Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

**Бакалаврська дипломна робота на тему:**

«Порівняльний аналіз методів нечіткого гешування»

Виконав: студент 4 курсу групи 1БС-20 б
спеціальності 125 Кібербезпека
_____ В'ячеслав МАЦЬКИЙ

Керівник: к. т. н., професор каф. ЗІ
_____ Наталія КОНДРАТЕНКО

«17» червня 2024 р.

Рецензент: к. т. н. доцент каф. ПЗ
_____ Ганна РАКИТЯНСЬКА
«17» червня 2024 р.

**Допущено до захисту**
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«17» червня 2024 р.

Вінниця ВНТУ – 2024 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри ЗІ, д. т. н., проф.**
**___Володимир ЛУЖЕЦЬКИЙ**
**«14» березня 2024 року**

# ЗАВДАННЯ
# НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Мацькому В'ячеславу Юрійовичу

1. Тема роботи: «Порівняльний аналіз методів нечіткого гешування»
   керівник роботи: Кондратенко Наталія Романівна, к. т. н., професор кафедри ЗІ,
   затверджені наказом ректора ВНТУ від 11 березня 2024 року №80.
2. Строк подання студентом роботи 17.06. 2024 р.
3. Вихідні дані до роботи:
   - Існуючі методи криптографічного гешування;
   - Існуючі методи нечіткого гешування;
   - Класифікації алгоритмів криптографічного та нечіткого гешування;
   - Існуючі методи виявлення нечітких дублікатів.
4. Зміст текстової частини: Вступ. Аналіз методів криптографічного та нечіткого гешування. Побудова базового методу та вибір модифікації. Опис програмної реалізації методу. Висновки. Перелік використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: Схема роботи алгоритму MD5 (плакат, А4). Схема однієї ітерації алгоритмів SHA-1 (плакат, А4). Схема однієї ітерації алгоритмів SHA-2 (плакат, А4). Схема роботи алгоритму SHA-3 (плакат, А4). Схема роботи сигнатурних методів (плакат, А4). Схема генерації нечіткого геш-значення в методі нечіткого гешування (плакат, А4).

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| 1 | Кондратенко Н.Р., к.т.н., проф. каф.ЗІ | 11.03.24 | 31.03 |
| 2 | Кондратенко Н.Р., к.т.н., проф. каф.ЗІ | 11.03.24 | 15.04.2 |
| 3 | Кондратенко Н.Р., к.т.н., проф. каф.ЗІ | 11.03.24 | 04.05.2 |

7. Дата видачі завдання 14.03. 2024 року.

КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів бакалаврської дипломної роботи | Строк виконання етапів роботи | Приміт |
|---|---|---|---|
| 1 | Аналіз завдання. Вступ | 12.03.24 - 14.03.24 | |
| 2 | Аналіз літературних джерел за напрямком бакалаврської дипломної роботи | 15.03.24 - 31.03.24 | |
| 3 | Порівняльний аналіз | 01.04.24 - 15.04.24 | |
| 4 | Практична реалізація, моделювання, експериментування, результати | 16.04.24 - 01.05.24 | |
| 5 | Розробка розділу тестування і обґрунтування доцільності розробки | 02.05.24 - 10.05.24 | |
| 6 | Аналіз виконання завдання, висновки | 30.05.24 - 04.06.24 | |
| 7 | Оформлення пояснювальної записки | 10.06.24 - 12.06.24 | |
| 8 | Попередній захист та доопрацювання БДР | 13.06.24 - 14.06.24 | |
| 9 | Представлення БДР до захисту | 18.06.24 - 19.06.24 | |
| 10 | Захист БДР | 19.06.24 - 21.06.24 | |

Студент _____ В'ячеслав МАЦЬКИ

Керівник роботи _____ Наталія КОНДРАТЕНК

# АНОТАЦІЯ

Бакалаврська дипломна робота складається з 61 сторінки формату А4, на яких є 10 рисунків, 5 таблиць, список використаних джерел містить 18 найменувань.

Бакалаврська робота присвячена порівняльному аналізу методів нечіткого гешування та розробці покращеного методу для виявлення нечітких дублікатів. В ході дослідження був проведений детальний аналіз існуючих алгоритмів криптографічного та нечіткого гешування для виявлення найбільш результативного методу. Дослідження охопило переваги та недоліки кожного з розглянутих алгоритмів, що дозволило зробити обґрунтований вибір базового методу виявлення нечітких дублікатів.

У рамках роботи було розроблено модифікацію базового методу виявлення нечітких дублікатів. Ця модифікація використовує основні етапи базового методу, проте робить це більш результативно.

Розроблена модифікація дозволяє забезпечити швидкість пошуку на великих об’ємах даних, враховувати синонічні входження та забезпечити максимальну унікальність термів методом уникнення повторень.

Ключові слова: нечітке гешування, нечіткі дублікати.

## ABSTRACT

The bachelor's thesis consists of 61 pages in A4 format, on which there are 10 figures, 5 tables, a list of words containing 18 names.

The bachelor's work is dedicated to the thorough analysis of fuzzy methods and the development of the colored method for identifying fuzzy duplicates. During the investigation, a detailed analysis of the main cryptographic and fuzzy algorithms was carried out to identify the most effective method. The research took into account the advantages and shortcomings of the skin from the examined algorithms, which allowed the development of a basic method for identifying fuzzy duplicates.

The work included a modification of the basic method for identifying fuzzy duplicates. This modification follows the basic steps of the basic method and is more effective.

The expanded modification makes it possible to ensure the fluidity of search on large volumes of data, to ensure synonic input and to ensure maximum uniqueness of terms using the method of unique repetition.



Keywords: unclear translation, unclear duplicate.

# ЗМІСТ

ВСТУП ..... 3

1 АНАЛІЗ МЕТОДІВ КРИПТОГРАФІЧНОГО ТА НЕЧІТКОГО ГЕШУВАННННЯ 5

1.1 Методи криптографічного гешування ..... 5
1.2 Методи нечіткого гешування ..... 10
1.3 Проблеми методів нечіткого гешування ..... 12

2 ПОБУДОВА БАЗОВОГО МЕТОДУ ТА ВИБІР МОДИФІКАЦІЇ ..... 15

2.1 Аргументація вибору базового методу ..... 15
2.2 Можливості покращення методу ..... 16
2.3 Модифікація методу виявлення нечітких дублікатів ..... 18
2.4 Особливості алгоритму реалізації модифікованого методу ..... 19

3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДУ ..... 24

3.1 Опис засобів розробки програмного забезпечення ..... 24
3.2 Архітектура розробленого програмного забезпечення ..... 33
3.3 Особливості програмної реалізації застосунку ..... 35
3.4 Аналіз отриманих результатів ..... 38
3.5 Вибір наборів даних для тестування розробленого методу ..... 39
3.6 Результати роботи модифікованого методу ..... 41

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 48

ДОДАТКИ ..... 51

Додаток А ..... 52
Додаток Б ..... 53

# ВСТУП

Гешування є однією з ключових технологій у комп'ютерних науках та інформаційних системах. Від ефективного гешування залежать продуктивність, безпека та надійність багатьох систем, включаючи бази даних, системи управління пам'яттю, мережеві протоколи та криптографічні застосування.

Геш-таблиці дозволяють швидко знаходити елементи у великих наборах даних, знижуючи час доступу до них. Контрольні суми та геш-функції використовуються для виявлення змін у даних, що є критично важливим для зберігання та передачі інформації. Геш-функції є основою багатьох криптографічних алгоритмів, забезпечуючи безпеку цифрових підписів, аутентифікації та інших методів захисту даних.

З розвитком технологій збільшуються обсяги даних, а також вимоги до швидкості та безпеки обробки інформації. Це створює нові виклики для методів гешування.

Геш-функції повинні ефективно працювати з великими обсягами даних без втрати продуктивності. Мінімізація ймовірності зіткнень (коли різні вхідні дані дають однаковий геш) є критично важливою для багатьох застосувань, особливо в криптографії. Сучасні алгоритми гешування повинні бути стійкими до різних видів атак, таких як атаки на колізії та обернені атаки. Геш-функції повинні забезпечувати високу швидкість обчислень при мінімальних ресурсах пам'яті.

Порівняльний аналіз методів гешування має на меті розглянути продуктивність, стійкість до атак, та інші характеристики різних геш-функцій, визначити, які методи гешування найкраще підходять для конкретних завдань та умов експлуатації, визначити слабкі місця сучасних алгоритмів та запропонувати шляхи їх поліпшення.

Актуальність теми порівняльного аналізу методів гешування зумовлена зростаючими вимогами до продуктивності, безпеки та ефективності інформаційних систем. Розуміння переваг та недоліків різних методів дозволить розробникам

вибирати оптимальні рішення для конкретних задач, сприяючи розвитку технологій та забезпечуючи надійність і безпеку сучасних інформаційних систем.

Об'єктом дослідження є процес порівняння реальних систем та програмних засобів, в яких застосовуються методи гешування. Це можуть бути бази даних, файлові системи, мережеві протоколи, системи контролю версій, криптографічні системи та інші інформаційні системи.

Предметом дослідження є методи гешування, включаючи їх алгоритмічні основи, принципи роботи, характеристики та сфери застосування.

Метою дослідження є проведення всебічного порівняльного аналізу методів гешування з урахуванням їхніх продуктивності, стійкості до атак, ефективності обчислень і придатності для різних завдань. До конкретних завдань дослідження належать:

1. Проаналізувати методи криптографічного та нечіткого гешування.
2. Сформулювати модифікацію.
3. Розглянути опис програмної реалізації обраного методу.
4. Проаналізувати отримані дані.

# 1 АНАЛІЗ МЕТОДІВ КРИПТОГРАФІЧНОГО ТА НЕЧІТКОГО ГЕШУВАННННЯ

## 1.1 Методи криптографічного гешування

Криптографічне гешування є ключовим аспектом сучасної криптографії та інформаційної безпеки, основною метою якого є перетворення вхідних даних довільного розміру у фіксований розмір гешу, що забезпечує ефективну перевірку цілісності та автентичності даних. У цій частині було розглянуто основні методи криптографічного гешування, їх властивості, переваги та недоліки, а також приклади застосування.

MD5, розроблений Рональдом Рівестом у 1991 році, є одним із найпопулярніших і широко використовуваних алгоритмів гешування. Алгоритм починає з ініціалізації чотирьох 32-бітних змінних. Ці змінні визначаються константами:

A = 0x67452301;

B = 0xEFCDAB89;

C = 0x98BADCFE;

D = 0x10325476;

Повідомлення доповнюється таким чином, щоб його довжина стала кратною 512. Це робиться у два кроки:

Додається один біт зі значенням 1.

Додається стільки нульових бітів, скільки необхідно, щоб довжина повідомлення стала 64 біти менше, ніж кратна 512.

У останні 64 біти додається початкова довжина повідомлення (у бітах), подана у вигляді 64-бітного числа з молодшими байтами вперед.

Повідомлення розбивається на блоки по 512 біт (16 слів по 32 біти). Кожен блок обробляється окремо за допомогою основного циклу. Для кожного блоку застосовується основний цикл, який складається з чотирьох раундів, кожен з яких має по 16 операцій.

Основний цикл використовує такі нелінійні функції:

F (B, C, D) = (B & C) | (~B & D);

G (B, C, D) = (B & D) | (C & ~D);

H (B, C, D) = B ^ C ^ D;

I (B, C, D) = C ^ (B | ~D);

Кожна операція складається з таких кроків:

Визначення функції і операндів.

Додавання результату функції до поточного значення одного з чотирьох основних змінних (A, B, C, D).

Додавання значення блоку і константи.

Ліва циклічна ротація отриманого значення на певне число бітів.

Додавання результату до однієї з основних змінних.

Після кожного блоку результуючі значення змінних A, B, C, D додаються до значень попереднього блоку. Після обробки всіх блоків, отримані значення A, B, C, D об'єднуються для формування кінцевого 128-бітного (16-байтного) гешу. Ці значення об’єднуються у маленькому порядку байтів.

Схема роботи алгоритму MD5 зображена на рисунку 1.1

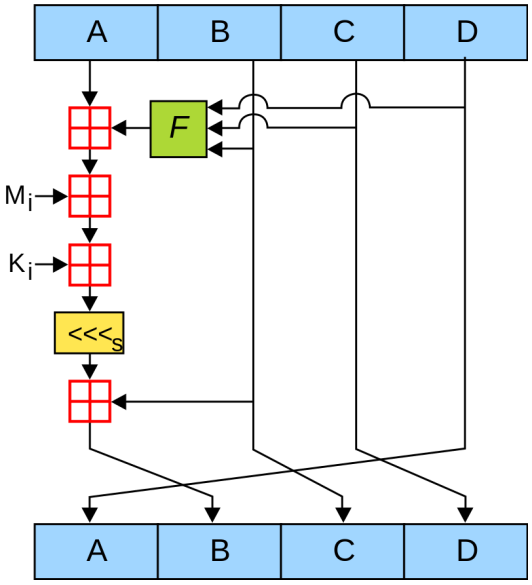



Рисунок 1.1 – Схема роботи алгоритму MD5

MD5 досить застарілий, але доволі відомий своєю швидкістю та ефективністю, що зробило його популярним для перевірки цілісності файлів і повідомлень. Ця швидкість досягається завдяки відносно простому алгоритму

обчислення гешу. Однак головним недоліком MD5 є його вразливість до колізій. Це означає, що можуть бути створені два різних повідомлення з однаковим гешем, що підриває безпеку цього алгоритму. Незважаючи на цей недолік, MD5 все ще використовується у певних невимогливих до безпеки потреб, наприклад, для перевірки цілісності завантажених файлів [1].

