Одержано редакцією 15.04.2021 р.
Прийнято до публікації 10.06.2021 р.

УДК 519.688

DOI 10.31651/2076-5886-2021-1-68-84

PACS 02.60.-x, 02.60.Pn

ХАРЧЕНКО Віталій Вікторович,
студент спеціальності «Прикладна математика» Черкаського національного університету імені Богдана Хмельницького

ДІДКОВСЬКИЙ Руслан Михайлович,
доктор технічних наук, доцент, доцент кафедри прикладної математики та інформатики Черкаського національного університету імені Богдана Хмельницького
e-mail: didkovskyirm@vu.cdu.edu.ua
ORCID 0000-0002-5166-7564

СЕРДЮК Олександр Анатолійович,
кандидат економічних наук, доцент, доцент кафедри прикладної математики та інформатики Черкаського національного університету імені Богдана Хмельницького
e-mail: serdyuk@ukr.net
ORCID 0000-0002-3919-4661

РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ ПЕРЕВІРКИ ПРОГРАМНОГО КОДУ НА НАЯВНІСТЬ ПЛАГІАТУ

У роботі описано аналіз передумов розробки та структуру розробленої серверної системи перевірки програмного коду, що є результатом виконання студентами практичних завдань з програмування, на наявність плагіату. Умови розробки системи складено на основі аналізу чотирьох наявних систем пошуку плагіату: MOSS, Codeqiury, Unicheck, CCFinderX. Для кожної з розглянутих систем визначено її переваги та недоліки, на основі чого визначено потрібні властивості розроблюваної системи. Визначено структуру серверної частини та потрібне програмне забезпечення й мову програмування для практичної реалізації. Визначено та наведено алгоритми для токенізації програмного коду та порівняння двох різних фрагментів коду з метою оцінки їх подібності. Реалізована у результаті система може використовуватись для надання API програмам пошуку плагіату у програмному коді.

***Ключові слова**: плагіат, плагіат у програмному коді, система перевірки на наявність плагіату, серверна система.*

Вступ

Запозичення коду серед студентів – це проблема, яка непокоїть багатьох викладачів університету, що викладають дисципліни, пов'язані з вивченням програмування. На жаль, викладачу важко особисто прослідкувати за унікальністю всіх робіт студентів. Окрім того, велику частку робочого часу доводиться витрачати на перевірку робіт, замість того, щоб цей час присвятити, наприклад, підготовці ще якісніших завдань для студентів та ознайомленню з новітніми трендами у

програмуванні. Для полегшення визначення наявності запозичень у програмному коді серед студентів можуть використовуватись системи перевірки робіт на плагіат, але у своїй більшості вони призначені для пошуку плагіату у звичайному тексті, і, відповідно, не пристосовані до аналізу програмного коду, який має свої особливості.

Мета статті – провести аналіз деяких сучасних доступних систем пошуку плагіату, розробити вимоги до власної системи та описати власну розроблену систему оцінки плагіату.

Практичне значення одержаних результатів. Результати, отримані в ході роботи над статтею, можна використовувати у практичних цілях для розробки систем пошуку плагіату у програмному коді і як навчальний матеріал.

Виклад основного матеріалу

1. Плагіат у програмному коді

Швидкий розвиток ІТ-індустрії та розширення її сфери впливу на інші технології, які до не давнього часу здавалися б вільні та незалежні від комп'ютерів, потребують нових кадрових спеціалістів. На фоні появи нових професій, виникла потреба в навчанні спеціалістів пов'язаних з програмуванням. Оскільки кількість студентів, які обрали програмування збільшилась, з'являється тенденція запозичення програмного коду, та видача його як власного авторського продукту. Тому з'являється потреба в якісній боротьбі з плагіатом в програмному продукті. На допомогу викладачам приходять системи виявлення плагіату. Від простих програм, які порівнюють два файли між собою, до автоматичних систем без безпосередньої участі викладача в ролі регулятора.

Плагіат програмного коду – це копіювання або відтворення вихідного коду без письмового дозволу від оригінального творця. Включає мінімальну адаптацію коду або включення фрагментів оригінального коду в свій код. Перетворення оригінального коду на іншу мову програмування можна вважати плагіатом, але тут все залежить від контексту. Використання генераторів коду студентами підпадає під зону ризику, зазвичай, виконання завдання в університетах не потребують використовування сторонніх засобів [27].

Замість того, щоб підозрювати всі роботи на плагіат, можна доручити цю роботу спеціальним системам для визначення плагіату. Розробці подібній системі, як раз і присвячена дана робота.

Плагіат в коді можна визначити як копіювання вихідного коду або окремих його частин без внесення якихось суттєвих змін в його структуру або з незначною його доробкою. Додавання коментарів до існуючого коду, зміни назв змінних, перестановка блоків функцій або класів.

2. Огляд програм для виявлення плагіату

Щоб створити власну системи оцінки програмного коду на плагіат, потрібно дослідити відомі системи для того, щоб мати представлення, як виглядають такі системи, та дослідити їх переваги й недоліки. На основі отриманих результатів спроектувати власний програмний продукт з уникненням типових недоліків. Для цього дослідимо такі відомі системи:

- MOSS [18];
- Codeqiury [5];
- Unicheck [23];
- CCFinderX [4].

Measure Of Software Similarity

MOSS – перша в світі програма для обчислення плагіату створена в 1994 Алексом Ейкеном [18]. Являє собою достатньо надійною програмою, дозволяє якісно перевірити програмний код на ознаки плагіату (рис. 1). Опирається на власну закриту базу даних з програмними кодами. Підтримує багато сучасних мов, включаючи Java, C, C++, Python, JavaScript. Використовує алгоритм відсіву – це алгоритм працює на основі виявлення унікальних особливостей документів, перетворюючи файли в набір хеш-значень, які називаються відбитками [1, 8].

Принцип роботи полягає в тому, що обраний файл порівнюється з власною базою даних. Виявляє подібність у програмах, та надає користувачу перелік подібних файлів до файлу, який перевіряють.

Переваги:

– легкий в роботі та безкоштовний;
– будь-хто може користуватися;
– підтримує багато мов програмування;
– власний ефективний алгоритм.

Недоліки:

– не дозволяє автоматично перевіряти файли на плагіат;
– оцінка плагіату на розсуд того, хто перевіряє;
– не має веб-версії.
– запускається за допомогою скриптів.

