МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна технологія виявлення нечітких дублікатів текстів програмного забезпечення»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Кузіков Б.О.**

Студента групи ІН.м – 02/1 **Пільгуй І.І.**

СУМИ 2021

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:

зав.кафедрою ____________________

____

"________" ____________________ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Пільгуй Ірині Іванівні

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія виявлення нечітких дублікатів текстів програмного забезпечення*

затверджую наказом по інституту від "____" ________________ 20 ___ р. № _____________

2. Термін здачі студентом закінченого проекту (роботи)________________________

3. Вхідні данні до проекту (роботи) ____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Огляд технологій, що застосовуються для виявлення нечітких дублікатів; 2) Постановка завдання й формування завдань дослідження; 3) Моделювання технології виявлення нечітких дублікатів програмного забезпечення; 4) Розробка додатку; 5) Аналіз результатів.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання ____

Керівник ________________________
(підпис)

Завдання прийняв до виконання ________________________
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд технологій, що застосовуються для виявлення нечітких дублікатів.		
2.	Постановка задачі та формування завдань дослідження.		
3.	Моделювання технології виявлення нечітких дублікатів програмного забезпечення		
4.	Розробка додатку		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник ________________________
(підпис)

Керівник проекту ________________________
(підпис)

РЕФЕРАТ

Записка: 52 стор., 15 рис., 3 таблиця, 3 додаток, 20 літературних джерел.

Об’єкт дослідження — Інформаційна технологія виявлення нечітких дублікатів текстів програмного забезпечення.

Мета роботи — розробка технології виявлення нечітких дублікатів текстів програмного забезпечення.

Результати — проведено аналіз літератури, методів та інструментів, які дозволяють виявити дублікати текстів програмного забезпечення у великих проектах та у студентських роботах, що використовують різні підходи, заснованих на метриках, узгодженні шаблонів послідовності токенів, аналізі абстрактного синтаксичного дерева (AST) або графа залежностей програми (PDG). У даній роботі описано та реалізовано технологію виявлення дублікатів засновану на абстрактному синтаксичному дереві, обчислюючи хеш-значення вузлів синтаксичного дерева та порівнюючи їх. Для оптимізації використаної пам’яті було застосовано алгоритм Winnowing. Дана розробка дозволяє виявляти дублікати коду студентських робіт в автоматичному режимі, на різних мовах програмування з достатнім рівнем точності.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ВИЯВЛЕННЯ НЕЧІТКИХ ДУБЛІКАТІВ ТЕКСТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, INFORMATION TECHNOLOGY FOR DETECTING FUZZY DUPLICATES IN SOURCE CODE, АБСТРАКТНЕ СИНТАКСИЧНЕ ДЕРЕВО, ЦИФРОВИЙ ВІДБИТОК, АЛГОРИТМ WINNOWING

ЗМІСТ

ВСТУП 6

1. АНАЛІЗ ПРОБЛЕМИ 8

1.1 Аналіз видів дублювання коду 8

1.2 Аналітичні огляд методів виявлення нечітких дублікатів......10

1.3 Постановка задачі...14

2. МЕТОДИКА ДОСЛІДЖЕНЬ 16

2.1 Аналіз вхідного коду 16

2.2 Цифровий відбиток піддерев 18

2.3 Індексація цифрових відбитків 21

2.4 Вилучення кластерів збігів..21

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ 23

3.1 Реалізація абстрактного синтаксичного дерева 24

3.2 Реалізація цифрових відбитків піддерев...................................26

3.3 Реалізація індексування цифрових відбитків 28

3.4 Представлення результатів аналізу 29

3.5 Тестування системи 30

ВИСНОВКИ... 34

ЛІТЕРАТУРА 35

ДОДАТКИ.. 37

ВСТУП

Дублювання коду – це термін в програмуванні, що позначає повторювання послідовності вихідного коду в рамках однієї або декількох програм. Послідовності повторюваного коду іноді називають клонами коду або просто клонами. Автоматизований процес пошуку дублікатів у вихідному коді називається виявленням клонів. Копіювання коду без вагомих причин є небажаним і може призвести до значних проблем таких, як [1,2]:

- зростання важкості підтримки;
- зниження читабельності коду;
- зростання розміру системи;
- зниження продуктивності системи;

Проблема клонування коду полягає в тому, що при виявлені помилки у фрагменті коду, що копіювався, всі схожі фрагменті потрібно перевірити на наявність тієї ж самої помилки. Інші види технічного обслуговування, наприклад, розширення або адаптації, також слід застосовувати кілька разів. Проте зазвичай такі копії дуже важко виявити у великих системах, якщо тільки вони не були задокументовані. Для великих систем виявлення можливе лише за допомогою автоматичних методів.

Різні дослідження показують, що значна частина, від 7% до 23% коду у великих системах програмного забезпечення є дублікатами [3,4]. Більшість великих компаній витрачають більше на утримання існуючих систем, ніж на розвиток нових систем.

Дублювання коду - це проблема, що часто зустрічається на академічних курсах, коли студенти використовують чужу роботу для отримання більш високих оцінок. Учні можуть копіювати чужий код, з або без змін і отримувати некоректні оцінки і тим самим загрожувати цілісності академічної системи. Вкрай важливо розпізнавати та карати таку поведінку студентів [5,6], не тільки

для підтримки академічної чесності, але й для зменшення кількості дублікатів коду в наступних завданнях з програмування у тому ж курсі.

У даній роботі буде розглянуто проблему дублювання коду при проведені олімпіади з програмування, а також при перевірці навчальних робіт студентів.

Отже, стає актуальною проблема визначення ідентичних ділянок програмного коду або ділянок, що мало відрізняються в автоматичному режимі, на різних мовах програмування з достатнім рівнем точності.

1. АНАЛІЗ ПРОБЛЕМИ

1.1 Аналіз видів дублювання коду

Існують 4 типи дублювання коду (див рис. 1.1):

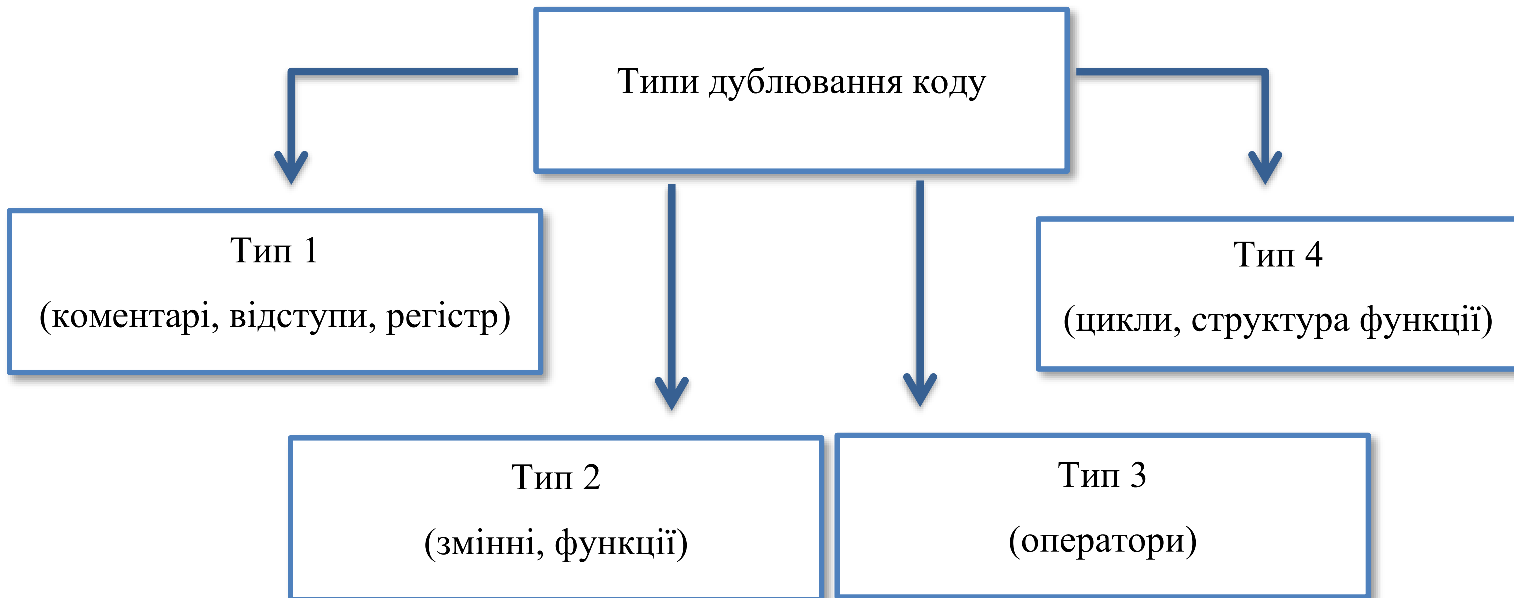

Рисунок 1.1 - типи копіювання коду

Розглянемо типи копіювання коду як за текстовими так і за функціональними ознаками, що були запропоновані авторами праць [4,7]:

- Тип 1 – дублювання коду з мінорними змінами такими як зміни у коментарях, відступах чи регістру.

```
1. int gcd(int a, int b) {
2.     if (b == 0) return a;
3.     else return gcd(b, a % b);
4. }
```

```
1. int gcd(int a, int b) {
2.     if (b == 0)
3.         return a;
4.     else
5.         return gcd(b, a % b);
6. }
```

Рисунок 1.2 - приклад копіювання типу 1

- Тип 2 - фрагмент коду типу 1, зі змінами в ідентифікаторах таких, як: тип змінної, назва змінної чи назва функції.

```
1. int gcd(int a, int b) {
2.     if (b == 0) return a;
3.     else return gcd(b, a % b);
4. }
```

```
1. long gcd(long c, long d) {
2.     if (d == 0) return c;
3.     else return gcd(d, c % d);
4. }
```

Рисунок 1.3 - приклад копіювання типу 2

- Тип 3 - фрагменти коду типу 2 з подальшими модифікаціями такими як, зміною, додаванні чи вилучені операторів.

```
1. int gcd(int a, int b) {
2.     if (b == 0) return a;
3.     else return gcd(b, a % b);
4. }
```

```
1. int gcd(int a, int b) {
2.     if(b == 0)
3.         return a;
4.     if(a == 0)
5.         return b;
6.     return gcd(b, a % b);
7. }
```

Рисунок 1.4 - приклад копіювання типу 3

- Тип 4 - два чи більше фрагментів коду, що виконують одні й ті ж самі розрахунки але імплементовані різними синтетичними конструкціями.

```
1. int gcd(int a, int b) {
2.     if (b == 0) return a;
3.     else return gcd(b, a % b);
4. }
```

```
1. int gcd(int a, int b) {
2.     while(b != 0) {
3.         swap(a, b);
4.         b = b % a;
5.     }
6.     return a;
7. }
```

Рисунок 1.5 - приклад копіювання типу 4

Клони 4-го типу показують лише те що програми вирішують одну й ту ж саму задачу, що відомо заздалегідь, тому даний тип у вирішенні поточної задачі не може бути врахований. Клони 1 та 2-го типів засвідчують про ймовірне копіювання, клони 3-го типу можливі і в роботах у яких відсутні запозичення, наприклад у випадках рішення типових алгоритмічних задач.