SHA-1, розроблений Національним інститутом стандартів і технологій США (NIST) у 1993 році. Вхідне повідомлення розбивається на блоки по 512 біт у кожному. Останній блок доповнюється до довжини кратної 512 біт. Спочатку додається 1, а потім нулі, щоб довжина блоку стала рівною (512-64=448) біт. В останні 64 біта записується довжина вихідного повідомлення у бітах. Якщо останній блок має довжину понад 448, але менше 512 біт, то додаток виконується в такий спосіб: спочатку додається 1, потім нулі аж до кінця 512-бітного блоку; після цього створюється ще один 512-бітний блок, який заповнюється аж до 448 біта нулями, після чого в 64 біта, що залишилися, записується довжина вихідного повідомлення в бітах. Доповнення останнього блоку здійснюється завжди, навіть якщо повідомлення вже має потрібну довжину.

Схема однієї ітерації алгоритмів SHA-1 зображена на рисунку 1.2

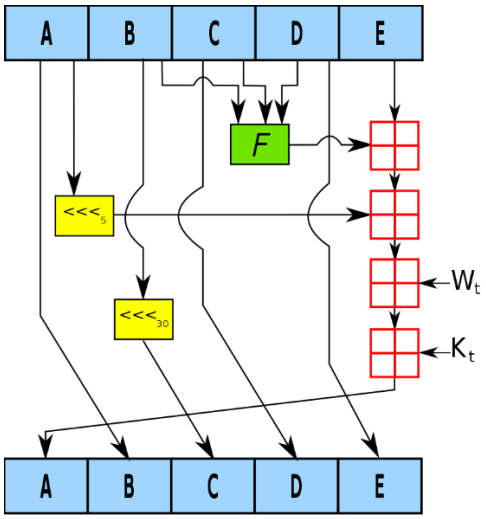



Рисунок 1.2 – Схема однієї ітерації алгоритмів SHA-1

На момент розробки SHA-1 вважався досить безпечним, але з часом були виявлені його вразливості до колізій. У 2005 році були опубліковані результати

досліджень, що демонструють можливість ефективних атак на SHA-1, що призвело до втрати довіри до цього алгоритму. Попри це, SHA-1 широко використовувався в сертифікатах SSL і цифрових підписах, але через вразливості був замінений більш безпечними алгоритмами. До 2017 року більшість організацій перейшли на використання SHA-2 або SHA-3 [2].

SHA-2 є сімейством геш-функцій, включаючи SHA-224, SHA-256, SHA-384, і SHA-512, розроблених як покращення над SHA-1. Початкове повідомлення після доповнення розбивається на блоки, кожен блок — на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64-ма чи 80-ма ітераціями (раундами). На кожній ітерації 2 слова перетворюються, функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням геш-функції.

Схема однієї ітерації алгоритмів SHA-2 зображена на рисунку 1.3

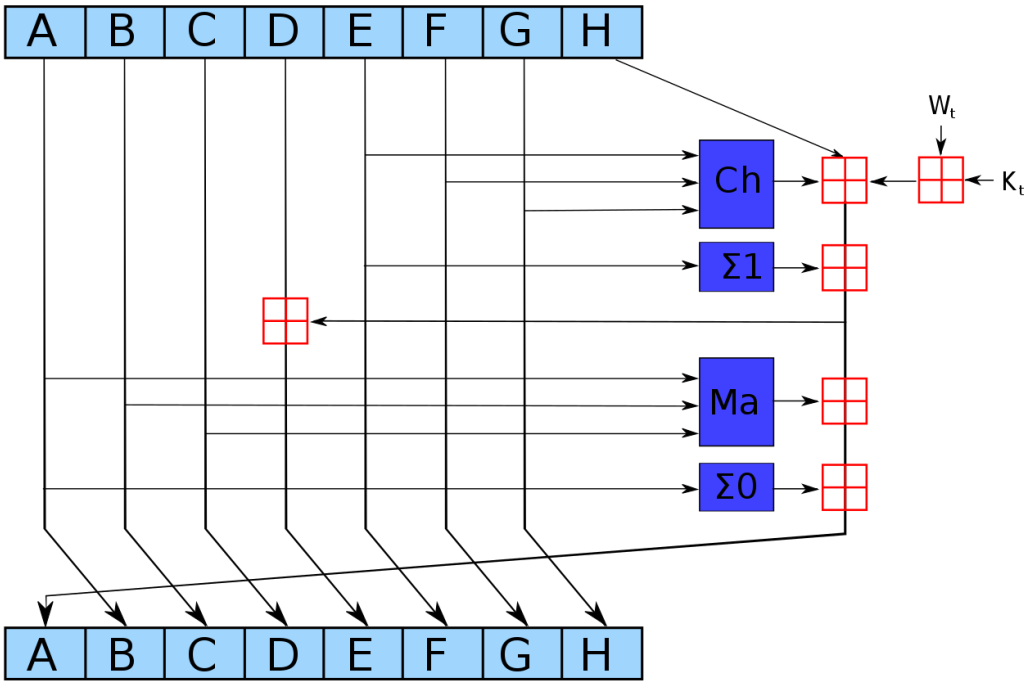



Рисунок 1.3 – Схема однієї ітерації алгоритмів SHA-2

Значно краща безпека порівняно з MD5 і SHA-1 забезпечила SHA-2 стійкість до колізій і більш ефективне застосування для криптографічних додатків. Ці геш-

функції використовують складнішу архітектуру, що робить їх менш вразливими до атак. SHA-2 широко використовується в різних галузях, включаючи електронну комерцію, блокчейн і цифрові підписи. Наприклад, Bitcoin використовує SHA-256 для забезпечення безпеки транзакцій у своїй блокчейн-мережі. Різні варіанти SHA-2 (SHA-224, SHA-256, SHA-384, SHA-512) дозволяють вибирати баланс між швидкістю та рівнем безпеки відповідно до конкретних потреб.

SHA-3 (Keccak) був затверджений NIST у 2015 році як новий стандарт гешування, що доповнює сімейство SHA-2 [3].

Keccak заснований на конструкції Sponge. Це означає, що для отримання гешу потрібно виконати такі дії: Взяти вихідне повідомлення M і доповнити його до довжини кратної r. У вигляді формули (1.1) їх можна зобразити в такий спосіб:

$$M=M||0x01||0x00||..||0x00||0x80. \qquad (1.1)$$

де M – вихідне повідомлення.

Тобто до повідомлення дописується одиничний байт, необхідна кількість нулів та завершується байт зі значенням 0x80, проте це справедливо лише для випадків, коли додається більше одного байту.

Однак у разі, якщо необхідно доповнити лише один байт, достатньо додати лише 0x81. Потім для кожного блоку Mi довжиною r біт виконуємо:

Додавання по модулю 2 з першими r-бітами набору початкових станів S. Перед початком роботи функції всі елементи S дорівнюватимуть нулю.

N разів застосовуємо до отриманих в результаті даних функції f. Набір початкових станів S для блоку Mi+1 буде результатом останнього раунду блоку Mi.

Після того, як всі блоки Mi закінчаться взяти підсумковий результат і повернути його як геш-значення.

Схема роботи алгоритму SHA-3 зображена на рисунку 1.4.

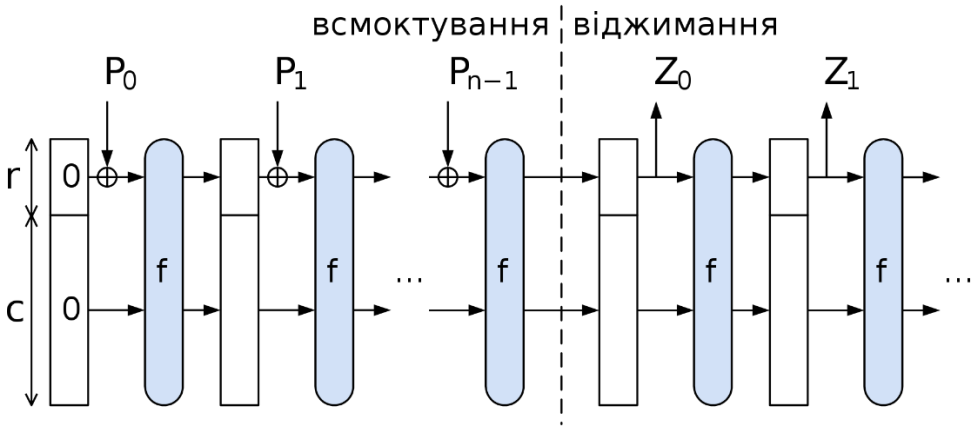



Рисунок 1.4 – Схема роботи алгоритму SHA-3

Використання алгоритму Keccak забезпечує різноманітність архітектури порівняно з попередніми SHA, що знижує ймовірність вразливостей. Алгоритм розроблений з урахуванням уроків, отриманих із недоліків попередніх алгоритмів, і є одним із найсучасніших і найбезпечніших методів гешування. Він включає механізми захисту від різних атак, включаючи атаки на структуру самого алгоритму . SHA-3 починає знаходити застосування у нових криптографічних рішеннях, де потрібна максимальна безпека. Він також розглядається як потенційна альтернатива SHA-2 у випадку виявлення нових вразливостей у останнього.

Методи криптографічного гешування є основоположними для забезпечення цілісності та безпеки даних у цифровому світі. Кожен з описаних методів має свої переваги та недоліки, що впливають на їх вибір у конкретних застосуваннях. З розвитком технологій і підвищенням рівня кіберзагроз важливо використовувати найсучасніші та найбезпечніші методи гешування для захисту інформації. Перехід від MD5 і SHA-1 до SHA-2 і SHA-3 демонструє постійний прогрес у розробці геш-функцій, які відповідають вимогам безпеки сьогодення.

### 1.2 Методи нечіткого гешування

Якщо в криптографічних геш-функціях суть алгоритму полягає в тому, що при найменшій зміні вхідних даних їх геш також значно змінюється, то в нечітких геш результат змінюється незначно або не змінюється зовсім. Тобто нечіткі геші

більш стійкі до невеликих змін у файлі. Тому подібні функції дозволяють набагато ефективніше виявляти нові модифікації шкідливого ПЗ і вимагають великих ресурсів до розрахунку.

Нечітке гешування - це метод, при якому програма обчислює шматкові геші від вхідних даних, тобто використовує так зване шматкове гешування, що контекстно викликається. В англомовних джерелах цей метод називається context triggered piecewise hashing (CTPH) [4].

Класифікацій нечітких гешів досить багато. Наприклад, за механізмом роботи алгоритми діляться на piecewise hashing, context triggered piecewise hashing, statistically improbable features, block-based rebuilding. За типом оброблюваної інформації їх можна розділити на побайтові, синтаксичні та семантичні. Але якщо йдеться про нечіткі геші, це, зазвичай, CTPH.

Алгоритм SSDeep [4] розроблений Джессі Корнблюмом для використання в комп'ютерній криміналістиці та заснований на алгоритмі spamsum. SSDeep обчислює кілька традиційних криптографічних гешів фіксованого розміру для окремих сегментів файлу і тим самим дозволяє виявляти схожі об'єкти. Алгоритм використовує механізм ковзного вікна для обчислення кільцевого гешу. Такий тип гешу був обраний з метою забезпечення більшої продуктивності, так як він дозволяє швидко отримати геш наступного вікна, якщо відомий геш попереднього. Байти, через які проходить ковзне вікно, додаються в деякий буфер, що представляє черговий “шматок”. Далі, коли чергове значення кільцевого гешу задовольняє умову, спрацьовує тригер до гешування поточного буферу. Буфер гешується, а молодші 6 біт отриманого гешу переводяться в кодування base64 та додаються до результуючого гешу.

Не менш відомим є й алгоритм sdhash. Геш, використовуваний а даному алгоритмі, заснований на ідеї будування гешу не на файлі в цілому, а на його найбільш характерних особливостях. Так, використовується ковзне вікно, розраховуючи на для кожної ділянки значення ентропії. Далі по отриманим значенням проходить інше ковзне вікно, яке вибирає серед своїх елементів найбільш характерне. Ці елементи, зберігаються в геш, але не цілком. Замість цього

для їх збереження використовується набір фільтрів Блума [5], які й представляють собою результуючий геш. Цей нечіткий геш дозволяє вирішити більшість проблем пов’язаних з SSDeep, і є другим по частоті використання нечітким гешем.

Алгоритм mrsh [6] був створений як спроба поєднати в собі SSDeep і sdhash, зберігши переваги обох та усунувши їх недоліки. Так само, як і в SSDeep, шматкові геші обчислюються при спрацьовуванні певної умови, проте результат не конкатенується до спільного результуючого гешу, а заноситься в блум-фільтри з фіксованою кількістю елементів. Наступною зміною порівняно з оригінальним SSDeep є заміна гешів, використовуваних в SSDeep, на більш криптостійкі.

Алгоритм mrsh-v2 [7] є розвитком mrsh. Зокрема, mrsh не виправдав ідею використання криптографічно стійких гешів, в зв’язку із чим в mrsh-v2 використовується кільцеве гешування як і в SSDeep.

Для роботи алгоритму bbhash [8] використовується деяке попередньо задана ініціалізуюча множина блоків. Ціль алгоритму – виявити ознаки того, що в аналізованих файлах містяться дані, які не є спільними для аналізованої множини. Робота алгоритму полягає в поблочній обробці файлу. Кожен черговий блок порівнюється зі всіма блоками з ініціалізаціоної множини. Порівняння відбувається за допомогою вимірювання відстані Хеммінга. Вибирається мінімальне значення з усіх, при цьому значення додається в результуючий геш тільки у випадку, якщо воно менше деякого порогового значення. Таким чином, в роботі даного алгоритму прослідковується ідея пошуку малоймовірних особливостей. Безліч блоків ініціалізації може бути обрано випадково, проте для коректного порівняння двох файлів безлічі блоків, використовуваних при розрахунку їх гешів, повинні бути однаковими. Для порівняння блоків може використовуватись як механізм порівняння, використовуваний в sdhash, так і механізм порівняння із SSDeep.