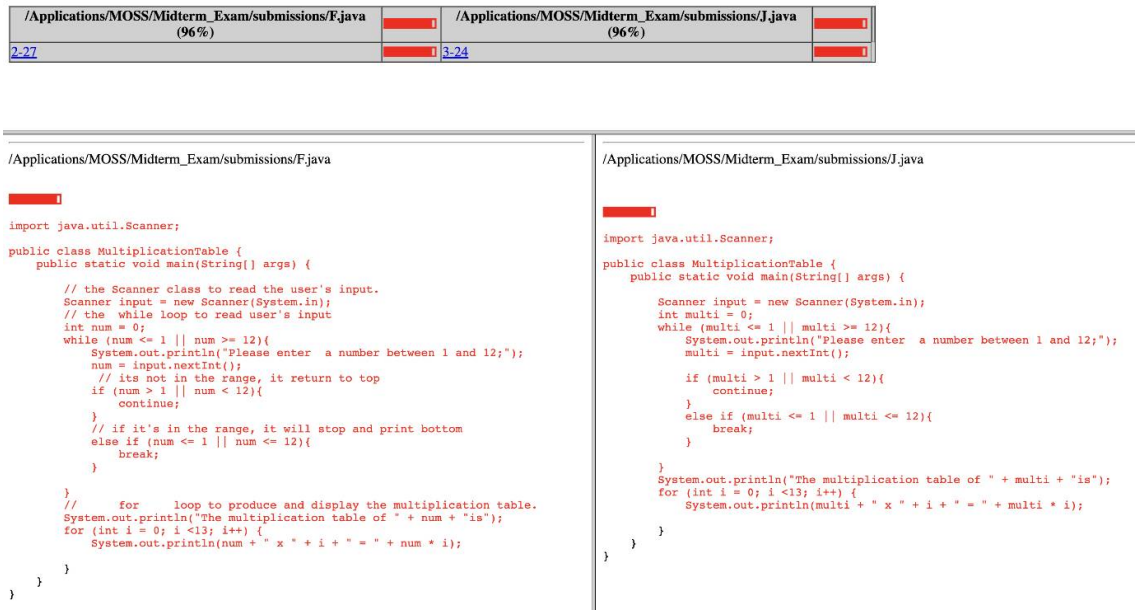

Рис 1. Інтерфейс системи MOSS

Codequiry

Codequiry – одна з найсучасніших системи перевірки плагіату в програмному коді. Детально вираховує унікальність коду, намагається виявити логічні шаблони та унікальний стиль коду (рис. 2). Перетворює програмний код у списки токенів, на другому етапі, шукає подібні кластери токенів у схожих групах [3]. Надає детальний інформацію про плагіат користувачу [5].

Переваги:

– має сучасний веб-інтерфейс, що спрощує роботу з системою;
– має велику базу публічного коду для перевірки на плагіат;

– підтримує велику кількість сучасних мов програмування;
– надає детальну інформацію про свою роботу.

Недоліки:

– платна система, передбачає помісячну підписку;
– демоверсія дозволяє перевірити до 20 файлів;

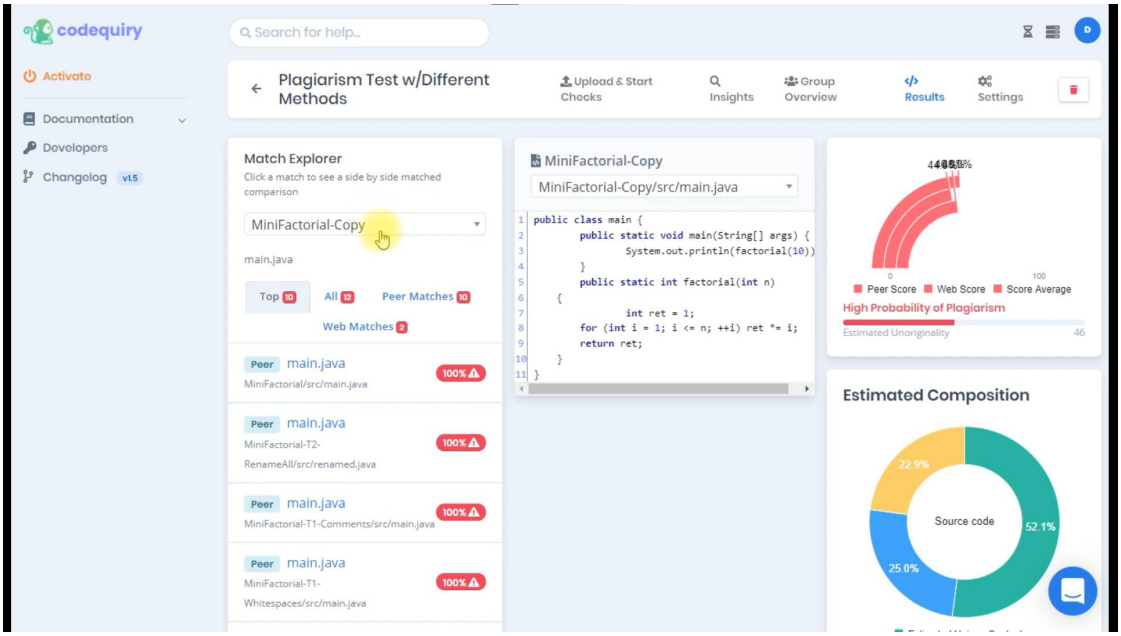

Рис 2. Інтерфейс системи перевірки плагіату Codequiry

Unicheck

Unicheck – широко відома система для перевірки плагіату в текстах. З підвищенням популярності на курси пов'язані з програмуванням, з 2019 року запустили бета-версію системи перевірки плагіату в програмному коді. Використовують власний алгоритм на основі штучного інтелекту [9]. Алгоритм базується на розумінні програмного коду так, як це робить комп'ютер. На момент написання роботи в 2021 році, система знаходиться в бета-тестуванні [23].

Переваги:

– надає детальний звіт про плагіат, відсотки, першоджерело;
– може порівнювати як два файли між собою, так і з базою даних;
– підтримує більшість сучасних мов програмування.

Недоліки:

– знаходиться в тестуванні;
– має ліміт на безкоштовну перевірку файлів, потрібно купувати додаткові спроби тестування.

CCFinderX

CCFinderX – детектор запозичення коду, виявляє клонування коду на різних мовах програмування (рис. 3). Покращена версія попереднього продукту CCFinder, нова архітектура дозволяє використовувати мультизадачність процесора користувача в виявленні плагіату, надає динамічну інформацію про запозичення [4].

Переваги:

– відкритий програмний продукт, користувач може модифікувати його під різні мови програмування;
– надає аналітичні дані про запозичення;

– має графічний інтерфейс.

Недоліки:

– не може працювати з файлами, які написані на декількох мовах програмування;
– складний запуск, необхідні знання роботи з терміналом;
– може порівнювати тільки два файли між собою.