Студент може одночасно застосовувати кілька типів змін і може застосовувати додаткові зміни до певної частини програми, наприклад, шляхом додавання деяких невідповідних функцій. Усі ці фактори сприяють виявленню плагіату коду на практиці.

1.2 Аналітичні огляд методів виявлення нечітких дублікатів

Методи виявлення дублікатів пов'язані з представленнями коду на яких вони базуються. Таким чином, клони типу 1 виявляються за допомогою текстових підходів, тип 2 - за допомогою лексичних підходів або підходів на основі токенів. Клони типу 3 виявляються за допомогою синтаксичних підходів, в основі яких лежать абстрактні синтаксичні дерева, з залученням інформації з метрики програмного забезпечення. Для виявлення клонів 4 типу використовують семантичні підходи. Семантичні підходи поділяються на підходи на основі графів і гібридні. Підходи на основі графів створюють графи

програмних залежностей, а гібридні підходи об'єднують кілька різних методів [8,9].

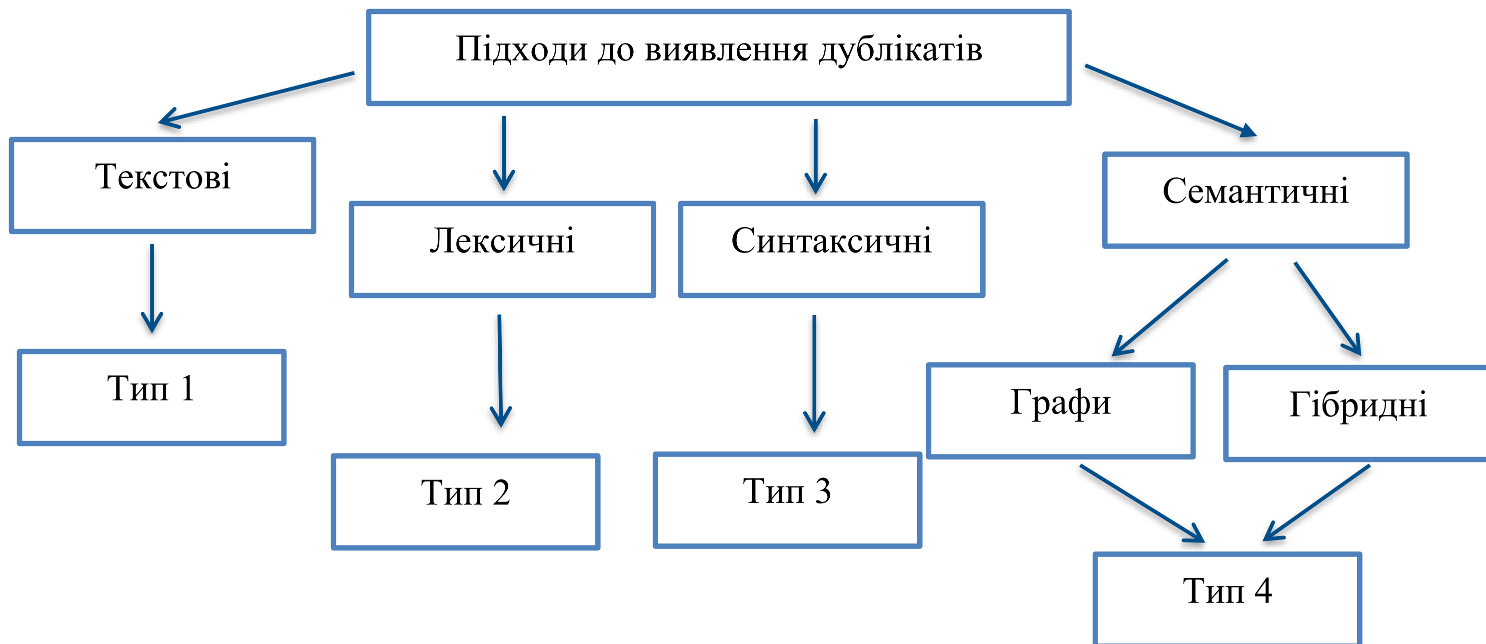

Рисунок 1.6 – Підходи до виявлення дублікатів

1.2.1 Текстові підходи

Текстові підходи ґрунтуються на текстовому порівнянні всіх рядків коду програми. Такий підхід практично не використовує трансформацію та нормалізацію вихідного коду перед фактичним порівнянням, крім видалення коментарів та зайвих символів пробілу. За допомогою алгоритму хешування чи суффіксного дерева виконуються пошук схожих підстрок належної довжини. Даний алгоритм використовують автори у роботах [10,11]. Текстові підходи не залежать від мови програмування, але зазвичай вони можуть виявити лише дублікати типу 1, тому рідко застосовуються для вирішення даної задачі.

1.2.2 Лексичні підходи

Алгоритми із застосуванням лексичного аналізу базуються на перетворенні вихідного коду в послідовність лексичних «токенів» з урахуванням мови програмування. Потім послідовність сканується на наявність дубльованих

підпослідовностей токенів. Методи з лексичним аналізом зазвичай більш стійкі до незначних змін коду, таких як форматування, інтервали і перейменування, ніж текстові методи.

Бренда Бейкер у роботі [3] запропонувала знаходження клонів на основі токенів. Вихідний код конвертується у послідовність токенів за допомогою лексичного аналізатора. Токени поділяються на параметричні та непараметричні, перші кодуються із використанням індекса входження у строку, другі підсумовуються за допомогою функтора хешування . Далі всі префікси отриманої послідовності символів представляються у вигляді суфіксного дерева. Якщо два суфікси мають загальний префікс, то це означає, що префікс зустрічається більше одного разу і може вважатися дублікатом.

1.2.3 Синтаксичні підходи

Синтаксичні підходи використовують синтаксичний аналізатор для перетворення вихідного коду у абстрактні синтаксичні дерева, які потім можуть бути оброблені за допомогою порівняння дерев або структурних метрик для пошуку клонів.

Підходи на основі порівняння дерев: методи на основі дерев визначають клони шляхом знаходження схожих піддерев. Імена змінних, літерали та інші токени в програмі можуть бути абстраговані в деревовидному поданні, що забезпечує більш точне виявлення клонів. Підхід виявлення подібних дерев було представлено авторами [12] у інструменті DECKARD. Основна ідея підходу полягає в обчисленні певних характеристичних векторів для апроксимації структурної інформації в абстрактних синтаксичних деревах в евклідовому просторі. Потім використовується хешування з урахуванням розташування (Locality-sensitive hashing метод - LSH) для кластеризації схожих векторів з використанням метрики евклідової відстані. Результуючі кластери векторів вважаються дублікатами.

Підходи на основі метрик: методи на основі метрик порівнюють вектори показників, зібраних із фрагментів коду, замість безпосереднього порівняння коду. Для однієї або декількох синтаксичних одиниць, таких як клас, функція, метод або оператор, обчислюється набір програмних метрик, які називаються функціями відбитків пальців (або цифрових відбитків), потім значення метрик порівнюються для пошуку клонів відповідних синтаксичних одиниць. У більшості випадків вихідний код спочатку аналізується у вигляді абстрактного синтетичного дерева або графу потоку управління, на основі яких обчислюються метрики [13,14].

Метод виявлення клонів запропонований авторами [15] використовує представлення під назвою Intermediate Representation Language (IRL) для характеристики кожної функції у вихідному коді. Клон визначається лише як пара цілих тіл функцій, які мають подібні метричні значення.

1.2.4 Семантичні підходи

Існують підходи, орієнтовані на семантику, з використанням статичного аналізу програм для надання більш точної інформації, ніж просто синтаксична подібність. У цих методах код представлений у вигляді графа залежностей програми (PDG). Граф залежностей програми містить інформацію про потік керування та потоку даних і, отже, містить семантичну інформацію. Вузли такого графа представляють вирази та оператори, а ребра представляють залежності керування та даних. Це уявлення абстрагується від лексичного порядку, в якому зустрічаються вирази та висловлювання. Після отримання набору PDG алгоритм узгодження ізоморфних підграфів застосовується для пошуку подібних підграфів, які повертаються як клони.

Один з провідних інструментів виявлення клонів на основі графів залежностей програми запропонований Komondoor і Horwitz [16], який знаходить ізоморфні підграфи PDG за допомогою зворотного програмного зрізу.

Krinke [17] використовує ітераційний підхід для виявлення максимально подібних підграфів у PDG.

1.2.5 Гібридні підходи

Крім перерахованих вище підходів існують також методи виявлення клонів, які використовують комбінацію синтаксичних і семантичних характеристик. Leitao [18] пропонує гібридний підхід, який поєднує синтаксичні методи, засновані на метриках абстрактних синтаксичних дерев, і семантичні методи з використанням графів викликів у поєднанні зі спеціалізованими функціями порівняння.

