поточний рівень і на наступній ітерації ми будемо використовувати ці дані для виключення з плану дисциплін, що дублюються.

Основною метою даної статті є демонстрацію нового підходу до побудови індивідуального навчального плану студента на основі його компетентностей та побажань. Так як, методи побудови навчальних планів, що використовуються зараз є дещо застарілими і не відповідають теперішнім вимогам.

Список використаних джерел:

1. Кривицька М.А., Бушмелева К.И. Підходи до проектування задачі складання робочого навчального плану спеціальності вищої професійної освіти. [Текст]

2. Тім Лорман, Джоан Депплер, Девід Харві. Інклюзивний підхід до побудови навчального процесу. [Текст]

3. Кристофидес Н. Теория графов. Алгоритмический подход. Москва, 1978.

Скороходов К.Ф.

студент,

Харківський національний університет радіоелектроніки

ЗАСТОСУВАННЯ МЕТОДУ ВІДБИТКІВ ДЛЯ ШВИДКОГО ПОШУКУ ПЛАГІАТУ В КОДІ ПРОГРАМИ

Цифрові документи, в тому числі і програмний код, дуже легко скопіювати. На жаль, деякі люди користуються цим для того, щоб видавати чужі роботи за свої. Проблема плагіату є дуже важливою, наприклад, в галузі освіти, коли студенти, замість того, щоб виконувати завдання самостійно, копіюють роботи інших студентів, можливо, лише вносячи при цьому незначні зміни. Вручну перевіряти на плагіат велику кількість робіт дуже складно, тому, в такому випадку, викладачеві було б зручно мати інструмент, який би дозволив автоматично перевірити роботи та знайти серед них підозріло схожі один до одного. Таке програмне забезпечення може приймати на вхід вихідні коди, аналізувати їх, і видавати ранжируваний список пар схожих файлів, а також якимось чином виділяти схожі рядки, щоб користувач мав можливість оцінити, чи дійсно плагіат має місце в цьому випадку.

На сьогодні було розроблено багато різноманітних підходів для пошуку плагіату в вихідному коді програм. Одним із підходів є підрахунок певних атрибутів програмного коду, наприклад кількості унікальних операторів та операндів мови програмування, якою написана програма, загальної кількості вживань певних операторів та операндів та ін. Іншим підходом є складання дерев розбору програм та порівняння їх за допомогою складних алгоритмів [1].

Одними з найбільш ефективних алгоритмів для швидкого пошуку однакових частин програмного коду у багатьох документах є алгоритми на основі відбитків. Ми розглянемо так званий алгоритм провіювання (англ. winnowing), який відноситься до цієї категорії [2].

Першим етапом алгоритму є токенізація. В ході цього процесу документ (файл програмного коду) зчитується, та кожному ключовому слову та операторі мови приписується код, призначений заздалегідь для кожного класу операторів. Рядкові та чисельні літерали також замінюються на коди. Будується рядок з отриманих кодів, зберігаючи порядок проходження їх у вихідному коді програми. Один символ рядка (токен) – код одного оператора. Наприклад, рядок коду «int x = a + 5» перетворюється на рядок токенів «int identifier assign identifier add int_literal». Таким чином ми автоматично ігноруємо назви функцій і змінних (класів, об'єктів тощо), а також роздільників, не зважаючи увагу на дрібні зміни коду програми. Потрібно відзначити, що процес токенізації і розбиття операторів на класи залежить від використовуваного в вихідному коді мови програмування. До одного класу операторів зазвичай відносять ті, яким відповідає один ідентифікатор мови програмування, всі виклики функцій, виклики методів класів, оголошення змінних елементарних типів та примірників класів.

На наступному кроці отриманий рядок токенів програми поділяється на k-грами. K-грам – це послідовність токенів довжиною k. Кожен наступний k-грам отримується з попереднього додаванням наступного токену спереду та відніманням останнього токену ззаду. Для наведеного вище прикладу та k = 3 ми отримуємо наступні k-грами: «int identifier assign», «identifier assign identifier», «assign identifier add», «identifier add int_literal».