3. Результати дослідів систем з оцінки плагіату

Дослідивши системи представлені вище, виявилось такі характерні ознаки:

– більшість з наведених систем мають сервер, окрім CCFinderX, він працює в локальному режимі з файлами, які користувач йому надав;
– більшість подібних систем мають власні публічні або приватні бази з зразками, з якими відбувається порівнювання, окрім CCFinderX, оскільки він автономний;
– мають власні вдосконалені алгоритми на основі відомих, деякі мають навіть штучний інтелект;
– всі наведені системи мають графічний інтерфейс.

Також виявились і недоліки подібних систем, наприклад:

– деякі екземпляри мають десктопний клієнт, тобто, потребують завантаження з мережі, та додаткових налаштувань, як MOSS та CCFinderX;
– сучасні системи такі, як Unicheck та Codequiry, передбачають користування на платній основі;
– не володіють централізованою системою відображення результатів, тобто викладач не може слідкувати за тими, хто здав завдання.

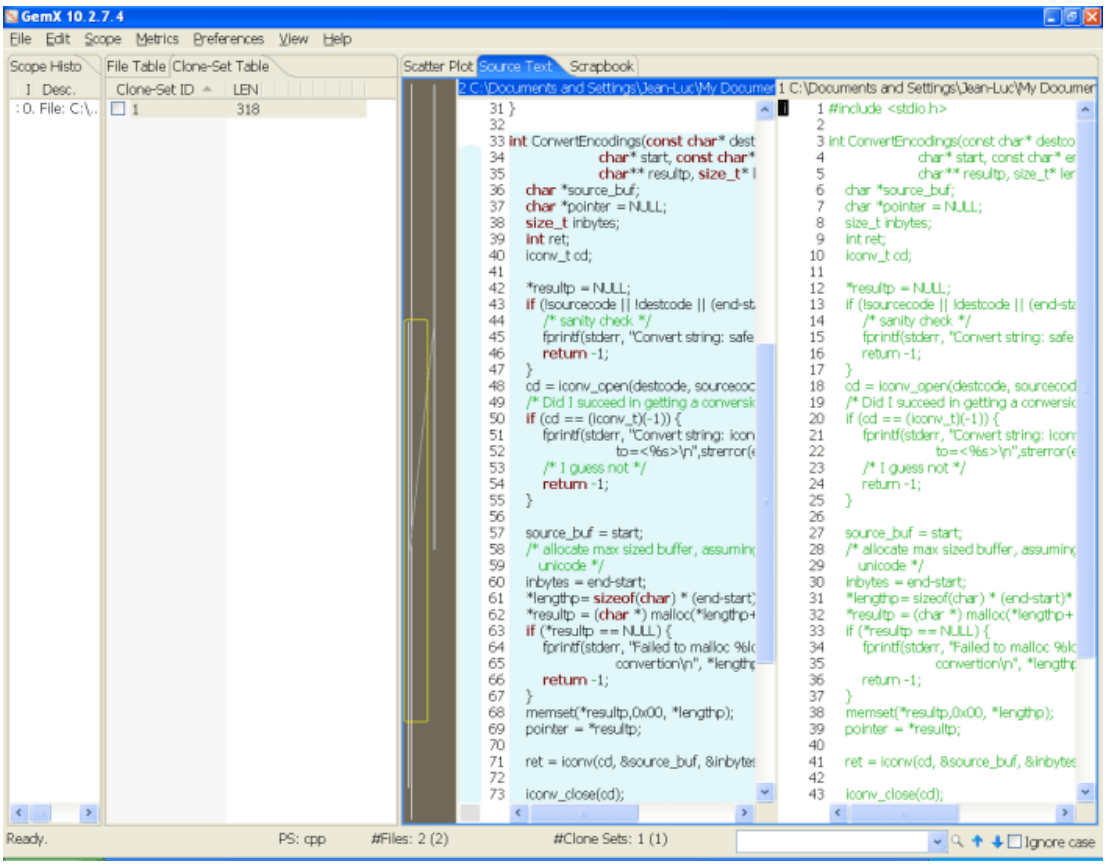

Рис 3. Інтерфейс системи перевірки коду на плагіат CCFinderX.

4. Визначення структури сервера

На основі оцінки вище зазначених систем, було прийняте рішення про створення власної системи оцінки плагіату у програмному коді, у вигляді серверу. Сервер реалізований в ході роботи, має такі властивості:

- можливість реєстрації та авторизації користувача;
- перевірка файлів на плагіат та отримання результату.

У сервері існує два типи ролей для користувачів:

- User – має такі можливості:
  - реєстрація та авторизація;
  - зберігання файлів та перевірка їх на плагіат;
  - перегляд свого акаунта та своїх результатів перевірки файлів.
- Admin – має всі вище зазначені можливості, а також:
  - перегляд всіх користувачів зареєстрованих на сервері;
  - перегляд їх результатів перевірки коду на плагіат.

Щоб досягти результату в ході роботи, будуть використані наступні технології:

- Java 11 – основна мова програмування [13];
- IntelliJ IDEA – середовище розробки з підтримкою мови програмування Java [11];
- Gradle – збирач пакетів [10];
- Spring Boot 2 Framework – контейнер на основі якого розгортається сервер [22];
- REST API – визначення загальної архітектури сервера [21];
- JSON – обмін інформацією з сервером [14];
- OAuth2 – реєстрації та авторизації користувачів на сервері [20];
- JWT – обмін приватною зашифрованою інформацією між користувачем та сервером [15];

5. RESTful API

Application Programing Interface (API) – це набір визначень і протоколів, за допомогою якого відбувається взаємодія між компонентами програмного забезпечення. Цей архітектурний підхід обертається навколо надання програмного інтерфейсу набору послуг для різних програм, які обслуговують різних споживачів [21].

При використанні архітектурного підходу в контексті веб-розробки, API визначається як набір специфікацій, таких як передача повідомлення запиту до ресурсу, за допомогою Hypertext Transfer Protocol (HTTP), визначенням структури запитів-відповідей, з використанням технологій Extensible Markup Language (XML) або JavaScript Object Notation (JSON).

Representational State Transfer (REST) – архітектурний стиль для розробки програмного забезпечення, містить набір архітектурних обмежень, не являє собою протокол чи стандарт. Розробники API можуть реалізовувати цей стиль декількома шляхами.

Коли відбувається запит до сервера чи програмного забезпечення за допомогою RESTful API, клієнт передає інформацію про стан ресурсу, та подає інформацію у одному з форматів через HTTP: JSON, XML, JavaScript-запит, звичайний текст, тощо.