1.3 Постановка задачі

Для виявлення дублікатів у студентських роботах важливо врахувати наступні аспекти при розробці:

- роботи можуть бути реалізовані на різних мовах програмування;
- студент може одночасно застосовувати кілька типів змін і може застосовувати додаткові зміни до певної частини програми, потрібно враховувати синтаксичний фактор вхідного коду;
- висока точність виявлення дублікатів для програм з невеликим об’ємом вихідного коду;

Метою даної роботи є розробка інформаційної технології, котра буде виявляти дублікати коду у студентських роботах. За допомогою даної системи користувачі зможуть виявляти порушення студентами академічної доброчесності, що підвищить рівень якості освіти.

Для досягнення поставленої мети сформульовані наступні задачі:

1) виявлення дублікатів для різних мов програмування;
2) виявлення клонів 1-3 типів;
3) представлення результатів у вигляді звіту;

Для пошуку клонів 1-3 типів найкращим є підхід на основі абстрактного синтаксичного дерева. Алгоритм аналізу на основі даного методу не потребує значної кількості часу для виконання перевірки, що є важливим при перевірці великої кількості робіт. Для підтримки різних мов програмування алгоритм потребує словники токенів. Для реалізації можна використовувати готовий генератор токенів «tree-sitter».

2. МЕТОДИКА ДОСЛІДЖЕНЬ

У даній роботі буде представлений алгоритм виявлення клонів, що складається з наступних кроків (Рис. 2.1):

- аналіз вхідного коду і представлення його у вигляді абстрактного синтаксичного дерева;
- представлення кожного піддерева у вигляді цифрового відбитку;
- індексація цифрових відбитків;
- отримання статистичних результатів.

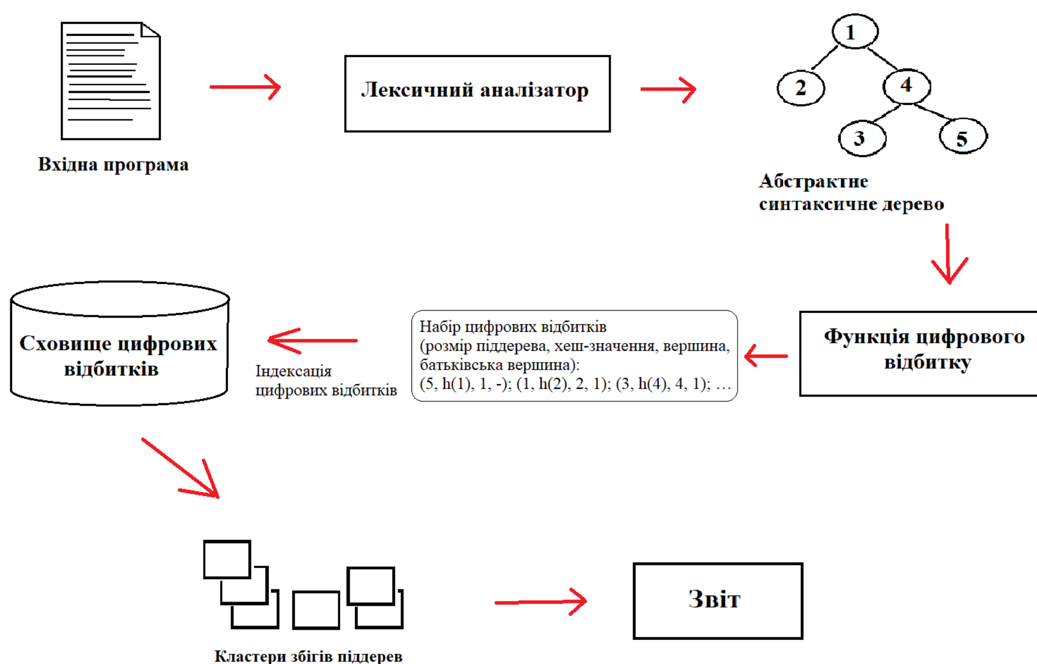

Рисунок 2.1 – Алгоритм виявлення клонів

2.1 Аналіз вхідного коду

Перед побудовою абстрактного синтаксичного дерева вхідна програма аналізується за допомогою лексичного аналізу.

Лексичний аналіз виконується шляхом перетворення вхідного рядку із послідовності символів в послідовність токенів, що потім поділяються на класи (Рис. 2.2).

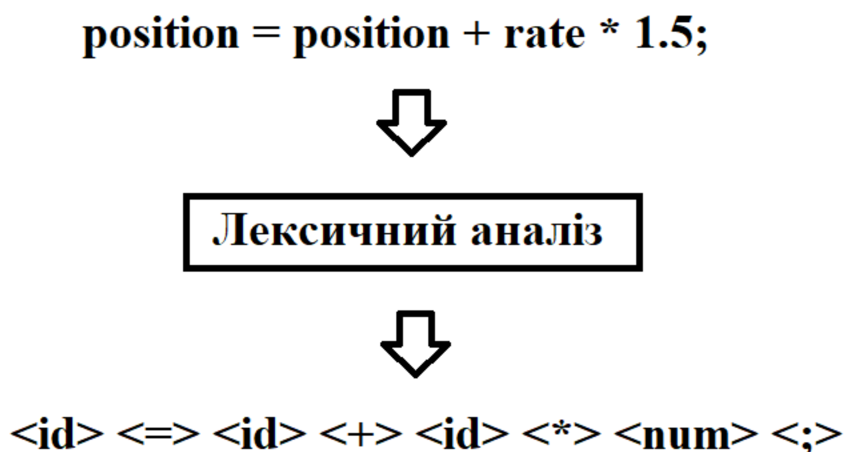

Рисунок 2.2 – Лексичний аналіз вихідного коду

У прикладі (Рис. 2.2) після лексичного аналізу було отримано 6 класів токенів: <id> - ідентифікатор, <num> - число, <=> - оператор присвоєння, <+> - оператор додавання, <*> - оператор множення, <;> - «розділювач». Далі отримані класи представляються у вигляді абстрактного синтаксичного дерева (Рис. 2.3).

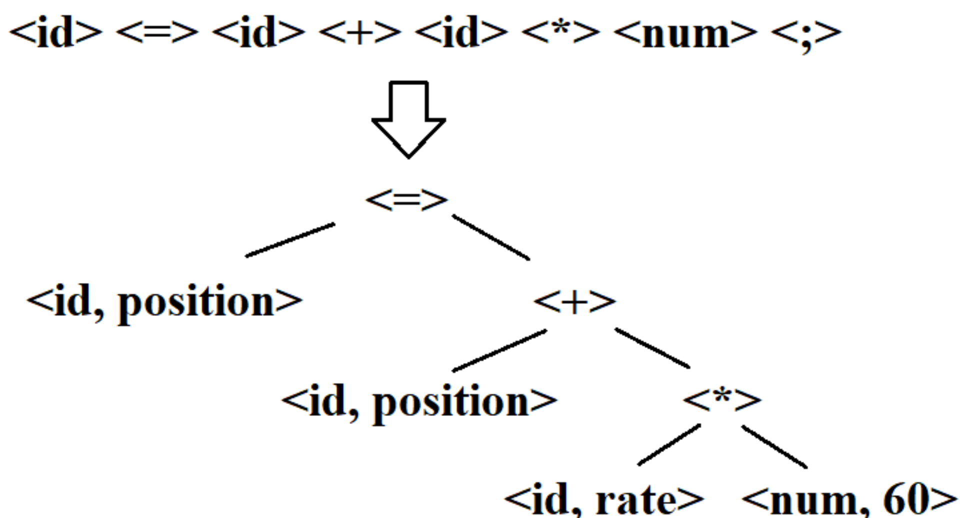

Рисунок 2.3 – Приклад абстрактного синтаксичного дерева

Абстрактне синтаксичне дерево є структурним представленням вхідної програми, очищене від елементів конкретного синтаксису за допомогою порівняння лінійної послідовності токенів з формальною мовою. Вершинами синтаксичного дерева є оператори, до яких приєднуються їх аргументи, які у свою чергу можуть бути складними вершинами. У прикладі (Рис. 2.2) абстрактне синтаксичне дерево побудоване з токенів, вершини містять додаткові атрибути – імена ідентифікаторів та значення чисел. У дерево не потрапив «розділювач», тому що він не має відношення безпосередньо до семантики даного фрагмента програми, а лише до конкретного синтаксису язика.

2.2 Цифровий відбиток піддерев

Кожне піддерево отриманого абстрактного синтаксичного дерева потрібно порівняти. Оскільки структура більшості синтаксичних дерев є досить складною і громіздкою, пряме порівняння кожного піддерева синтаксичного дерева є складним та неефективним. Отже, для зменшення використовування пам'яті та ефективного порівняння усі піддерева будуть представлені у вигляді хеша, за допомогою поліноміального кільцевого хеша. Для оптимізації підрахунку хешів використовується ковзаюча хеш функція. Далі отриманий набір хешів фільтрується за допомогою алгоритму Winnowing [19].

2.2.1 Ковзаюча хеш функція

Ковзаюча хеш функція – це функція, в якій вхідні дані хешуються у рамках деякого вікна. Для перерахунку значення хеша потрібно знати лише попереднє значення хешу, значення вхідних даних, що залишилися за межами вікна, та значення даних, що потрапили у вікно. Тобто, якщо $x = h(a_1 a_2 \dots a_n)$ представляє собою хеш послідовності $a_1 a_2 \dots a_n$, то хеш $h(a_2 a_3 \dots a_n a_{n+1})$ для

наступної послідовності $a_2 a_3 \dots a_n a_{n+1}$ може бути розрахований за допомогою функції $f(x, a_1, a_{n+1})$ [20].

Формула поліноміального кільцевого хеша:

$$h(a_1 a_2 \dots a_n) = a_1 x^{n-1} + a_2 x^{n-2} + \dots + a_n x^0, \quad (2.1)$$

де n – довжина послідовності, x – константа.