## 1.3 Проблеми методів нечіткого гешування

Нечітке гешування, незважаючи на свої значні переваги у роботі з даними, що піддаються незначним змінам або шумам, має також і низки проблем, які обмежують його ефективність і застосування в реальних умовах.

Однією з головних проблем нечіткого гешування є забезпечення балансу між стійкістю до змін і унікальністю гешів. Занадто нечітке гешування може призвести до ситуації, коли різні вхідні дані генерують однакові геші, що призводить до колізій. Це може бути особливо критично у випадках, коли точність ідентифікації є вирішальною, наприклад, у біометричних системах. З іншого боку, недостатньо нечітке гешування може не забезпечити необхідної стійкості до невеликих змін у вхідних даних, що знижує його ефективність у практичних застосуваннях.

Іншою важливою проблемою є потреба у великих обчислювальних ресурсах. Методи нечіткого гешування, такі як нейронні мережі або складні алгоритми локально-чутливого гешування, потребують значних обчислювальних потужностей для обробки даних і створення гешів. Це може бути серйозним обмеженням для застосування таких методів у реальному часі або на пристроях з обмеженими ресурсами, таких як мобільні телефони або вбудовані системи.

Питання ефективності обробки файлів в реальному часі також є критичним, адже великі файли можуть оброблятись дуже повільно із використанням деяких функцій, наприклад bbhash обробляє файл порядку 10 Мб за сотні секунд, тоді як SSDeep – за десяті долі секунд. До того ж аналіз і маленьких файлів є цілком неможливим внаслідок того, що довжина будівельного блока перевищує довжину файла. Проблемою також є адаптація нечіткого гешування до різноманітних типів даних. Хоча деякі методи ефективно працюють із зображеннями або біометричними даними, вони можуть бути менш ефективними при роботі з іншими типами даних, такими як аудіо або текст. Наприклад, алгоритми, що добре справляються з зображеннями, можуть не враховувати специфічні особливості аудіосигналів, такі як частотний спектр або тимчасові зміни. Це вимагає розробки спеціалізованих алгоритмів для кожного типу даних, що збільшує складність і вартість впровадження нечіткого гешування.

Окремо варто визначити критичність конфіденційності даних і інтеграції нечіткого гешування в існуючі системи. Використання нечітких гешів може вимагати зберігання додаткової інформації, яка допомагає відновити або перевірити оригінальні дані. Наприклад, у біометричних системах можуть зберігатися проміжні дані, які використовуються для перевірки автентичності. Це створює ризики для конфіденційності, оскільки зловмисники можуть використовувати ці дані для відновлення або підробки біометричних характеристик. Забезпечення конфіденційності таких проміжних даних є важливим завданням для розробників систем нечіткого гешування. Тим не менш, багато організацій вже використовують стійкі методи гешування, такі як SHA-2 або SHA-3, і перехід на нечітке гешування може вимагати значних змін у інфраструктурі та програмному забезпеченні. Це включає оновлення алгоритмів гешування, адаптацію баз даних для зберігання нечітких гешів та навчання персоналу. Високі витрати на такі зміни можуть стримувати організації від впровадження нечіткого гешування, навіть якщо воно може запропонувати покращену функціональність.

Результатом аналізу файлів різного розміру будуть нечіткі підписи різної довжини. І коректне співставлення їх є додатковою задачею [9-10].

# 2 ПОБУДОВА БАЗОВОГО МЕТОДУ ТА ВИБІР МОДИФІКАЦІЇ

## 2.1 Аргументація вибору базового методу

Методи, що використовують шингли [11], залучають підмножини послідовних термінів із вхідних документів для їх порівняння та визначення частини спільних шинглів. Основний недолік цих методів полягає у великій кількості порівнянь, що необхідні. Зі збільшенням розміру шинглів (кількості термінів у шинглі) алгоритмічна складність зростає квадратично. Проблема стає особливо помітною при порівнянні більше двох документів для оцінки їх подібності. Однак однією з переваг є можливість визначення міри подібності у відсотках.

Інша група методів застосовує стандартні метрики відстані, що використовуються в комп'ютерній лінгвістиці для порівняння термінів у текстах і оцінки міри їх подібності. Водночас, ці методи часто демонструють недостатню стійкість і не завжди ефективні для обробки великих обсягів даних.

Кожна з цих стратегій порівняння текстів має свої плюси та мінуси, і вибір між ними залежить від конкретних вимог задачі, таких як тип текстів, обсяг даних і вимоги до швидкодії алгоритму.

Сигнатурний підхід використовує спеціальну геш-функцію для обчислення унікального значення для кожного документа. Якщо два документи мають однакове значення гешу, вони вважаються дублікатами [12]. Головна перевага цього методу — використання широко відомих і часто оптимізованих алгоритмів гешування.

Втім, головний недолік сигнатурних методів полягає в їхній нездатності точно визначити міру подібності між двома документами. Ці методи лише вказують, чи є документи ідентичними, але не здатні визначити відсоткову міру подібності.

Сигнатурні методи також відрізняються за способами попередньої обробки документів перед застосуванням геш-функції. Загалом, схема роботи сигнатурних методів зображена на рисунку 2.1.

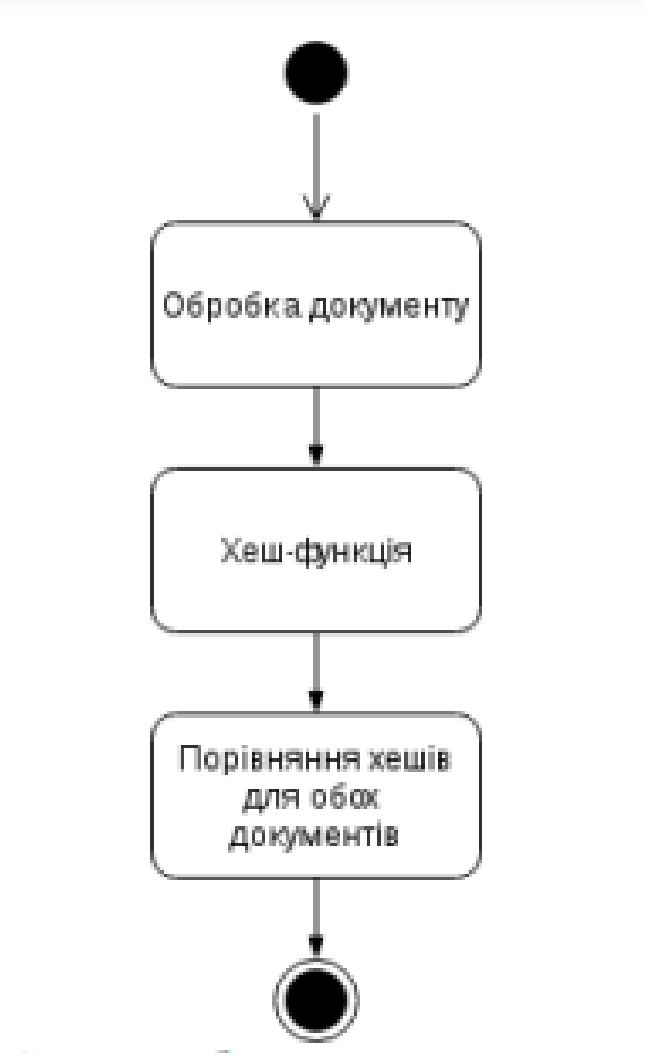



Рисунок 2.1 – Схема роботи сигнатурних методів

Метод I-Match відзначається високою швидкістю та точністю серед сигнатурних методів. Ця ефективність досягається завдяки попередньому розрахунку таблиці IDF-значень на заздалегідь вибраному корпусі документів, відомому як «лексикон». Цей «лексикон» не залежить від документів, що порівнюються, і може бути підготовлений розробником на етапі реалізації методу.

Це відкриває можливості для модифікацій як під час створення «лексикону», так і при безпосередньому обчисленні сигнатур для порівнюваних документів.

## 2.2 Можливості покращення методу

Для визначення можливостей покращення базового методу варто детально розглянути сам метод I-Match [13] та його наявні модифікації. Основні етапи базового методу I-Match включають:

1. На підготовчому етапі обчислюється так званий «лексикон» — таблиця IDF-значень, обчислених на великому корпусі документів. Величина корпусу значно впливає на точність методу, оскільки після обчислення пропонується відкинути найвищі і найнижчі значення IDF, щоб уникнути впливу на дактилограму документа частовживаних слів, сполучників тощо.
2. Для документів, що порівнюються, визначають набір унікальних термінів. Порядок слів та речень у тексті не впливає на значення дактилограми. Рекомендується також видалити знаки пунктуації, стоп-слова, провести стемінг та видалити слова зі спеціальними символами.
3. Для кожного терміна у вхідних документах знаходиться значення IDF у «лексиконі». Це значення IDF подається на вхід геш-функції, яка працює в кумулятивному режимі, поступово накопичуючи та переобчислюючи дактилограму документа.
4. В результаті кожен документ отримує значення геш-функції — «відбиток». Якщо у порівнюваних документів ці значення збігаються, вони вважаються дублікатами.

Одна з модифікацій рекомендує відкидати значення з високими та низькими IDF [13], оскільки це дозволяє виключати не лише дуже рідкісні слова, а й слова з помилками, підвищуючи стійкість методу до таких помилок. Порогові значення IDF мають бути визначені експериментально й залежать від конкретного корпусу документів. Наприклад, у наукових текстах виключення високих IDF може означати видалення важливих термінів, а в художніх текстах — рідкісних зворотів або авторських неологізмів.

Інша модифікація вирішує проблему низької ефективності при порівнянні великих документів, коли перетин між термінами документа і лексикону є незначним. Пропонується ввести додатковий «вторинний лексикон», який є

більшим за основний і не проходить фільтрацію термінів. Якщо пошук в основному лексиконі не дав результатів, виконується пошук у вторинному лексиконі. Експериментальні результати також свідчать, що змішування основного та додаткового лексиконів нормалізує значення IDF для різних корпусів документів.

Крім того, можливе створення кількох лексиконів для різних корпусів документів та обчислення кількох «відбитків» документів за допомогою різних лексиконів.

Отже, аналіз базового методу I-Match та його модифікацій показує, що модифікації, спрямовані на підвищення точності, зазвичай стосуються процесу обчислення початкового «лексикону» до початку роботи алгоритму обчислення «відбитку» документа. З іншого боку, процес обчислення «лексикону» хоча й враховує повторювані терміни, використовує прості методи порівняння термінів.

В оригінальній статті зазначено, що для обчислення «відбитку» документа слід використовувати алгоритм SHA-1 (NIST 1995), але не надається аргументація цього вибору. Тому, варто дослідити, як вибір геш-функції впливає на точність і швидкодію методу.

### 2.3 Модифікація методу виявлення нечітких дублікатів

Сформулюємо наступну модифікацію:

- Під час обчислення «лексикону» враховувати синонімічні колізії у вхідних документах. Це потребує зміни процедури порівняння термінів при їх включенні в таблицю «лексикону». Порівняння має передбачати пошук у словнику синонімів та виключення їх із результатуючої таблиці IDF. Словник синонімів слід реалізувати з використанням структур даних та алгоритмів, що дозволяють швидкий пошук у великих обсягах даних. Видалення синонімічних входжень з таблиці зменшить використання пам’яті та підвищить точність методу завдяки тому, що значення IDF для синонімів буде представлене одним значенням у таблиці «лексикону».

– На етапі попередньої обробки документу, що порівнюється, враховувати синонімічні входження при виділенні унікальних термінів. Це зменшить кількість унікальних термінів у документі, знижуючи затрати процесорного часу на його обробку.
– Перед обчисленням «відбитку» документа під час пошуку значення IDF у таблиці «лексикону» вважати синоніми однаковими термінами. Це дозволить уникнути повторень та забезпечить максимальну унікальність термінів, що подаються на вхід геш-функції.

## 2.4 Особливості алгоритму реалізації модифікованого методу

Отже, на основі запропонованої модифікації, слід використовувати швидкі алгоритми та ефективні структури даних для реалізації методу виявлення нечітких дублікатів у великих обсягах даних. Пошук інформації у програмній інженерії є складним та важливим завданням, оскільки він спрямований на виявлення документів або текстів, що відповідають заданим критеріям чи запитам. Це вимагає здійснення пошуку за ключовими словами або іншими визначеними параметрами для задоволення інформаційних потреб користувача.

Основна мета інформаційного пошуку полягає у задоволенні потреб користувача у отриманні необхідної інформації. Це може включати виявлення документів у великій колекції даних, які містять відповідну інформацію або відповідають запиту.

Завдання інформаційного пошуку постійно розширюються і включають такі аспекти:

– моделювання запитів;
– класифікація документів;
– фільтрація документів;
– кластеризація документів;
– проектування архітектури пошукових систем та інтерфейсів користувача;

- отримання інформації, включаючи анотування та реферування документів;
- використання мови запитів та інше.

Процес пошуку інформації включає послідовність операцій, спрямованих на збір, обробку та надання потрібної інформації користувачам. Етапи цього процесу:

- формулювання запиту природною мовою, вибір пошукових систем і сервісів, формалізація запиту згідно з відповідними інформаційно-пошуковими моделями;
- проведення пошуку в одній або кількох пошукових системах;
- перегляд отриманих результатів (посилань);
- попередня обробка результатів: перегляд змісту посилань, вилучення та збереження відповідних і релевантних даних;
- за необхідності, модифікація запиту і повторний пошук з подальшою обробкою результатів.

Інформаційний пошук використовує різні стратегії, методи, механізми і засоби для задоволення потреб користувачів у знаннях. Поведінка користувача, який здійснює пошук, визначається не лише інформаційними потребами, але й різноманіттям доступних інструментів.