Важливо розуміти дещо про використання HTTP у RESTful API: заголовки та параметри відіграють важливу роль у обміні інформацією між ресурсами, оскільки вони містять важливу інформацію ідентифікатора про метадані запиту, авторизацію, тип контенту, єдиний ідентифікатор ресурсу, кешування, куки, код відповідей, інше. Існують заголовки запитів та відповідей, кожні з них містять власну інформацію про з'єднання HTTP та коди станів.

Найбільш популярний варіант надсилання тіла інформації – це JSON, оскільки він не прив'язаний до певної мови програмування, зручний у використанні, легкий у розумінні як людьми, так і машинами. У своїй основі містить пари — ключ та значення, які використовуються у формуванні тіл запитів та відповідей, у вигляді масивів відповідних пар [14].

Для того, щоб API розглядався як RESTful, має слідувати таким критеріям:

- архітектура, що складається з клієнтів, серверів та ресурсів, із запитами, які використовують протокол HTTP;
- всі комунікації між клієнтом та сервером відбуваються без зберігання стану та сесій, кожен запит та відповідь є відокремленими та незалежним;
- дані кешуються, для полегшення навантаження на сервер;
- стандартизація запитів та відповідей між клієнтом та сервером відповідно;
- багатошарова система з розподіленням обов'язків, ієрархічна структура невидима для клієнтів.

6. JSON Web Token

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний та автономний спосіб передачі безпечної інформації між сторонами з використанням стандарту JSON. Інформації яку містить в собі токен можна довіряти, оскільки вона має цифровий підпис. Токен може бути підписаний за допомогою публічного або секретного ключа з використанням шифрувальних алгоритмів [15].

Найбільш поширений спосіб використання JWT – авторизація. Єдиний вхід (Single Sign-in) – це функція, яка широко використовує JWT, що дозволяє один раз ввійти в систему та використовувати токен, допоки не пройде час його існування. Як тільки користувач ввійде в систему, він отримає токен з його особистою інформацією, яку зможе використовувати для доступу до захищених служб та ресурсів системи.

7. Структура JSON Web Token

У компактній формі, токен складається з трьох частин, які шифруються алгоритмом Base64Url та відділені крапкою [15]:

- заголовок – складається з двох частин: тип токену та алгоритм який використовується для кодування його;
- корисне навантаження – інформація для та про користувача, ким виданий токен, кому виданий, тривалість життя токена та інші;
- підпис – вираховується на основі заголовку, корисного навантаження та обраного алгоритму кодування з використанням публічного або приватного ключа.

Переваги використання JWT в порівнянні з підходом використовування сесій такі:

- JWT – не потребує створення сесії та зберігання додаткових даних про сесію. Все що необхідно серверу – це перевірити підпис токена;
- сервер може не займатися створенням токенів, а надати їх іншим ресурсам, наприклад серверу авторизації;
- в токенах можна зберігати додаткову інформацію про користувачів;
- токен надає можливість доступу до різних сервісів, які зможуть прочитати цей токен.

8. Протокол авторизації OAuth2

OAuth2 – відкритий стандарт авторизації, який дозволяє делегувати логіку авторизації перевіреним сервісам, як Google, Amazon, Facebook та іншим, які

виступають в ролі посередника між користувачем та сторонніми сервісам. Дозволяє користувачам отримати доступ до сторонніх сервісів, та ділитися своєю конфіденційною інформацією з ними (рис. 4). Перевага такого способу полягає в тому, що ви не розкриваєте свої облікові дані сторонньому ресурсу [20].

Рис 4. Абстрактний опис протоколу OAuth2

OAuth2 визначає чотири ролі:

- Власник ресурсу – користувач, який авторизує додаток для доступу до свого акаунту;
- Клієнт – додаток, який хоче отримати доступ до акаунту користувача, перед доступом до інформації користувача, додаток має бути авторизований користувачем, а авторизація має буде схвалена сервером авторизації;
- Сервер авторизації – сервер який видає токени клієнту для доступу до інформації користувача;
- Сервер ресурсів – сервер, на якому знаходяться захищені ресурси, до яких клієнт звертається використовуючи токен доступу користувача.

Послідовність роботи протоколу:

- додаток запитує у користувача дозвіл на авторизацію для доступу до серверу ресурсів;
- користувач надає дозвіл додатку на авторизацію;
- додаток запитує авторизаційний токен у сервера авторизації, шляхом надання інформації про себе та дозвіл користувача;
- якщо істинність додатку підтверджена і дозвіл на авторизацію дійсний, то сервер створює токен доступу. Процес авторизації завершений;
- додаток запитує захищений ресурс у серверу ресурсів, надаючи при цьому токен доступу;
- якщо токен дійсний, сервер ресурсів надає ресурс додатку.

Перед тим, як почати використовувати OAuth2, необхідно зареєструвати свій додаток на сервісі. В роботі в якості авторизаційного сервісу використовується Google API. Тому далі буде приклад як зареєструвати додаток в Google. Це робиться в розділі «developer» сайту Google API, де необхідно надати наступну інформацію:

- назва додатку;
- сайт додатку (глобальний або локальний шлях);
- callback URL – це шлях, на який сервіс буде перенаправляти користувача після авторизації або відмови.

Після реєстрації додатку, сервіс створить облікові дані клієнта — ідентифікатор клієнту та секрет клієнта. Ідентифікатор клієнта являє собою публічну доступну строку, яка використовується API сервісом для автентифікації додатку, а також для створення авторизаційних URL для користувачів. Секрет клієнта використовується для автентифікації істинності додатку для API сервісу, коли додаток запитує доступ до акаунту користувача. Секрет повинний бути відомим тільки додатку і API сервісу.

9. Нормалізація та токенізація програмного коду

Щоб зрозуміти як правильно працювати з програмним кодом, потрібно зрозуміти, що він представляє з себе [16].

Програма – це набір команд, інструкцій, даних, призначених для виконання функцій необхідних користувачу, використовуючи для цього апаратне забезпечення операційної системи, для якої вона розроблена. Програми складаються з синтаксичних одиниць, які відповідають правилам конкретної мови програмування: методів, класів та бібліотек, які, у свою чергу, складаються з операторів або інструкцій для розв’язання конкретної задачі (стандарт ISO/IEC 2382-1:1993) [12].

