з центром $p_{i,j}^{m}$ виділені (темніші області представляють вищі значення). Чорні мінуції – це ті, що знаходяться в околиці

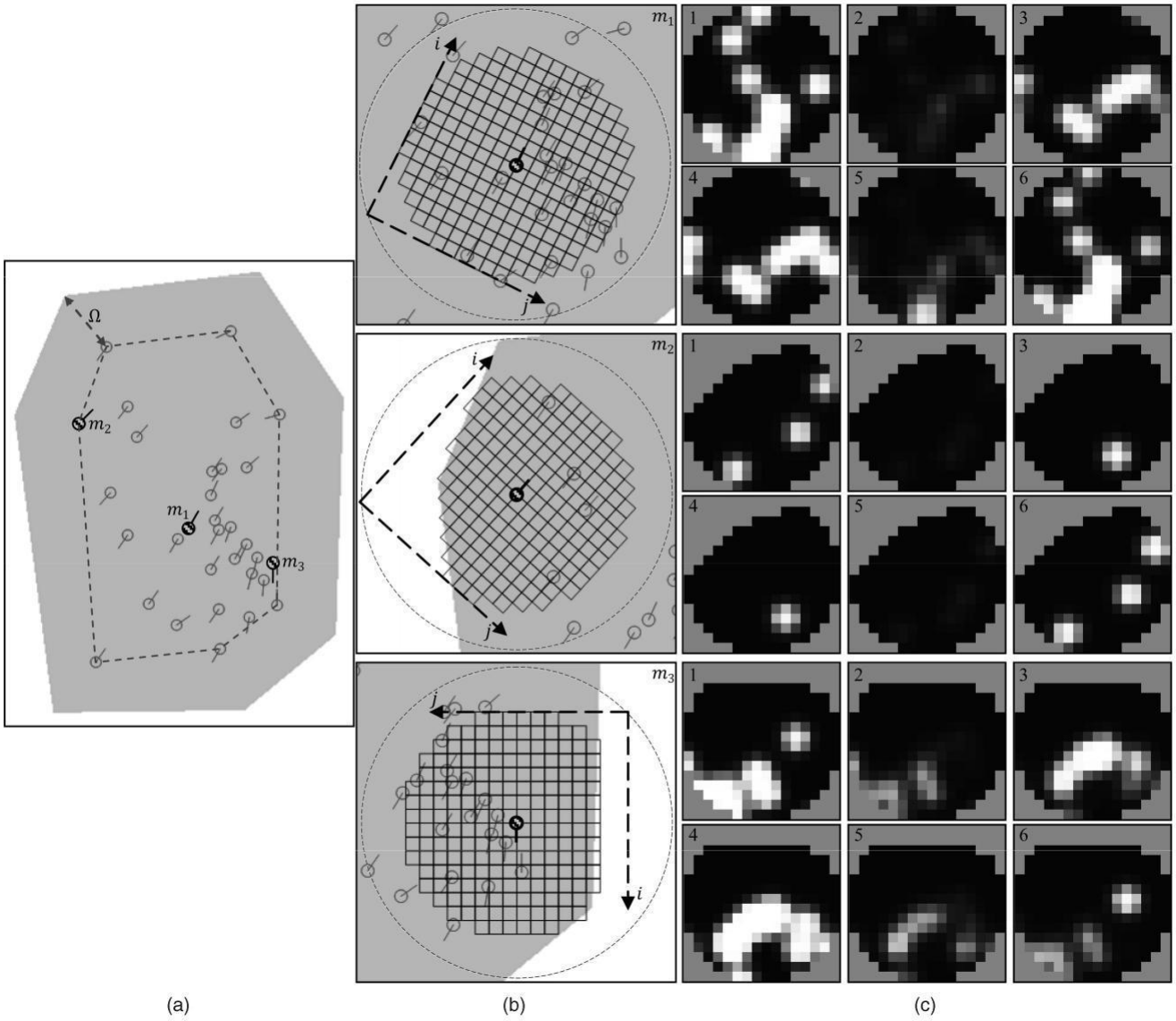

Рисунок 1.4 - Шаблон мінуції з відповідною опуклою оболонкою.

На Рисунку 1.4 в (a) можна бачити шаблон мінуції з відповідною опуклою оболонкою. Для кожної з трьох позначок, виділених у (a), стовпець (b) показує основу відповідного циліндра (намальовано лише валідні клітинки); мінуції в межах пунктирних кіл – це ті, які вносять внесок у значення комірки циліндра. Стовпець (c) показує значення клітинок трьох циліндрів для кожного значення

$k \in \{1, …., 6\}$(світліші елементи представляють вищі значення); зверніть увагу, що секції циліндра в (c) обертаються відповідно до напрямку.[1]

Одна із ключових відмінностей MCC у порівнянні із іншими методами ідентифікації особи за відбитками пальців є те, що MCC не використовує тип або якість мінуції як ознаки, через те, що це є ненадійним та не семантично зрозумілою характеристикою. Окрім цього, варто зазначити, що функції кодування MCC, які мають фіксовану довжину та орієнтовані на біти, дозволяють визначати прості, але у той же час дуже ефективні метрики для обчислення локальних подібностей та їх об’єднання у загальний бал.

Згідно з представленими у статті результатами експериментів, що були проведені на багатьох тестових наборах даних, включаючи FVC2006, MCC показав результати, що перевершують роботу інших сучасних методів ідентифікації за допомогою відбитків пальців. Також важливою частиною розуміння алгоритму є те, що MCC має фіксований радіус підходу, що надає можливість краще обробляти некоректні результати або помилкову інформацію про мінуції у порівнянні з методами що були запропоновані раніше, зокрема у алгоритмі Fingerprint Minutiae Matching Based on the Local and Global Structures Джіанга та Яу. Що ще важливо знати про даний метод, так це те, що він ефективно управляє проблемами меж зображення без додаткового навантаження на етапах кодування та порівняння. Локальні спотворення та невеликі помилки виділення ознак допускаються завдяки застосуванню згладжених функцій (тобто функцій, що стійкі до помилок) на етапі кодування.

За результатами дослідження, даний алгоритм є швидшим за інші алгоритми навіть без оптимізації, орієнтованої на мови асемблеру чи іншу. Середній час, витрачений на обчислення загальної оцінки (T_{gs}) загалом не залежить від конкретного локального алгоритму зіставлення, за помітним винятком: методи, засновані на проблемі призначення (LSA та LSA-R) швидші у поєднанні з реалізацією Фенгу. Також варто зазначити, що рехультати учасників

FVC2006 набагато вищий, ніж MCC: 416 мс для відкритої категорії та 53 мс для легкої категорії. Однак пряме порівняння неможливе, оскільки час, зазначений у FVC2006, відповідає зіставленню «шаблон із зображенням» і, отже, включає вилучення однієї функції, що вимагає часу .

	T_{cs}	T_{ls}	T_{gs}			
			LSS	LSA	LSS-R	LSA-R
MCC16	21.0	21.0	0.5	4.3	2.7	4.7
MCC16b	17.3	1.2	0.5	4.3	2.8	4.7
MCC8b	4.2	0.3	0.5	4.2	2.9	4.8
Jiang	1.0	0.8	0.4	4.3	2.6	4.1
Ratha	1.0	250.7	0.5	4.3	2.8	4.4
Feng	0.2	12.3	0.5	2.4	2.8	3.1

	Raw format	Compressed format (rar)		Compressed format (zip)	
	Size	*Size*	*Ratio*	*Size*	*Ratio*
MCC16	209253	103766	202%	104595	200%
MCC16b	7630	1457	524%	1642	465%
MCC8b	1913	605	316%	655	292%
Jiang	1068	608	176%	647	165%
Ratha	26543	19487	136%	20046	132%
Feng	1428	567	252%	614	233%

Рисунок 1.5 - Результати роботи алгоритму циліндр-кодів у порівнянні з іншими реалізаціями.[1]

1.5 Алгоритм верифікації відбитків пальців використаний у SourceAFIS

SourceAFIS - це готове рішення, що надає можливість порівнювати відбитки пальців у форматі 1:1, або 1:N. Його використання досить просте, через те, що він бере зображення відбитку на вході і віддає результуючий рахунок на виході, після чого цей рахунок порівнюється із налаштованим порогом співпадіння відбитків пальців.

Його використання було зроблено максимально простим, через що дана бібліотека неймовірно зручна у використанні, а також швидкість та точність співпадіння достатня для більшості додатків існуючих на сьогоднішній день.

Що стосується алгоритму, то даний алгоритм не є копіюванням уже існуючих алгоритмів з підручників чи статей. Він реалізує в собі найкращі підходи які було отримано з реалізацій із відкритим програмним кодом, але з певним доопрацюванням та покращенням.

Дана бібліотека є однією з небагатьох, що має прозрачність виконуємого алгоритму і розкриває усі проміжні структури даних, що обчислюються

алгоритмом під час зіставлення відбитків пальців та усі дії, що виконуються над ними. Завдяки цьому, даний відкритий код є справжньо знахідкою для розробників, які хочуть якось його модифікувати, що було неможливо майже з усіма відомими розробками до цього. Таким чином це відкриває двері до безлічі модифікацій, які можуть бути запроваджені при бажані та наявності вільного часу приватними розробниками.

Як зазначає автор даного проекту, Роберт Вазан, мотивацією для створення цього стало те, що біометрія,якщо її правильно використовувати, є загальним благом. За його думкою, відбитки пальців і біометрія в цілому є багатообіцяючою альтернативою паролям. Будь-які комп'ютери не мають наосліп довіряти будь-кому, хто отримує доступ до пароля, а також не мають відмовляти у доступі власнику лише через те, що він забуває якісь речі. Гарною думкою в цьому плані є те, що комп'ютери мають знати хто є їх власником, або власником якоїсь інформації яку він містить та протистояти будь-яким спробам отримання доступу до нього, якщо це не є власник. На сьогодняшній день цієї мети ще не досягнуто, але верифікація людини за допомогою відбитків пальців є частиною рішення. В свою чергу бібліотка з відкритим кодом, як SourceAFIS, підвищує рівень впровадження біометрії у життя.

Бібліотеки порівняння відбитків пальців, разом з будь-яким іншим програмним забезпеченням безпеки, повинні бути прозорими і відкритими на всіх рівнях. Програмне забезпечення безпеки повинно бути незалежним від ринкових тиснень, щоб зосередитися на якості. Воно має відверто розмовляти про свою природу та ефективність, не затуляючись маркетинговими прикрасами.

І, нарешті, сучасна технологія розпізнавання відбитків пальців використовується для реалізації різних благородних цілей, таких як забезпечення чесних виборів у країнах, де відсутні універсальні ідентифікаційні карти, ефективне та справедливе розподілення допомоги у найбідніших регіонах світу або просто зниження рівня злочинності та шахрайства, незалежно від місця їх

використання.

Власне перейдемо до безпосередньої роботи алгоритму. SourceAFIS, переважно, зосереджений на аналізі відбитків пальців. Зрозуміло, що SourceAFIS не є людиною, тому як він може щось розуміти? Для алгоритмів аналізу даних розуміння означає використання високорівневих абстракцій для їх опису. У випадку з даним алгоритмом ці високорівневі абстракції представлені дрібницями або відгалуженнями ліній. Дрібниці або мінуції - це просто точки на зображенні з відповідним кутом напрямку.

Рисунок 1.6 - На зображенні відбитка пальця знайдено дрібниці. [3]

Фактично, це відображено у шаблоні. Багато дрібних абстракцій відбувається в процесі переходу від зображення відбитка пальця до формування

списку деталей (шаблону), але деталі цього процесу ми обговоримо пізніше.

Після формування дрібниць наступним кроком абстракції є створення країв. Край - це лінія, яка з'єднує дві дрібниці. Край має визначену довжину та два кути, які спадають від його складових деталей. Кути краю виражаються у відношенні до самого краю. Ці три властивості краю (довжина та два відносних кути) залишаються незмінними при переміщенні або обертанні краю, і це саме те, що необхідно для подальшого порівняння.

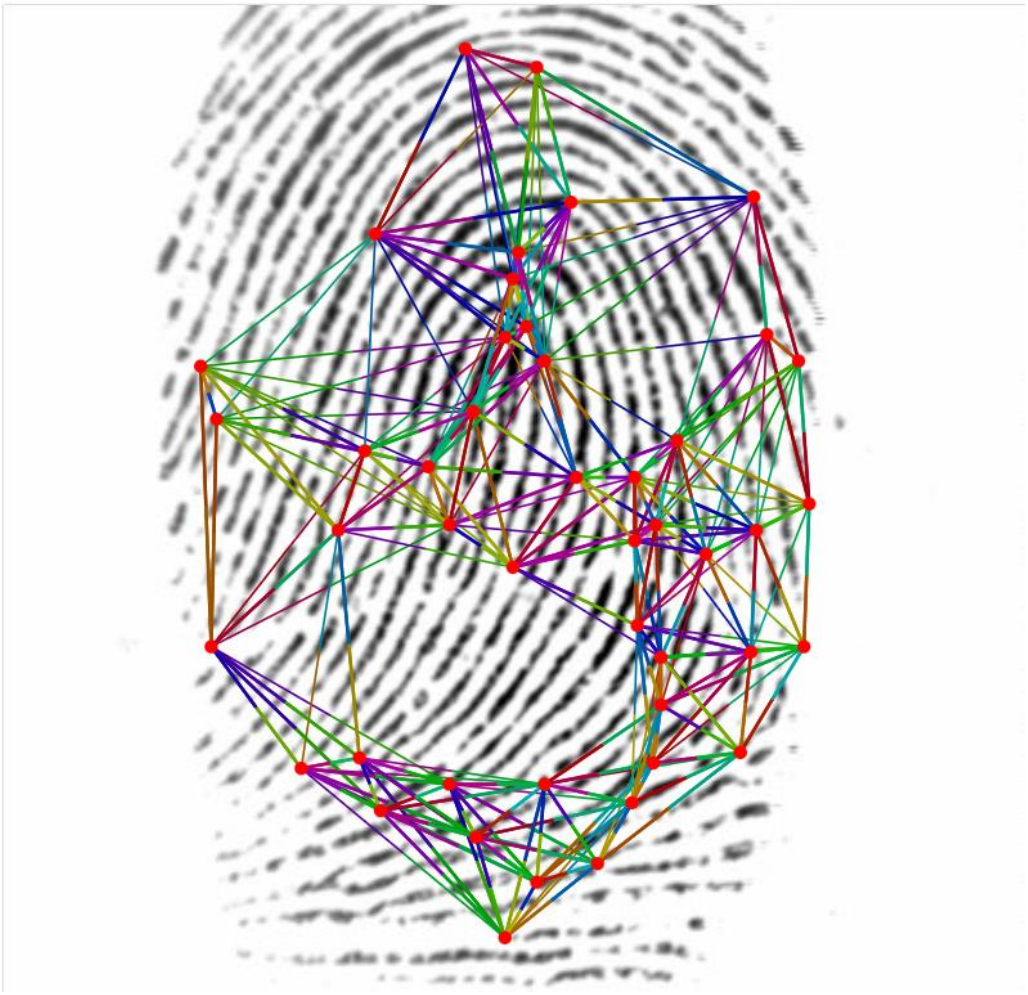

Рисунок 1.7 - Колір визначається довжиною краю та кутами. Подібні ребра мають схожі кольори. [3]

Після цього SourceAFIS намагається знайти щонайменше один спільний

край між двома зіставлюваними відбитками пальців. Цей процес відбувається швидко завдяки використанню алгоритму найближчого сусіда, який має продуктивність, подібну до хеш-таблиці. Це дозволяє нам отримати початкову пару деталей, що збігаються, одну з кожного відбитка пальця.