Стратегія пошуку представляє загальний план або концепцію, яка визначає, як система або користувач буде задовольняти інформаційні потреби. Це включає вибір типу та методів пошуку, архітектурні рішення баз даних та інші стратегічні аспекти. Вибір оптимальної стратегії залежить від конкретних умов і обмежень, і на практиці часто вимагає компромісу між практичними потребами та доступними ресурсами.

Метод пошуку описує сукупність моделей і алгоритмів, які використовуються на різних етапах пошукового процесу. Це включає побудову пошукового образу запиту, відбір документів за цим образом, розширення або переформулювання запиту, локалізацію результатів і оцінку їх відповідності потребам користувача.

Пошуковий образ запиту є текстом, сформульованим за інформаційно-пошуковою моделлю (ІПМ), який відображає смисловий зміст інформаційного запиту і містить необхідні вказівки для ефективного пошуку. Методи пошуку, які виділяють підмножину документів, що потенційно містять відповіді на запит, є важливою частиною процесу і залежать від конкретного завдання і області дослідження.

Як ітеративний процес, пошук інформації часто включає методи скорочення простору пошуку, тобто визначення підмножини документів, які варто переглянути для досягнення максимально ефективного результату. Ці методи формують методологічну основу стратегії пошуку і можуть бути класифіковані на такі типи:

- пошук у єдиному просторі (зазвичай, за темою);
- ієрархічно упорядкований пошук;
- пошук в альтернативних просторах;
- динамічний пошук, що змінюється в процесі пошуку.

Ці методи дозволяють використовувати різні підходи до пошуку інформації, адаптуючи їх до специфіки конкретного завдання.

Метод побудови пошукового образу запиту повинен забезпечувати ефективні способи формулювання запитів для досягнення різноманітних цілей. Механізми пошуку описують моделі і алгоритми, які формують результати пошуку документів відповідно до запиту. Засоби пошуку включають інформаційно-пошукові мови і мови визначення/управління даними, які забезпечують обробку об’єктів, таких як документи, словники і набори результатів пошуку. Вони також включають об’єкти користувацького інтерфейсу, які дозволяють користувачеві керувати вибором операційних об’єктів в конкретній інформаційно-пошуковій системі (ІПС).

Пошукові технології — це уніфіковані послідовності для ефективного використання засобів пошуку під час взаємодії користувача з системою, для стабільного отримання кінцевих і проміжних результатів. Навігація в цьому контексті включає реалізацію пошуку за запитом у вибраній базі даних, з урахуванням стратегії використання методів, засобів і технологій конкретної ІПС

для отримання і оцінки результатів. Засоби навігації допомагають користувачам керувати процесом пошуку через інтерфейс, який спрощує взаємодію з базою даних, забезпечуючи вибір різних операційних об’єктів.

Процес пошуку інформації передбачає послідовність дій, які через систему приводять до певного результату і дозволяють оцінити його повноту. Оскільки користувач зазвичай не має повного розуміння інформаційного наповнення ресурсу, на якому він проводить пошук, він оцінює адекватність формулювання запиту та повноту отриманого результату, спираючись на зовнішні оцінки, проміжні результати і узагальнення, порівнюючи їх, наприклад, з попередніми.

Типологія пошукових завдань може бути класифікована залежно від ступеня семантичної невизначеності та співвідношення між відомим та невідомим у предметі пошуку на три основні типи:

1. Предметний (або атрибутивний) пошук — пошук конкретного об'єкта, про існування якого відомо (наприклад, фактографія або твори конкретного автора). Пошукова модель полягає в ідентифікації цього об'єкта за його атрибутами або відомими характеристиками.
2. Тематичний пошук — пошук інформації з певної теми або предметної області, наприклад, методів вирішення конкретної задачі. Тематичний пошук орієнтований на виявлення описів об'єктів, які пов'язані з темою і можуть бути визначені за відомими атрибутами. Модель пошуку означає пошук за частковим поняттям або зв'язками, що частково визначені комбінацією характеристичних ознак. Тематичний пошук реалізується як послідовність атрибутивних пошуків.
3. Проблемний пошук полягає у виявленні описів об'єктів або їх складових, які можуть існувати у основній діяльності та разом утворюють ціле, властивості якого перевищують суму властивостей його частин. Таким чином, ці властивості не є явними атрибутами, а нові властивості можуть бути визначені комбінацією вже відомих атрибутів. Логічна модель пошуку передбачає пошук «схожих» документів, зміст яких асоціюється з завданням користувача.

Види пошуку:

- Повнотекстовий пошук: пошук по всьому вмісту документа, прикладом якого є пошукові системи Інтернету, такі як Google чи Bing. Для прискорення пошуку зазвичай використовують попередньо побудовані індекси, серед яких найпоширенішими є інвертовані індекси.
- Пошук за метаданими: пошук за певними атрибутами документа, такими як назва, дата створення, розмір, автор тощо. Прикладом є пошук у файловій системі (наприклад, у Windows).
- Пошук за зображеннями: пошук за змістом зображень. Пошукові системи розпізнають зміст завантажених або зазначених URL зображень і показують схожі зображення в результатах пошуку.
- Пошук в аудіо: пошук у фрагментах аудіофайлів, як це надає, наприклад, Google.

Компоненти інформаційного пошуку: Для забезпечення інформаційного пошуку можна виділити два рівні: абстрактний і конкретний. Абстрактна ІПС складається з моделей обробки інформації, правил індексації та критеріїв видачі, а також критеріїв семантичної відповідності інформації. Конкретна ІПС є практично реалізованою системою, що включає масив документів для пошуку, технічні засоби та людей, які взаємодіють із системою. Виходячи з цього, інформаційний пошук включає:

- Семантичне осмислення запиту і видача адрес відповідних документів.
- Пошук самого документа.

В абстрактному вигляді ІПС представляє сукупність Інформаційних Пошукових Моделей (ІПМ), правил індексації та логіки, критеріїв смислової відповідності.

# 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДУ

## 3.1 Опис засобів розробки програмного забезпечення

Так як більшість лабораторій обладнані комп’ютерами з встановленою операційною системою Windows та часом мають поганий інтернет зв’язок, було вирішено розробляти програмне забезпечення у вигляді Windows застосунку, який має усі необхідні компоненти для належної роботи застосунку.

Було обрано кілька найпопулярніших інструментів для створення Windows застосунків.

Swing — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. Це частина бібліотеки базових класів Java (JFC, Java Foundation Classes). Swing розробляли для забезпечення функціональнішого набору програмних компонентів для створення графічного інтерфейсу користувача, ніж у ранішого інструментарію AWT. Компоненти Swing підтримують специфічні look-and-feel модулі, що динамічно підключаються. Завдяки ним можлива емуляція графічного інтерфейсу платформи (тобто до компоненту можна динамічно підключити інші, специфічні для даної операційної системи вигляд і поведінку). Основним недоліком таких компонентів є відносно повільна робота, хоча останнім часом це не вдалося підтвердити через зростання потужності персональних комп'ютерів. Позитивна сторона - універсальність інтерфейсу створених програм на всіх платформах.

Переваги:

- Платформенна незалежність: Swing базується на Java, тому програми, що використовують Swing, можуть запускатися на будь-якій платформі, де є встановлене Java Runtime Environment (JRE).;
- Розширюваність: Swing має багатий набір компонентів і можливостей для створення різноманітних GUI, включаючи кнопки, тексти, таблиці, вкладки і т.д.
- Легкість навчання: Для розробників Java вивчення Swing є відносно простим, особливо для тих, хто вже знайомий з основами Java.

- Можливості настроювання: Swing надає великі можливості для налаштування вигляду та поведінки GUI за допомогою платформено-незалежних стилів (Look and Feel).
- Документація та підтримка: Через те, що Swing є частиною стандартного API Java, для нього існує обширна документація і підтримка від розробників.

Недоліки:

- Видимий вік: Swing інколи сприймається як застарілий інструмент через більш сучасні альтернативи, такі як JavaFX.
- Високий рівень абстракції: Деякі розробники можуть відчувати, що Swing занадто абстрактний і важко контролювати деталі.
- Продуктивність: У деяких випадках Swing може мати обмеження на продуктивність, особливо при обробці великої кількості даних або при інтенсивному використанні анімацій.
- Відсутність підтримки для деяких сучасних функцій: Наприклад, підтримка швидкісного рендерингу або підтримка сучасного дизайну інтерфейсу може бути обмеженою порівняно з новішими технологіями.

Qt - це крос-платформовий інструментарій для створення класичних і вбудованих графічних інтерфейсів і програм, що працюють на різних програмних та апаратних платформах, і є стандартною програмою зі стандартними можливостями і швидкістю.

Бібліотека Qt була розроблена в 1996 році і з тих пір активно розвивається. Її використовують такі відомі IT-корпорації, як Autodesk, Google, Microsoft, Nokia, Panasonic, Siemens, Walt Disney Animation Studios та інші.

Вигляд інтерфейсу зображений на рисунку 3.1.

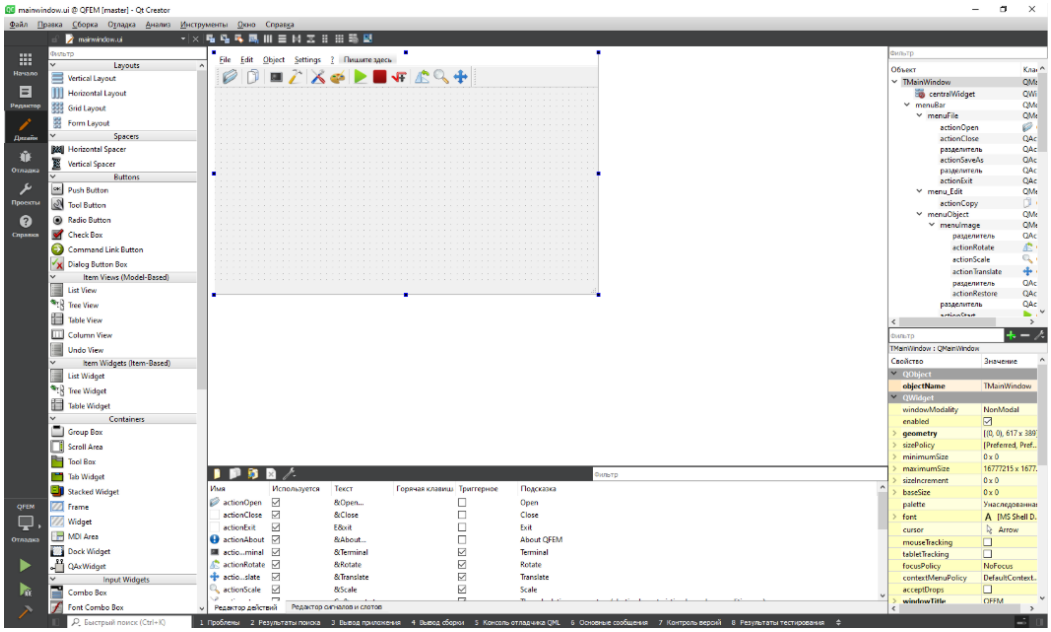


Рисунок 3.1 – QT Creator

Переваги:

- Кросплатформенність: Qt дозволяє розробляти додатки, які можуть працювати на різних операційних системах, таких як Windows, macOS, Linux, Android та iOS, з мінімальними змінами коду.
- Гнучкий інтерфейс користувача: Qt надає можливості для створення привабливих і сучасних користувацьких інтерфейсів з використанням Qt Widgets або QML для декларативного програмування інтерфейсів.
- Продуктивність: Qt відомий своєю високою продуктивністю, особливо для додатків з високими вимогами до графіки та інтерактивності.
- Модульність: Qt складається з модулів, які можна використовувати окремо або разом, що дозволяє розробляти різноманітні типи додатків — від простих GUI до складних багатоплатформенних систем.
- Документація та підтримка: Qt має добре структуровану і детальну документацію, а також активну спільноту та підтримку, що робить процес розробки ефективнішим.
- Інтеграція з C++: Qt використовує C++, що дозволяє розробникам легко інтегрувати Qt з існуючим C++ кодом і користуватися перевагами цього потужного мови програмування.

- Інструменти розробки: Qt надає інструменти, такі як Qt Creator (інтегроване середовище розробки), які спрощують процес розробки, тестування та налагодження додатків.
- Вбудовані ресурси для розробки GUI: Qt містить великий набір готових компонентів і віджетів для побудови GUI, а також підтримує створення власних компонентів.
- Декларативне програмування: Використання QML та Qt Quick дозволяє розробляти динамічні інтерфейси з меншими витратами коду та більшою виразністю.
- Багатомовна підтримка: Qt дозволяє легко локалізувати додатки для різних мов та регіонів.

Недоліки:

- Ліцензування: Qt доступний за комерційною ліцензією або під GPL/LGPL. Використання Qt у комерційних продуктах може вимагати придбання ліцензії, що може бути дорого для деяких компаній.
- Розмір бібліотеки: Qt є великою бібліотекою, і додатки, побудовані з її використанням, можуть мати великий розмір, що може вплинути на розмір кінцевих програмних продуктів.
- Крута крива навчання: Для нових користувачів, особливо тих, хто не знайомий з C++ [16], навчання роботі з Qt може бути складним, оскільки включає освоєння нових концепцій і методів.
- Платформозалежні відмінності: Хоча Qt забезпечує кросплатформенність, можуть виникати деякі нюанси та відмінності у поведінці додатків на різних платформах, які потрібно враховувати при розробці.
- Обмежена інтеграція з іншими мовами: Хоча Qt має підтримку для Python (PyQt, PySide) та інших мов, ця підтримка може бути менш зрілою або повноцінною порівняно з нативною підтримкою C++.

- Перевантаження функціоналом: Іноді розробники можуть відчувати, що в Qt надто багато можливостей, що може ускладнювати вибір оптимальних інструментів і методів для конкретних завдань.
- Складність оновлень: Підтримка та оновлення додатків на нові версії Qt може бути складною, особливо якщо використовується велика кількість специфічних для старих версій API.
- Нестандартний синтаксис: Використання сигналів і слотів в Qt може спочатку виглядати незвично і вимагає додаткового часу для звикання.