Розглянувши, що таке програма, зрозуміло, що програма – це не просто текст, який можна було б порівняти послівно, вона своєрідну структуру, синтаксис, який залежить від мови програмування, але в більшості, якщо це не вузькоспеціалізована мова під конкретну ситуацію, в них можна легко простежити закономірності, наприклад, створення циклу чи оператора розгалуження. Тобто програма містить спеціальний набір не змінних шаблонів, які відрізняються за синтаксисом, але не за суттю, тому можливо привести до одного стандартизованого виду, що буде зручний для аналізу та подальшого порівняння в алгоритмі. Тому перед цим потрібно провести нормалізацію та токенізацію.

Нормалізація – процес перетворення вихідного коду, таким чином, щоб опустити усі несуттєві частини програмного коду для алгоритму, який буде перевіряти код на плагіат [16]. Включає:

- переведення усіх символів в нижній регістр;
- заміну усіх символів табуляції на пробіли, видалення символів переводу рядка;
- заміну роздільних знаків на пробіли;
- видалення однорядкових та багаторядкових коментарів;
- заміну багатьох пробільних символів одним.

Нормалізація дозволяє уникнути хибного спрацювання або навпаки не спрацювання алгоритму при підрахуванні алгоритму. Наприклад, хибне спрацювання може виникнути, якщо додати коментарі, які збільшать розмір коду, або змінити назву змінної, тому для цього приводимо програмний код до одного уніфікованого виду. Після проведення нормалізації вихідний код готовий для подальшої процедури, перетворення його на список токенів.

Лексема (токен) – послідовність машинних символів вихідного коду програми,

яке має спільне сукупне значення. В мові програмування, частіше використовують слово «токен», тому для зручності, в подальшому будемо використовувати його [26].

Будь-яка мова програмування складається з наступних лексем:

- зарезервовані слова (модифікатори доступу, типи змінних та інше);
- ідентифікатори (назва змінних, методів, класів);
- числові, буквенні, рядкові константи;
- знаки операторів;
- коментарі;
- роздільники.

Процес токенізації буде проходити з використанням спеціальної бібліотеки для парсингу програмного коду – ANTLR4 [2]. Потужний інструмент, використовується для аналізу текстів, створення власних мов програмування, та компіляторів. В випадку системи для оцінки програмного коду на плагіат, дозволяє на 100% правильно перетворити вихідний код в список токенів. Для роботи цієї бібліотеки необхідно мати два файли. В одному файлі містяться всі можливі лексеми для конкретної мови програмування, в іншому містяться типове використання лексем в конструкціях. Алгоритм роботи бібліотеки ANTLR:

- генерація класу, який буде займатися розбором коду на токени;
- генерація синтаксичного аналізатора;
- перевірка вихідного коду, на наявність помилок, з подальшим повідомленням користувачу;
- створює синтаксичне дерево, генерує два об'єкта Visitor та Listener, для обходу дерева.

У нашому випадку нам потрібно отримати з дерева послідовність токенів, для подальшого використання їх в алгоритмі визначення плагіату.

10. Алгоритм Вагнера-Фішера

Головним алгоритмом, який буде виконувати перевірку програмного коду на плагіат – алгоритм Вагнера-Фішера. В основу якого покладене таке поняття як відстань Левенштейна.

Відстань Левенштейна – числове значення, яке дозволяє вирахувати різницю між двома послідовностями символів. Визначається як мінімальна кількість операцій (вставки, видалення, заміни), необхідних для перетворення одної послідовності у іншу [25].

Нехай два рядки S_1 і S_2 – довжиною m і n – рядки відповідно над деяким алфавітом (у нашому випадку множина лексем мови програмування Java), тоді відстань Левенштейна $lev(S_1, S_2)$ можна підрахувати за наступною формулою:

$$lev(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ min\begin{cases} lev(i, j-1)+1 \\ lev(i-1, j)+1 \\ lev(i-1, j-1)+m(S_1[i], S_2[i]) \end{cases} & j > 0, i > 0 \end{cases},$$

де $m(a, b)$ дорівнює нулю, якщо $a = b$ і одиниці в інакшому випадку; $min\{a, b, c\}$

повертає найменше з трьох аргументів.

Тут крок по *i* означає видалення (D) із першого рядка, по *j* – вставку (I) у перший рядок, а крок по обох індексах означає заміну символу (R) або відсутність змін (M). Відстань Левенштейна знаходиться у останній комірці матриці *lev (lev(m, n))*.

Приклад розрахунку відстані Левенштейна подано у таблиці 1.

Алгоритм, запропонований Вагнером та Фішером, містить в собі покращення внесені в оригінальний алгоритми і дозволяє вирахувати найкоротшу відстань Левенштейна між двома рядками, з меншими витратами пам'яті за допомогою динамічного обчислення масивів, скоротивши необхідну пам'ять до *min{m, n}* [9]. Тобто, не потрібно тримати в пам'яті матрицю розміром *m·n*, обчислення достатньо мати два вектори довжиною *m* або *n*.

Таблиця 1

Результат роботи алгоритму Левенштейна

		М	**О**	**Р**	**Г**	**Е**	**Н**	**Ш**	**Т**	**Е**	**Р**	**Н**
	0	1	2	3	4	5	6	7	8	9	10	11
Л	1	1	2	3	4	5	6	7	8	9	10	11
Е	2	2	2	3	4	4	5	6	7	8	9	10
В	3	3	3	3	4	5	5	6	7	8	9	10
Е	4	4	4	4	4	4	5	6	7	7	8	9
Н	5	5	5	5	5	5	4	5	6	7	8	8
Ш	6	6	6	6	6	6	5	4	5	6	7	8
Т	7	7	7	7	7	7	6	5	4	5	6	7
Е	8	8	8	8	8	7	7	6	5	4	5	6
Й	9	9	9	9	9	8	8	7	6	5	5	6
Н	10	10	10	10	10	9	8	8	7	6	6	**5**

Псевдокод алгоритму Вагнера-Фішера:

```
function WagnerFischer(char s[1..m],char t[1..n]):
  declare int v0[n+1]
  declare int v1[n+1]
  for i from 0 to n:
    v0[i] := i;
  for i from 0 to m-1:
    v1[0] := i+1;
  for j from 0 to n-1:
    deletionCost := v0[j+1] + 1
    insertionCost := v1[j] + 1
```

```
      if s[i] = t[j]:
      substitutionCost := v0[j]
    else:
      substitutionCost := v0[j] + 1
    v1[j+1] := minimum(deletionCost,
                       insertionCost,
                       substitutionCost)
  swap v0 with v1
return v0[n]
```

З метою отримання результату роботи алгоритму у прийнятному і зрозумілому виді, перетворимо число Левенштейна у відсотки за допомогою наступної формули:

$$PlagiarizedValue = \left\{1 - \frac{LD}{Max(FFLS, SFLS)}\right\} \cdot 100,$$

де

LD (Levenshtein distance) – відстань Левенштейна між двома файлами;
FFLS (First token list size) – довжина першого списку токенів;
SFLS (Second token list size) – довжина другого списку токенів.