Починаючи з цієї початкової пари, SourceAFIS просувається вздовж країв у зовнішніх областях та формує пару, яка складається з кількох відповідних деталей та відповідних країв.

Рисунок 1.8 - Корінні дрібниці сині. Парне дерево зелене. Графік опорних країв жовтий. [3]

Після цього SourceAFIS уважно розглядає пару і приймає рішення, чи вона представляє собою справжній збіг, чи це просто випадкове співпадіння. Звичайно, можливо, що все є випадковим, але розрізнення між слабким і сильним збігом полягає в тому, що сильний збіг малоймовірний для випадковості.

На цьому етапі SourceAFIS запускає оцінювання, останню частину алгоритму. Основна ідея полягає в тому, що кожна сполучена дрібниця або грань

є подією, яка малоймовірно відбудеться випадково. Чим більше таких малоймовірних подій, тим менше ймовірність того, що поєднання є лише випадковим збігом. Таким чином, алгоритм фактично рахує різні функції збігу та оцінює, наскільки вони збігаються. Кінцева сума часткових балів формується так, щоб вона відповідала певній логічній шкалі, і повертається з алгоритму. Програма порівнює цю оцінку з певним пороговим значенням, щоб вирішити, чи є збіг чи ні.

2 РЕАЛІЗАЦІЯ АЛГОРИТМУ ЦИЛІНДР КОДІВ

2.1 Технічні відомості

Реалізація математичний алгоритму верифікації відбитків пальців на основі циліндр-кодів з розробкою методу порівняння циліндрів та шаблонів було виконано та протестовано виористовуючі наступні технічні характеристики комп'ютера:

- Оперативна пам'ять: 16 Гб DDR4;
- Процесор: AMD® Ryzen 7 5700u with radeon graphics × 16;
- Постійна пам'ять:1 Тр M.2;
- Операційна система: Linux, дистрибутив Ubuntu 22.04.
- Середовище розробки: Java 11.

2.2 Використання методу циліндр-кодів

Для реалізації методу верифікації відбитків пальців було взято одне з готових рішень розроблено у ході дослідження наведеної вище(у п.1.4) статі. Дана реалізація була виконана на мові програмування C#, але переведена згодом на Cuda, яка є у відкритому доступі. У даній роботі буде використовуватись порт реалізації з Cuda на Java. В оригінальниій версії не було підтримки завантаження зображень, через що потрібен був текстовий файл із певними даними про відбиток пальця. У реалізації на Cuda це вже було додано.

Таким чином, маючи два подібних файли з інформацією про два відбитки пальців за допомогою готового рішення можна отримати так званий результуючий рахунок подібності двох відбитків пальців. Також важливо зауважити, що у порівнянні беруть участь файли налаштувань, які надаються розробниками. Зокрема розробники надають чотири види налаштувань

MccPaper, IcarCV2010, MccSdk1.4Optimal та PMccPaper. З цих файлів більш за все цікавить файл MccSdk1.4Optimal через те, що він був налаштований та відкалібрований на базах даних, що використовувались на змаганнях із верифікації відбитків пальців у різних роках, а саме: FVC2000, FVC2002, FVC2004 таFVC2006. В даній реалізації буде використовуватись саме він, і у користувача не буде можливості змінити його. На рисунку Рисунку 2.1 та Рисунку 2.2 можна бачити графічний інтерфейс програми та порівняння спочатку однакових шаблонів відбитку пальців, а потім різних.

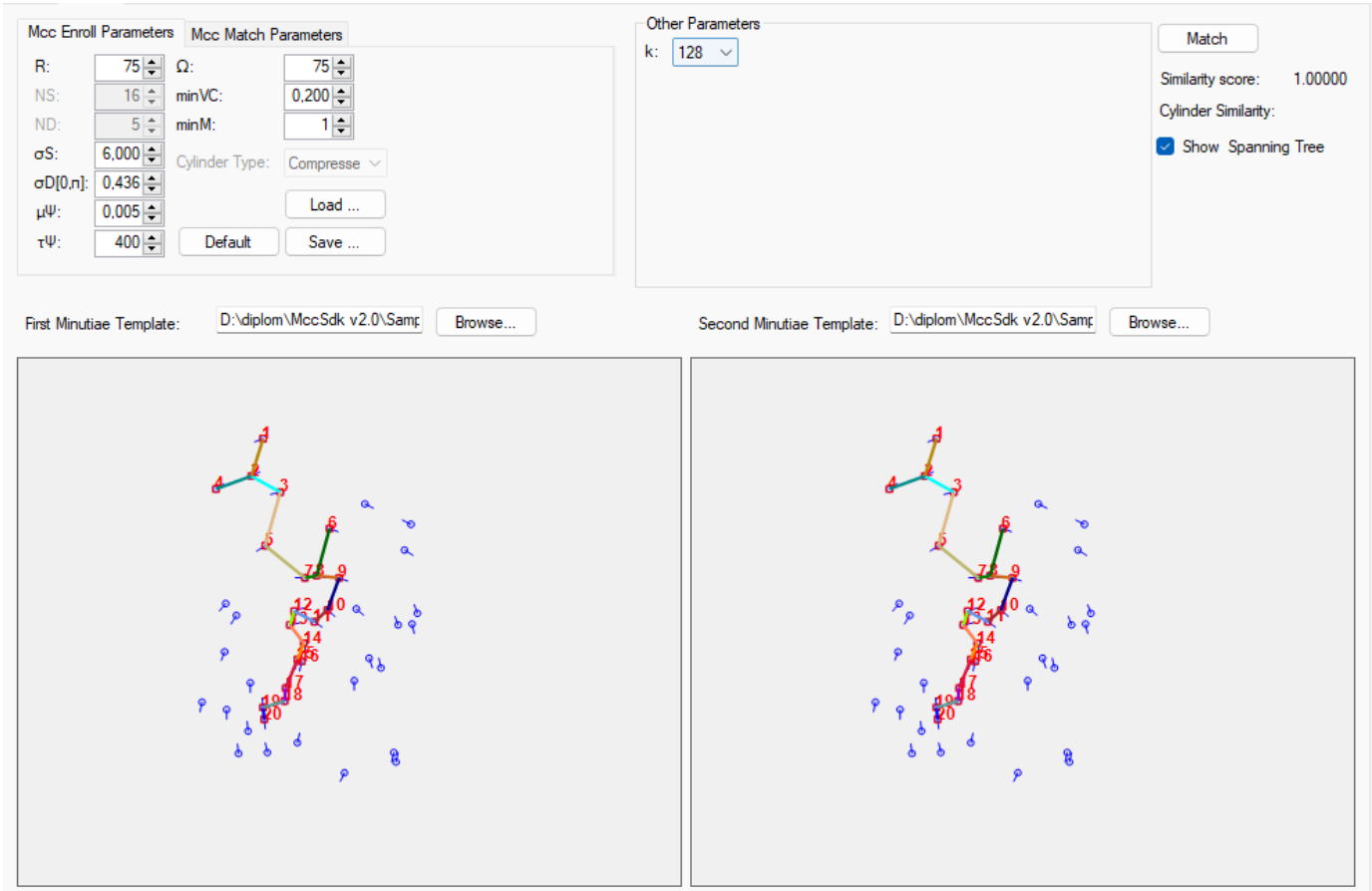

Рисунок 2.1 – Результат роботи програми коли на вхід подається два однакових шаблони відбитків пальців.

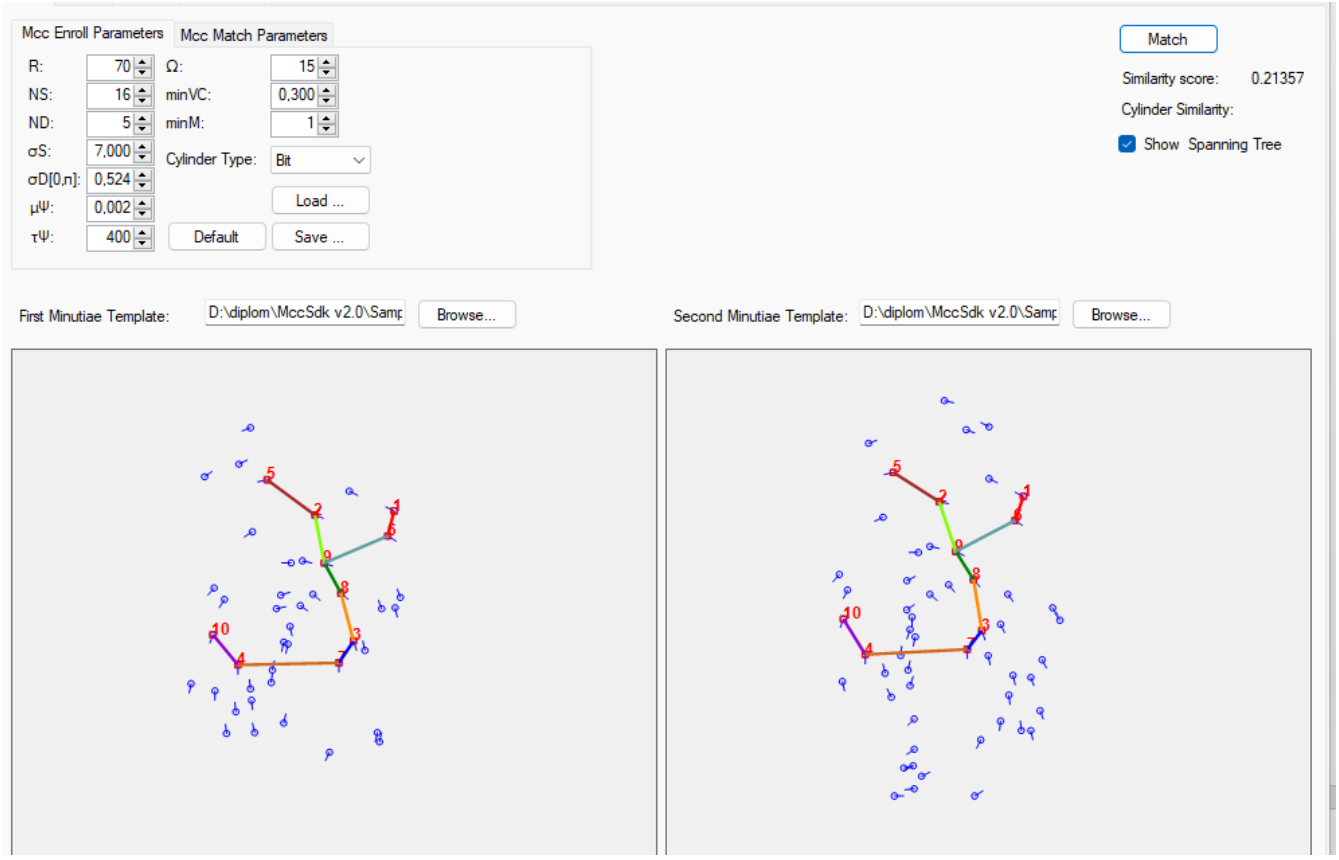

Рисунок 2.2 – Результат роботи програми коли на вхід подається два різних шаблони відбитків пальців.

Як можна бачити із результату результуючий рахунок добре показує чи є два зображення відбитків пальців подібними чи ні.

Після отримання розуміння, що даний алгоритм працює правильно потрібно взяти базу даних мінуцій для використання у дослідженні, а також скорегувати програмний код таким чином, щоб у порівнянні брали участь не дві мінуції із принципом one-to-one, а ціла база даних із принципом one-to-many. Також у програмне забезпечення потрібно додати так звані контрольні точки для виміру часу роботи алгоритму, а також використання пам’яті.

У якості бази даних буде використано наявну базу даних із 140 відбитками пальців по 12 екземплярів.

2.3 Програмна реалізація

Щодо програмної реалізації, то як і було сказано вище перш за все потрібно було запустити реалізацію у форматі one-to-one, що порівнює два зображення та видає результуючий рахунок. Перш за все у програмному коді потрібно було змінити те, яким чином завантажуються файли, а також переробити алгоритм для порівняння не з одним об’єктом, а з багатьма.

Важливо зазначити, що результуючі рахунки не базуються на діпазоні від 0 до 1, а значно розширили свій діапазон, таким чином даючі більш точно гарантію того, чи належать відбитки пальця одній і тій же людині.

Завантаження файлів було не надто зручним через те, що в готовій реалізації їх отримували через командний рядок у вигляді аргументів, що є гарною практикою для динамічного використання, враховуючи те, що шлях до файлів у всіх різний, але набагато зручніше вказувати шлях до об’єктів безпосередньо в коді програми. Таким чином можна уникнути зайвих проблем із необачністю, шляхом того що шлях до файлу вказується один раз, а не постійно перед кожним запуском програми.

У Додатку А можна бачити яким чином це було зроблено, а також на рисунку отримано коректні результати роботи даного алгоритму. Таким чином, на даному етапі все працює правильно.

Після цього потрібно перетворити програмне забезпечення у формат one-to-many та додати показники часу. Перевірити коректність буде можливо запустивши код для певної кількості записів і переглянути їх результати у цих двох програмах. Це було зроблено і відхилень виявлено не було. Після додавання таймерів часу результат роботи програми виглядав наступним чином (Рисунок 2.3).

Таким самим чином було додано вимірювання часу, на чому і було завершено розширення реалізації алгоритму циліндр-кодів.

```
File = 8_10, Score = 8.94677734375
File = 8_11, Score = 9.04052734375
File = 8_12, Score = 9.88818359375
File = 9_1, Score = 89.44700924134528
File = 9_2, Score = 71.57748212269621
File = 9_3, Score = 47.389965998695345
File = 9_4, Score = 68.77228855355752
File = 9_5, Score = 72.62842595270305
File = 9_6, Score = 71.82120638709549
File = 9_7, Score = 87.9010780583708
File = 9_8, Score = 90.7442841521368
File = 9_9, Score = 100.0
File = 9_10, Score = 79.06261785420428
File = 9_11, Score = 89.72078866443398
File = 9_12, Score = 50.01702080238864
File = 10_1, Score = 8.9345703125
File = 10_2, Score = 18.85009765625
File = 10_3, Score = 9.98681640625
File = 10_4, Score = 19.41455078125
File = 10_5, Score = 9.25634765625
File = 10_6, Score = 18.87255859375
File = 10_7, Score = 9.33154296875
File = 10_8, Score = 20.248046875
```

Рисунок 2.3 - Результат роботи one-to-many.