Windows Forms - інтерфейс програмування додатків, що відповідає за графічний інтерфейс користувача, є частиною Microsoft.NET Framework [17] і спрощує доступ до елементів інтерфейсу Microsoft Windows.

В Windows Forms форма - це візуальна поверхня, на якій виводиться інформація для користувача. Обычно додаток Windows Forms будується шляхом розміщення елементів керування формуванням і написанням коду для реагування на дії користувача, наприклад, миші або натискання клавіш. Елемент управління - це окремий елемент користувацького інтерфейсу, призначений для відображення або введення даних.

При виконанні користувачем будь-якої дії з формою або одним з її елементів управління створюється подія. Програма реагує на ці події за допомогою коду і обробляє події при їх виникненні. Подробнее см. в розділі Создание обработчиков событий в Windows Forms.

Windows Forms включає в себе широкий набір елементів управління, які можна додати на форми: текстові поля, кнопки, розкриваючі списки, перемикачі та навіть веб-сторінки. Список всіх елементів управління, які можна використовувати в формі, представлені в розділі елементи управління для використання у формах Windows Forms. Якщо існуючий елемент управління не задовольняє потреби, в Windows Forms можна створити користувацькі елементи керування за допомогою класу UserControl.

В складі Windows Forms входять багатофункціональні елементи користувацького інтерфейсу, що дозволяють відтворювати можливості таких

складних додатків, як Microsoft Office. Використовуючи елементи керування ToolStrip і MenuStrip, можна створити панелі інструментів і меню, що містять текст та малюнки, підменю та інші елементи керування, такі як текстові поля та поля з списками.

Переваги:

- Нативна підтримка у Windows.
- Використання MS Visual Studio для розробки застосувань.
- Широкий набір компонентів для розробки.
- Подійно-орієнтована архітектура.

З недоліків можна виділити складність створення складних інтерфейсних рішень.

Windows Presentation Foundation (WPF) - це Microsoft графічна підсистема для відображення користувацьких інтерфейсів у додатках Windows. WPF, раніше відомий як "Авалон", спочатку був випущений в рамках .NET Framework 3.0 у 2006 році. WPF використовує DirectX. WPF намагається забезпечити послідовну модель програмування для побудови додатків та розділяє користувальницький інтерфейс з бізнес-логіки. Він нагадує схожі XML-орієнтовані об'єктні моделі, такі як ті, що реалізовані в XUL і SVG.

WPF використовує XAML, мову на базі XML, для визначення та поєднання різних елементів інтерфейсу [!]. Програми WPF можуть бути розгорнуті як окремі настільні програми або розміщені як вбудований об'єкт на веб-сайті. WPF має на меті об'єднати ряд загальних елементів користувальницького інтерфейсу, таких як 2D / 3D рендеринг, фіксовані та адаптивні документи, типографіка, векторна графіка, анімація часу виконання та попередньо відтворені носії. Потім ці елементи можуть бути пов'язаними та маніпулювати на основі різних подій, взаємодії користувачів та прив'язування даних.

В якості графічної технології WPF використовує технологію DirectX, що дозволяє використовувати апаратне прискорення при відмальовуванні графічних елементів інтерфейсу користувача.

У порівнянні з попередником, Windows Forms, WPF пропонує нову командну модель, що надає дві наступних важливих можливості:

– делегування подій відповідними командам;

– підтримка включеного стану елемента керування в синхронізованому виді за допомогою стану відповідної команди.

Вигляд розробки WPF форми зображено на рисунку 3.2

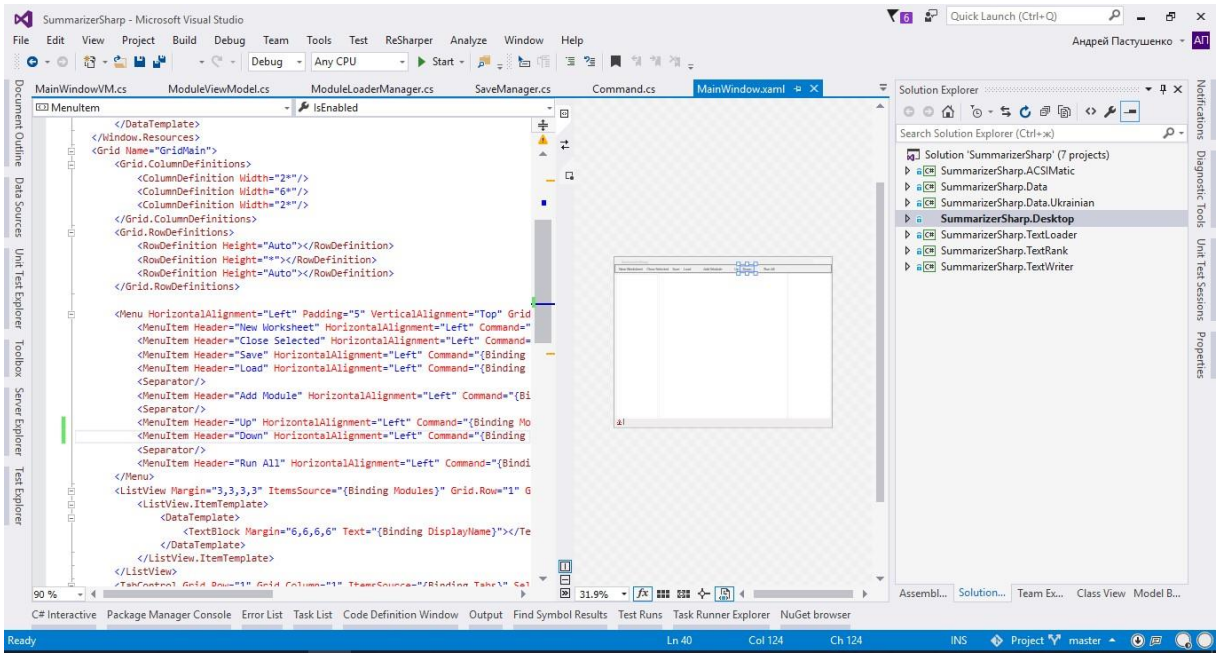



Рисунок. 3.2 – Розробка WPF форми у Visual Studio 2022

Прив'язка даних – це метод доступу до інформації та відображення її на користувацькому інтерфейсі додатка без написання коду для виконання цієї роботи. Зазвичай «товсті» клієнти використовують двонаправлену прив'язку даних, що дозволяє зворотну передачу інформації з інтерфейсу до об'єкта логічної моделі. Оскільки багато Windows-додатків потребують роботи з даними, прив'язка даних є ключовою технологією для створення користувацьких інтерфейсів, таких як у WPF.

Клас Binding у WPF відповідає за реалізацію прив'язки даних.

Переваги:

– Гнучкість у створенні інтерфейсів.

– Підтримка шаблону MVVM.
– Використання GDI+.
– Підтримка розробки у Microsoft Visual Studio.

Недоліки:

– Складність освоєння бібліотеки при переході з Windows Forms.
– Вищі вимоги до апаратного забезпечення порівняно з Windows Forms.

За результатами дослідження було вирішено розробляти програмне забезпечення у вигляді Windows-додатка з використанням WPF. Це дозволить створювати складний інтерфейс для налаштування параметрів алгоритмів реферування. Додатково, застосування шаблону MVVM забезпечить розділення розробки інтерфейсу користувача (представлення) та ядра системи (моделі), що зменшить зв’язаність модулів системи.

MVVM та розподіл ролей:

Шаблон MVVM був розроблений для розділення завдань дизайнера і програміста, що неможливо, коли Java-розробник створює GUI у Swing або розробник на Visual C++ створює інтерфейс у MFC. Розробники часто не мають необхідних навичок для створення зручних і привабливих інтерфейсів, що більше підходить дизайнерам. Дизайнери інтерфейсів краще розуміють, що бажають користувачі, ніж програмісти, які пишуть код. Тому, оптимально, якщо дизайнер створює інтерфейс, а розробник пише код для реалізації його логіки. Проте, технології типу Swing або MFC не дозволяють такої гнучкості. [18]

Для реалізації проекту було обрано кілька сторонніх бібліотек, що спрощують реалізацію певних компонентів програмного забезпечення:

SharpNL — портована на .NET Framework бібліотека Apache OpenNLP для обробки текстових даних. Вона підтримує основні задачі, критичні для обробки тексту:

– Токенізація.
– Розбиття на речення.
– Тегування (визначення лінгвістичних одиниць).

Для розробки програмного продукту на основі технології WPF доцільно обрати одну з мов програмування:

- C#
- Visual Basic .NET
- Microsoft Visual C++

Враховуючи поставлені завдання, було обрано мову C#, яка є об'єктно-орієнтованою. C# (вимовляється як "See Sharp") — це проста, сучасна, об'єктно-орієнтована та безпечна мова програмування, яка має корені в сімействі мов C, тому вона знайома програмістам C, C++, Java та JavaScript.

C# забезпечує підтримку компонентного програмування, що включає створення програмних компонентів як самодостатніх і самореалізованих пакетів функціональності з властивостями, методами, подіями та документацією.

Мова C# допомагає створювати надійні додатки через такі механізми як збірка сміття (автоматичне звільнення пам'яті), обробка винятків (структурований підхід до відновлення помилок) та типовий дизайн, що виключає можливість читання з неініціалізованих змінних, вихід за межі масивів або виконання неперевірених типових перетворень. C# має єдину систему типів, яка підтримує як типи посилань, так і значень.

Інтегроване середовище розробки:

Microsoft Visual Studio — це серія продуктів, що включає інтегроване середовище розробки та інші інструменти. Вона дозволяє розробляти як консольні програми, так і програми з графічним інтерфейсом, веб-застосунки, веб-служби, як у рідному, так і у керованому кодах для різних платформ (Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework, Microsoft Silverlight).

Visual Studio 2022 — остання версія, яка включає .NET Framework 4.6 та підтримку універсальної платформи Windows 10. C++ розробникам доступні нові функціональні можливості стандарту C++14 та покращення з C++17.

Для розробки було обрано платформу .NET Framework та мову програмування C#, що дозволяє створювати ефективні Windows-додатки.

Для реалізації інтерфейсу користувача було обрано бібліотеку Windows Presentation Foundation, що забезпечує потужні можливості для створення складних інтерфейсних рішень.

В ході розробки архітектури додатка було вирішено використовувати ряд сторонніх бібліотек для реалізації необхідних функціональних можливостей.

Microsoft Visual Studio було обрано як інтегроване середовище розробки для C#, забезпечуючи візуальне середовище розробки інтерфейсів користувача для WPF.

### 3.2 Архітектура розробленого програмного забезпечення

Беручи до уваги сферу застосування розроблюваної автоматизованої системи, було вирішено, що її архітектура буде побудована за клієнт-серверною моделлю. При цьому користувач встановлює на комп'ютері клієнтське застосування, яке здійснює лише мінімальну попередню обробку документа та готує його до відправки на сервер. Сервер, у свою чергу, аналізує документи та надсилає на клієнтську частину результуючі метадані з результатами роботи реалізованого методу. Клієнтська частина візуалізує ці результати на вхідних документах і відображає інформацію у зручному для користувача вигляді.

Клієнт-серверну архітектуру можна описати як концепцію інформаційної системи, в якій основна частина її ресурсів зосереджена на серверах, що обслуговують клієнтів. Ця архітектура включає такі компоненти:

- Сервери, які надають інформацію або інші послуги програмам, що до них звертаються.
- Клієнти, які використовують сервіси, що надаються серверами.
- Мережа, що забезпечує взаємодію між клієнтами та серверами.

Правила взаємодії між клієнтом і сервером називаються протоколом взаємодії.

Модель клієнт-серверної взаємодії визначається розподілом обов'язків між клієнтом та сервером і логічно складається з трьох рівнів операцій:

- Рівень представлення даних — відповідає за відображення даних користувачеві та отримання від нього команд і даних.
- Прикладний рівень — реалізує основну логіку додатка, здійснюючи необхідну обробку інформації.
- Рівень доступу до даних — забезпечує зберігання та доступ до даних.

Модель клієнт-сервер не визначає, що сервери повинні мати більше ресурсів, ніж клієнти. Вона дозволяє будь-якому комп'ютеру використовувати спільні ресурси інших хостів для розширення своїх можливостей. При централізованих обчисленнях велика кількість ресурсів зосереджується на невеликій кількості комп'ютерів. Чим більше обчислень виконується на серверах замість клієнтів, тим простішим може бути клієнт-хост. Він залежить від мережевих ресурсів (серверів та інфраструктури) для обчислення та зберігання даних. Наприклад, бездисковий вузол завантажує свою операційну систему з мережі, а комп'ютерний термінал є лише інтерфейсом вводу/виводу для сервера.

На противагу цьому, товстий клієнт, як персональний комп'ютер, має більше ресурсів і не залежить від сервера для виконання важливих функцій.

З 1980-х до кінця 1990-х років, коли мікрокомп'ютери ставали доступнішими та потужнішими, багато організацій переміщували обчислення з централізованих серверів на товсті клієнти. Це дозволило більш персоналізовано керувати комп'ютерними ресурсами, але ускладнило управління IT-інфраструктурою. Протягом 2000-х років вебзастосунки стали достатньо потужними, щоб конкурувати з додатками, розробленими для конкретної архітектури. Це зрілість вебзастосунків, більш доступне зберігання великих обсягів даних та поява сервісорієнтованої архітектури сприяли розвитку хмарних обчислень у 2010-х роках.

### 3.3 Особливості програмної реалізації застосунку

Основна ідея модифікації базового методу полягає в додаванні кроків для видалення синонімів із вхідних документів. Головним завданням при реалізації цього модифікованого методу є забезпечення максимальної швидкодії нового кроку, щоб вплив на загальну продуктивність модифікованого методу був мінімальним у порівнянні з базовим методом.