11. Структура пакетів сервера

Сервер складається з чотирьох головних пакетів:

- auth – містить в собі логіку реєстрації та авторизації користувачів;
- detector – містить в собі логіку, для перевірки програмного коду на плагіат, нормалізацію, токенізацію та алгоритм Вагнера-Фішера;
- uploadfile – відповідає за можливість завантаження файлів з програмним кодом на сервер;
- plagiarism – відповідає за виведення інформації про плагіат для користувача;
- resources – містить службові файли для роботи серверу.

Пакет auth

Так, як всі шляхи на сервері захищені, і потребують автентифікації, було прийняте рішення додати на сервер логіку реєстрації та авторизації.

В даному пакеті знаходиться логіка, яка відповідає за авторизацію на сервері та реєстрацію користувача на сервері. Реєстрація відбувається двома шляхами: Basic Auth та OAuth2.

Перша, відбувається за специфікацією Basic Auth, тобто, користувач створює власний обліковий запис з використанням логіну та паролю. Цей процес відбувається між користувачем та сервером безпосередньо, вся інформація про користувача зберігається на сервері.

Друга, відбувається за специфікацією OAuth2, тобто між користувачем та сервером, знаходиться третя сторона, яка бере на себе відповідальність за реєстрацію. Користувач ініціює авторизацію, надаючи серверу згоду про авторизацію, в даному випадку компанії Google. За згодою користувача сервер бере необхідні облікові дані про особу в провайдера Google, та оброблює їх. Перевага даного методу реєстрації та авторизації в тому, що на сервері не зберігається пароль користувача, і весь процес проходить між серверами. Також за для безпеки, токен доступу (Access token) на сервері перетворюється на власний токен за стандартом JWT. Навіть якщо зловмисник поцупить цей токен і розкодує, максимум з корисної інформації, він отримає ім’я користувача та його електронний адрес. З даним токеном зловмисник має можливість доступу тільки до сервера перевірки вихідного коду на плагіат і більше нікуди,

відредагувати акаунт він немає можливості, такий функціонал не передбачений на сервері.

Після авторизації, обмін інформацією між клієнтом та сервером відбувається за допомогою закодованого токена стандарту JWT. З нього сервер дістає всю необхідну інформацію, перевіряє права доступу та час життя токену. Якщо все пройшло успішно, користувач має право користуватися захищеними ресурсами серверу. Якщо сервер не зможе прочитати токен, чи час його життя закінчився, користувачу потрібно буде знову авторизуватися в системі.

Пакет detector

В ньому знаходиться вся логіка, яка відповідає за нормалізацію та токенізацію вихідних файлів, а також за їх перевірку на плагіат.

Процес знаходження плагіату відбувається наступним чином:

- відбувається нормалізація та токенізація вхідного файлу;
- поточний вхідний файл у вигляді списку токенів, порівнюється за допомогою алгоритму з списками токенів в базі даних;
- отриманий результат передається користувачу.

Нормалізація та токенізація проходить з використанням бібліотеки ANTLR. На основі двох файлів JavaLexer.g4 та JavaParser.g4, де описані основні граматичні лексеми та правила створення синтаксичних конструкцій мови програмування Java, створюється Lexer та Parser. Лексер в свою чергу за допомогою парсера розбиває файл на токени, які в подальшому використовує алгоритм Вагнера-Фішера. Перевіряючи всі файли з поточним, виявляє найбільший відсоток плагіату та повертає результат. Результат має наступний вигляд:

- plagiarism – відсоток запозичень в коді від 0 до 100;
- lastCheckedTime – дата коли відбувалась перевірка файлу.

Пакет uploadfile

Даний модуль відповідає за завантаження файлу на сервер. Коли користувач завантажив файл на сервер, на сервері копія файлу відправляється в модуль detector, де перетворюється в список токенів та повертається назад в модуль uploadfile, зі свого боку модуль зберігає файл та його токенізований вигляд в базу даних.

Також володіє можливістю віддавати файли на завантаження користувачу. Надає можливість адміністратору переглядати файли, які користувач завантажив.

Пакет plagiarism

Пакет plagiarism відповідає за співпрацю API серверу з модулем детектора. Містить контролер в який користувач надсилає інформацію про файл, який хоче перевірити на плагіат.

Всі результати перевірок зберігаються в базі даних та прикріпляються до відповідного файлу, який користувач хоче перевірити. Дана можливість надає користувачу переглянути пізніше результати роботи, не очікуючи певний час, який може знадобитися на перевірку. Користувач має змогу перевіряти тільки ті файли, які він надав серверу. Файли інших користувачів та їх результати перевірки доступні для перегляду належать тим користувачам, які їх передали до системи, а також адміністратору.

Пакет resources

Містить в собі службові файли для роботи серверу. Наприклад, sql-файли, так звані файли міграції, необхідні при першому запуску сервера в новому місці. В даних

файлах описана структура бази даних, структура таблиць та всі зв'язки між ними.

Також тут знаходяться конфігураційні файли, які містять важливі властивості для роботи сервера, наприклад, шлях до бази даних, список дозволених адрес, які можуть звертатися до сервера, налаштування OAuth2.

Висновки

У статті розглянуто відомі системи для виявлення плагіату в програмному коді, а саме: MOSS, Codeqiury, Unicheck, CCFinderX. Представлено опис переваг та недоліків кожної системи. Кожна з цих систем має унікальну архітектуру та інтерфейс, але всі мають спільний алгоритм роботи:

- приймають вхідний файл на оцінку;
- перетворюють його в список токенів;
- використовують власний алгоритм для оцінки коду на плагіат;
- надають результат користувачу;

Створено власний сервер для оцінки коду на плагіат з урахуванням переваг та недоліків.

У ході роботи було розглянуло та з'ясовано наступне:

- розглянуто, що таке плагіат у програмному коді;
- розглянуто, відомі системи пошуку плагіату та їх реалізації;
- розроблено алгоритм на основі токенізованого представлення;
- реалізовано сервер для визначення плагіату у коді, розроблених мовою програмування Java, яка може виступати в ролі API для додатків.