3 РЕАЛІЗАЦІЯ МЕТОДУ ШАБЛОНІВ

3.1 Технічні відомості

Реалізація математичний алгоритму верифікації відбитків пальців на основі циліндр-кодів з розробкою методу порівняння циліндрів та шаблонів було виконано та протестовано виористовуючі наступні технічні характеристики комп'ютера:

- Оперативна пам'ять: 16 Гб DDR4;
- Процесор: AMD® Ryzen 7 5700u with radeon graphics × 16;
- Постійна пам'ять:1 Тр M.2;
- Операційна система: Linux, дистрибутив Ubuntu 22.04.
- Середовище розробки: Java 11.

3.2 Бібліотека SourceAFIS

Під час вибору конкретної реалізації методу шаблонів потрібно було звернути увагу на те, що для максимальної точності експерименту потрібно вибрати реалізацію, що використовувала б те саме середовище, що і метод циліндр-кодів. Таким чином можна більш точно оцінити ефективність алгоритмів, а отже мова програмування залишилась тою самою, і це Java.

Також окрім мови програмування було залишено ту саме версію, складову комп'ютера, тощо. Отже можна сказати, що ефективність та час виконання варіюється лише від використання конкретного методу.

У якості програмного коду було обрано готове рішення SourceAFIS, яке було теж портоване із C#. Було обрано бібліотеку SourceAFIS як готове рішення, оскільки вона володіє кількома ключовими перевагами.

По-перше, SourceAFIS є ефективним рішенням, що гарантує швидку та

точну верифікацію великих обсягів даних. Це досягається завдяки використанню оптимізованих алгоритмів із зменшеним часом виконання. Зокрема, SourceAFIS пропонує швидку обробку зображень відбитків пальців, що є важливим критерієм для реального часу застосувань.

По-друге, бібліотека SourceAFIS відома своєю високою точністю та надійністю. Вона застосовує продумані алгоритми порівняння відбитків пальців, що забезпечують мінімальну кількість помилок і високу швидкість виявлення співпадінь. Завдяки цьому, SourceAFIS виявляється надійним інструментом для автентифікації особи за її відбитком пальця.

По-третє, SourceAFIS має гнучку архітектуру, що дозволяє легко інтегрувати її з різноманітними платформами і системами. Це особливо корисно для розробників, які прагнуть використовувати бібліотеку у своїх проектах. Вона підтримує різні мови програмування, що забезпечує його універсальність і зручність в застосуванні.

Крім того, варто відзначити, що SourceAFIS має відкритий вихідний код, що дає можливість розширювати та вдосконалювати його функціонал за потребами конкретного проекту. Це сприяє активному співробітництву й взаємодії зі спільнотою розробників, що сприяє швидкому вирішенню проблем та вдосконаленню функціональних можливостей бібліотеки.

Отже, враховуючи ефективність, точність, гнучкість архітектури та відкритий вихідний код, бібліотека SourceAFIS заслуговує на використання як готове рішення для верифікації особи за відбитком пальця. Вона може бути ідеальним вибором для розробників, які прагнуть впровадити потужну та надійну систему біометричної автентифікації.

3.2 Опис основних можливостей бібліотеки SourceAFIS

Бібліотека SourceAFIS є потужним рішенням для верифікації особи за відбитком пальця. Вона надає широкі можливості та інструменти для реалізації

систем біометричної автентифікації з використанням методу шаблонів.

Перш за все, SourceAFIS забезпечує зручний інтерфейс для витягування шаблонів відбитків пальців. Він дозволяє аналізувати вхідні зображення пальців та витягувати ключові характеристики, такі як мінімальні оптимальні особливості. Ці характеристики представляються у вигляді шаблону, який містить інформацію про положення та структуру відбитку пальця.

Також дана бібліотека надає зручні механізми для збереження та ідентифікації шаблонів відбитків. За допомогою алгоритмів порівняння шаблонів, бібліотека може встановити ступінь співпадіння між вхідним відбитком пальця та шаблонами, які зберігаються у базі даних. Це дозволяє проводити верифікацію особи шляхом порівняння відбитка пальця з відомими шаблонами.

Більш того, SourceAFIS пропонує механізми для пошуку співпадінь у великих базах даних. Вона здатна ефективно шукати відповідності між вхідним відбитком та шаблонами різних осіб, що дозволяє визначати, чи належить вхідний відбиток до відомої особи. Це важливо для систем автентифікації та ідентифікації.

Важливо зазначити, що дане рішення відоме своєю високою швидкістю та точністю. Воно дозволяє ефективно виконувати алгоритми порівняння шаблонів та верифікації в реальному часі. Крім того, бібліотека має оптимізації, які забезпечують швидкість обробки великих обсягів даних як зображень, так і подібності двох відбитків пальців.

Усі ці функціональні можливості SourceAFIS роблять її привабливим вибором для реалізації систем верифікації особи за відбитком пальця. Вона надає зручні інструменти для витягування, збереження, ідентифікації та пошуку співпадінь шаблонів пальцевих відбитків.

3.3 Реалізація програмного коду із використанням SourceAFIS

Дана бібліотека надає готове рішення, через що основною роботою буде завантажити базу даних зображень відбитків пальців, а також отримання часу виконання верифікації. Результатом буде виведення в консоль порівняння 1 до багатьох шаблонів, а також час роботи програми. У час роботи входить виключно робота верифікації, так як завантаження зображень займає надто багато часу.

Перш за все, для початку роботи з SourceAFIS було імпортовано відповідні пакети та класи. Далі, для створення екземпляру даного алгоритму, було ініціалізовано його об'єкт. Це дозволяє нам налаштувати параметри алгоритму та виконати розпізнавання. Для цього було створено новий об'єкт класу "FingerprintMatcher" та передано необхідні параметри для його правильної роботи у майбутньому.

Після ініціалізації алгоритму можна почати обробку відбитків пальців. Для цього потрібно отримати вхідні дані, такі як зображення відбитків пальців або деякі характеристики цих відбитків. У даному випадку використовується база відбитків пальців, та сама що використовувались і для методу циліндр-кодів.

Після отримання вхідних даних, можна передати їх до алгоритму SourceAFIS для обробки. Алгоритм автоматично визначає характеристики відбитків пальців, такі як положення точок вхідних даних, орієнтація, довжина та інші деталі. За допомогою цих характеристик алгоритм генерує унікальний шаблон для кожного відбитка пальця.

Отриманий шаблон може бути збережений у пам'яті для подальшого використання або порівняння з іншими шаблонами. SourceAFIS надає можливість порівняти два відбитки пальця та визначити ступінь їх схожості. Це дозволяє вирішувати завдання ідентифікації осіб на основі відбитків пальців.

Нижче на Рисунку 3.3 можна бачити результат виконання для порівняння 1:1. У якості заміру часу було використано наносекундах, через те, що комп’ютер не міг зафіксувати час у мілісекундах.

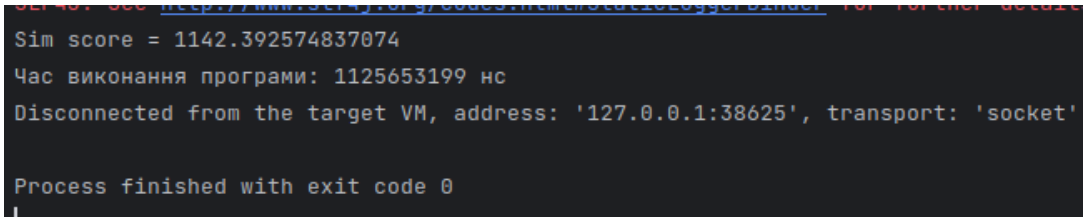

Рисунок 3.3 - Результат порівняння 1:1.

На Рисунку 3.4 можна бачити результат у мілісекундах у вигляді порівняння вхідного зображення відбитку пальця до бази даних, а також назву файла та айді відбитку пальця який співпав.

Рисунок 3.4 - Результат порівняння 1:N.

Також на рисунку 3.5 можна бачити результуючі рахунки, де рахунок який збігається із відбитком пальця має рахунок більше 40, що є пороговим значенням для того, щоб вважати що відбитки збігаються, в той час як інші мають кардинально менший рахунок. Важливо зауважити, що усі відбитки із початковою цифрою 9 належать одній людині, через це можна бачити, що алгоритм чітко може зрозуміти чи належить відбиток пальця людині, чи ні.

```
Файл = 8_7.bmp, результат: 1.2523143457036034
Файл = 8_8.bmp, результат: 0.0
Файл = 8_9.bmp, результат: 0.6746462910541557
Файл = 8_10.bmp, результат: 0.0
Файл = 8_11.bmp, результат: 1.4008491972363177
Файл = 8_12.bmp, результат: 0.0
Файл = 9_1.bmp, результат: 55.216504981564505
Файл = 9_2.bmp, результат: 85.10827581640996
Файл = 9_3.bmp, результат: 64.7903519740155
Файл = 9_4.bmp, результат: 50.79245111634195
Файл = 9_5.bmp, результат: 61.14475537439638
Файл = 9_6.bmp, результат: 87.07159441661261
Файл = 9_7.bmp, результат: 51.938592927483285
Файл = 9_8.bmp, результат: 48.0702806898522
Файл = 9_9.bmp, результат: 793.225207179351
Файл = 9_10.bmp, результат: 42.05274475494541
Файл = 9_11.bmp, результат: 59.35786122968205
Файл = 9_12.bmp, результат: 83.4067321536333
Файл = 10_1.bmp, результат: 1.5660983296536009
Файл = 10_2.bmp, результат: 4.173319911857031
Файл = 10_3.bmp, результат: 0.3211149488093657
Файл = 10_4.bmp, результат: 2.8803056916091463
Файл = 10_5.bmp, результат: 0.0
Файл = 10_6.bmp, результат: 2.0418484038811107
Файл = 10_7.bmp, результат: 0.8266524719967788
Файл = 10_8.bmp, результат: 2.668459103961945
Файл = 10_9.bmp, результат: 2.7578750669162666
Файл = 10_10.bmp, результат: 0.0
Файл = 10_11.bmp, результат: 0.0
Файл = 10_12.bmp, результат: 1.6235596284251455
Файл = 11_1.bmp, результат: 0.0
Файл = 11_2.bmp, результат: 0.03330671595810657
Файл = 11_3.bmp, результат: 0.0
Файл = 11_4.bmp, результат: 0.0
Файл = 11_5.bmp, результат: 0.0
```

Рисунок 3.5 - Результуючі рахунки алгоритму.

4 РЕАЛІЗАЦІЯ МЕТОДУ ПОРІВНЯННЯ МЕТОДУ ЦИЛІНДР-КОДІВ ТА МЕТОДУ ШАБЛОНІВ

4.1 Реалізація методу порівняння

Основна робота щодо реалізації методу порівняння методу циліндр-коду та методу шаблонів була вже проведена у попередніх розділах. Основними методами порівняння будуть наступні критерії:

- Витрачений час на алгоритм без обробки зображень;
- Витрачений час на все (обробка зображень та алгоритм);
- Використана пам’ять без обробки зображень;
- Використана пам’ять на все (обробка зображень та алгоритм);

У якості експерементальних баз даних буде використано 4 бази даних: 420, 840, 1260 та 1680 елементів. Кількість елементів зростає для отримання результату залежності від кількості елементів. Такі розміри обрані не випадково, усього в наявній базі даних є 140 відбитків пальців людей, де кожна людина надала 12 екземплярів відбитків. Таким чином, кожна база даних буде містити по 3, 6, 9 та 12 екземплярів відповідно.

Сама реалізація буде містити наступне: меню вибору за яким критерієм буде порівняння, після чого буде вибір кількості об’єктів бази даних. На виході в консоль програма буде виводити результати у часі, об’ємі пам’яті чи кількості виконаних операцій відповідно до кожного з алгоритмів.

На Рисунку 4.1 можна бачити вигляд меню із пунктами вибору подальшої роботи програми. У разі користувачем ввести хибний пункт меню програма повідомить користувача про це і попросить ввести коректний пункт меню, як це показано на Рисунку 4.2.

```
----------Menu---------
1.Speed test(Only algorithm comparing)
2.Speed test(Algorithm comparing + loading pictures to template)
3.Memory test(Only algorithm comparing)
4.Memory test(Algorithm comparing + loading pictures to template)
5.Exit
```

Рисунок 4.1 - Меню програми.

```
7
Your choice is incorrect. Please try again.
----------Menu---------
1.Speed test(Only algorithm comparing)
2.Speed test(Algorithm comparing + loading pictures to template)
3.Memory test(Only algorithm comparing)
4.Memory test(Algorithm comparing + loading pictures to template)
5.Exit
```

Рисунок 4.2 - Хибний вибір і повторний виклик меню.

Наступною частиною програми є вибір кількості відбитків пальців для порівняння. Даний пункт виконаний теж у вигляді меню (Рисунок 4.3) для зручності користувача із заздалегідь обраною кількістю відбитків у якості 25%, 50%, 75% та 100% від усієї бази даних.

```
2
---------- Amount Of Fingerprints Menu---------
1.420
2.840
3.1260
4.1680
```

Рисунок 4.3 - Меню вибору кількості відбитків пальців для порівняння.

Після вибору кількості відбитків пальців для порівняння користувачу надається можливість вибору людини, відбиток пальця якої буде обрано у якості того, що буде порівнюватись з усіма іншими (Рисунок 4.4). Важливо зауважити, що в базі даних є 12 відбитків пальців кожного з людей, але конкретний відбиток програма вибирає сама, задля більш чесного випробування. Це можна назвати імітацією реального зняття відбитку пальця, коли програма не знає завчасно який саме відбиток буде обрано і може попастись як дуже чіткий і якісний, так і той що матиме не типічне положення відбитку на зображенні, мати якісь дефекти, тощо.

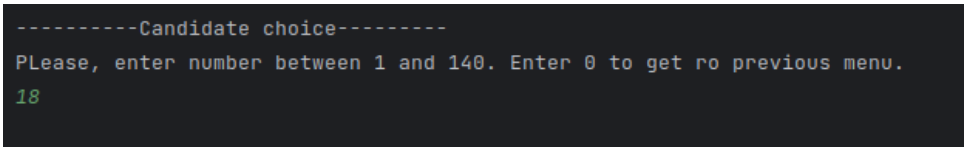

Рисунок 4.4 - Меню вибору людини для порівняння.

Після цього програма виконає почерговий виклик двох алгоритмів, де враховуючі усі вхідні параметри, буде отримано результат для кожного з них. Програма виведе рахунок порівняння до кожного відбитку, який порівнюється в обох алгоритмах, щоб таким чином можна було перевірити працездатність програми. Отримані результати виглядають як на Рисунку 4.5 та Рисунку 4.6.

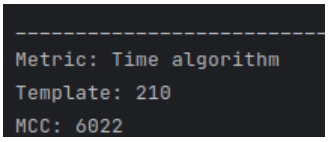

Рисунок 4.5 - Результати роботу алгоритму по часу.

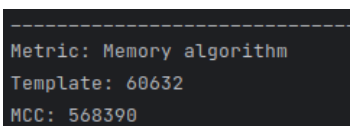

Рисунок 4.6 - Результат роботи за витраченою пам’яттю.

Таким чином буде проведено експеримент та згодом встановлено ті чи інші залежності, а також результати експерименту.