Аналізуючи запропоновану модифікацію, можна зробити висновок, що найбільший вплив на швидкодію модифікованого методу матиме пошук у словнику синонімів. Тому при реалізації слід зосередитися на максимальному прискоренні цього пошуку. Це можна досягти кількома способами:

- Використання спеціалізованих структур даних, призначених для виконання пошукових операцій на великих обсягах даних.
- Попередня обробка та структуризація словника синонімів, що сприятиме прискоренню пошуку.

Розглянемо кожен із цих етапів детальніше.

До спеціалізованих структур даних та алгоритмів, що підтримують швидкий пошук на великих об’ємах даних, можна віднести:

- Геш-таблиці
- Бінарні дерева пошуку
- Купи
- Геш-таблиця

Геш-таблиця – це структура даних, яка реалізує інтерфейс асоціативного масиву, зберігаючи пари (ключ, значення) і дозволяючи виконувати три операції: додавання нової пари, пошук та видалення за ключем. Існує два основних типи геш-таблиць: з ланцюжками та з відкритою адресацією. Геш-таблиця складається з масиву ННН, елементами якого є пари (у випадку відкритої адресації) або списки пар (у випадку з ланцюжками).

Операції в геш-таблиці починаються з обчислення геш-функції від ключа. Отримане геш-значення i=hash(key)i = hash(key)i=hash(key) виконує роль індексу в

масиві HHH. Далі операція (додавання, видалення, пошук) перенаправляється до об’єкта, розташованого у відповідній комірці масиву H[i]H[i]H[i].

Колізія виникає, коли для різних ключів отримується одне й те саме геш-значення. Такі випадки є досить поширеними: наприклад, якщо в геш-таблицю розміром 365 комірок додати 23 елементи, ймовірність колізії перевищує 50% (при рівній ймовірності потрапляння елемента в будь-яку комірку) — див. парадокс днів народження. Тому механізм розв’язання колізій є важливою складовою будь-якої геш-таблиці.

У деяких особливих випадках можна уникнути колізій взагалі. Наприклад, якщо всі ключі відомі заздалегідь (або змінюються дуже рідко), для них можна знайти досконалу геш-функцію, яка розподіляє їх по комірках без колізій. Геш-таблиці з такими функціями називаються геш-таблицями з прямою адресацією і не потребують механізму розв’язання колізій.

Бінарне дерево пошуку — це структура даних, призначена для реалізації швидкого пошуку елементів. Кожна гілка дерева має дві дочірні гілки. Структура бінарного дерева зображена на рисунку 3.3.

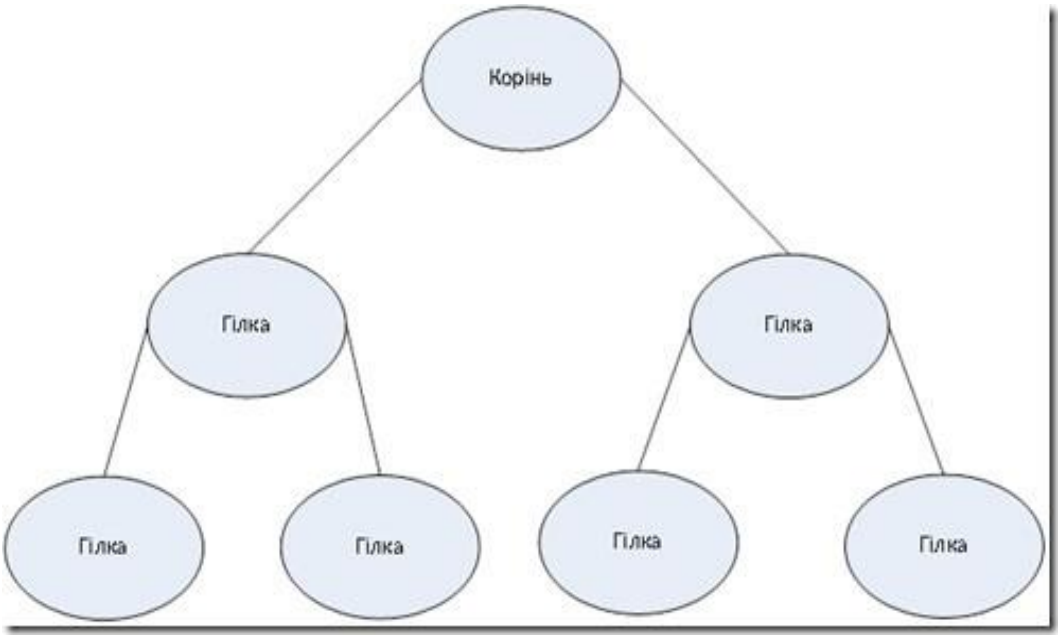



Рисунок 3.3 – Структура бінарного дерева

Тепер, для того щоб зрозуміти різницю між бінарним деревом і бінарним деревом пошуку було представлено наступну діаграму на рисунку 3.4.

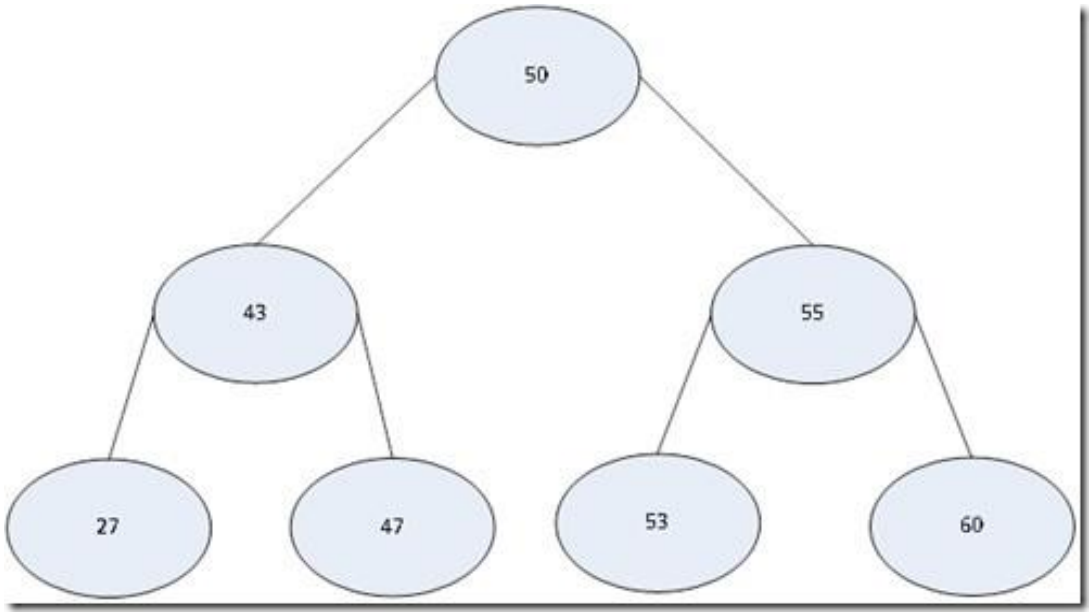



Рисунок 3.4 – Структура бінарного дерева з ключами

На попередній діаграмі показане дерево, де кожна гілка містить певний ключ (у нашому випадку ціле число). Важливо зазначити, що це дерево побудоване так, що кожна гілка має дочірні гілки з меншими або більшими значеннями ключа. Наприклад, гілка з ключем 43 має дві дочірні гілки: з меншим значенням ключа — 27 (ліва гілка) і з більшим значенням ключа — 47 (права гілка).

Купа — це спеціалізована структура даних типу дерево, яка задовольняє властивості купи: якщо вузол BBB є нащадком вузла AAA, то ключ AAA ≥ ключ BBB. Це означає, що елемент із найбільшим ключем завжди є кореневим вузлом купи, тому такі купи іноді називають max-купами (якщо ж порядок порівняння змінити, то найменший елемент буде кореневим вузлом, і такі купи називають min-купами).

У купі немає обмежень щодо кількості дочірніх вузлів у кожного вузла, хоча на практиці їх кількість зазвичай не перевищує двох. Купа є ефективною реалізацією абстрактного типу даних, відомого як черга з пріоритетом. Купи відіграють ключову роль у деяких ефективних алгоритмах на графах, таких як

алгоритм Дейкстри з використанням d-куп і пірамідальне сортування (сортування методом піраміди).

### 3.4 Аналіз отриманих результатів

Для оцінки якості роботи методів визначення нечітких дублікатів застосовуються різноманітні метрики, що грунтуються на аналізі результатів роботи методу. При цьому ідеальними результатами вважаються такі, що повністю співпадають з думкою експертів, що також проводять оцінку. Існують такі метрики оцінювання якості:

- повнота (recall);
- точність (precision);
- акуратність (accuracy);
- F-міра.

Дані метрики базуються на матриці класифікації, наданій в таблиці 3.1

Таблиця 3.1 – Матриця класифікації

| | Є дублікатом | Не є дублікатом |
|---|---|---|
| Знайдено методом | *a* | *b* |
| Не знайдено методом | *c* | *d* |

Де *a* – кількість документів, знайдених методом і є дублікатами з точки зору експертів; *b* – кількість документів, знайдених методом, але не є дублікатами на думку експертів; *c* – кількість документів, що є дублікатами з точки зору експертів, але не знайдених методом; *d* – документи, що не є дублікатами з точки зору експертів і виявлені методом як не дублікати.

Повнота (recall) обчислюється як відношення знайдених дублікатів до їх загальної кількості:

$$R = \frac{a}{a + c}$$

Повнота характеризує можливість методу знаходити дублікати, але не враховує кількість документів, які не є дублікатами з точки зору експертів. Наприклад, якщо повнота дорівнює 75% то це означає, що 25% дублікатів не знайдено.

Точність (precision) визначається як відношення знайдених дублікатів до загальної кількості документів:

$$P = \frac{a}{a + b}$$

Точність характеризує можливості методу видавати в списку результатів тільки дублікати. Наприклад, якщо точність дорівнює 25% то це означає що із усього набору результатів тільки чверть окументів є дублікатами.

Акуратність (accuracy) визначається відношенням правильно знайдених методом дублікатів до загального числа документів:

$$A = \frac{a + d}{a + b + c + d}$$

F-міра використовується для того, щоб об'єднати показники точності і повноти в одній величині. Для цього F-міра обчислюється за формулою:

$$F_\beta = \frac{(1 + \beta^2) * P * R}{\beta^2 * P + R}$$

При $\beta$ = 1 F-міра надає однакову вагу як точності так і повноті і називається збалансованою, або $F_1$ -мірою. Для неї формула значно спрощується:

При дослідженнях дана метрика використовується як єдина метрика, по значенню якої можна робити висновок про якість роботи методу.

### 3.5 Вибір наборів даних для тестування розробленого методу

Для дослідження ефективності розробленого модифікованого методу визначення нечітких дублікатів потрібно обрати відповідні набори даних. Для аналізу дублікатів у веб-документах було обрано корпус WT10G, створений університетом Глазго протягом кількох років шляхом збору веб-сторінок з 11,680 серверів. Цей корпус містить 1,692,096 документів. Однією з його переваг є проведений аналіз зв'язків між документами та дослідження близькості документів за змістом, що можна вважати експертними оцінками в обраних метриках.

Корпус WT10G доступний на веб-сайті університету Глазго та містить документи у форматі SGML, відомому як TREC веб-формат. TREC – це серія конференцій, присвячених інформаційному пошуку. Формат SGML передбачає наявність відкриваючих і закриваючих тегів, а також додаткових тегів, таких як TEXT, HL, HEAD, HEADLINE, TTL, та LP для форматування документа і поділу на логічні частини.

CMD є частиною Common Crawl dataset, який збирався протягом 7 років і включає понад петабайт даних, що містять веб-сторінки, метадані та виділений текст. Набір даних Common Crawl розміщений на Amazon S3 і доступний для вільного завантаження як частина програми Amazon Public Datasets. Дані Common Crawl зберігаються у форматі WARC (web-ARChive), що дозволяє ефективно обробляти веб-архіви з мільярдами веб-сторінок. Формат WARC містить:

WARC-файли з необробленими даними сканування

WAT-файли з обчислювальними метаданими

WET-файли з витягнутим текстом

Формат WARC забезпечує відображення процесу сканування, зберігаючи HTTP-відповіді, запити та метадані сканування (типи WARC: response, request, metadata).

### 3.6 Результати роботи модифікованого методу

Основними параметрами досліджень ефективності методу були швидкість його роботи на однакових наборах даних та набір метрик, що визначають ефективність. Було обрано метрики, які застосовуються для оцінки методів класифікації та базуються на визначенні значень false-positive та false-negative у порівнянні з експертною оцінкою.

Тестування проводилось для:

- базового методу I-Match.
- модифікованого методу I-Match, запропонованого в даній дисертації, який використовував відповідний словник синонімів, спеціально підготовлений та переформатований для роботи з цим методом.
- методу шинглів, реалізованого у бібліотеці text-shingles.
- методу winnowing, реалізованого у бібліотеці winnowing.

Для тестування модифікованого методу пошуку нечітких дублікатів використовувалась система з наступною конфігурацією:

CPU: Intel Core i7-3612QM

RAM: 8 GB

OS: Linux Debian 9 x64

Таблиця 3.2 показує швидкодію розробленого методу у порівнянні з існуючими аналогами:

Таблиця 3.2 – Швидкодія протестованих методів

| Корпус | I-Match* | I-Match (мод)* | Метод шинглів | Winnowing |
|---|---|---|---|---|
| WT10G | 19.340 | 22.119 | 57.083 | 101.742 |
| CMD | 8.430 | 9.012 | 18.023 | 18.340 |
| Crosswikis | 7.236 | 8.120 | 17.620 | 12.874 |

З результатів роботи видно, що модифікований метод поступається в швидкодії базовому аналогу. Це пов'язано з додаванням кроку пошуку синонімів при побудові "відбитку" документа. Проте втрати в швидкодії не є значними у порівнянні з іншими аналогами, складаючи в середньому близько 10%. Зважаючи на те, що більшість методів не орієнтуються на покращення швидкодії, а користувачі не потребують надшвидких результатів, такі втрати в швидкодії є незначними.