Список використаної літератури:

1. Aiken A., Schleimer S., Wikerson D. Winnowing: Local Algorithm for Document Fingerprinting. // Proceeding of ACMSIGMOD Int. Conference on Management of Data. San Diego. 2003. P. 76-85. ACM Press. New York, USA. 2003.
2. ANTLR 4, ANother Tool for Language Recognition [Електронний ресурс] – Режим доступу: https://www.antlr.org/
3. Baxter I., Yahin A., Moura L., Anna M.S., Bier L. Clone Detection Using Abstract Syntax Trees. // Proceedings of ICSM. IEEE. 1998.
4. CCFinderX. [Електронний ресурс] – https://github.com/gpoo/ccfinderx
5. Codequiry. [Електронний ресурс] – Режим доступу: https://codequiry.com
6. Faidhi J.A.W., Robinson S.K. An Empirical Approach for Detecting Program Similarity within a University Programming Environment. //Computer and Education. 1987. 11(1). P. 11-19.
7. Heckel P. A. Techique for Isolationg Differences Between File. // Communications of the ACM 21(4). April 1978. P. 264-268.
8. Heinzte N. Scalable Document Fingerprinting. // In 1996 USENIX Workshop of Electronic Commerce, 1996.
9. Huang X., Hardison R.C., Miller W. A Space-efficient Algorithm for Local Similarities. // Computer Applications in the Biosciences 6. 1990. P 373-381.
10. Gradle. [Електронний ресурс] – Режим доступу: https://gradle.org/
11. IntelliJ IDEA Ultimate Edition. [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/idea/
12. ISO/IEC 2382-1:1993 Information Technology – Vocabulary – Part1: Fundamental terms. [Електронний ресурс] – Режим доступу: https://www.iso.org/ru/standard/7229.html
13. Java 11 JDK. Oracle. [Електронний ресурс] – Режим доступу: https://www.oracle.com/java/technologies/downloads/
14. JSON. [Електронний ресурс] – Режим доступу: https://www.json.org/json-en.html
15. JWT. [Електронний ресурс] – Режим доступу: https://jwt.io/
16. Lexical Analysis. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Lexical_analysis
17. Mishne G., M. de Rijke. Source Code Retrieval using Conceptual Similarity // Proceedings RIAO. Vaucluse. 2004. P. 539-555.
18. MOSS A System for Detecting Software Similarity. [Електронний ресурс] – Режим доступу: https://theory.stanford.edu/~aiken /moss/

19. Prechelt L., Malpohl G., Philippsen M. JPlag: Finding Plagiarism Among a Set of Programs. // Technical Report No. 1/00, University of Karlsruhe, Department of Informatics. March 2000.
20. OAuth2. RFC 6749 [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749
21. RESTful API Tutorial. [Електронний ресурс] – Режим доступу: https://restfulapi.net/
22. Spring Boot 2 Framework. [Електронний ресурс] – Режим доступу: https://spring.io/
23. Unicheck. [Електронний ресурс] – Режим доступу: https://unicheck.com/ua/blog/innovative-tool-for-checking-source-code-for-plagiarism
24. Wise M.J. String similarity via greedy string tiling and running Karb-Rabin matching. // Dept. of CS, University of Sidney. December 1993.
25. Відстань Левенштейна. [Електронний ресурс] – Режим доступу: http://uk.wikipedia.org/wiki/Відстань Левенштейна
26. Лексема. [Електронний ресурс] – 2021. – Режим доступу: http://uk.wikipedia.org/wiki/Лексема
27. Плагіат. [Електронний ресурс] – 2021. – Режим доступу: http://uk.wikipedia.org/wiki/Плагіат

References:

1. Aiken A., Schleimer S., Wikerson D. Winnowing: Local Algorithm for Document Fingerprinting. // Proceeding of ACMSIGMOD Int. Conference on Management of Data. San Diego. 2003. P. 76-85. ACM Press. New York, USA. 2003.
2. ANTLR 4, ANother Tool for Language Recognition [Електронний ресурс] – Режим доступу: https://www.antlr.org/
3. Baxter I., Yahin A., Moura L., Anna M.S., Bier L. Clone Detection Using Abstract Syntax Trees. // Proceedings of ICSM. IEEE. 1998.
4. CCFinderX. [Електронний ресурс] – https://github.com/gpoo/ccfinderx
5. Codequiry. [Електронний ресурс] – Режим доступу: https://codequiry.com
6. Faidhi J.A.W., Robinson S.K. An Empirical Approach for Detecting Program Similarity within a University Programming Environment. //Computer and Education. 1987. 11(1). P. 11-19.
7. Heckel P. A. Techique for Isolationg Differences Between File. // Communications of the ACM 21(4). April 1978. P. 264-268.
8. Heinzte N. Scalable Document Fingerprinting. // In 1996 USENIX Workshop of Electronic Commerce, 1996.
9. Huang X., Hardison R.C., Miller W. A Space-efficient Algorithm for Local Similarities. // Computer Applications in the Biosciences 6. 1990. P 373-381.
10. Gradle. [Електронний ресурс] – Режим доступу: https://gradle.org/
11. IntelliJ IDEA Ultimate Edition. [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ru-ru/idea/
12. ISO/IEC 2382-1:1993 Information Technology – Vocabulary – Part1: Fundamental terms. [Електронний ресурс] – Режим доступу: https://www.iso.org/ru/standard/7229.html
13. Java 11 JDK. Oracle. [Електронний ресурс] – Режим доступу: https://www.oracle.com/java/technologies/downloads/
14. JSON. [Електронний ресурс] – Режим доступу: https://www.json.org/json-en.html
15. JWT. [Електронний ресурс] – Режим доступу: https://jwt.io/
16. Lexical Analysis. [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Lexical_analysis
17. Mishne G., M. de Rijke. Source Code Retrieval using Conceptual Similarity // Proceedings RIAO. Vaucluse. 2004. P. 539-555.
18. MOSS A System for Detecting Software Similarity. [Електронний ресурс] – Режим доступу: https://theory.stanford.edu/~aiken /moss/
19. Prechelt L., Malpohl G., Philippsen M. JPlag: Finding Plagiarism Among a Set of Programs. // Technical Report No. 1/00, University of Karlsruhe, Department of Informatics. March 2000.
20. OAuth2. RFC 6749 [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc6749
21. RESTful API Tutorial. [Електронний ресурс] – Режим доступу: https://restfulapi.net/
22. Spring Boot 2 Framework. [Електронний ресурс] – Режим доступу: https://spring.io/
23. Unicheck. [Електронний ресурс] – Режим доступу: https://unicheck.com/ua/blog/innovative-tool-for-checking-source-code-for-plagiarism
24. Wise M.J. String similarity via greedy string tiling and running Karb-Rabin matching. // Dept. of CS, University of Sidney. December 1993.
25. Levenstein's distance: http://uk.wikipedia.org/wiki/Відстань_Левенштейна. [in Ukrainian]
26. A token: http://uk.wikipedia.org/wiki/Лексема. [in Ukrainian]
27. Plagiarism: http://uk.wikipedia.org/wiki/Плагіат. [in Ukrainian]