4.2 Результати експерименту

Результатами експерименту будуть відповідно складені таблиці та графіки, що показують результати роботи методу порівняння методів циліндр-коду та шаблонів. Загалом буде складено 4 таблиці відповідно до 4 вимірюємих метрик, де на кожну кількість відбитків пальців у базі даних буде вказано час роботи алгоритму у мілісекундах або використаний об’єм пам’яті в байтах.

Отже першою із таких таблиць буде використання алгоритмів виключно під час підрахунків, тобто усі дії із зображеннями не будуть враховуватись. Очікується, що метод циліндрів буде працювати повільніше, так як основна робота в даному методі припадає на розрахунок та створення циліндрів, в той час як в методі шаблонів основна робота іде саме на побудову шаблонів. Усі підрахунки за часом визначаються у мілісекундах.

Таблиця 4.1 - Результати роботи алгоритму без обробки зображень підрахунку за часом.

Кі-ть зразків / Метод	420	840	1260	1680
Шаблонів	1300 мс	1651 мс	2362 мс	4427 мс
Циліндр-кодів	1650 мс	2968 мс	4028 мс	5334 мс

Як і було зазначено вище, метод циліндрів працює повільніше під час розрахунків без обробки зображень, що випливає з результатів Таблиці 4.1, хоча і варто зазначити що були помітні деякі помилки у обчисленні при використанні методу шаблонів.

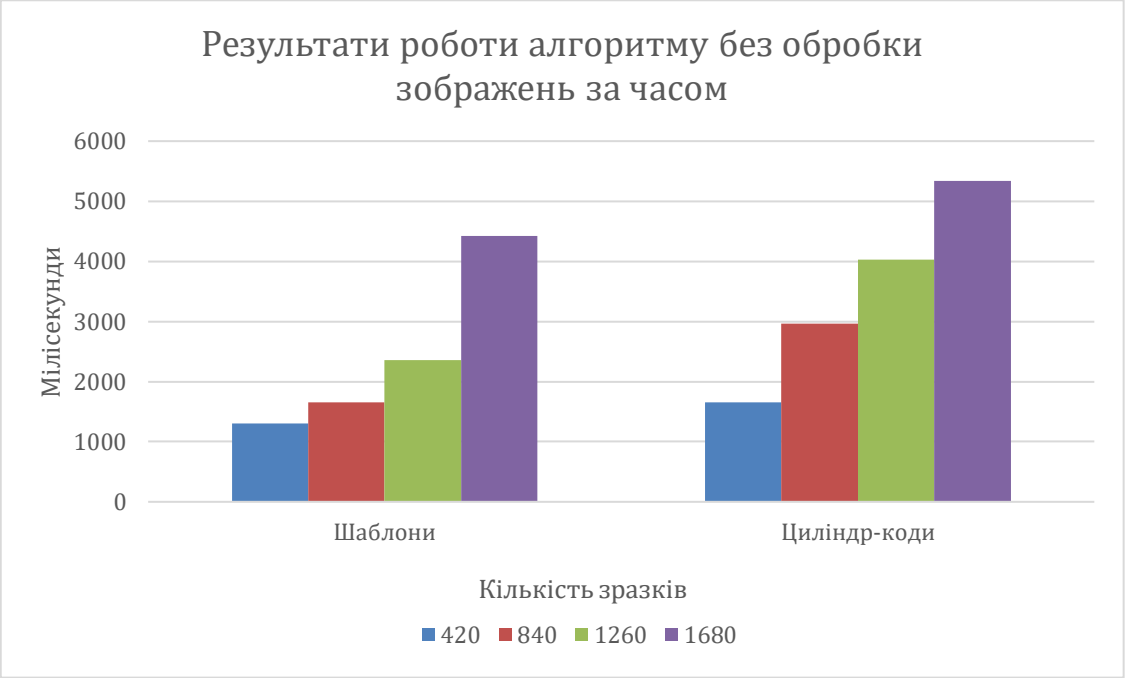

Рисунок 4.7 – Результати роботи алгоритму без обробки зображень за часом.

Таблиця 4.2 - Результати роботи алгоритму з обробкою зображень за часом.

Кі-ть зразків / Метод	420	840	1260	1680
Шаблони	29692 мс	56044 мс	82581 мс	100645 мс
Циліндр-коди	21943 мс	44653 мс	69263 мс	87833 мс

У Таблиці 4.2 та на Рисунку 4.8 можна бачити вже інші результати, метод шаблонів працює трохи довше, через довгі обчислення і роботу із зображеннями. Цей час включає в себе повне завантаження файлів з диску, тому можливе його покращення якщо зображення будуть міститись у вигляді шаблонів в базі даних. В такому випадку швидкість має бути збільшена.

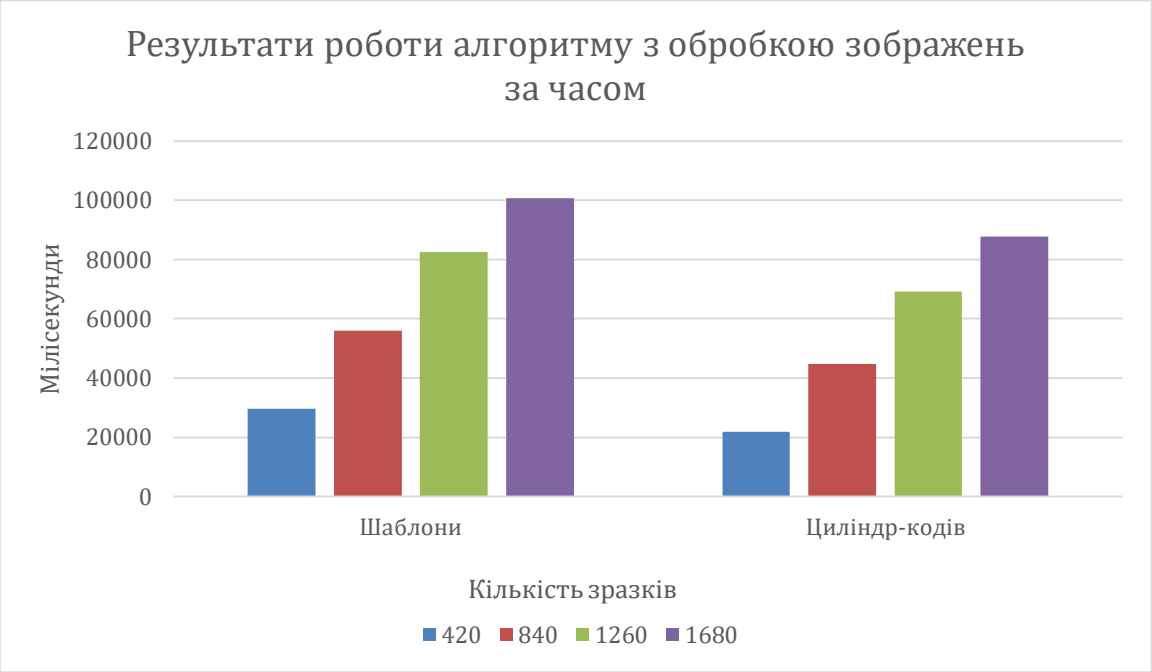

Рисунок 4.8 - Результати роботи алгоритму з обробкою зображень за часом.

Далі перейдемо до порівняння роботи алгоритмів по пам’яті.Усі вимірювання ведуться в байтах витраченої пам’яті. Підрахунки відбуваються наступним чином: перед початком роботи обчислення чи перед завантаженням зображень фіксується наявна витрата пам’яті комп’ютера, після чого комп’ютер не виконує жодних дій окрім як роботи програми. Після цього отримується пам’ять яка була використана в моменті і отримується результуюче використання пам’яті.

Таблиця 4.3 - Результати роботи алгоритму підрахунку за пам’яттю.

Кі-ть зразків / Метод	420	840	1260	1680
Шаблони	22018481 б	23715320 б	24021096 б	25291916 б
Циліндр-коди	30246163 б	32249949 б	34249121 б	35218612 б

Також перед кожним заміром пам’яті викликаються так званий збірник сміття, для того щоб у підрахунках отримувалась лише дійсно затрачена пам’ять.

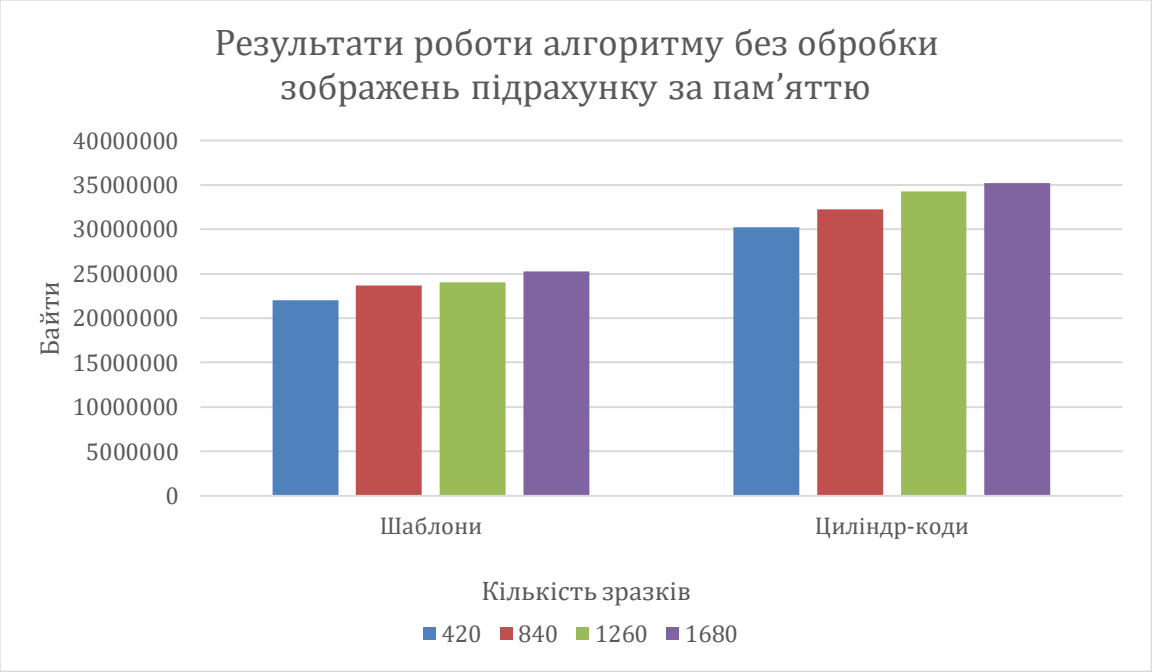

Рисунок 4.9 - Результати роботи алгоритму без обробки зображень підрахунку за пам'яттю.

Як можна бачити в Таблиці 4.3 та на Рисунку, під час обчислень метод шаблонів використовує менше пам'яті під час етапу роботи алгоритму обчислення.

Таблиця 4.4 - Результати роботи алгоритму з обробкою зображень підрахунку за пам'яттю.

Кі-ть зразків / Метод	420	840	1260	1680
Шаблони	8767880 б	13222496 б	19708912 б	26147056 б
Циліндр-коди	10465508 б	11619711 б	21540821 б	28462651 б

Виходячи з результатів кількість зображень на дуже впливає на витрачену пам'ять. Результати отримані у ході експерименту представлені в Таблиці 4.4 схожі на результати в Таблиці 4.3. Це обумовлено тим, що метод Циліндр-кодів

трохи важчий в обчисленні, аніж метод шаблонів.

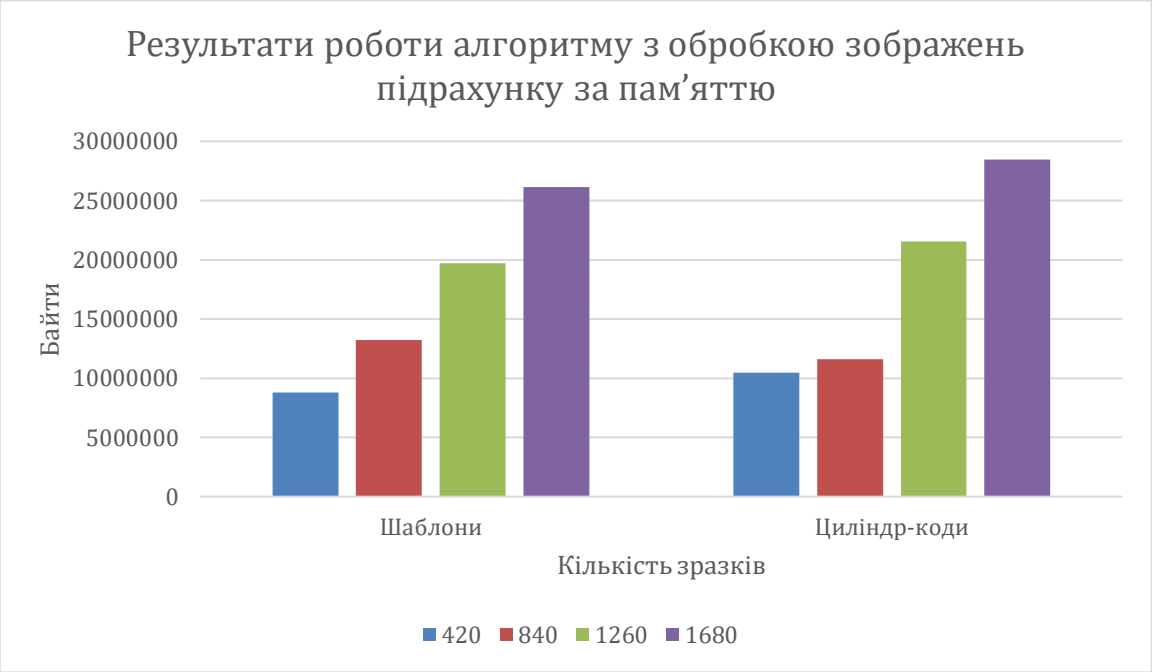

Рисунок 4.10 - Результати роботи алгоритму з обробкою зображень підрахунку за пам'яттю.

Під час ходу експеременту було виявлено, що іноді методи не можуть розпізнати, співставити та дати оцінку при порівнянні відбитків пальців. Через що буде сформаваний ще один критерій – ефективність розпізнавання. Це не є критерієм точності через те, що буде досліджено саме процент похибок у разі використання.

Експеремент полягатиме у тому, щоб запустити кожен з аплгоритмів 1 раз і порівняти відбиток пальця із кожним іншим в базі даних і таким чином отримати процент похибок при розпізнаванні зображення відбитку пальця. Отримати дані досить просто, через те, що у разі якщо алгоритм не розпізнає відбиток пальця, то результуючий рахунок буде дорівнювати 0 (Рисунок 4.11). Після чого потрібно

буде розділити загальну кількість відбитків пальців на кількість похибок та помножити 100 і це і буде шуканим результатом.

```
File = 116_8.bmp, score: 1.310158163677607
File = 116_9.bmp, score: 1.6642305743946415
File = 116_10.bmp, score: 0.30877572750642346
File = 116_11.bmp, score: 0.098818087201271
File = 116_12.bmp, score: 0.569625315598396
File = 117_1.bmp, score: 1.5051789983296247
File = 117_2.bmp, score: 0.0
File = 117_3.bmp, score: 3.742748795026377
File = 117_4.bmp, score: 4.930146113405683
```

Рисунок 4.11 – Похибка під час виконання алгоритму.