Формула ковзаючої хеш функції:

$$h = (h_p - a_p x^{n-1}) \cdot x + a_n , \quad (2.2)$$

де h_p – хеш попередньої послідовності, a_p – попередній символ, a_n – новий символ. Складність ковзаючої хеш функції лінійна - O(n), де n – довжина послідовності.

2.2.2 Алгоритм Winnowing

Winnowing використовує ковзаюче вікно розміру w, що проходить по списку хешів та фільтрує їх. Алгоритм на кожному кроці знаходить мінімальне значення хеша у вікні, якщо таких декілька, використовує крайнє праве та записує його до списку результатів, якщо даний хеш ще не був використаний. Коли вікно досягає кінця послідовності, записаний набір хешів приймається як цифрові відбитки документа.

Нижче наведено приклад роботи алгоритму Winnowing на короткому прикладі тексту (Рис. 2.4).

1) Вхідний текст

A do run run run, a do run run

2) Послідовність 5-грам отриманих з тексту

adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun

3) Гіпотетична послідовність хешей 5-грам

77 74 42 17 98 50 17 98 8 88 67 39 77 74 42 17 98

4) Вікна хешей, при w = 4

(77, 74, 42, **17**) (74, 42, 17, 98)
(42, 17, 98, 50) (17, 98, 50, **17**)
(98, 50, 17, 98) (50, 17, 98, **8**)
(17, 98, 8, 88) (98, 8, 88, 67)
(8, 88, 67, 39) (88, 67, **39**, 77)
(67, 39, 77, 74) (39, 77, 74, 42)
(77, 74, 42, **17**) (74, 42, 17, 98)

5) Цифрові відбитки обрані Winnowing

17 17 8 39 17

Рисунок 2.4 – Приклад використання алгоритму Winnowing

Алгоритм Winnowing призводить до отримання цифрових відбитків з кількома корисними властивостями. Використовуючи k-грами та розмір вікна w, зберігатимуться такі властивості [19]:

- Нечутливість до шуму: збіги коротші за k не виявляються. Алгоритм хешує підрядки довжини k, що означає, що відповідні сегменти повинні мати принаймні довжину k, щоб бути хешовані.
- Рівномірний розподіл хешів: Winnowing гарантовано вибере принаймні один хеш для кожної довжини вікна. До того моменту, коли вікно зміститься на одну одиницю, попередній найнижчий хеш повинен

вийти за рамки вікна, і, таким чином, алгоритм змушений вибрати новий найнижчий хеш з поточного вікна. Ця властивість корисна, оскільки гарантує, що жодна частина документа не буде пропущена з набору цифрових відбитків.

- Гарантовано виявлення довгих збігів: дві вищенаведені властивості гарантують, що збіги довжиною принаймні w + k - 1 завжди виявляються.

2.3 Індексація цифрових відбитків

У результаті пункту 2.2 кожне піддерево абстрактного синтаксичного дерева представлено у вигляді цифрового відбитка. Цифровий відбиток піддерева x — це кортеж, що включає його розмір w(t), його хеш-значення h(x), його кореневий вузол та батьківський вузол, а також файл і відповідний номер рядка. У сховищі цифрових відбитків піддерев підтримується подвійний індекс: кортежі спочатку сортуються у порядку зменшення ваги, потім за значенням хеша та батьківським вузлом. Індексування реалізується за допомогою деревовидної структури даних B+. За допомогою цієї структури можливо ітеруватися по сховищу цифрових відбитків отримуючи найбільш вагомі клони, а також отримати всі відбитки дочірніх піддерев даного вузла. Обчислювальна складність даної структури для кожної операції у найгіршому випадку складає $O(\log\frac{t}{2n})$, де t — порядок дерева чи коефіцієнт розгалуження; n - кількість елементів у дереві.

2.4 Вилучення кластерів збігів

Ітерація по сховищу цифрових відбитків дозволяє отримати кластери точних збігів піддерева. Кластери вважаються однаковими, якщо вони мають однаковий вагу та значення хеша. Однак, є невелика ймовірність отримати хибнопозитивні результати. Для зменшення кількості хибнопозитивних

результатів можна збільшити довжину хеш-значень, що негативно вплинуть на розмір сховища.

Вирішенням даної проблеми є додаткова ітерація по фіксованій кількості дочірніх елементів. Наприклад, розглянемо два піддерева, з вершинами x і y, та відповідними дочірніми елементами $x_1, \ldots, x_i$ і $y_1, \ldots, y_i$. Якщо x і y мають однакові значення ваги та хеша, то додатково застосовується порівняння значення ваги та хеша дочірніх пар $(x_1, y_1), \ldots, (x_i, y_i)$. Даний підхід може бути реалізований за допомогою вказівника на батьківський вузол цифрового відбитка пальця, який дозволяє, заданому вузлу, отримати цифрові відбитки його дочірніх елементів. Цей підхід хибнопозитивного виявлення не вимагає десеріалізації синтаксичних дерев.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

Для виявлення нечітких дублікатів коду, запропонований алгоритм використовує підхід на основі абстрактного синтаксичного дерева. Під час розробки технології виявлення клонів з використанням даного підходу, потрібно врахувати наступні питання: аналіз вхідних програм та побудова абстрактного синтаксичного дерева, хешування піддерев, індексація цифрових відбитків та звітування про результати. Алгоритм виявлення клонів та відповідна архітектура системи представлена на Рисунку 3.1-3.2.