Таблиця 3.3 містить аналіз точності на корпусі WT10G із застосуванням раніше описаних метрик:

Таблиця. 3.3 – Значення метрик для корпусу WT10G

| Метод | Повнота | Точність | Акуратність | $F_{0.5}$-міра |
|---|---|---|---|---|
| I-Match | 0,894 | 0,927 | 0,912 | 0,9 |
| I-Match (мод). | 0,924 | 0,935 | 0,93 | 0,926 |
| Метод шинглів | 0,824 | 0,803 | 0,811 | 0,819 |

| | | | | |
|---|---|---|---|---|
| Winnowing | 0,75 | 0,733 | 0,739 | 0,746 |

Для корпусу CMD результати занесені в таблицю 3.4.

Таблиця 3.4 – Значення метрик для корпусу CMD

| Метод | Повнота | Точність | Акуратність | $F_{0.5}$-міра |
|---|---|---|---|---|
| I-Match | 0,824 | 0,915 | 0,874 | 0,84 |
| I-Match (мод). | 0,814 | 0,927 | 0,875 | 0,834 |
| Метод шинглів | 0,864 | 0,822 | 0,839 | 0,855 |
| Winnowing | 0,75 | 0,733 | 0,739 | 0,746 |

Для корпусу Crosswikis результати занесені в таблицю 3.5.

Таблиця 3.5 – Значення метрик для корпусу Crosswikis

| Метод | Повнота | Точність | Акуратність | $F_{0.5}$-міра |
|---|---|---|---|---|
| I-Match | 0,768 | 0,739 | 0,749 | 0,762 |
| I-Match (мод). | 0,734 | 0,731 | 0,732 | 0,733 |
| Метод шинглів | 0,758 | 0,735 | 0,743 | 0,753 |
| Winnowing | 0,768 | 0,738 | 0,748 | 0,761 |

З результатів видно, що модифікований метод демонструє кращі показники метрик для більшості корпусів, на яких він був протестований. Проте для корпусу Crosswikis розроблений метод показав дещо гірші результати. Це може бути

обумовлено тим, що статті на ресурсі Wikipedia мають переважно науковий характер, де використання довгих синонімічних рядів не є характерним.

Розвиток Інтернету та активне впровадження цифрових технологій у повсякденне життя призвели до стрімкого збільшення обсягу цифрових текстових даних. Виявлення нечітких дублікатів документів є однією з важливих і складних задач у галузі автоматизованого аналізу природномовних текстів. Це завдання актуальне через численні застосування, де потрібно враховувати "подібність" документів, наприклад, для поліпшення якості пошукових індексів та архівів, об'єднання статей за змістом, фільтрації поштового й пошукового спаму, визначення порушень авторських прав тощо.

Розроблений у даній роботі метод визначення нечітких дублікатів може бути застосований у багатьох сферах. Однією з перших галузей, де використовувалися методи визначення дублікатів, було визначення порушень авторських прав. Тому першою асоціацією при згадці про методи визначення дублікатів є боротьба за авторські права. Однак ці методи не повинні використовуватися лише для пост-аналізу написаних робіт, щоб запобігти неавторизованому копіюванню. Вони також можуть надати науковцям можливість розширення кола джерел для підготовки матеріалів під час написання робіт.

Пошукові індекси сьогодні є одними з найважливіших систем у мережі Інтернет, проте обсяги даних у них перевищують можливості ручної обробки. Проблема дублікації документів з великою схожістю є актуальною. Використання методів визначення нечітких дублікатів дозволить зменшити обсяги даних у пошукових індексах та підвищити ефективність пошуку.

Цією проблемою зацікавлені різні групи осіб. Передусім це пошукові системи як загального, так і спеціалізованого призначення. Системи загального призначення прагнуть скоротити обсяги інформації при збереженні точності пошуку, тоді як спеціалізовані системи прагнуть підвищити точність пошуку на наявних даних. Зацікавлені також постачальники текстових документів до спеціалізованих пошукових систем, оскільки висока швидкість оцінки документів на відповідність критеріям дозволяє швидко коригувати тексти.

Опосередковано зацікавлені державні органи та замовники, які прагнуть покращення роботи спеціалізованих систем, таких як юридичні пошукові системи, а також швидшого виконання замовлень.

Спеціалізований програмний продукт, що реалізує описану технологію пошуку нечітких дублікатів, дозволяє ефективно обробляти та зберігати текстові дані, підвищуючи швидкість пошуку та забезпечуючи високу точність. Основними клієнтами є університети, видавництва, новинні агенції та організації, що займаються написанням дипломних та курсових робіт. Програмне рішення повинно бути представлене у вигляді веб-сервісу з зовнішніми API, що забезпечить гнучкість рішення та легку інтеграцію в існуючі системи великих організацій.

# ВИСНОВКИ

Гешування є однією з ключових технологій у комп'ютерних науках та інформаційних системах. Від ефективного гешування залежать продуктивність, безпека та надійність багатьох систем, включаючи бази даних, системи управління пам'яттю, мережеві протоколи та криптографічні застосування.

Геш-таблиці дозволяють швидко знаходити елементи у великих наборах даних, знижуючи час доступу до них. Контрольні суми та геш-функції використовуються для виявлення змін у даних, що є критично важливим для зберігання та передачі інформації. Геш-функції є основою багатьох криптографічних алгоритмів, забезпечуючи безпеку цифрових підписів, аутентифікації та інших методів захисту даних.

З розвитком технологій збільшуються обсяги даних, а також вимоги до швидкості та безпеки обробки інформації. Це створює нові виклики для методів гешування.

Порівняльний аналіз методів гешування є важливим інструментом для розуміння і вибору оптимальних алгоритмів для різних інформаційних систем. На основі проведеного аналізу можна зробити такі висновки:

Алгоритми гешування, такі як MurmurHash і CityHash, демонструють високу продуктивність та є ефективними для великих обсягів даних. Класичні геш-функції, такі як MD5 та SHA-1, хоч і все ще використовуються, поступаються сучасним алгоритмам в швидкості та безпеці. MD5 та SHA-1 більше не вважаються безпечними для криптографічних застосувань через вразливість до колізій. SHA-2 та SHA-3 пропонують значно вищий рівень безпеки, будучи стійкими до більшості відомих атак.

Використання геш-таблиць та інвертованих індексів значно прискорює пошук даних, що є критично важливим для великих баз даних та інформаційних систем. Алгоритми, такі як SipHash, забезпечують баланс між продуктивністю та безпекою, що робить їх ідеальними для використання у мережевих протоколах та інших системах реального часу.

Методи гешування, що ефективно масштабуються, такі як Consistent Hashing, є незамінними для розподілених систем та мережевих сервісів, де потрібне динамічне додавання або видалення вузлів. Для криптографічних цілей рекомендується використовувати алгоритми, що мають високий рівень безпеки, такі як SHA-3 та BLAKE2. Для загальних цілей з високими вимогами до швидкості, але не до безпеки, можуть бути використані швидкі алгоритми, такі як MurmurHash.

Аналіз методів гешування показав, що жоден алгоритм не є універсальним рішенням для всіх можливих застосувань. Вибір конкретного алгоритму повинен ґрунтуватися на розумінні його властивостей, переваг та недоліків у контексті конкретного застосування. Завдяки порівняльному аналізу можна зробити обґрунтований вибір, що забезпечить оптимальну продуктивність, безпеку та ефективність інформаційних систем.

# ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Pandey V., K Mishra V. Architecture based on MD5 and MD5-512 Bit Applications. International Journal of Computer Applications. 2013. Vol. 74, no. 9. P. 29–33. URL: https://doi.org/10.5120/12914-9884 (date of access: 09.05.2024).
2. Secure Hash Algorithm. Encyclopedia of Cryptography and Security. Boston, MA, 2011. P. 1116. URL: https://doi.org/10.1007/978-1-4419-5906-5_1209 (date of access: 09.05.2024).
3. Impeccable Keccak / I. Gavrilan et al. IACR Transactions on Cryptographic Hardware and Embedded Systems. 2024. Vol. 2024, no. 2. P. 154–189. URL: https://doi.org/10.46586/tches.v2024.i2.154-189 (date of access: 10.05.2024).
4. Kornblum J. Identifying almost identical files using context triggered piecewise hashing. Digital Investigation. 2006. Vol. 3. P. 91–97. URL: https://doi.org/10.1016/j.diin.2006.06.015 (date of access: 11.05.2024).
5. Нікітін В., Крилов Є. Спектральний фільтр блума на основі простих чисел для використання в active-anti entropy механізмі узгодження даних у розподіленій документоорієнтованій нереляційній базі даних. Computer systems and information technologies. 2023. № 3. С. 75–80. URL: https://doi.org/10.31891/csit-2023-3-9 (дата звернення: 11.05.2024).
6. Jakobs C., Lambertz M., Hilgert J.-N. ssdeeper: Evaluating and improving ssdeep. Forensic Science International: Digital Investigation. 2022. Vol. 42. P. 301402. URL: https://doi.org/10.1016/j.fsidi.2022.301402 (date of access: 12.05.2024).
7. Husham Ali B., Jalal A. A., Al-Obaydy Al-Obaydy W. N. I. Data loss prevention (DLP) by using MRSH-v2 algorithm. International Journal of Electrical and Computer Engineering (IJECE). 2020. Vol. 10, no. 4. P. 3615. URL: https://doi.org/10.11591/ijece.v10i4.pp3615-3622 (date of access: 14.05.2024).

8. mvHash-B - A New Approach for Similarity Preserving Hashing / F. Breitinger et al. 2013 Seventh International Conference on lT Security lncident Management and lT Forensics (IMF), Nuremberg, Germany, 12–14 March 2013. 2013. URL: https://doi.org/10.1109/imf.2013.18 (date of access: 14.05.2024).
9. Єремізін, О. Перспективи використання нечіткого гешування в антивірусному захисті / О. Єремізін, І. Стьопочкіна // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні : науково-технічний збірник. – 2016. – Вип. 1(31). URL: https://ela.kpi.ua/handle/ /123456789/25509 (дата звернення: 20.05.2024).
10. Grant G. Bioinformatics – The Machine Learning Approach. Computers & Chemistry. 2000. Vol. 24, no. 1. P. 139–141. URL: https://doi.org/10.1016/s0097-8485(00)80015-7 (date of access: 20.05.2024).
11. Зиберт Андрей Оскарович, Мирошниченко Виктория Валентиновна Алгоритм определения наличия заимствований в тексте с использованием эталонного множества слов // Universum: технические науки. 2014. №12 (13). URL: https://cyberleninka.ru/article/n/algoritm-opredeleniya-nalichiya-zaimstvovaniy-v-tekste-s-ispolzovaniem-etalonnogo-mnozhestva-slov (дата обращения: 20.06.2024).
12. Near Duplicate Document Detection using Document Image / G. Uwamahoro et al. International Journal of Multimedia and Ubiquitous Engineering. 2016. Vol. 11, no. 7. P. 159–168. URL: https://doi.org/10.14257/ijmue.2016.11.7.17 (date of access: 20.06.2024).
13. Kołcz A., Chowdhury A. Lexicon randomization for near-duplicate detection with I-Match. The Journal of Supercomputing. 2008. Vol. 45, no. 3. P. 255–276. URL: https://doi.org/10.1007/s11227-007-0171-z (date of access: 20.06.2024).
14. Stanford NLP. URL: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html (дата звернення 20.05.2024).
15. TIOBE Index for May 2018. URLd: https://www.tiobe.com/tiobe-index/ – (date of access: 27.05.2024).

16. Elementary C++ Programming. *C++ Programming*. 2019. P. 25–78. URL: https://doi.org/10.1515/9783110471977-002 (date of access: 27.06.2024).
17. Дударчук В. С., Dudarchuk V. Розробка системи уніфікації потоків даних на мові програмування C#, платформа .Net : master's thesis. 2020. URL: http://elartu.tntu.edu.ua/handle/lib/33790 (дата звернення: 27.06.2024).
18. Тлумачний словник з інформатики / Г.Г. Півняк, Б.С. Бусигін, М.М. Дівізінюк та ін. – Д., Нац. гірнич. ун-т, 2010. – С. 495][ Міжнародний союз електрозв'язку; переклад – Юрій Пероганич (2006-11-23). Словник термінів. Всесвітній саміт з питань інформаційного суспільства. Підсумкові документи. с. 84. Архів оригіналу за 2013-06-25. (дата звернення: 27.05.2024).

# ДОДАТКИ

52

## Додаток А

## Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: Порівняльний аналіз методів нечіткого гешування

Автор роботи: Мацький В'ячеслав Юрійович

Тип роботи: бакалаврська дипломна робота .