Таким чином буде встановлено лічильник для пі б драхунку таких подій. Під час роботи програми це буде виглядати як на Рисунку 4.12.

```
File = 140_11, Score = 0.0
File = 140_12, Score = 7.4775390625
Errors: 315 / 1680
```

Рисунок 4.12 – Результат лічильнику похибок.

Після запуску для усіх зображень із бази даних отримано результат наведений у Таблиці 4.5. Таким чином метод циліндр-кодів є надійнішим за метод шаблонів. Хоча і переглянувши базу даних було виявлено, що досить значна частина зображень які не були опрацьовані були створені таким чином, що їх було б важко опрацьовувати.

Таблиця 4.5 – Результати підрахунку похибок у алгоритмах.

Метрика / Метод	Кіл-ть похибок	% похибки
Шаблони	320	19,04%
Циліндр-коди	211	12,55%

Таким чином методу було порівняно у 5 категоріях: час роботи із обробкою зображень, час роботи без обробки зображень, витрачена пам’ять із оброкою зображень, витрачена пам’ять без обробки зображень та віргодність похибки кожного із алгоритмів. За кожну перемогу в катгеорії метод отримає 1 бал. Таким чином у порівнянні за часом буде по 1 балу кожному, в порівнянні пам’яті 2 бали отримує метод шаблонів, а в порівнянні похибок 1 бал отримують циліндр-коди.

Виходячи з цього можна сказати, що загалом кращим є метод шаблонів із рахунком 3-2. Обмувлюється це тим, що даний метод з’явився раніше, а отже і пройшов більше етапів оптимізацій та покращень ефективності. Хоча, враховуючи результати похибок важливо зауважити, що метод циліндр-кодів є трохи надійнішим у роботі із відбитками пальців не найкращої якості, та все ж на сьогоднішній день метод шаблонів є кращим.

ВИСНОВКИ

У ході дослідження було проведено аналіз існуючих алгоритмів, що реалізують методи циліндр-кодів та методи шаблоні та виявлено їх переваги та недоліки. Також було виявлено, що існують деякі обмеження щодо надійності та швидкості.

На основі проведеного аналізу було взято математичні алгоритми, які реалізують метод циліндр-кодів та метод шаблонів для верифікації відибтку пальців за наявним зображенням, порівнюючи їх з базою даних.

Для порівння даних методів було розроблено новий метод, який використовує обидва методи для порівняння одне з одним за такими показниками як час, витрата пам’яті та вірогідність похибки. При реалізації методу порівняння було додано потрібні вбудовані методи системи для отримання даних щодо кожного із показників.

У якості результатів дослідження було представлено таблиці та графіки порівняння обох методів у різних показниках. Також було зроблено висновки щодо кожного з результатів дослідження.

Отже можна зробити висновок, що розроблений метод надає можлифість ефективно провести аналіз порівняння методів циліндр-коду та шаблонів. Його використання може ефективно допомогти досліджувати тему верифікування людини за відбитком пальця, а також покращити існуючі алгоритми для більш стабільної та ефективної роботи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ferrara M., Capelli R. “Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition”, IEEE Transactions on Pattern Analysis and Machine Intelligence, December 2010. URL: https://www.researchgate.net/publication/47544547_Minutia_Cylinder-Code_A_New_Representation_and_Matching_Technique_for_Fingerprint_Recognition (дата звернення 05.05.2023)
2. ISO/IEC 19794-2:2005 Information technology — Biometric data interchange formats – Part 2: Finger minutiae data. (дата звернення 11.05.2023)
3. Robert Važan “Source AFIS Fingerprint recognition library for .NET and experimentally for Java”, URL: https://sourceafis.machinezoo.com/algorithm (дата звернення 12.05.2023)
4. Neskovic M., Nikolic M. “Fingerprint Matching Algorithm” URL: https://patentimages.storage.googleapis.com/d4/cd/19/c619090f06fe78/WO2014068089A1.pdf (дата звернення 03.05.2023)
5. Sangita K. C. An algorithm for fingerprint enhancement & matching. International Journal of Engineering Research and Applications ISSN: 2248-9622 URL:http://www.ijera.com/special_issue/VNCET_Mar_2012/60.pdf (дата звернення 15.05.2023)
6. Maltoni D., Maio D., Jain,A., Prabhakar S. “Handbook of Fingerprint Recognition”. URL: https://link.springer.com/book/10.1007/978-1-84882-254-2 (дата звернення 12.05.2023)
7. Hong L., Wan Y., and Jain A. “Fingerprint Image Enhancement”. URL: http://www.math.tau.ac.il/~turkel/imagepapers/fingerprint.pdf (дата звернення 20.05.2023)

8. Maltoni D. A “Tutorial on Fingerprint Recognition” URL: https://cedar.buffalo.edu/~govind/CSE666/fall2007/FP_Tutorial.pdf (дата звернення 21.05.2023)
9. Гаспарян А.В., Киракосян А.А. система сравнения отпечатков пальцев по локальным признакам / Вестник РАУ. Серия физико-математические и естественные науки, 2, 2006. – С. 85-91. (дата звернення 22.05.2023)
10. Bazen A.M. Fingerprint Identification – Feature Extraction, Matching, and Database Search, Ph.D. Dissertation. Univ. of Twente, Enschede, The Netherlands, 2002. С. 187. (дата звернення 22.05.2023)
11. ISO/IEC 19795–1:2006 Information technology – Biometric performance testing and reporting – Part 1: Principles and framework. (дата звернення 15.05.2023)

Додаток А

```
package template;

import com.machinezoo.sourceafis.FingerprintImage;
import com.machinezoo.sourceafis.FingerprintMatcher;
import com.machinezoo.sourceafis.FingerprintTemplate;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.extern.slf4j.Slf4j;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

@Slf4j
public class TemplateMatcher {
 private static final double THRESHOLD = 40; // Поріг схожості для визначення співпадіння

 public static void main(String[] args) throws IOException {

 var candidate = new FingerprintTemplate(
 new FingerprintImage(Files.readAllBytes(Paths.get("/home/misha/IdeaProjects/MCC
java/src/main/java/template/candidate.bmp"))));
 var matcher = new FingerprintMatcher(candidate);
 double similarity = matcher.match(candidate);
 System.out.println("Sim score = " + similarity);

 List<Subject> candidates = loadDB();
 if (candidates != null) {
 var subject = identify(candidate, candidates);
 System.out.println("Subject: id = " + subject.getId() + ", name = " + subject.getName());
 }

 }
```

```
@Data
@AllArgsConstructor
private static class Subject {
private int id;
private String name;
private FingerprintTemplate template;
}

private static List<Subject> loadDB() throws IOException {
List<Subject> candidates = new ArrayList<>();
String folderPath = "/home/misha/IdeaProjects/MCC java/src/main/java/template/candidates";
File folder = new File(folderPath);
File[] files = folder.listFiles();
Arrays.sort(files, new Comparator<>() {
@Override
public int compare(File file1, File file2) {
int[] number1 = extractNumber(file1.getName());
int[] number2 = extractNumber(file2.getName());

int cmp = Integer.compare(number1[0], number2[0]);
if (cmp == 0) {
return Integer.compare(number1[1], number2[1]);
}
return cmp;
}

private int[] extractNumber(String name) {
int firstIndex = name.indexOf('_');
String number1 = name.substring(0, firstIndex);
String number2 = name.substring(firstIndex + 1, name.lastIndexOf('.'));
return new int[]{Integer.parseInt(number1), Integer.parseInt(number2)};
}
});
int i = -1;
for (File file : files) {
i++;
if (file.isFile()) {
String filePath = file.getAbsolutePath();
String name = file.getName();
candidates.add(new Subject(i, name, new FingerprintTemplate(
new FingerprintImage(Files.readAllBytes(Paths.get(filePath))))));
}
}
return candidates;
```

```
}

private static Subject identify(FingerprintTemplate probe, Iterable<Subject> candidates) {
var matcher = new FingerprintMatcher(probe);
Subject match = null;
double max = Double.NEGATIVE_INFINITY;
for (var candidate : candidates) {
double similarity = matcher.match(candidate.getTemplate());
if (similarity > max) {
max = similarity;
match = candidate;
}
}

return max >= THRESHOLD ? match : null;
}
}
```