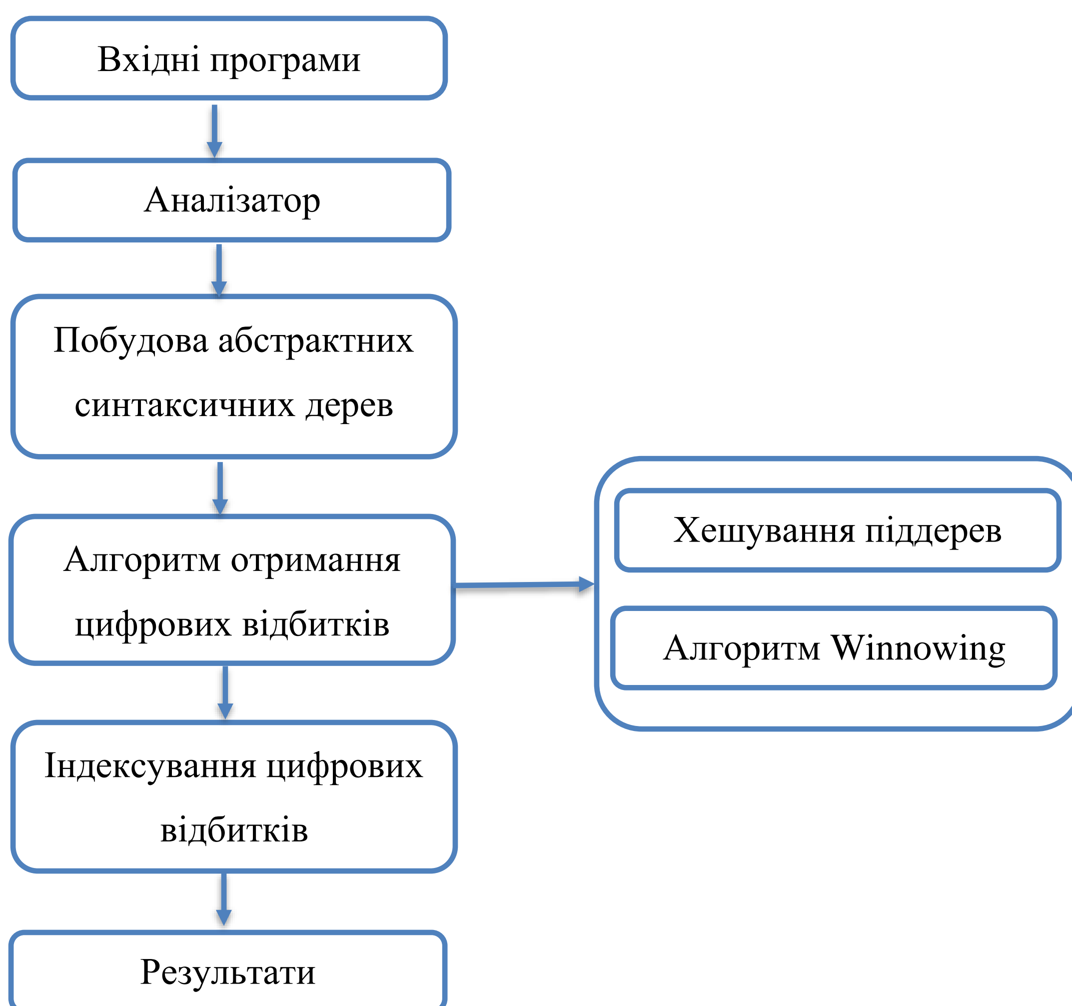

Рисунок 3.1 – Алгоритм виявлення клонів

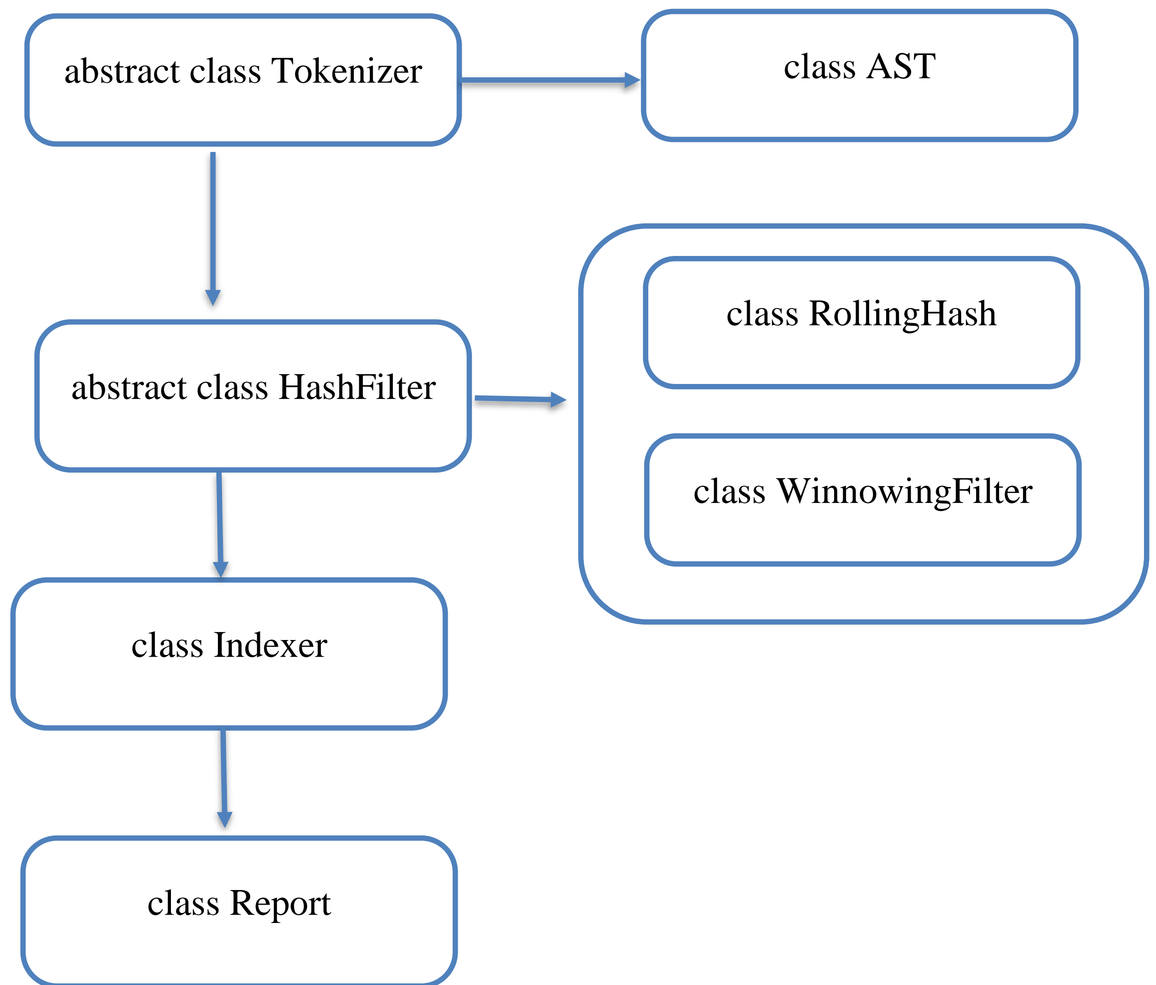

Рисунок 3.2 – Архітектура системи

Завдяки модульної архітектури розробленої системи, вона може бути доповнена новим функціоналом без великих затрат часу, включаючи легку інтеграцію підтримки нових мов програмування. Дана система була реалізована за допомогою мови програмування TypeScript.

3.1 Реалізація абстрактного синтаксичного дерева

Для побудови абстрактного синтаксичного дерева використовується бібліотека «Tree-sitter». Дана бібліотека може побудувати конкретне синтаксичне дерево для вихідного файлу та підтримує більш ніж 40 мов

програмування. Реалізація абстрактного синтаксичного дерева за допомогою бібліотеки «tree-sitter»:

```
import { default as Parser, SyntaxNode } from "tree-sitter";

export class AST {
  public static supportedLanguages = ["c", "c-sharp", "java",
"javascript", "python"];

  public static IsSupportedLanguage(language: string): boolean {
    return this.supportedLanguages.includes(language);
  }

  public static RegisterLanguage(language: string): void {
     try {
       require("tree-sitter-" + language);
     }
     catch (error) {
        throw new Error("tree-sitter-${language} language not found.");
     }
     this.supportedLanguages.push(language);
   }

  public AST(text: string): string {
    const tree = this.parser.parse(text);
    return tree.rootNode.toString();
  }
}
```

Нижче наведено приклад абстрактного синтаксичного дерева (Рис. 3.3), отриманого за допомогою бібліотеки «tree-sitter».

```
1. def sum(a, b):
2.     return a + b;
```

```
module [0, 0] - [2, 0]
  function_definition [0, 0] - [1, 17]
    name: identifier [0, 4] - [0, 7]
    parameters: parameters [0, 7] - [0, 13]
      identifier [0, 8] - [0, 9]
      identifier [0, 11] - [0, 12]
    body: block [1, 4] - [1, 17]
      return_statement [1, 4] - [1, 16]
        binary_operator [1, 11] - [1, 16]
          left: identifier [1, 11] - [1, 12]
          right: identifier [1, 15] - [1, 16]
```

Рисунок 3.3 – Приклад побудови абстрактного синтаксичного дерева

Додати до списку нову мову програмування можливо за допомогою команди yarn global add tree-sitter-[мова програмування].

3.2 Реалізація цифрових відбитків піддерев

Кожне піддерево абстрактного синтаксичного дерево представляється у вигляді хеш значення за допомогою функції поліноміального кільцевого хеша. Для оптимізації підрахунку хешів використовується ковзаюча хеш функція. Далі отриманий набір хешів фільтрується за допомогою алгоритму Winnowing.

Для уникнення перезаповнення типів даних підрахунок хеша виконується з використанням модульної арифметики. Оскільки у підрахунку застосовується операція множення, модуль помножений сам на себе, не повинен перевищувати допустимого числового значення мови програмування TypeScript. Максимальне числове значення, що може бути використане в TypeScript є $2^{53} - 1$. Тоді $mod = 33554393$ – максимальне 26-бітне просте число.

При хешуванні також використовується константа. Оскільки символи вхідного токену в основному мають значення менше за 128, доцільно обрати значення константи, таке що $127 \cdot base < mod$. Отже, для підрахунку поліноміального кільцевого хешу $base = 747287$.

Реалізація підрахунку поліноміального кільцевого хеша:

```
public hashToken(token: string): number {
  let hash = 0;
  for (let i = 0; i < token.length; i++) {
    hash = ((hash + token.charCodeAt(i)) * this.base) % this.mod;
  }
  return hash;
}
```

Для уникнення колізій в однакових підпослідовностях токенів при підрахунку наступного хеша використано різні значення констант. Для ковзаючої хеш функції $base = 9999991$.

Реалізацію ковзаючої хеш функції:

```
  public NextHash(token: number): number {
    this.hash = (this.hash * this.base + this.hashes[index] *
this.maxBase + token)  % this.mod;
    this.hashes[index] = token;
    this.index = (this.index + 1) % this.k;
    return this.hash;
  }
```

Реалізацію алгоритму Winnowing:

```
public async *fingerprints(tokens: string[]):
AsyncIterableIterator<Fingerprint> {
    const hash = new RollingHash(this.k);
    let window: string[] = [];
    let filePos: number = -this.k;
    let bufferPos = 0;
    let minPos = 0;
    const buffer: number[] = new
Array(this.windowSize).fill(Number.MAX_SAFE_INTEGER);

    for await (const [hashedToken, token] of this.hashTokens(tokens)) {
      filePos++;
      window = window.slice(-this.k + 1);
      window.push(token);
      if (filePos < 0) {
        hash.nextHash(hashedToken);
        continue;
      }
```

```
      // minPos define far right hashing.
      bufferPos = (bufferPos + 1) % this.windowSize;
      buffer[bufferPos] = hash.nextHash(hashedToken);
      if (minPos === bufferPos) {
        // Scan buffer starting from bufferPos for the far right
minimal hashing.
        let i = (bufferPos + 1) % this.windowSize;
        for (; i !== bufferPos; i = (i + 1) % this.windowSize) {
          if (buffer[i] <= buffer[minPos]) {
            minPos = i;
          }
        }

        const offset = (minPos - bufferPos - this.windowSize) %
this.windowSize;
        const start = filePos + offset;
      }
      else {
        if (buffer[bufferPos] <= buffer[minPos]) {
          minPos = bufferPos;
          const start = filePos + ((minPos - bufferPos -
this.windowSize) % this.windowSize);
        }
       }
      yield {
         hash: buffer[minPos],
         start,
         stop: start + this.k - 1,
      };
    }
```

3.3 Реалізація індексування цифрових відбитків

Функція *cloneDetector()* виконує фактичне порівняння файлів. Спочатку файли представляються у вигляді токенів за допомогою абстрактних синтаксичних дерев. Далі кожен токен представляється у вигляді цифрового відбитку. Для кожного цифрового відбутку виконується перевірка на наявність такого ж хешу в дереві. Якщо хешу не було виявлено, то даний цифровий відбиток додається до дерева, інакше цифровий відбиток додається до звіту. В кінці функцію виконується формування звіту. Реалізацію функції *cloneDetector()*:

```
public async cloneDetector(files: File[]): Promise<Report> {
    const report = new Report();
    const tokenizedFiles = files.map(f =>
this.tokenizer.tokenizeFile(f));
    for (const file of tokenizedFiles) {
      let kgram = 0;
      for await (const { hash, start, stop  } of
hashFilter.fingerprints(file.Ast)) {
        // add kgram to file
        file.kgrams.push(new Range(start, stop));

        const part: Occurrence = {
          file,
          side: { index: kgram, start, stop, data, Region.merge(
          file.mapping[start],
          file.mapping[stop]
        ) }
        };

        // look if the index already contains the given hashing
        const matches = this.index.get(hash);

        if (matches) {
          report.addOccurrences(hash, part, ...matches);
          matches.push(part);
        } else {
          this.index.set(hash, [part]);
        }
        kgram += 1;
      }
    }
    report.create();
    return report;
  }
```

3.4 Представлення результатів аналізу

У результаті індексування всі цифрові відбитки, які зустрічаються в кількох файлах, збираються і об’єднуються у результуючий звіт.

Даній звіт містить всі пари файлів, які мають принаймні один спільний цифровий відбиток, а також показники:

- схожість - представляє частку спільних відбитків пальців між двома файлами;

- найдовший фрагмент – це найбільша довжина послідовних цифрових відбитків, що збігаються між двома файлами;
- повне перекриття - це абсолютне значення спільних цифрових відбитків.

Приклад звіту представлено в таблиці 3.1.