Для кожного отриманого на попередньому кроці k-грама розраховується контрольна сума (хеш-сума, або хеш). Оскільки кожен k-грам отримується з попереднього лише дованням та відніманням токеів, немає необхідності розраховувати хеш повністю для кожного k-грама, а можна використати кільцевий хеш, наприклад хеш Рабина – Карпа, що значно прискорює роботу алгоритму [3]. Нехай, для k-грама $c_1 \ldots c_k$ хеш-сума $H(c_1 \ldots c_k)$ буде дорівнювати:

$$c_1 * b^{k-1} + c_2 * b^{k-2} + \cdots + c_{k-1} * b + c_k, \quad (1)$$

де b – якесь просте число.

Для того, щоб розрахувати хеш k-грама $c_2 \ldots c_{k+1}$, необхідно лише відняти перший доданок, помножити результат на b, та додати наступний доданок:

$$H(c_2 \ldots c_{k+1}) = (H(c_1 \ldots c_k) - c_1 * b^{k-1}) * b + c_{k+1}. \quad (2)$$

Оскільки b^{k-1} – це константа, то хеш кожного наступного k-грама можна розрахувати лише двома операціями додавання та двома операціями множення.

Далі вибирається певна підмножина отриманих хешів, які складають відбиток документу. Відбиток містить лише частину всіх хешів для того, щоб він не мав занадто великий розмір. Також відбиток містить інформацію про розташування кожного токену у файлі вихідного коду. Підмножина хеш-сум складається наступним чином. Нехай, вікно хеш-сум має розмір w. Розглянемо послідовність хешів $h_1 h_2 \ldots h_n$, яка представляє собою документ. Для кожної позиції $1 \le i \le n - w + 1$ визначається вікно хеш-сум $h_i \ldots h_{i+w-1}$. В кожному вікні вибирається найменше значення хеш-суми. Якщо таких значень декілька, вибирається саме праве значення. Кожне значення вибирається лише один раз. Тоді сукупність усіх вибраних хешів буде скадати відбиток документу.

Пошук схожих документів виконується наступним чином. На першому кроці будується індекс, який кожному хешу ставить у відповідність список

документів, у яких він зустрічається. Далі кожен хеш кожного відбитку документу шукається у інших документах за допомогою індексу. Для кожного документу d ми знаходимо список документів d_1, d_2, …, відбитки яких містять ті ж самі хеш-суми, що і d. Далі будується список пар документів (d, d_1), (d, d_2), …, який сортується за числом збігів хеш-сум. Пари з найбільшим числом збігів видаються користувачеві.

Список використаних джерел:

1. M. Mozgovoy. Desktop tools for offline plagiarism detection in computer programs. Informatics in Education, 2006.

2. Saul Schleimer, Daniel S. Wilkerson, Alex Aiken. Winnowing: local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 09-12, 2003.

3. Richard M. Karp, Michael O. Rabin. Pattern-matching algorithms. IBM Journal of Research and Development, 1987.

Скороходов К.Ф.

студент,

Харківський національний університет радіоелектроніки

ЗАСТОСУВАННЯ ЛАТЕНТНО-СЕМАНТИЧНОГО АНАЛІЗУ ДЛЯ ПОШУКУ ПЛАГІАТУ В КОДІ ПРОГРАМИ

Проблема плагіату, зокрема програмного коду, є дуже актуальною на сьогодні. Ручна перевірка великої кількості робіт на плагіат є дуже складною задачею. Існує багато програмних систем для автоматичної перевірки робіт, які використовують різноманітні алгоритми та підходи. Одним із підходів до пошуку плагіату в вихідному коді програми є застосування латентно-семантичного аналізу [1].

Латентно-семантичний аналіз є одним з методів інформаційного пошуку (англ. information retrieval). Інформаційний пошук – це наука про пошук неструктурованої інформації в колекції інформаційних ресурсів. За останні роки галузь інформаційного пошуку значно розширилась та включає в себе такі задачі, як класифікація та категоризація документів, візуалізація даних, проектування архітектур пошукових систем та інтерфейсів користувача, фільтрація інформації та міжмовний пошук інформації (англ. cross-language information retrieval) [2]. Одним із способів представлення колекції вхідних документів (корпусу) в інформаційному пошуку є так звана векторна модель (англ. vector space model). У векторній моделі корпус документів представлений як m × n матриця термів-на-документи A, де m (кількість рядків матриці) – це число унікальних слів в корпусі (термів), n (кількість стовпців матриці) – це число документів, а кожен елемент матриці a_{ij} – це частота, з якою терм і