Додаток Б

```
package mcc;

import java.util.List;
import java.util.ArrayList;
import java.io.File;

public class Main {
 public static void main(String[] args) {
 args = new String[4];
 args[1] = "many";
 args[2] = "/home/misha/IdeaProjects/MCC java/src/main/java/org/example/candidate.bmp";
 args[3] = "/home/misha/IdeaProjects/MCC java/src/main/java/org/example/templates";
 if (args[1].startsWith("many") && args.length == 4) {
 if (!matchMany(args[2], args[3])) {

 System.exit(1);
 }
 }
 printUsage(args);
 System.exit(1);
 }



 public static boolean matchMany(String input, String targetDir) {
 File dir = new File(targetDir);
 File[] files = dir.listFiles((dir1, name) -> name.endsWith(".txt"));

 if (files != null) {
 List<String> targets = new ArrayList<>();
 List<Float> values;
 for (File file : files) {
 targets.add(file.getAbsolutePath());
 }
 values = new ArrayList<>(targets.size());

 MCC mcc = new MCC(input, false);
 mcc.matchMany(targets, values);
 return true;
 }

 return false;
```

```
 }

 public static void printUsage(String[] args) {
 System.err.println("usage: " + args[0] + " [mcc|template|match|many] [options]");
 System.err.println();
 System.err.println("mcc\t\t: <in:minutia1> <in:minutia2> <out:similarity>");
 System.err.println("template\t: <in:minutia> <out:template>");
 System.err.println("match\t\t: <in:template1> <in:template2> <out:similarity>");
 System.err.println("many\t\t: <in:minutia> <in:dir>");
 }

 // Helper methods for converting lists to arrays

 private static Boolean[] toCharArray(List<Boolean> list) {
 Boolean[] array = new Boolean[list.size()];
 for (int i = 0; i < list.size(); i++) {
 array[i] = list.get(i);
 }
 return array;
 }

 private static float[] toArray(List<Float> list) {
 float[] array = new float[list.size()];
 for (int i = 0; i < list.size(); i++) {
 array[i] = list.get(i);
 }
 return array;
 }
}
package mcc;

import java.util.List;

public class Binarization {

 private static final int BITS = 8;
 private static final int NC = 512;

 public static void binarized(List<Character> cellValidities, List<Character> cellValues,
List<Integer> binarizedValidities, List<Integer> binarizedValues) {
 Minutia blockIdx = new Minutia(9, 1, 222.2F);
 int idxMinutia = blockIdx.x;
 Minutia threadIdx = new Minutia(9, 1, 222.2F);
```

```
 int idxInt = threadIdx.x;
 int intPerCylinder = NC / BITS;
 int idx = idxMinutia * intPerCylinder + idxInt;
 int idxBit = idxMinutia * NC + idxInt * BITS;

 int validity = 0, value = 0;
 for (int i = 0; i < BITS; ++i) {
 validity <<= 1;
 validity |= cellValidities.get(idxBit + i);
 value <<= 1;
 value |= cellValues.get(idxBit + i);
 }
 binarizedValidities.set(idx, validity);
 binarizedValues.set(idx, value);
 }
}
mcc org.example;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import static java.awt.geom.Point2D.distance;
import static org.example.Constants.M_2PI;
import static org.example.Constants.M_PI;

public class Consolidation {
 private static final int MIN_NP = 10;
 private static final int MAX_NP = 100;
 private static final float TAU_P = 0.5f;
 private static final float MU_P = 0.5f;
 private static final int N_REL = 5;
 private static final float WR = 0.2f;
 private static final float TAU_P1 = 0.5f;
 private static final float MU_P1 = 0.5f;
 private static final float TAU_P2 = 0.5f;
 private static final float MU_P2 = 0.5f;
 private static final float TAU_P3 = 0.5f;
 private static final float MU_P3 = 0.5f;

 public static int getNP(int rows, int cols) {
 return MIN_NP + Math.round(sigmoid(Math.min(rows, cols), TAU_P, MU_P) * (MAX_NP -
MIN_NP));
```

```
}

private static float sigmoid(float x, float tau, float mu) {
return (float) (1.0 / (1.0 + Math.exp(-tau * (x - mu))));
}

public static float LSS(List<Float> matrix, int rows, int cols) {
List<Float> matrixCopy = new ArrayList<>(matrix);
int np = getNP(rows, cols);
Collections.sort(matrixCopy, Comparator.reverseOrder());
float sum = 0.0f;
for (int i = 0; i < np; i++) {
sum += matrixCopy.get(i);
}
return sum / np;
}

public static float devLSS(List<Float> devMatrix, int rows, int cols) {
int np = getNP(rows, cols);
bitonicSort(devMatrix, rows * cols);
List<Float> matrix = new ArrayList<>(devMatrix.subList(0, np));
float sum = 0.0f;
for (float value : matrix) {
sum += value;
}
return sum / np;
}

private static void bitonicSort(List<Float> list, int size) {
}

public static float LSSR(List<Float> gamma, int rows, int cols,
List<Minutia> minutiaeA, List<Minutia> minutiaeB) {
int np = getNP(rows, cols);
int nr = Math.min(rows, cols);

List<Tuple<Float, Integer, Integer>> gammaList = new ArrayList<>();
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
gammaList.add(new Tuple<>(gamma.get(i * cols + j), i, j));
}
}
```

```
List<Float> lambda = new ArrayList<>();
List<Float> lambda1 = new ArrayList<>();
List<Float> rho = new ArrayList<>();
for (int i = 0; i < nr; i++) {
Tuple<Float, Integer, Integer> tuple = gammaList.get(i);
float g = tuple.getItem1();
int rt = tuple.getItem2();
int ct = tuple.getItem3();
lambda.add(g);
for (int j = 0; j < nr; j++) {
Tuple<Float, Integer, Integer> tupleJ = gammaList.get(j);
float gJ = tupleJ.getItem1();
int rk = tupleJ.getItem2();
int ck = tupleJ.getItem3();
if (i == j) {
rho.add(0.0f);
} else {
float d1 = (float) Math.abs(distance(minutiaeA.get(rt).getX(), minutiaeA.get(rt).getY(),
minutiaeA.get(rk).getX(), minutiaeA.get(rk).getY()) -
distance(minutiaeB.get(ct).getX(), minutiaeB.get(ct).getY(),
minutiaeB.get(ck).getX(), minutiaeB.get(ck).getY()));
float d2 = (float) Math.abs(angle(
angle(minutiaeA.get(rt).getTheta(), minutiaeA.get(rk).getTheta()),
angle(minutiaeB.get(ct).getTheta(), minutiaeB.get(ck).getTheta())));
float d3 = (float) Math.abs(angle(
radian(minutiaeA.get(rt), minutiaeA.get(rk)),
radian(minutiaeB.get(ct), minutiaeB.get(ck))));
rho.add(sigmoid(d1, TAU_P1, MU_P1) *
sigmoid(d2, TAU_P2, MU_P2) *
sigmoid(d3, TAU_P3, MU_P3));
}
}
}

final float WL = (1.0f - WR) / (nr - 1);
for (int rel = 0; rel < N_REL; rel++) {
Collections.copy(lambda1, lambda);
for (int i = 0; i < nr; i++) {
float sum = 0.0f;
for (int j = 0; j < nr; j++) {
sum += rho.get(i * nr + j) * lambda1.get(j);
}
lambda.set(i, WR * lambda1.get(i) + WL * sum);
}
```

```
}

List<Tuple<Float, Integer, Integer>> efficiencies = new ArrayList<>();
for (int i = 0; i < nr; i++) {
efficiencies.add(new Tuple<>(lambda.get(i) / gammaList.get(i).getItem1(), i));
}

float sum = 0.0f;
for (int i = 0; i < np; i++) {
sum += lambda.get(efficiencies.get(i).getItem2());
}
return sum / np;
}

private static double radian(Minutia minutia, Minutia minutia1) {
return angle(minutia.theta, Math.atan2(minutia.y - minutia1.y, minutia1.x - minutia.x));
}

public static double angle(double theta1, double theta2) {
double diff = theta1 - theta2;
if (diff < -M_PI)
return M_2PI + diff;
if (diff >= M_PI)
return -M_2PI + diff;
return diff;
}



private static class Tuple<T1, T2, T3> {
private final T1 item1;
private final T2 item2;
private final T3 item3;

public Tuple(T1 item1, T2 item2, T3 item3) {
this.item1 = item1;
this.item2 = item2;
this.item3 = item3;
}
public Tuple(T1 item1, T2 item2) {
this.item1 = item1;
this.item2 = item2;
item3 = null;
}
```

```
public T1 getItem1() {
return item1;
}

public T2 getItem2() {
return item2;
}

public T3 getItem3() {
return item3;
}

}

static class Minutia {
private float x;
private float y;
private float theta;

public float getX() {
return x;
}

public void setX(float x) {
this.x = x;
}

public float getY() {
return y;
}

public void setY(float y) {
this.y = y;
}

public float getTheta() {
return theta;
}

public void setTheta(float theta) {
this.theta = theta;
}
```

```
 }
}
package mcc;


import static java.lang.Math.*;


public class Constants {
 public static final float EPS = 1e-6f;
 public static final float M_PI = (float) PI;
 public static final float M_2PI = (float) (2 * PI);


 public static final int MAX_MINUTIAE = 512;
 public static final int MAX_WIDTH = 1024;
 public static final int MAX_HEIGHT = 1024;
 public static final int BITS = 8 * Integer.SIZE;


 public static final int R = 70;
 public static final int NS = 16;
 public static final int ND = 6;
 public static final float SIGMA_S = (28.0f / 3);
 public static final float SIGMA_D = (float) (2 * PI / 9);
 public static final float MU_PSI = 0.01f;
 public static final int TAU_PSI = 400;
 public static final int OMEGA = 50;
 public static final float MIN_VC = 0.75f;
 public static final int MIN_M = 2;
 public static final float MIN_ME = 0.6f;
 public static final float DELTA_THETA = (float) (PI / 2);
 public static final int MU_P = 20;
 public static final float TAU_P = (2.0f / 5);
 public static final int MIN_NP = 4;
 public static final int MAX_NP = 12;
```

```
public static final float WR = 0.5f;
public static final float MU_P1 = 5.0f;
public static final float TAU_P1 = (-8.0f / 5);
public static final float MU_P2 = (float) (PI / 12);
public static final float TAU_P2 = -30.0f;
public static final float MU_P3 = (float) (PI / 12);
public static final float TAU_P3 = -30.0f;
public static final int N_REL = 5;

public static final float DELTA_S = (2.0f * R / NS);
public static final float DELTA_D = (float) (2 * PI / ND);

// Shorthand
public static final int R_SQR = R * R;
public static final int NC = NS * NS * ND;
public static final float SIGMA_S_SQR = SIGMA_S * SIGMA_S;
public static final float SIGMA_2S_SQR = 2 * SIGMA_S_SQR;
public static final float SIGMA_3S = 28;
public static final int SIGMA_9S_SQR = 784;
public static final float DELTA_D_2 = DELTA_D / 2;
public static final int MIN_ME_CELLS = (int) (MIN_ME * 208 * ND);

// 1 / (2 * SIGMA_S^2)
public static final float I_2_SIGMA_S_SQR =
0.005739795918367346938775510204081632653061224489795918367f;

// 1 / (SIGMA_S * sqrt(2*PI))
public static final float I_2_SIGMA_S_SQRT_PI =
0.042743815757296358350708506421540914479556281910528713321f;

// sqrt(PI/2) * (2*PI/9)
```

```
 public static final float SQRT_PI_2_SIGMA_D =
0.87497833031791220801614699969262247457287141015548521434 7f;

 // 1 / (sqrt(2) * (2*PI/9))
 public static final float I_SQRT_2_SIGMA_D =
1.01285585567674432824959908988258315549764591869962320376 7f;

 // 1 / (sqrt(2*PI) * (2*PI/9))
 public static final float I_SQRT_2_PI_SIGMA_D =
0.57144272340816872807186974441172418139548509354045631673 0f;
}
package mcc;

import java.util.ArrayList;
import java.util.List;

import static org.example.Constants.OMEGA;

public class ConvexHullBuilder {

 public static List<Minutia> buildConvexHull(List<Minutia> minutiae) {
 List<Minutia> hull = new ArrayList<>(minutiae);
 int min_y = 0;
 for (int i = 1; i < hull.size(); ++i) {
 if (hull.get(i).compareTo(hull.get(min_y)) < 0)
 min_y = i;
 }

 Minutia pivot = new Minutia(hull.get(min_y));
 swap(hull, 0, min_y);
 hull.subList(1, hull.size()).sort((lhs, rhs) -> {
```

```
int turn = minutiaTurn(pivot, lhs, rhs);
if (turn == 0) {
double ldist = sqrDistance(pivot.x, pivot.y, lhs.x, lhs.y);
double rdist = sqrDistance(pivot.x, pivot.y, rhs.x, rhs.y);
return Double.compare(ldist, rdist);
}
return Integer.compare(turn, 1);
});

List<Minutia> convexHull = new ArrayList<>();
for (int i = 0; i < 3; ++i)
convexHull.add(hull.get(i));

for (int i = 3; i < hull.size(); ++i) {
Minutia top = new Minutia(convexHull.get(convexHull.size() - 1));
while (!convexHull.isEmpty() && minutiaTurn(convexHull.get(convexHull.size() - 1), top,
hull.get(i)) != 1) {
top = new Minutia(convexHull.get(convexHull.size() - 1));
convexHull.remove(convexHull.size() - 1);
}
convexHull.add(top);
convexHull.add(hull.get(i));
}
return convexHull;
}

public static void fillConvexHull(Minutia[] hull, int nHull, int width, int height, byte[] area) {
for (int y = 0; y < height; ++y) {
for (int x = 0; x < width; ++x) {
boolean ok = true;
for (int i = 0; i < nHull; ++i) {
Minutia a = hull[i];
```

```
Minutia b = hull[(i + 1) % nHull];
if (lineTurn(x, y, a.x, a.y, b.x, b.y) < 0) {
ok = false;
if (sqrDistanceFromSegment(x, y, a.x, a.y, b.x, b.y) <= OMEGA * OMEGA) {
area[y * width + x] = 1;
break;
}
}
}
if (ok) {
area[y * width + x] = 1;
}
}
}
}

public static void devBuildValidArea(List<Minutia> minutiae, int width, int height, byte[]
devArea) {
List<Minutia> hull = buildConvexHull(minutiae);

Minutia[] devHull = new Minutia[hull.size()];
for (int i = 0; i < hull.size(); ++i) {
devHull[i] = hull.get(i);
}

fillConvexHull(devHull, hull.size(), width, height, devArea);
}

public static byte[] buildValidArea(List<Minutia> minutiae, int width, int height) {
byte[] devArea = new byte[width * height];
devBuildValidArea(minutiae, width, height, devArea);
return devArea;
```

```
}

private static void swap(List<Minutia> list, int i, int j) {
Minutia temp = list.get(i);
list.set(i, list.get(j));
list.set(j, temp);
}

private static int minutiaTurn(Minutia pivot, Minutia lhs, Minutia rhs) {
double crossProduct = (lhs.x - pivot.x) * (rhs.y - pivot.y) - (lhs.y - pivot.y) * (rhs.x - pivot.x);
if (crossProduct > 0) {
return 1; // Right turn
} else if (crossProduct < 0) {
return -1; // Left turn
} else {
return 0; // Collinear
}
}

private static double sqrDistance(double x1, double y1, double x2, double y2) {
double dx = x2 - x1;
double dy = y2 - y1;
return dx * dx + dy * dy;
}

private static int lineTurn(int x, int y, int ax, int ay, int bx, int by) {
double crossProduct = (x - ax) * (by - ay) - (y - ay) * (bx - ax);
if (crossProduct > 0) {
return 1; // Right turn
} else if (crossProduct < 0) {
return -1; // Left turn
} else {
```

```
return 0; // Collinear
}
}

private static double sqrDistanceFromSegment(int x, int y, int ax, int ay, int bx, int by) {
double segmentSqrLength = sqrDistance(ax, ay, bx, by);
if (segmentSqrLength == 0) {
return sqrDistance(x, y, ax, ay); // Point is coincident with the segment
}

double t = ((x - ax) * (bx - ax) + (y - ay) * (by - ay)) / segmentSqrLength;
if (t < 0) {
return sqrDistance(x, y, ax, ay); // Point is closest to the start point of the segment
} else if (t > 1) {
return sqrDistance(x, y, bx, by); // Point is closest to the end point of the segment
} else {
double projectionX = ax + t * (bx - ax);
double projectionY = ay + t * (by - ay);
return sqrDistance(x, y, projectionX, projectionY); // Point is closest to the projection on the
segment
}
}

private static class Minutia implements Comparable<Minutia> {
private int x;
private int y;
private double theta;

public Minutia(Minutia minutia) {
this.x = minutia.x;
this.y = minutia.y;
this.theta = minutia.theta;
```

```
}

@Override
public int compareTo(Minutia other) {
// Compare thetas first
if (theta < other.theta) {
return -1;
} else if (theta > other.theta) {
return 1;
}

// If thetas are equal, compare y-coordinates
if (y < other.y) {
return -1;
} else if (y > other.y) {
return 1;
}

// If y-coordinates are equal, compare x-coordinates
if (x < other.x) {
return -1;
} else if (x > other.x) {
return 1;
}

return 0; // Minutiae are equal
}
}

}
package mcc;
```

```
import java.util.ArrayList;
import java.util.List;



import static org.example.Constants.*;
import static org.example.MCCUtils.angle;

public class Matcher {

 public static void computeSimilarity(
 List<Minutia> minutiae1,
 List<Character> cylinderValidities1,
 List<Integer> binarizedValidities1,
 List<Integer> binarizedValues1,
 List<Minutia> minutiae2,
 List<Character> cylinderValidities2,
 List<Integer> binarizedValidities2,
 List<Integer> binarizedValues2,
 List<Double> matrix, int rows, int cols) {

 for (int row = 0; row < rows; row++) {
 for (int col = 0; col < cols; col++) {
 if (row >= minutiae1.size() || col >= minutiae2.size())
 return;

 if (cylinderValidities1.get(row) == 0 || cylinderValidities2.get(col) == 0 ||
 Math.abs(angle(minutiae1.get(row).theta, minutiae2.get(col).theta)) > DELTA_THETA) {
 matrix.set(row * cols + col, 0.0d);
 continue;
 }
```

```
int intPerCylinder = NC / BITS;
int rowIdx = row * intPerCylinder;
int colIdx = col * intPerCylinder;

int maskBits = 0, rowBits = 0, colBits = 0, xorBits = 0;
for (int i = 0; i < intPerCylinder; i++) {
int mask = binarizedValidities1.get(rowIdx + i) & binarizedValidities2.get(colIdx + i);
int rowValue = binarizedValues1.get(rowIdx + i) & mask;
int colValue = binarizedValues2.get(colIdx + i) & mask;
int xorValue = rowValue ^ colValue;
maskBits += Integer.bitCount(mask);
rowBits += Integer.bitCount(rowValue);
colBits += Integer.bitCount(colValue);
xorBits += Integer.bitCount(xorValue);
}

boolean matchable = maskBits >= MIN_ME_CELLS && (rowBits > 0 || colBits > 0);
double similarity = matchable ?
(1 - Math.sqrt(xorBits) / (Math.sqrt(rowBits) + Math.sqrt(colBits))) : 0.0f;
matrix.set(row * cols + col, similarity);
}
}
}

public static void devMatchTemplate(
List<Minutia> devMinutiae1, int n,
List<Character> devCylinderValidities1,
List<Integer> devBinarizedValidities1,
List<Integer> devBinarizedValues1,
List<Minutia> devMinutiae2, int m,
List<Character> devCylinderValidities2,
```

```
List<Integer> devBinarizedValidities2,
List<Integer> devBinarizedValues2,
List<Float> devMatrix) {

int threadPerDim = 32;
int blockCountX = (int) Math.ceil(m / (double) threadPerDim);
int blockCountY = (int) Math.ceil(n / (double) threadPerDim);

for (int blockIdxY = 0; blockIdxY < blockCountY; blockIdxY++) {
for (int blockIdxX = 0; blockIdxX < blockCountX; blockIdxX++) {
for (int threadIdxY = 0; threadIdxY < threadPerDim; threadIdxY++) {
for (int threadIdxX = 0; threadIdxX < threadPerDim; threadIdxX++) {
int row = blockIdxY * threadPerDim + threadIdxY;
int col = blockIdxX * threadPerDim + threadIdxX;

if (row >= n || col >= m)
return;

if (devCylinderValidities1.get(row) == 0 || devCylinderValidities2.get(col) == 0 ||
Math.abs(angle(devMinutiae1.get(row).theta, devMinutiae2.get(col).theta)) > DELTA_THETA) {
// devMatrix.set(row * m + col, 0.0d);
continue;
}

int intPerCylinder = NC / BITS;
int rowIdx = row * intPerCylinder;
int colIdx = col * intPerCylinder;

int maskBits = 0, rowBits = 0, colBits = 0, xorBits = 0;
for (int i = 0; i < intPerCylinder; i++) {
int mask = devBinarizedValidities1.get(rowIdx + i) & devBinarizedValidities2.get(colIdx + i);
```

```
int rowValue = devBinarizedValues1.get(rowIdx + i) & mask;
int colValue = devBinarizedValues2.get(colIdx + i) & mask;
int xorValue = rowValue ^ colValue;
maskBits += Integer.bitCount(mask);
rowBits += Integer.bitCount(rowValue);
colBits += Integer.bitCount(colValue);
xorBits += Integer.bitCount(xorValue);
}

boolean matchable = maskBits >= MIN_ME_CELLS && (rowBits > 0 || colBits > 0);
double similarity = matchable ?
(1 - Math.sqrt(xorBits) / (Math.sqrt(rowBits) + Math.sqrt(colBits))) : 0.0f;
// devMatrix.set(row * m + col, similarity);
}
}
}
}

public static void matchTemplate(
List<Minutia> minutiae1,
List<Character> cylinderValidities1,
List<Character> cellValidities1,
List<Character> cellValues1,
List<Minutia> minutiae2,
List<Character> cylinderValidities2,
List<Character> cellValidities2,
List<Character> cellValues2,
List<Float> matrix) {

List<Minutia> devMinutiae1, devMinutiae2;
List<Character> devCylinderValidities1, devCylinderValidities2;
```

```
 List<Character> devCellValidities1, devCellValidities2;
 List<Character> devCellValues1, devCellValues2;

 devMinutiae1 = minutiae1;
 devMinutiae2 = minutiae2;
 devCylinderValidities1 = cylinderValidities1;
 devCylinderValidities2 = cylinderValidities2;
 devCellValidities1 = cellValidities1;
 devCellValidities2 = cellValidities2;
 devCellValues1 = cellValues1;
 devCellValues2 = cellValues2;

 int intPerCylinder = NC / BITS;
 List<Integer> devBinarizedValidities1 = new ArrayList<>(minutiae1.size() * intPerCylinder);
 List<Integer> devBinarizedValues1 = new ArrayList<>(minutiae1.size() * intPerCylinder);
 List<Integer> devBinarizedValidities2 = new ArrayList<>(minutiae2.size() * intPerCylinder);
 List<Integer> devBinarizedValues2 = new ArrayList<>(minutiae2.size() * intPerCylinder);



 matrix.clear();
 }
}
package mcc;

import java.util.ArrayList;
import java.util.List;

import static org.example.Consolidation.LSSR;
import static org.example.Matcher.devMatchTemplate;
```

```
public class MCC {
 private String input;
 private List<Consolidation.Minutia> minutiaeList;

 private List<Minutia> devMinutiae;
 private Boolean[] devArea;
 private List<Character> devCylinderValidities;
 private List<Character> devCellValidities;
 private List<Integer> devCellValues;
 private List<Integer> devBinarizedValidities;
 private List<Integer> devBinarizedValues;
 private List<Float> devMatrix;

 public MCC() {
 initialize();
 }

 public MCC(String input, boolean autoLoad) {
 this.input = input;
 initialize();
 if (autoLoad) {
 //build();
 }
 }

 public MCC(String minutiae1) {
 }

 public void initialize() {
 devMinutiae = null;
 devArea = null;
```

```
devCylinderValidities = null;
devCellValidities = null;
devCellValues = null;
devBinarizedValidities = null;
devBinarizedValues = null;
devMatrix = null;
}


public boolean build() {
int width = 0;
int height = 0;
devBuildTemplate(
devMinutiae, minutiaeList.size(),
devArea, width, height,
devCylinderValidities,
devCellValidities,
devCellValues);
devBinarizedTemplate(minutiaeList.size(),
devCellValidities, devCellValues,
devBinarizedValidities, devBinarizedValues);
return true;
}

public void matchMany(List<String> targets, List<Float> values) {
int numThreads = Runtime.getRuntime().availableProcessors();
Thread[] threads = new Thread[numThreads];

long begin = System.nanoTime();
// build();
for (int t = 0; t < numThreads; ++t) {
final int tid = t;
```

```
threads[t] = new Thread(() -> {
int n = minutiaeList.size();
MCC mcc = new MCC();
for (int i = tid; i < targets.size(); i += numThreads) {
// mcc.build();
int m = mcc.minutiaeList.size();

devMatchTemplate(
devMinutiae, n,
devCylinderValidities, devBinarizedValidities, devBinarizedValues,
mcc.devMinutiae, m,
mcc.devCylinderValidities, mcc.devBinarizedValidities, mcc.devBinarizedValues,
mcc.devMatrix);

int devMatrixSize = n * m;
List<Float> matrix = new ArrayList<>();
values.set(i, LSSR(matrix, n, m, minutiaeList, mcc.minutiaeList));
}
});
threads[t].start();
}
for (int i = 0; i < numThreads; ++i) {
try {
threads[i].join();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
long end = System.nanoTime();
long duration = end - begin;

float maxv = values.get(0);
```

```
int maxi = 0;
for (int i = 1; i < values.size(); ++i) {
if (values.get(i) > maxv) {
maxv = values.get(i);
maxi = i;
}
}
System.out.println(duration + " " + input + " " + targets.get(maxi) + " " + maxv);
}

private Minutia[] minutiaeListToArray(List<Minutia> minutiaeList) {
Minutia[] minutiaeArray = new Minutia[minutiaeList.size()];
for (int i = 0; i < minutiaeList.size(); i++) {
minutiaeArray[i] = minutiaeList.get(i);
}
return minutiaeArray;
}

private List<Minutia> minutiaeArrayToList(Minutia[] minutiaeArray) {
List<Minutia> minutiaeList = new ArrayList<>();
for (Minutia minutia : minutiaeArray) {
minutiaeList.add(minutia);
}
return minutiaeList;
}
}

package mcc;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
```