Таблиця 3.1 – Приклад звіту

Файл 1	**Файл 2**	**Схожість**	**Найдовший фрагмент**	**Повне перекриття**
Example1.c	Example2.c	0.94	63	218
Example2.c	Example3.c	0.89	56	208
Example1.c	Example3.c	0.88	56	204

Текс файлів Example1.c, Example2.c, Example3.c представлено у додатку Б.

3.5 Тестування системи

Результати перевірки системи на виявлення дублікатів різних типів представлені у таблиці 3.2. Перевірку було виконано з залученням 40 файлів вихідного коду з системи ejudge.

Таблиця 3.2 – Статистика виявлення дублікатів різних типів

Тип дублікату	**Процент розпізнавання**
Повна копія	100%
Зміни у коментарях, відступах та ідентифікаторах	100%
Зміна порядку рядків коду	92%
Зміна, додавання чи вилучення операторів	91%

Отже, даний алгоритм успішно виявляє клони 1-3 типів. Клони 4 типу не брались до уваги тому що, вони показують лише те що програми вирішують одну й ту ж саму задачу, що відомо заздалегідь.

Система підтримує 5 мов програмування: C, C#, Java, JavaScript та Python. Приклад роботи системи для перерахованих мов програмування представлені у таблиці 3.3. Текст файлів представлені у додатку Б.

Таблиця 3.3 – Результати роботи системи для різних мов програмування

Файл 1	**Файл 2**	**Схожість**	**Найдовший фрагмент**	**Повне перекриття**
java_sample.java	java_sample-copy.java	0.98	9	38
c_sample.c	c_sample-copy.c	0.78	12	28
c-sharp_sample.cs	c-sharp_sample-copy.cs	0.89	29	90
python_sample.py	python_sample-copy.py	0.99	67	134
js_sample.js	js_sample-copy.js	1	36	72

Систему було застосовано до 259 файлів вихідного коду, що представляють собою вісім завдань з програмування на мові C. Дані файли було взято з системи ejudge з курсу «Програмування». Два рішення завдання, що мають схожість менше 70%, вважаються незалежними роботами.

На Рисунку 3.3 показано розподіл подібності коду студентських посилань.

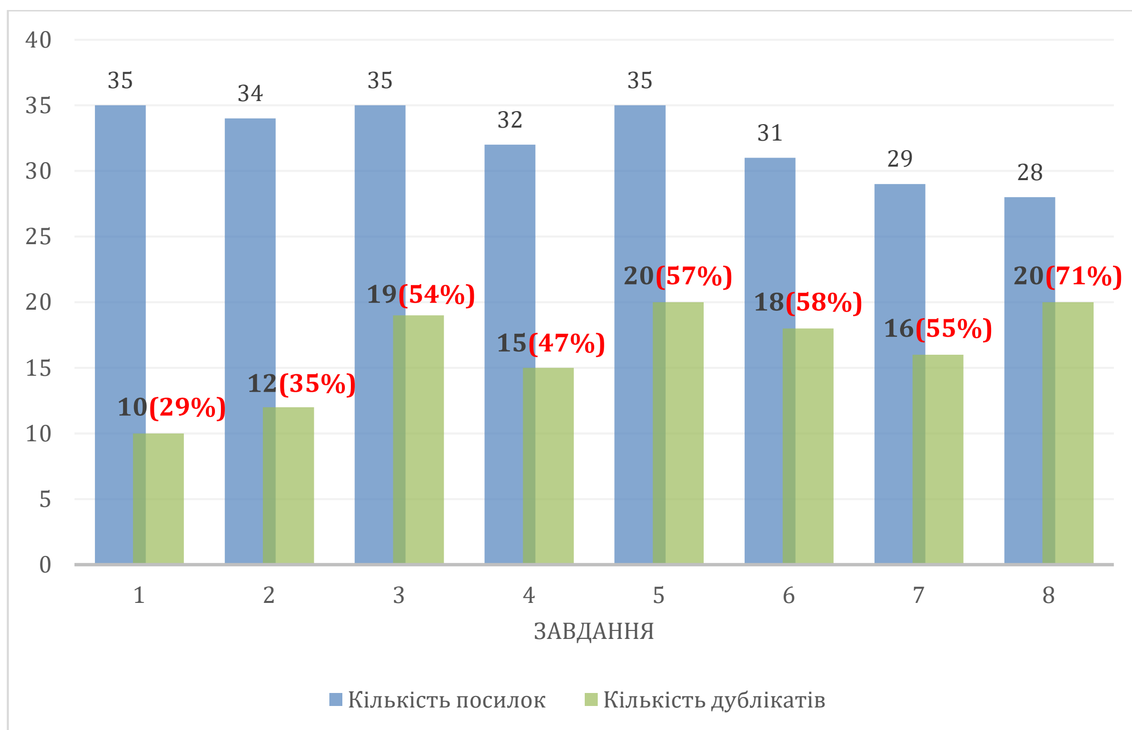

Рисунок 3.4 – Статистика дублікатів коду у студентських роботах

Перевірка показала, що 130 робіт є дублікатами, що складає 50% всіх робіт. Серед них 382 пари мають схожість більше ніж 98%. Така схожість свідчить про пряму копію без значних змін. Роботи, що мають найнижчу недопустиму оцінку подібності в деяких місяцях відрізняються, але в той же час містять велику кількість ідентичних допоміжних функцій та місць спільного коду.

Нижче на Рисунку 3.5 наведено приклад двох робіт, що мають коефіцієнт схожості 0,85. Дані роботи вважаються такими що містять дублікати.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/times.h>
#include <strings.h>
#include <ctype.h>

int ()
{
 int t1, t2, t;
 int timeinsec, nofattempts;
 char url[100], url1[80];
 strcpy(url, "url1");
 strcpy(url1, " url2");
 char word[15], *chk;
 chk = "word";
 FILE *fp;
 int syst = 1;
 fp = fopen("words", "r");
 t1 = time();
 while(chk != NULL)
 {
  chk = fgets(word, 15, fp);
  if (chk == NULL) exit(1);
  word [ strlen(word) - 1 ] = '\0';
  strcat(url, word);
  strcat(url, url1);
  nofattempts = nofattempts + 1;
```

```
#include <sys/times.h>
#include <sys/time.h>
#include <strings.h>
#include <ctype.h>

int ()
{
 int time1, time2, time_var;
 int timeinsec, nofattempts;
 char url[100], url1[80];
 strcpy(url, "u1");
 strcpy(url1, " u2");
 char word[15], *chk;
 chk = "word";
 FILE *fp;
 int syst = 1;
 fp = fopen("words", "r");
 time1 = time();
 while(chk != NULL)
 {
  chk = fgets(word, 15, fp);
  if (chk == NULL) exit(1);
  word [ strlen(word) - 1 ] = '\0';
  strcat(url, word);
  strcat(url, url1);
```

Рисунок 3.5 – Приклад дублікату вихідного коду

Текст програм, що зображені на рисунку 3.5, представлено у додатку В.

ВИСНОВКИ

У даній роботі було описано та реалізовано технологію виявлення нечітких дублікатів вихідного коду на основі абстрактного синтаксичного дерева. Алгоритм може ефективно виявляти нечіткі дублікати коду за допомогою процесу генерування хеш-значень та їх порівняння. Дана система дає можливість аналізувати програми, виявляючи частку спільних цифрових відбитків між файлами. При тестуванні системи було встановлено, що даний алгоритм може розпізнавати клони 1-2 типів на 100%, а клони 3 типу на 91%, що свідчить про можливе застосування даної технології у сфері освіти. Клони 4 типу у дані задачі не розглядались.

Результати дослідження вказують на те що, дублікати коду у студентських роботах можуть значною мірою загрожувати академічній доброчесності. Оскільки при тестуванні 259 студентських робіт виявилось, що 50% є дублікатами. Застосування системи виявлення клонів коду пом’якшує дану проблему. Дана технологія враховує підтримку кількох мов програмування та вимоги до швидкості проведення аналізу для використання під час перевірки програм у рамках навчального процесу.

ЛІТЕРАТУРА

1. Allen E. Bug Patterns in Java.
2. Vashisht A. et al. A DETAILED STUDY OF SOFTWARE CODE CLONING. 2018. Vol. 9. P. 20–32.
3. Baker B.S. On finding duplication and near-duplication in large software systems // Reverse Eng. - Work. Conf. Proc. IEEE, 1995. P. 86–95.
4. Roy C.K., Cordy J.R. An empirical study of function clones in open source software // Proc. - Work. Conf. Reverse Eng. WCRE. 2018. P. 81–90.
5. Mariani L., Micucci D. AuDeNTES // ACM Trans. Comput. Educ. ACM PUB27 New York, NY, USA , 2012. Vol. 12, № 1.
6. Pierce J., Zilles C. Investigating student plagiarism patterns and correlations to grades // Proc. Conf. Integr. Technol. into Comput. Sci. Educ. ITiCSE. Association for Computing Machinery, 2017. P. 471–476.
7. Bellon S. et al. Comparison and evaluation of clone detection tools // IEEE Trans. Softw. Eng. 2007. Vol. 33, № 9. P. 577–591.
8. Vislavski T. et al. LICCA: A tool for cross-language clone detection // 25th IEEE Int. Conf. Softw. Anal. Evol. Reengineering, SANER 2018 - Proc. Institute of Electrical and Electronics Engineers Inc., 2018. Vol. 2018-March. P. 512–516.
9. Manso A. et al. Plagiarism Detection in Algorithms-a Case Study Using Algorithmi. 2020.
10. Ducasse S., Rieger M., Demeyer S. Language independent approach for detecting duplicated code // Conf. Softw. Maint. 1999. P. 109–118.
11. for J.J.-P. of the 1993 conference of the C., 1993 undefined. Identifying redundancy in source code using fingerprints // dl.acm.org.
12. Jiang L. et al. DECKARD: Scalable and accurate tree-based detection of code clones // Proc. - Int. Conf. Softw. Eng. 2007. P. 96–105.
13. Novak M., Joy M., Kermek D. Source-code Similarity Detection and Detection Tools Used in Academia // ACM Trans. Comput. Educ. ACM

PUB27 New York, NY, USA , 2019. Vol. 19, № 3.

14. Salman Khan M., Salman M. UNF Digital Commons A Topic Modeling approach for Code Clone Detection Suggested Citation. 2019.
15. Mayrand J., Leblanc C., Merlo E.M. Experiment on the automatic detection of function clones in a software system using metrics // Conf. Softw. Maint. IEEE, 1996. P. 244–253.
16. Komondoor R., Horwitz S. Using Slicing to Identify Duplication in Source Code // Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). Springer, Berlin, Heidelberg, 2019. Vol. 2126 LNCS. P. 40–56.
17. Krinke J. Identifying similar code with program dependence graphs // Reverse Eng. - Work. Conf. Proc. 2001. P. 301–309.
18. Leitão A.M. Detection of Redundant Code Using R 2 D 2 // Softw. Qual. J. 2004 124. Springer, 2004. Vol. 12, № 4. P. 361–382.
19. Schleimer S., Wilkerson D.S., Aiken A. Winnowing: Local Algorithms for Document Fingerprinting. 2003.
20. Xia W. et al. Ddelta: A deduplication-inspired fast delta compression approach // Perform. Eval. North-Holland, 2018. Vol. 79. P. 258–272.

ДОДАТКИ

Додаток А. Лістинг програмного коду

AST.ts