Підрозділ кафедра захисту інформації ФІТКІ .
(кафедра, факультет)

### Показники звіту подібності Unicheck

Оригінальність – 79,0%. Схожість – 21%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Валентина КАПЛУН
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ В'ячеслав МАЦЬКИЙ
(підпис) (прізвище,ініціали)

Керівник роботи _____ Наталія КОНДРАТЕНКО
(підпис) (прізвище, ініціали)

53

**Додаток Б**

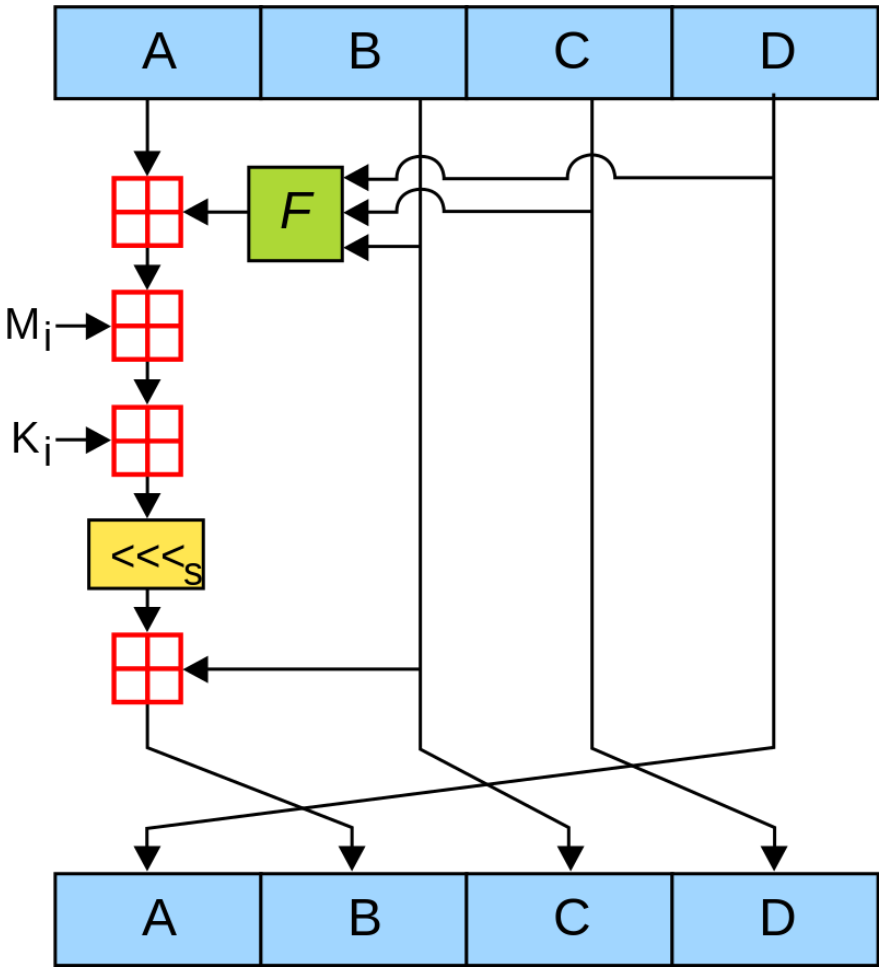
**ІЛЮСТРАТИВНА ЧАСТИНА**

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ НЕЧІТКОГО ГЕШУВАННЯ

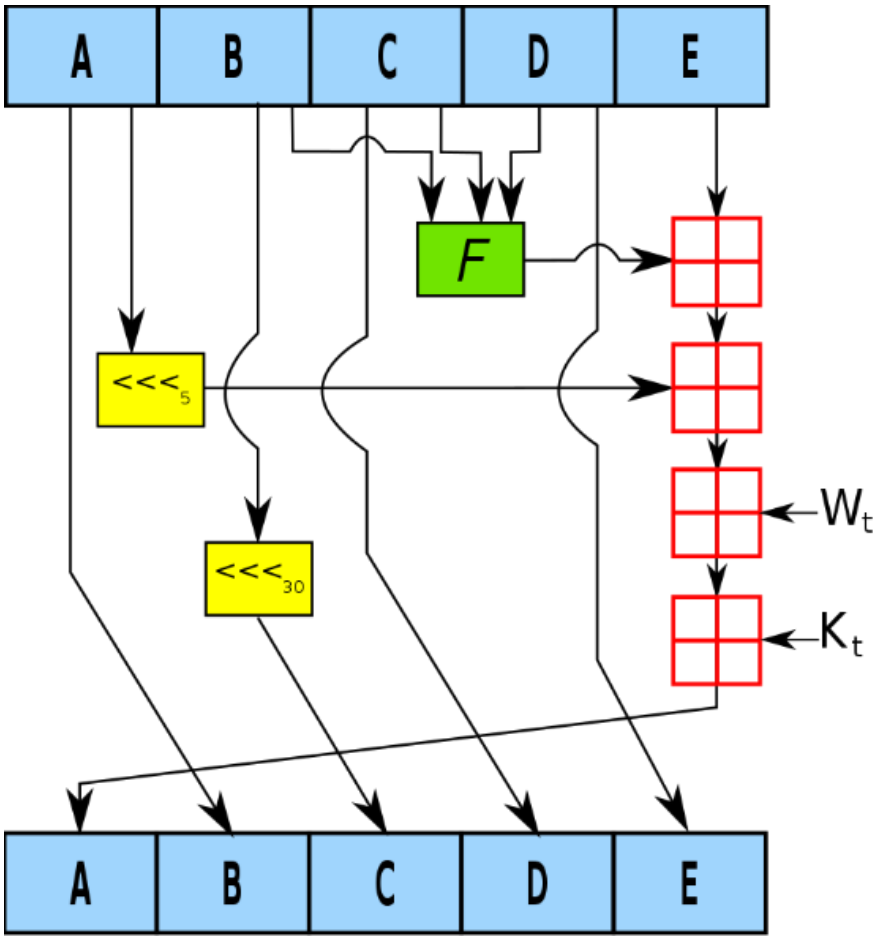
Виконав: студент 4 курсу групи 1БС-20б
спеціальності 125 Кібербезпека
_____В’ячеслав МАЦЬКИЙ
_____2024 р.

Керівник: к. т. н., професор каф. ЗІ
_____Наталія КОНДРАТЕНКО
_____2024 р.

СХЕМА РОБОТИ АЛГОРИТМУ MD5



# СХЕМА ОДНІЄЇ ІТЕРАЦІЇ АЛГОРИТМІВ SHA-1



# СХЕМА ОДНІЄЇ ІТЕРАЦІЇ АЛГОРИТМІВ SHA-2

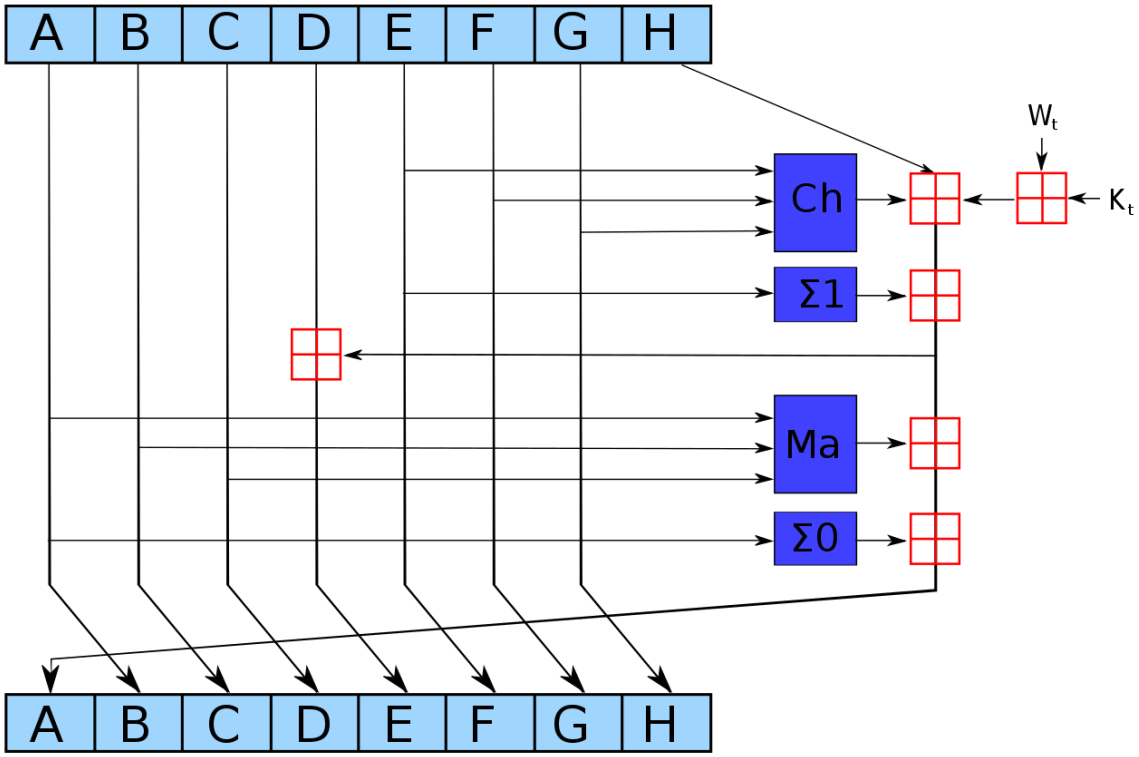
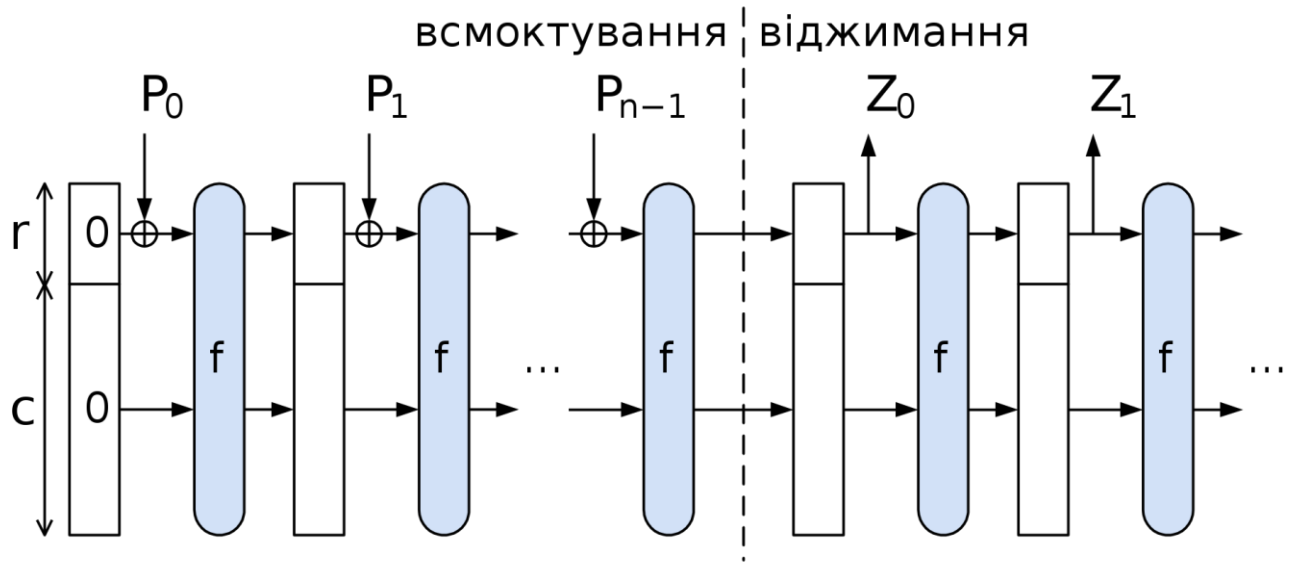
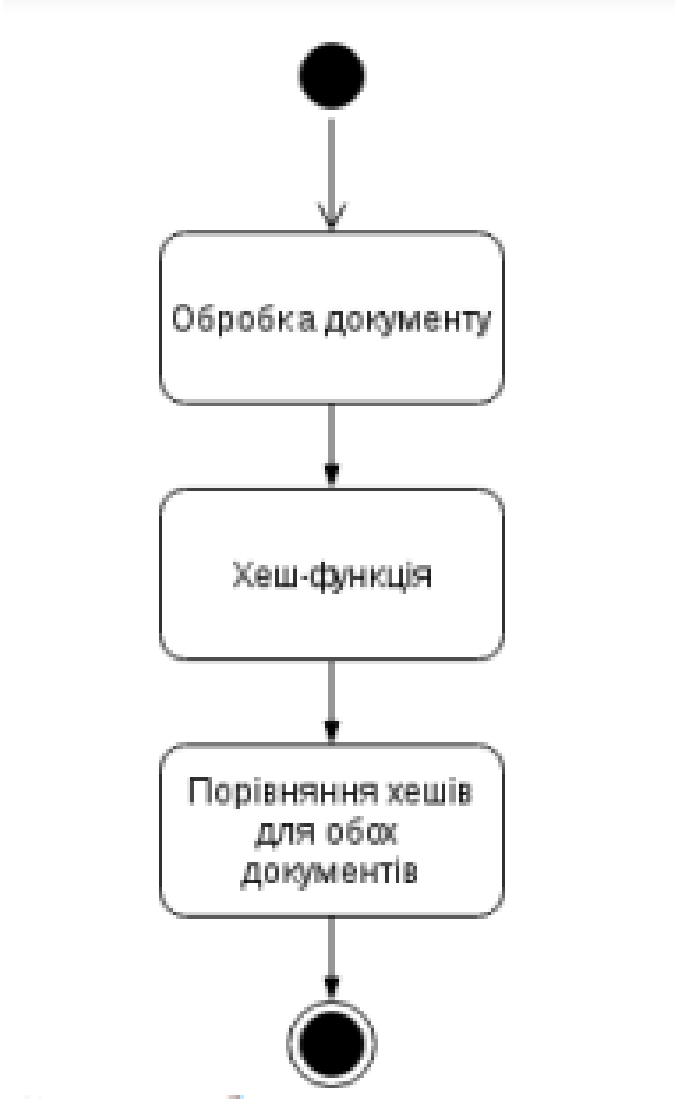


СХЕМА РОБОТИ АЛГОРИТМУ SHA-3



# СХЕМА РОБОТИ СИГНАТУРНИХ МЕТОДІВ



# СХЕМА ГЕНЕРАЦІЇ НЕЧІТКОГО ГЕШ-ЗНАЧЕННЯ В МЕТОДІ НЕЧІТКОГО ГЕШУВАННЯ