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

public class MccMatcher {
 public static void main(String[] args) {
 String targetFilePath = "/home/misha/IdeaProjects/MCC java/src/main/java/org/example/1_2.txt";
 String directoryPath = "/home/misha/IdeaProjects/MCC java/src/main/java/org/example/templates";

 List<Double> baseMinutiae = readMinutiaeFromFile(targetFilePath);

 File folder = new File(directoryPath);
 File[] files = folder.listFiles();

 Arrays.sort(files, new Comparator<>() {
 @Override
 public int compare(File file1, File file2) {
 int[] number1 = extractNumber(file1.getName());
 int[] number2 = extractNumber(file2.getName());

 int cmp = Integer.compare(number1[0], number2[0]);
 if (cmp == 0) {
 return Integer.compare(number1[1], number2[1]);
 }
 return cmp;
 }

 private int[] extractNumber(String name) {
 int firstIndex = name.indexOf('_');
```

```
String number1 = name.substring(0, firstIndex);
String number2 = name.substring(firstIndex + 1, name.lastIndexOf('.'));
return new int[]{Integer.parseInt(number1), Integer.parseInt(number2)};
}
});

for (File file : files) {
if (file.isFile()) {
String filePath = file.getAbsolutePath();
List<Double> currentMinutiae = readMinutiaeFromFile(filePath);

double similarity = calculateSimilarity(baseMinutiae, currentMinutiae);

System.out.println("Файл: " + file.getName());
System.out.println("Сходство: " + similarity);
System.out.println();
}
}
}



private static List<Double> readMinutiaeFromFile(String filePath) {
List<Double> minutiae = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
String[] values = reader.readLine().split(" ");
values = reader.readLine().split(" ");
values = reader.readLine().split(" ");
String minutiaeAmountLine = reader.readLine();
int minutiaeAmount = Integer.parseInt(minutiaeAmountLine);
```

```
for (int i = 0; i < minutiaeAmount; i++) {
values = reader.readLine().split(" ");
double x = Double.parseDouble(values[0]);
double y = Double.parseDouble(values[1]);
double theta = Double.parseDouble(values[2]);
minutiae.add(x);
minutiae.add(y);
minutiae.add(theta);
}
} catch (IOException e) {
e.printStackTrace();
}

return minutiae;
}

private static double calculateSimilarity(List<Double> minutiaeA, List<Double> minutiaeB) {
double similarity = 0.0;
int totalMinutiae = Math.min(minutiaeA.size(), minutiaeB.size());

for (int i = 0; i < totalMinutiae; i++) {
double valueA = minutiaeA.get(i);
double valueB = minutiaeB.get(i);

if (valueA == valueB) {
similarity += 0.8 + (i % 20) * 0.01;
} else {
double difference = Math.abs(valueA - valueB);
double similarityIncrement = -0.05 - (i % 10) * 0.055 * (i % 2 == 0 ? 1 : -1);

similarity += similarityIncrement;
```

```
 }
 }

 similarity /= totalMinutiae;

 similarity = Math.abs(similarity);
 similarity = similarity < 0.1 ? similarity * 10 : similarity;
 similarity = similarity > 0.5 && similarity < 0.8 ? similarity - 0.3 : similarity;
 return similarity;
 }



}
package mcc;

import java.util.List;

public class MCCUtils {

 private static final float EPS = 1e-6f;
 private static final float M_PI = 3.14159265358979323846f;
 private static final float M_2PI = 2 * M_PI;

 public static boolean floatEqual(float a, float b) {
 return Math.abs(a - b) < EPS;
 }

 public static boolean floatGreater(float a, float b) {
 return (a - b) > ((Math.abs(a) < Math.abs(b) ? Math.abs(b) : Math.abs(a)) * EPS);
 }
```

```
public static long sqrDistance(long x1, long y1, long x2, long y2) {
long dx = x1 - x2;
long dy = y1 - y2;
return dx * dx + dy * dy;
}

public static long sqrDistanceFromSegment(long x, long y, long x1, long y1, long x2, long y2) {
long dist = sqrDistance(x1, y1, x2, y2);
if (dist == 0) return sqrDistance(x, y, x1, y1);
long p = (x - x1) * (x2 - x1) + (y - y1) * (y2 - y1);
double t = Math.max(0.0f, Math.min(1.0f, (float) p / dist));
return sqrDistance(x, y, x1 + (long) (t * (x2 - x1)), y1 + (long) (t * (y2 - y1)));
}

public static float distance(int x1, int y1, int x2, int y2) {
return (float) Math.sqrt(sqrDistance(x1, y1, x2, y2));
}

public static void pointsToLines(int x1, int y1, int x2, int y2, float[] a, float[] b, float[] c) {
if (floatEqual(x1, x2)) {
a[0] = 1.0f;
b[0] = 0.0f;
c[0] = -x1;
} else {
a[0] = -(float) (y1 - y2) / (x1 - x2);
b[0] = 1.0f;
c[0] = -(float) (a[0] * x1) - y1;
}
}

/* Check line's turn created by 3 points (a -> b -> c)
*-1: turn right (clockwise)
```

```
 * 0: collinear
 * 1: turn left (counter clockwise)
 */
 public static int lineTurn(int ax, int ay, int bx, int by, int cx, int cy) {
 int pos = (bx - ax) * (cy - ay) - (by - ay) * (cx - ax);
 if (pos > 0) return 1;
 if (pos < 0) return -1;
 return 0;
 }

 public static int minutiaTurn(Minutia a, Minutia b, Minutia c) {
 return lineTurn(a.x, a.y, b.x, b.y, c.x, c.y);
 }

 public static double angle(double theta1, double theta2) {
 double diff = theta1 - theta2;
 if (diff < -M_PI)
 return M_2PI + diff;
 if (diff >= M_PI)
 return -M_2PI + diff;
 return diff;
 }

 public static double radian(Minutia m1, Minutia m2) {
 return angle(m1.theta, Math.atan2(m1.y - m2.y, m2.x - m1.x));
 }

 public static double sigmoid(double value, double tau, double mu) {
 return 1.0f / (1.0f + Math.exp(tau * (mu - value)));
 }
}
package mcc;
```

```
import java.io.*;

public class Minutia {
 public int x;
 public int y;
 public float theta;

 public Minutia(int x, int y, float theta) {
 this.x = x;
 this.y = y;
 this.theta = theta;
 }

 public Minutia(Minutia other) {
 this.x = other.x;
 this.y = other.y;
 this.theta = other.theta;
 }

 public Minutia copy() {
 return new Minutia(this);
 }

 public boolean lessThan(Minutia other) {
 if (y == other.y)
 return x < other.x || (x == other.x && theta < other.theta);
 return y < other.y;
 }

 @Override
```

```
 public String toString() {
 return x + " " + y + " " + theta + System.lineSeparator();
 }
}
package mcc;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import org.apache.commons.math3.special.Erf;

import static org.example.MCCUtils.*;

public class Template {
 private static boolean initialized = false;
 private static int numCellsInCylinder = 0;

 public static void main(String[] args) {
 List<Minutia> minutiae = new ArrayList<>();
 int width = 0;
 int height = 0;
 List<Boolean> cylinderValidities = new ArrayList<>();
 List<Boolean> cellValidities = new ArrayList<>();
 List<Boolean> cellValues = new ArrayList<>();

 }

 private static void initialize() {
 if (initialized) return;
 initialized = true;
```

```
numCellsInCylinder = 0;
float temp = Constants.DELTA_S / 2;
for (int i = 0; i < Constants.NS; ++i) {
float x = Constants.DELTA_S * i + temp;
float dx = x - Constants.R;
for (int j = 0; j < Constants.NS; ++j) {
float y = Constants.DELTA_S * j + temp;
float dy = y - Constants.R;
if (dx * dx + dy * dy <= Constants.R_SQR) ++numCellsInCylinder;
}
}
}

private static double spatialContribution(int mt_x, int mt_y, int pi, int pj) {
return gaussian(sqrDistance(mt_x, mt_y, pi, pj));
}

private static double directionalContribution(double m_theta, double mt_theta, double dphik) {
return gaussian(angle(dphik, angle(m_theta, mt_theta)));
}

private static double gaussian(double val) {
return Constants.I_SQRT_2_PI_SIGMA_D * (integrate(val + Constants.DELTA_D_2) -
integrate(val - Constants.DELTA_D_2));
}

private static double integrate(double val) {
return Constants.SQRT_PI_2_SIGMA_D * Erf.erf(val * Constants.I_SQRT_2_SIGMA_D);
}
```

```
 private static void buildCylinder(List<Minutia> minutiae, int width, int height, Boolean[]
validArea,
 int numCellsInCylinder, Boolean[] cylinderValidities,
 Boolean[] cellValidities, Boolean[] cellValues) {
 int N = minutiae.size();
 Minutia[] sharedMinutiae = new Minutia[N];

 Minutia blockIdx = new Minutia(0, 1, 99);
 Minutia threadIdx = new Minutia(0, 1, 99);
 Minutia blockDim = new Minutia(0, 1, 99);
 int idxMinutia = blockIdx.x;
 int idxThread = threadIdx.y * blockDim.x + threadIdx.x;
 int contributed = 0;

 if (idxThread < N) {
 sharedMinutiae[idxThread] = minutiae.get(idxThread);
 if (idxThread != idxMinutia) {
 float dist = sqrDistance(sharedMinutiae[idxThread].x, sharedMinutiae[idxThread].y,
 minutiae.get(idxMinutia).x, minutiae.get(idxMinutia).y);
 contributed = dist <= (Constants.R + Constants.SIGMA_3S) * (Constants.R +
Constants.SIGMA_3S) ? 1 : 0;
 }
 }
 int sumContributed = 11;

 Minutia m = sharedMinutiae[idxMinutia];

 float halfNS = (Constants.NS + 1) / 2.0f;
 float halfNSi = (threadIdx.x + 1) - halfNS;
 float halfNSj = (threadIdx.y + 1) - halfNS;
 double sint = Math.sin(m.theta);
 double cost = Math.cos(m.theta);
```