```
import { default as Parser, SyntaxNode } from "tree-sitter";

export class AST {
  public static supportedLanguages = ["cpp", "c-sharp", "java",
"javascript", "python"];

  public static IsSupportedLanguage(language: string): boolean {
    return this.supportedLanguages.includes(language);
  }

  public static RegisterLanguage(language: string): void {
     try {
       require("tree-sitter-" + language);
     }
     catch (error) {
        throw new Error("tree-sitter-${language} language not found.");
     }
    this.supportedLanguages.push(language);
   }

  public AST(text: string): string {
    const tree = this.parser.parse(text);
    return tree.rootNode.toString();
  }
}
```

RollingHash.ts

```
export class RollingHash {

  private readonly hashes: number[];
  private readonly maxBase: number;
  private index = 0;
  private hash = 0;

  readonly k: number = 23;
  readonly mod: number = 33554393;
  readonly base: number = 9999991;
```

```
  constructor() {
    this.k = k;
    this.maxBase = this.mod - this.modPow(this.base, this.k, this.mod);
    this.hashes = new Array(this.k).fill(0);
  }

  private ModularPow(base: number, exp: number, mod: number): number {
    let result = 1;
    base = base % mod;
    while (exp > 1) {
      if (exp % 2 == 1) {
        result = (result * base) % mod;
      }
      exp = exp >> 1;
      base = (base * base) % mod;
    }
    return result;
  }

  public NextHash(token: number): number {
    this.hash = (this.hash * this.base + this.hashes[index] *
this.maxBase + token)  % this.mod;
    this.hashes[index] = token;
    this.index = (this.index + 1) % this.k;
    return this.hash;
  }
}
```

Winnowing.ts

```
import { Fingerprint, HashFilter } from "./hashFilter";
import { RollingHash } from "./rollingHash";

export class WinnowingFilter extends HashFilter {
  private readonly k: number;
  private readonly windowSize: number;

  constructor(k: number /*default is 23*/, windowSize: number) {
    this.k = k;
    this.windowSize = windowSize;
  }
```

```
  public async *fingerprints(tokens: string[]):
AsyncIterableIterator<Fingerprint> {
    const hash = new RollingHash(this.k);
    let window: string[] = [];
    let filePos: number = -this.k;
    let bufferPos = 0;
    let minPos = 0;
    const buffer: number[] = new
Array(this.windowSize).fill(Number.MAX_SAFE_INTEGER);

    for await (const [hashedToken, token] of this.hashTokens(tokens)) {
      filePos++;
      window = window.slice(-this.k + 1);
      window.push(token);
      if (filePos < 0) {
        hash.nextHash(hashedToken);
        continue;
      }
      // minPos define far right hashing.
      bufferPos = (bufferPos + 1) % this.windowSize;
      buffer[bufferPos] = hash.nextHash(hashedToken);
      if (minPos === bufferPos) {
        // Scan buffer starting from bufferPos for the far right
minimal hashing.
        let i = (bufferPos + 1) % this.windowSize;
        for (; i !== bufferPos; i = (i + 1) % this.windowSize) {
          if (buffer[i] <= buffer[minPos]) {
            minPos = i;
          }
        }

        const offset = (minPos - bufferPos - this.windowSize) %
this.windowSize;
        const start = filePos + offset;
      }
      else {
        if (buffer[bufferPos] <= buffer[minPos]) {
          minPos = bufferPos;
          const start = filePos + ((minPos - bufferPos -
this.windowSize) % this.windowSize);
        }
       }
      yield {
```

```
          hash: buffer[minPos],
          start,
          stop: start + this.k - 1,
        };
      }
    }
  }
```

```
Utils.ts
ranging(): Report {
    let isFound = false;
    let direction;
    if (this.enabled) {
      let acceptable: SnappingType[] | undefined;
      const activeProviders =
this.providersContainer.getActiveProviders();
      for (const provider of activeProviders) {
        if (acceptable && !acceptable.includes(provider.type)) {
          continue;
        }
        const result = provider(finalPoint, direction);

        provider.drawTooltip(result);
        if (result.found) {
          result.snappingType = provider.type;
         direction = direction || result.direction;

          if (!result.canBeModifiedBy) {
            return result;
          } else {
            finalPoint = result.point;
            acceptable = result.canBeModifiedBy;
            isFound = true;
            snappingType = provider.type;
          }
        }
      }
    }
    return { point: finalPoint, found: isFound, snappingType:
snappingType };
  }

async restoreSnapshot() {
```

```
    await this._viewerService.ready();
    const sortedCommandsList = await
this._snapshotProvider.getCommandsForScenario();
    this.originalCommandSequence = sortedCommandsList;
    this.savedCommandSequence =
sortedCommandsList.filter(command => command.isSaved);
    for (const commandSnapshot of sortedCommandsList) {
      this.executeSnapshotCommand(commandSnapshot)
    }

    const finalVirtualEntities =
this._drawingManager.virtualModel.entities;
    for (const entity of finalVirtualEntities) {
      const command: Command<any> | undefined =
FactoriesScope.drawingCommandFactory.build(
        entity.virtualCurve.getCreateCommandType(),
        entity.virtualCurve.generateToolConfig(),
        CommandScope.LOCAL
      );

      if (command) {
        command.commandLinkId = entity.commandLinkId;
        command.setupDependencies(this._factory);
        command.redo();
      }
    }
    this._onSnapshotRestored.next(true);
  }
```

Index.ts

```
import { Tokenizer } from "../tokenizer";
import { Report, Occurrence } from "./report";
import { WinnowwingFilter } from "../WinnowwingFilter";
import { Options } from "../options";
import { HashFilter } from "../hashFilter";

export interface DolosOptions {
  k: number;
  windowSize: number;
  language: string;
}
export class DefaultOptions implements Options{
```

```
  public static defaultLanguage = "c";
  public static defaultKgramLength = 23;
  public static defaultKgramsInWindow = 17;
}

export class Indexer {
  private readonly tokenizer: Tokenizer;
  private readonly hashFilter: HashFilter;
  private readonly index: Map<Hash, Array<Occurrence>> = new Map();

  constructor(options: Options = new Options()) {
    this.hashFilter = new WinnowFilter(options.k, options.windowSize);
  }

private createQuadTrees() {
    this.logger.info(`Creating new Quadtrees`);
    const quadTreeBounds = this.getViewerExtentsBox();
    if (!quadTreeBounds) {
      return;
    }

    this._backgroundQuadTree = new Quadtree(quadTreeBounds, 10, 6);
    this._foregroundQuadTree = new Quadtree(quadTreeBounds, 10, 6);
    this._ready.next();
  }

public async cloneDetector(files: File[]){
    const tokenizedFiles = files.map(f => this.tokenizeFile(f));
    const report = new Report(this.options);
    for (const file of tokenizedFiles) {
      let kgram = 0;
      for (const { hash, start, stop} of
hashFilter.fingerprints(file.Ast)) {
        file.kgrams.push(new Range(start, stop));
        const part: Occurrence = {
          fileside: { index: kgram, start, stop, data,
        Region.merge(file.mpp [start],file.mpp [stop])}
        };
        // look if the index already contains the given hashing
        const matches = this.index.get(hash);
        if (matches) {
          report.addOccurrences(hash, part, ...matches);
          matches.push(part);
        } else {
          this.index.set(hash, [part]);
```

```
          }
          kgram += 1;
        }
      }
      report.create();
      return report;
    }
  }
```

Додаток Б. Лістинг текстів файлів

Example1.c

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>

int binarysearch(int a, int mass[], int n);
void InsertionSort(int n, int mass[]);

int main()
{
    int N, a;
    printf("Input N: ");
    scanf_s("%d", &N);
    int* mass;
    mass = (int *)malloc(N * sizeof(int));
    printf("Input the array elements:\n");
    for (int i = 0; i < N; i++)
        scanf_s("%d", &mass[i]);
    InsertionSort(N, mass);
    printf("Sorted array:\n");
    for (int i = 0; i < N; i++)
        printf("%d ", mass[i]);
    printf("\n");
    printf("Input variable 'a' for search: ");
    scanf_s("%d", &a);
    int k;
    k = binarysearch(a, mass, N);
    if (k != -1)
    {
        printf("The index of the element is %d\n", k);
    }
    else
        printf("The element isn't found!\n");
    free(mass);
    _getch();
    return 0;
}

int binarysearch(int a, int mass[], int n)
{
    int low, high, middle;
    low = 0;
    high = n - 1;
    while (low <= high)
    {
        middle = (low + high) / 2;
        if (a < mass[middle])
```

```
            high = middle - 1;
        else if (a > mass[middle])
            low = middle + 1;
        else
            return middle;
    }
    return -1;
}

void InsertionSort(int n, int mass[])
{
    int newElement, location;

    for (int i = 1; i < n; i++)
    {
        newElement = mass[i];
        location = i - 1;
        while (location >= 0 && mass[location] > newElement)
        {
            mass[location + 1] = mass[location];
            location = location - 1;
        }
        mass[location + 1] = newElement;
    }
}
```