KHARCHENKO Vitalii,
Student, Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
DIDKOWSKY Ruslan,
Doctor of Technical Sciences, Associate Professor, Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine
SERDIUK Oleksandr,
Candidate of Economic Sciences, Associate Professor, Department of Informatics and Applied Mathematics, The Bohdan Khmelnytsky National University of Cherkasy, Ukraine

DEVELOPMENT THE SERVER-SIDE PART OF THE SYSTEM OF CHECKING SOFTWARE CODE FOR THE PRESENCE OF PLAGIARISM

Summary. Introduction. *Borrowing code among students is a problem that worries many university professors who teach disciplines related to the study of programming. Unfortunately, it is difficult for a teacher to personally follow the uniqueness of all students' work. In addition, a large part of the working time has to be spent on checking the work, instead of devoting this time, for example, to preparing even better tasks for students and getting acquainted with the latest trends in programming. To facilitate the detection of borrowings in the program code among students, plagiarism checking systems can be used, but most of them are designed to search for plagiarism in plain text, and, accordingly, are not adapted to the analysis of program code, which has its own characteristics.*

The purpose *of the article is to analyze some modern available plagiarism search systems, to develop requirements for one's own system and to describe one's own developed plagiarism evaluation system.*

Results. *The paper considers 4 programs designed to detect plagiarism: Measure Of Software Similarity (MOSS), Codequiry, Unicheck, CCFinderX. For each of the programs the identified advantages and disadvantages in comparison with other studied programs are given. The comparative analysis revealed the following characteristics of the studied programs: most of these systems have a server; most of these have their own public or private databases with samples to compare with; all systems have their own advanced algorithms based on known ones, some even have artificial intelligence; all of these systems have a graphical interface. The following shortcomings were also identified: some instances have a desktop client, i.e., require network downloads and additional settings, such as MOSS and CCFinderX; modern systems, such as Unicheck and Codequiry, provide for paid use; do not have a centralized system for displaying results, i.e. the teacher can not follow those who passed the task.*

Based on the comparative evaluation of the analyzed systems, a proprietary plagiarism evaluation system was created in the program code in the form of a server, which has the following properties: the ability to register and authorize the user; check files for plagiarism and get the result. There are two types of roles for users on the server: User and Admin. To achieve the result, the following technologies were used: the main programming language - Java 11; development environment - IntelliJ IDEA; Gradle package collector; Spring Boot 2 Framework container; technologies REST API, JSON, OAuth2, JWT. The main algorithm that checks the program code for plagiarism is the Wagner-Fisher algorithm, which is based on such a concept as Levenstein's distance. The article describes Levenstein's algorithm and gives an example of its use to calculate the distance between two lines.

Conclusion. *As a result of the work several known systems of plagiarism detection in the program code are considered, the description of advantages and disadvantages of each system is presented. It is determined that all considered systems have a common algorithm: accept the input file for evaluation; turn it into a list of tokens; use their own algorithm to evaluate the code for plagiarism; provide the result to the user. An algorithm based on a token representation is developed. Implemented its own server for evaluating plagiarism code, taking into account the advantages and disadvantages of the analyzed systems, the structure of which is given in the article. The resulting software product can act as a server that provides an API for applications designed to check for plagiarism in the software code.*

Keywords: *plagiarism, plagiarism in program code, plagiarism check system, server system.*

Одержано редакцією 20.10.2021 р.
Прийнято до публікації 08.12.2021 р.

УДК 378:517

DOI 10.31651/2076-5886-2021-1-85-91

PACS 02.60.-x

ДЗЮБА Вікторія Анатоліївна,
кандидат технічних наук, викладач кафедри прикладної математики та інформатики Черкаського національного університету імені Богдана Хмельницького
e-mail: viktoriya.dzyuba15@gmail.com
ORCID 0000-0003-1655-0333

ВИКОРИСТАННЯ АПАРАТУ АЛГЕБРИ ТА ГЕОМЕТРІЇ У ПРОЦЕСІ РОЗВ'ЯЗАННЯ ПРИКЛАДНИХ ЗАДАЧ ПІД ЧАС НАВЧАННЯ ФАХІВЦІВ У СФЕРІ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Вдале поєднання математичного апарату, сучасної інженерії та високих технологій дає змогу реалізувати сміливі ідеї та втілити надсучасні рішення у життя кожного з нас.

У статті показано, що оволодіння апаратом алгебри для фахівців з аналізу даних є вкрай важливим аспектом для успішного вирішення задач прикладного характеру, крім цього, це дозволить у майбутньому перспективно реалізувати себе в кар'єрному плані. В якості підтвердження актуальності обраного напряму дослідження, вказані провідні науковці та їх роботи по тематиці даної статті.

Окреслено основні сучасні напрямки використання математичних дисциплін для розвитку інформаційних систем та технологій. Розроблено алгоритм виконання елементарної задачі теорії матричних ігор та її програмну реалізацію у середовищах Python та Javascript.

Розглянуто розв'язання задачі прикладного характеру, із використанням алгебраїчних підходів, стосовно побудови моделі мережі транспортних потоків та проаналізовано отримані результати.

Ключові слова: *алгебра та геометрія, математична модель, аналіз даних, інформаційні системи, новітні технології.*

Вступ

Стрімкий розвиток інформаційних систем та новітніх технологій вносить динаміку позитивних змін у кожну сферу життєдіяльності людини. Прогресивні інженерні рішення формують нові підходи до автоматизації процесів, наприклад дозволяють комфортно подорожувати по новій місцевості із застосуванням карт, онлайн-перекладачів, дронів тощо. Для успішного функціонування вказаних новітніх технологій передбачається використання сучасного апарату математичних дисциплін [1].

Опанування дисципліни «Алгебра та геометрія» є фундаментальним етапом на шляху до вивчення предметів циклу професійного спрямування спеціальності 126 «Інформаційні системи та технології». Це можна пояснити тим, що більшість задач програмування зводиться до розв'язання систем лінійних рівнянь, побудови матриць та дій над ними, виконання базових операцій над векторами тощо [2].

Крім цього, для більшості ІТ-компаній, рівень математичних знань є індикатором до того, настільки майбутній працівник буде компетентний у своїй роботі.

Разом з цим, безпосереднє вивчення теорії та розв'язання математичних задач, під час навчання, призводить до того, що студент практично втрачає зв'язок між витраченими роками на оволодіння математичних дисциплін та їх значимістю у майбутній кар'єрі.