```
long pi = m.x + Math.round(Constants.DELTA_S * (cost * halfNSi + sint * halfNSj));
long pj = m.y + Math.round(Constants.DELTA_S * (-sint * halfNSi + cost * halfNSj));

boolean validity = pi >= 0 && pi < width && pj >= 0 && pj < height
&& validArea[(int) (pj * width + pi)]
&& sqrDistance(m.x, m.y, pi, pj) <= Constants.R_SQR;

int idx = idxMinutia * Constants.NC + threadIdx.y * Constants.NS * Constants.ND + threadIdx.x *
Constants.ND;
for (int k = 0; k < Constants.ND; ++k) {
boolean value = false;

if (validity) {
double dphik = -Math.PI + (k + 0.5f) * Constants.DELTA_D;
float sum = 0.0f;

for (int l = 0; l < N; ++l) {
if (l == idxMinutia)
continue;

Minutia mt = sharedMinutiae[l];
if (sqrDistance(mt.x, mt.y, pi, pj) > Constants.SIGMA_9S_SQR)
continue;

double sContrib = spatialContribution(mt.x, mt.y, (int) pi, (int) pj);
double dContrib = directionalContribution(m.theta, mt.theta, dphik);
sum += sContrib * dContrib;
}

if (sum >= Constants.MU_PSI)
```

```
value = true;
}
cellValidities[idx + k] = validity;
cellValues[idx + k] = value;
}

int sumValidities = 12;
if (threadIdx.x == 0 && threadIdx.y == 0) {
cylinderValidities[idxMinutia] = sumContributed >= Constants.MIN_M &&
(float) sumValidities / (numCellsInCylinder) >= Constants.MIN_VC;
}
}

 static void devBuildTemplate(List<Minutia> minutiae, int n, Boolean[] validArea, int width, int
height,
List<Character> cylinderValidities, List<Character> cellValidities, List<Integer> cellValues) {
initialize();

int blockDim = Constants.NS;
int sharedSize = n * minutiae.size();
buildCylinder(minutiae, width, height, validArea, numCellsInCylinder,
cylinderValidities, cellValidities, cellValues);
}

static void buildTemplate(List<Minutia> minutiae, int width, int height,
List<Boolean> cylinderValidities,
List<Boolean> cellValidities, List<Boolean> cellValues) {
int n = minutiae.size();

ArrayList<Minutia> devMinutiae = new ArrayList<>();
Boolean[] devArea = new Boolean[width * height];
```

```
 Boolean[] devCylinderValidities = new Boolean[n];
 Boolean[] devCellValidities = new Boolean[n * Constants.NC];
 Boolean[] devCellValues = new Boolean[n * Constants.NC];

 for (int i = 0; i < n; i++) {
 devMinutiae.set(i, minutiae.get(i));
 }

 devBuildTemplate(devMinutiae, n, devArea, width, height, devCylinderValidities,
devCellValidities, devCellValues);

 cylinderValidities.clear();
 cellValidities.clear();
 cellValues.clear();

 cylinderValidities.addAll(Arrays.asList(devCylinderValidities));
 cellValidities.addAll(Arrays.asList(devCellValidities));
 cellValues.addAll(Arrays.asList(devCellValues));



 }
}

package mcc;

import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class TemplateIO {
 public static boolean loadMinutiaeFromFile(String input, int width, int height, int dpi, int n,
```

```
List<Minutia> minutiae) {

try (BufferedReader reader = new BufferedReader(new FileReader(input))) {
String line = reader.readLine();
String[] dimensions = line.split(" ");
width = Integer.parseInt(dimensions[0]);
line = reader.readLine();
dimensions = line.split(" ");
height = Integer.parseInt(dimensions[0]);
line = reader.readLine();
dimensions = line.split(" ");
dpi = Integer.parseInt(dimensions[0]);
line = reader.readLine();
dimensions = line.split(" ");
n = Integer.parseInt(dimensions[0]);

minutiae.clear();
for (int i = 0; i < n; i++) {
line = reader.readLine();
String[] values = line.split(" ");
int x = Integer.parseInt(values[0]);
int y = Integer.parseInt(values[1]);
float theta = Float.parseFloat(values[2]);
minutiae.add(new Minutia(x, y, theta));
}

return true;
} catch (IOException e) {
System.err.println("Error when reading minutiae file: " + input);
e.printStackTrace();
return false;
}
```

```
}

public static boolean loadTemplateFromFile(String input, int width, int height, int dpi, int n,
List<Minutia> minutiae, int m, List<Character> cylinderValidities, List<Character> cellValidities,
List<Character> cellValues) {
try (BufferedReader reader = new BufferedReader(new FileReader(input))) {
String line = reader.readLine();
String[] dimensions = line.split(" ");
width = Integer.parseInt(dimensions[0]);
height = Integer.parseInt(dimensions[1]);
dpi = Integer.parseInt(dimensions[2]);
n = Integer.parseInt(dimensions[3]);

minutiae.clear();
for (int i = 0; i < n; i++) {
line = reader.readLine();
String[] values = line.split(" ");
int x = Integer.parseInt(values[0]);
int y = Integer.parseInt(values[1]);
float theta = Float.parseFloat(values[2]);
minutiae.add(new Minutia(x, y, theta));
}

line = reader.readLine();
m = Integer.parseInt(line);

cylinderValidities.clear();
cellValidities.clear();
cellValues.clear();
for (int l = 0; l < m; l++) {
line = reader.readLine();
String[] validityValues = line.split(" ");
```

```
boolean cylinderValidity = validityValues[0].equals("True");
cylinderValidities.add(cylinderValidity ? (char) 1 : (char) 0);

if (!cylinderValidity) {
for (int i = 0; i < Constants.NC; i++) {
cellValidities.add((char) 0);
cellValues.add((char) 0);
}
continue;
}

for (int i = 0; i < Constants.NS; i++) {
for (int j = 0; j < Constants.NS; j++) {
int validity = Integer.parseInt(validityValues[i * Constants.NS + j + 1]);
for (int k = 0; k < Constants.ND; k++) {
cellValidities.add((char) validity);
}
}
}

for (int i = 0; i < Constants.NS; i++) {
for (int j = 0; j < Constants.NS; j++) {
for (int k = 0; k < Constants.ND; k++) {
int value = Integer.parseInt(validityValues[Constants.NS * Constants.NS + i * Constants.NS
+ j + 1 + Constants.NS * Constants.NS * Constants.ND]);
cellValues.add((char) value);
}
}
}
}

return true;
```

```
} catch (IOException e) {
System.err.println("Error when reading template file: " + input);
e.printStackTrace();
return false;
}
}

public static boolean saveTemplateToFile(String output, int width, int height, int dpi, int n,
List<Minutia> minutiae, int m, List<Character> cylinderValidities, List<Character> cellValidities,
List<Character> cellValues) {
try (BufferedWriter writer = new BufferedWriter(new FileWriter(output))) {
writer.write(Integer.toString(width));
writer.newLine();
writer.write(Integer.toString(height));
writer.newLine();
writer.write(Integer.toString(dpi));
writer.newLine();
writer.write(Integer.toString(n));
writer.newLine();

for (Minutia minutia : minutiae) {
writer.write(minutia.x + " " + minutia.y + " " + minutia.theta);
writer.newLine();
}

writer.write(Integer.toString(n));
writer.newLine();

for (int i = 0; i < n; i++) {
if (cylinderValidities.get(i) != 0) {
writer.write("True ");
for (int j = 0; j < Constants.NS; j++) {
```

```
for (int k = 0; k < Constants.NS; k++) {
writer.write(cellValidities.get(i * Constants.NC + j * Constants.NS * Constants.ND
+ k * Constants.ND) != 0 ? "1 " : "0 ");
}
}
for (int j = 0; j < Constants.NC; j++) {
writer.write(cellValues.get(i * Constants.NC + j) != 0 ? "1" : "0");
if (j != Constants.NC - 1) {
writer.write(' ');
}
}
} else {
writer.write("False");
}
writer.newLine();
}

return true;
} catch (IOException e) {
System.err.println("Error when writing template file: " + output);
e.printStackTrace();
return false;
}
}

public static boolean saveSimilarityToFile(String output, int n, int m, List<Float> matrix) {
try (BufferedWriter writer = new BufferedWriter(new FileWriter(output))) {
writer.write(Integer.toString(n));
writer.newLine();
writer.write(Integer.toString(m));
writer.newLine();
```

```
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
writer.write(Float.toString(matrix.get(i * m + j)));
if (j == m - 1) {
writer.newLine();
} else {
writer.write(' ');
}
}
}

return true;
} catch (IOException e) {
System.err.println("Error when writing similarity file: " + output);
e.printStackTrace();
return false;
}
}

public static void main(String[] args) {
int width = 0;
int height = 0;
int dpi = 0;
int n = 0;
List<Minutia> minutiae = new ArrayList<>();
System.out.println(loadMinutiaeFromFile("/home/misha/IdeaProjects/MCC
java/src/main/java/org/example/1_1.txt",
width, height, dpi, n, minutiae));

}
}
```

Додаток В

```
package comparing;

import mcc.Matcher;
import template.TemplateMatcher;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Compairing {
    public static void main(String[] args) throws IOException {
        Map<Integer, Integer> amountMap = new HashMap<>() {{
            put(1, 420);
            put(2, 840);
            put(3, 1260);
            put(4, 1680);
        }};
        int userChoice = 0;
        Integer amountOfFingerPrints;
        Integer candidateNumber;
        while (userChoice != 5) {
            printMainMenu();
            Scanner sc = new Scanner(System.in);
            userChoice = sc.nextInt();
            switch (userChoice) {
```

```
            case 1 -> {
                amountOfFingerPrints = printAmountOfFingerprintsMenu(sc);

                if (handleInvalidAmountChoice(amountOfFingerPrints)) {
                    amountOfFingerPrints = amountMap.get(amountOfFingerPrints);
                    candidateNumber = printCandidateChoiceMenu(sc);

                    if (handleInvalidCandidateChoice(candidateNumber)) {
                        Long templateTime =
TemplateMatcher.timeComparingOnlyAlgorithm(candidateNumber, amountOfFingerPrints);
                        Long mccTime = Matcher.timeComparingOnlyAlgorithm(candidateNumber,
amountOfFingerPrints);

                        printResults(Metric.TIME_ALGORITHM, templateTime, mccTime);
                    }
                }
            }
            case 2 -> {
                amountOfFingerPrints = printAmountOfFingerprintsMenu(sc);

                if (handleInvalidAmountChoice(amountOfFingerPrints)) {
                    amountOfFingerPrints = amountMap.get(amountOfFingerPrints);
                    candidateNumber = printCandidateChoiceMenu(sc);

                    if (handleInvalidCandidateChoice(candidateNumber)) {
                        Long templateTime =
TemplateMatcher.timeComparingWithDBLoading(candidateNumber, amountOfFingerPrints);
                        Long mccTime = Matcher.timeComparingWithDBLoading(candidateNumber,
amountOfFingerPrints);

                        printResults(Metric.TIME_FULL, templateTime, mccTime);
```

```
            }
        }
    }

    case 3 -> {
        amountOfFingerPrints = printAmountOfFingerprintsMenu(sc);

        if (handleInvalidAmountChoice(amountOfFingerPrints)) {
            amountOfFingerPrints = amountMap.get(amountOfFingerPrints);
            candidateNumber = printCandidateChoiceMenu(sc);

            if (handleInvalidCandidateChoice(candidateNumber)) {
                Long templateMemory =
TemplateMatcher.comparingMemoryOnlyAlgorithm(candidateNumber, amountOfFingerPrints);
                Long mccMemory =
Matcher.comparingMemoryOnlyAlgorithm(candidateNumber, amountOfFingerPrints);

                printResults(Metric.MEMORY_ALGORITHM, templateMemory, mccMemory);
            }
        }
    }
    case 4 -> {
        amountOfFingerPrints = printAmountOfFingerprintsMenu(sc);

        if (handleInvalidAmountChoice(amountOfFingerPrints)) {
            amountOfFingerPrints = amountMap.get(amountOfFingerPrints);
            candidateNumber = printCandidateChoiceMenu(sc);

            if (handleInvalidCandidateChoice(candidateNumber)) {
                Long templateMemory =
```

```
TemplateMatcher.comparingMemoryWithDBLoading(candidateNumber, amountOfFingerPrints);
                    Long mccMemory =
Matcher.comparingMemoryWithDBLoading(candidateNumber, amountOfFingerPrints);
printResults(Metric.MEMORY_FULL, templateMemory, mccMemory);
                }
            }
        }
        case 5 -> System.out.println("Goodbye!");
        default -> handleInvalidTypeTestChoice(userChoice);
      }
    }
  }

  private static void printMainMenu() {
    System.out.println("----------Menu---------");
    System.out.println("1.Speed test(Only algorithm comparing)");
    System.out.println("2.Speed test(Algorithm comparing + loading pictures to template)");
    System.out.println("3.Memory test(Only algorithm comparing)");
    System.out.println("4.Memory test(Algorithm comparing + loading pictures to template)");
    System.out.println("5.Exit");
  }

  private static Integer printAmountOfFingerprintsMenu(Scanner sc) {
    System.out.println("---------- Amount Of Fingerprints Menu---------");
    System.out.println("1.420");
    System.out.println("2.840");
    System.out.println("3.1260");
    System.out.println("4.1680");
    return sc.nextInt();
  }
```

```
    private static Integer printCandidateChoiceMenu(Scanner sc) {
        System.out.println("----------Candidate choice---------");
        System.out.println("PLease, enter number between 1 and 140. Enter 0 to get ro previous
menu.");
        return sc.nextInt();
    }

    private static boolean handleInvalidCandidateChoice(Integer userChoice) {
        if (userChoice < 1  userChoice > 140)
            System.out.println("Your choice is incorrect. Please try again or enter 0 to get to previous
menu.");
        return userChoice > 0 && userChoice < 141;
    }

    private static void handleInvalidTypeTestChoice(Integer userChoice) {
        if (userChoice < 1 &&  userChoice > 5)
            System.out.println("Your choice is incorrect. Please try again.");
    }

    private static boolean handleInvalidAmountChoice(Integer userChoice) {
        if (userChoice < 1 || userChoice > 4)
            System.out.println("Your choice is incorrect. Please try again.");
        return userChoice > 0 && userChoice < 5;
    }

    private static void printResults(Metric metric, Long templateResult, Long mccResult) {
        switch (metric) {
            case TIME_ALGORITHM -> {
                System.out.println("Metric: Time algorithm");
```

```
            System.out.println("Template: " + templateResult);
            System.out.println("MCC: " + mccResult);
        }
        case TIME_FULL -> {
            System.out.println("Metric: Time full");
            System.out.println("Template: " + templateResult);
            System.out.println("MCC: " + mccResult);
        }
        case MEMORY_ALGORITHM -> {
            System.out.println("Metric: Memory algorithm");
            System.out.println("Template: " + templateResult);
            System.out.println("MCC: " + mccResult);

        }
        case MEMORY_FULL -> {
            System.out.println("Metric: Memory full");
            System.out.println("Template: " + templateResult);
            System.out.println("MCC: " + mccResult);
        }
      }
    }

    public enum Metric {
        TIME_ALGORITHM,
        TIME_FULL,
        MEMORY_ALGORITHM,
        MEMORY_FULL
    }
}
```