Example2.c

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>

int binarysearch(int a, int mass[], int n);
void InsertionSort(int n, int mass[]);

int main()
{
    int N, a;
    printf("Input N: ");
    scanf_s("%d", &N);
    int* mass;
    mass = (int *)malloc(N * sizeof(int));
    printf("Input the array elements:\n");
    for (int i = 0; i < N; i++)
        scanf_s("%d", &mass[i]);
    InsertionSort(N, mass);
    printf("Sorted array:\n");
    for (int i = 0; i < N; i++)
        printf("%d ", mass[i]);
    printf("\n");
    printf("Input variable 'a' for search: ");
```

```
    scanf_s("%d", &a);
    int k;
    k = binarysearch(a, mass, N);
    if (k != -1)
    {
        printf("The index of the element is %d\n", k);
    }
    else
        printf("The element isn't found!\n");
    free(mass);
    _getch();
    return 0;
}

//add some minor changes
int binarysearch(int a, int mass[], int n)
{
    int low = 0, high = n - 1, middle;
    while (low <= high)
    {
        middle = (low + high) * 0.5;
        if (a < mass[middle])
            high = middle - 1;
        else if (a > mass[middle])
            low = middle + 1;
        else
            return middle;
    }
    return -1;
}

void InsertionSort(int n, int mass[])
{
    int newElement, location;

    for (int i = 1; i < n; i++)
    {
        newElement = mass[i];
        location = i - 1;
        while (location >= 0 && mass[location] > newElement)
        {
            mass[location + 1] = mass[location];
            location = location - 1;
        }
        mass[location + 1] = newElement;
    }
}
```

Example3.c

```
#include <stdio.h>
#include <malloc.h>
#include <conio.h>

int binarysearch(int a, int mass[], int n);
void InsertionSort(int n, int mass[]);

int main()
{
    int N, a;
    printf("Input N: ");
    scanf_s("%d", &N);
    int* mass;
    mass = (int *)malloc(N * sizeof(int));
    printf("Input the array elements:\n");

    for (int i = 0; i < N; i++)
        scanf_s("%d", &mass[i]);
    InsertionSort(N, mass);

    printf("Sorted array:\n");
    for (int i = 0; i < N; i++)
        printf("%d ", mass[i]);
    printf("\n");
    printf("Input variable 'a' for search: ");
    scanf_s("%d", &a);
    int k;
    k = binarysearch(a, mass, N);

    if (k != -1)
        printf("The index of the element is %d\n", k);

    else
        printf("The element isn't found!\n");

    free(mass);
    _getch();
    return 0;
}

int binarysearch(int a, int mass[], int n)
{
    int low = 0, high = n - 1, middle;
    while (low <= high)
    {
        middle = (low + high) / 2;
        if (a < mass[middle])
            high = middle - 1;
        else if (a > mass[middle])
```

```
            low = middle + 1;
        else
            return middle;
    }
    return -1;
}

void InsertionSort(int n, int mass[])
{
    int newElement, location;
       int i = 1;
    while(i < n)
    {
        newElement = mass[i];
        location = i - 1;
        while (location >= 0 && mass[location] > newElement)
        {
            mass[location + 1] = mass[location];
            location = location - 1;
        }
        mass[location + 1] = newElement;
              i++;
    }
}
```

java_sample.java

```
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {
    // Create a HashMap object called capitalCities
    HashMap<String, String> capitalCities = new HashMap<String, String>();

    // Add keys and values (Country, City)
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
  }
}
```

java_sample-copy.java

```
import java.util.HashMap;

public class Main {
  public static void main(String[] args) {
    // Create a HashMap object called capitalCities
```

```
    HashMap<String, String> capitalCities = new HashMap<String, String>();

    // Add keys and values (Country, City)
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
  }
}
```

c_sample.c

```
#include<stdio.h>
#include<conio.h>

int fact(int f) {
  if (f==0 || f==1) {
    printf("Calculated Factorial");
    return 1;
  }
  return f * fact(f - 1);
}

int main(void) {
  int f = 12;
  printf("The factorial of %d is %d \n", f, fact(f));
  getch();
  return 0;
}
```

c_sample-copy.c

```
#include<stdio.h>
#include<conio.h>

long factorial(long f) {
  if (f==0 || f==1) {
    printf("Calculated Factorial");
    return 1;
  }
  return f * fact(f - 1);
}

int main(void) {
  long f = 12;
  printf("The factorial of %d is %d \n", f, fact(f));
  getch();
  return 0;
}
```

c-sharp_sample.cs

```
using System;
class bubblesort
{
	static void Main(string[] args)
	{
		int[] a = { 30, 20, 50, 40, 10 };
		int t;
		Console.WriteLine("The Array is : ");
		for (int i = 0; i < a.Length; i++)
		{
			Console.WriteLine(a[i]);
		}
		for (int j = 0; j <= a.Length - 2; j++)
		{
			for (int i = 0; i <= a.Length - 2; i++)
			{
				if (a[i] > a[i + 1])
				{
					t = a[i + 1];
					a[i + 1] = a[i];
					a[i] = t;
				}
			}
		}
		Console.WriteLine("The Sorted Array :");
		foreach (int aray in a)
			Console.Write(aray + " ");
		Console.ReadLine();
	}
}
```

c-sharp_sample-copy.cs

```
using System;
class bubblesort
{
	static void Main(string[] args)
	{
		int[] a = { 30, 20, 50, 40, 10 };
		int t;
		for (int i = 0; i < a.Length; i++)
		{
			Console.WriteLine(a[i]);
		}
		for (int j = 0; j <= a.Length - 2; j++)
		{
```

```
                for (int i = 0; i <= a.Length - 2; i++)
                {
                    if (a[i] > a[i + 1])
                    {
                        t = a[i + 1];
                        a[i + 1] = a[i];
                        a[i] = t;
                    }
                }
            }
            foreach (int aray in a)
                Console.Write(aray + " ");
            Console.ReadLine();
        }
}
```

python_sample.py

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

#print a list of the _id values of the inserted documents:
print(x.inserted_ids)
```

python_sample-copy.py

```
import pymongo
```

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 6, "name": "Michael", "address": "Valley 345"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)
print(x.inserted_ids)
```

js_sample.js

```
function compareName(a, b) {

    // converting to uppercase to have case-insensitive comparison
    const name1 = a.name.toUpperCase();
    const name2 = b.name.toUpperCase();

    let comparison = 0;

    if (name1 > name2) {
        comparison = 1;
    } else if (name1 < name2) {
        comparison = -1;
    }
    return comparison;
}

const students = [{name: 'Sara', age:24},{name: 'John', age:24}, {name:
'Jack', age:25}];

console.log(students.sort(compareName));
```

js_sample-copy.js

```
function compare(a, b) {

    // converting to uppercase to have case-insensitive comparison
    const name1 = a.name.toUpperCase();
    const name2 = b.name.toUpperCase();

    let comparison = 0;

    if (name1 > name2) {
        comparison = 1;
    } else if (name1 < name2) {
        comparison = -1;
    }
    return comparison;
}

const students = [{name: 'Sara', age:24},{name: 'John', age:24}, {name:
'Jack', age:25}];

console.log(students.sort(compareName));
```

Додаток В. Приклад програм з дублікатами.

Програма на рисунку 3.4 зліва:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/times.h>
#include <strings.h>
#include <ctype.h>

int ()
{
  int t1, t2, t;
  int timeinsec, nofattempts;
  char url[100], url1[80];
  strcpy(url, "url1");
  strcpy(url1, "url2");
  char word[15], *chk;
  chk = "word";
  FILE *fp;
  int syst = 1;
  fp = fopen("words", "r");
  t1 = time();
  while(chk != NULL)
  {
    chk = fgets(word, 15, fp);
    if (chk == NULL) exit(1);
    word [ strlen(word) - 1 ] = '\0';
    strcat(url, word);
    strcat(url, url1);
    nofattempts = nofattempts + 1;
    printf("\n %s %d\n",word,nofattempts);
    if (strlen(word) == 3)
    syst = system(url);
    if (syst == 0)
    {
       t2 = time();
       t = t2 - t1;
       timeinsec = t/1000000000;
       printf("\n !!! here's the passowrd:- %s",word);
       printf("\n Total .of atempts: %d\n",nofattempts);
       printf("\n The total time_var taken: %d seconds", timeinsec);
       exit(1);
    }
    strcpy(url, "");
    strcpy(url, "wget --http-user= --http-passwd=");
  }
}
```

Програма на рисунку 3.4 праворуч:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/times.h>
#include <sys/time.h>
#include <strings.h>
#include <ctype.h>

int ()
{
  int time1, time2, time_var;
  int timeinsec, nofattempts;
  char url[100], url1[80];
  strcpy(url, "u1");
  strcpy(url1, "u2");
  char word[15], *chk;
  chk = "word";
  FILE *fp;
  int syst = 1;
  fp = fopen("words", "r");
  time1 = time();
  while(chk != NULL)
  {
    chk = fgets(word, 15, fp);
    if (chk == NULL) exit(1);
    word [ strlen(word) - 1 ] = '\0';
    strcat(url, word);
    strcat(url, url1);
    if (strlen(word) == 3)
    {
      syst = system(url);
      nofattempts = nofattempts + 1;
      printf("\n %s %d\n",word,nofattempts);
    }
    if (syst == 0)
    {
       time2 = time();
       time_var = time2 - time1;
       timeinsec = time_var/1000000000;
       printf("\n The Password is: %s",word);
       printf("\n  of Attempts: %d\n",nofattempts);
       printf("\n Time Taken: %d seconds\n", timeinsec);
       exit(1);
    }
    strcpy(url, "");
    strcpy(url, "wget --http-user= --http-passwd=");
  }
}
```