УДК 007.51: 519.712.3

МОЛЧАНОВСЬКИЙ О.І.,
СТЕПАНОВ І.О.,
КОВТУН Д.І.

ПОРІВНЯННЯ АЛГОРИТМУ ЖАДІБНОГО РЯДКОВОГО ЗАМОЩЕННЯ І МЕТОДУ ВІДБИТКІВ ПРИ ВИЯВЛЕННІ ПЛАГІАТУ У ВИХІДНИХ КОДАХ ПРОГРАМ

У роботі наводиться порівняльний аналіз алгоритмів жадібного рядкового замощення і методу відбитків для попарного порівняння вихідних кодів програм для пошуку повних або часткових співпадінь у них. Наводяться експериментальні дані, що дозволяють порівняти ефективність і часові характеристики цих алгоритмів для різних вхідних даних і параметрів. На основі проведених експериментів обрано алгоритм для подальшої розробки системи пошуку плагіату у вихідних кодах програм.

In this research comparative analysis of the algorithms for pair comparison of the source codes is given. The received experimental data that gives capability to compare the efficiency and time characteristics of these algorithms are given for different input data and parameters. The optimal algorithm for future researches is selected based on taken experiments.

Вступ

В наш час інтенсивного розвитку інформаційних технологій інтелектуальна власність стає все більш цінною, тому виникла необхідність її захисту. Це означає, що потрібні потужні засоби для боротьби з плагіатом і, насамперед, виявлення його.

Задача пошуку плагіату, як і поняття плагіату, досить широка і нечітка, тому спробуємо для початку привести визначення цього поняття.

Плагіат – це вид порушення авторських прав, який являє собою протизаконне використання під своїм ім'ям чужого витвору (наукового, літературного, музичного), чи винаходу, раціоналізаторської ідеї (повністю або частково) не вказуючи джерело запозичення. Вимушення до співавторства теж є плагіатом.

Задачею системи пошуку плагіату є автоматичне виявлення (за заданими критеріями) того, чи була використана у програмі деяка чужа ідея. На практиці певним чином задаються деяка функція близькості і поріг, за якими можна визначити наскільки ймовірно, що деяка частина програмного коду була запозичена [1].

Для вирішення цієї задачі найчастіше використовують структурні методи аналізу, що передбачає порівняння програмних кодів з урахуванням їх структури. Такі методи досліджують структуру програми не ізольовано, а як би в контексті, встановлюють взаємозв'язок різних характеристик, їх спільну поведінку.

Недоліком структурних методів є їх складність та важкість обчислень. Класичним прикладом структурного підходу є побудова дерева програми з наступним порівнянням дерев для різних програм. Складність такого методу кубічна, що не дає можливості скільки-небудь ефективно його застосовувати для великої кількості досить довгих програм. Крім того, структурні методи зазвичай спираються на синтаксис конкретної мови програмування. Адаптація методу для іншої мови потребує значних зусиль. Складність реалізації алгоритмів, що порівнюють структуру програм, є платою за їх точність.

Тому для запланованої системи необхідно обрати максимально швидкий алгоритм, що можна визначити лише порівнянням існуючих алгоритмів між собою. Досліджуючи існуючі системи і створені алгоритми варто виділити два досить прості алгоритми – алгоритм жадібного рядкового замощення і метод відбитків.

Єдиним ефективним шляхом їх дослідження є реалізація цих алгоритмів і збір значної кількості даних на великій виборці програних кодів. Це дозволить порівняти отримані дані за допомогою таких загальновідомих продуктів як Microsoft Excel, використовуючи його зручні засоби для обробки

табличних даних. На основі цієї обробки і будуть зроблені певні висновки про реалізовані алгоритми і їх ефективність у контексті запланованої системи.

Опис алгоритмів, що порівнюються

Для початку розглянемо евристичний алгоритм отримання жадібного рядкового замощення (жадібного спільного покриття рядків). Він отримує на вході два рядки символів певного алфавіту, що відповідають двом програмним кодам. Для представлення програми у вигляді такого рядка застосовують процедуру токенізації [2], яка полягає у виділенні з програмного коду усіх значущих операторів і ключових слів (наприклад: *if, else, while, for, try* та інших) та побудові рядка із символів, відповідних вибраним словам. Такі символи називають токенами, а рядок – рядком токенів, чи токенізованим представленням программного коду. Результатом застосування жадібного алгоритму для таких рядків буде набір їх спільних підрядків, що не перетинаються. Причому можна сказати, що такий набір буде близьким до оптимального. Підрядок, що входить у цей набір, ми будемо називати тайлом.

В структурі цього алгоритму можна виділити 4 цикли:

- зовнішній цикл. Його результатом є додавання у набір чергового тайлу, причому тайли додаються у порядку зменшення, починаючи із найбільшого;
- внутрішній цикл по усім символам першого рядку;
- внутрішній цикл по усім символам другого рядку;
- цикл по співпадаючим символам, починаючи від першого співпадання поточних (відносно другого і третього циклів) символів даних рядків.

Слід зазначити, що цей алгоритм використовує дві евристики:

- спочатку ми знаходимо найбільші спільні підрядки, що складаються з непомічених символів (помічаються такі символи, які вже належать певному токену, були вибрані раніше), тому вважаємо, що довгі співпадання для нашої задачі найбільш значущі. Зрозуміло, що внаслідок цієї евристики ми іноді можемо не знайти найбільш оптимальне (в плані кількості покриваючих підрядків) спільне часткове покриття, але це було б для нас абсолютно несуттєво без наступної евристики;
- якщо в алгоритмі ми будемо дозволяти враховувати надто малі найбільші спільні префікси, то випадкові співпадання невеликої кількості токенів будуть впливати на визначення подібності програм, тому, щоб цього уникнути, ми вводимо константу мінімальної довжини тайлу, яка є вхідним параметром алгоритму.

Псевдокод алгоритму:

```
Greedy-String-Tiling(String P, String T, int
MinimumMatchLength) {
    tiles = {};
    do {
      maxmatch = MinimumMatchLength;
      matches = {};
      Forall unmarked tokens P_p in P {
        Forall unmarked tokens T_t in T {
        j = 0;
        while (P_{p+j} == T_{t+j}) &&
unmarked(P_{p+j}) && unmarked(T_{t+j})
        j++;
        if (j == maxmatch) {
          matches = matches U match(p, t, j);
          } else if (j > maxmatch) {
            matches = {match(p,t, j)}:
            maxmatch = j;
            }
         }
      }
    Forall match(p,t. maxmatch) ∈ matches {
      if (not occluded) {
        for j = 0 ... (maxmatch — 1) {
          mark(P_{p+j});
          mark(T_{t+j});
        }
        tiles = tiles U matches(a, b,
maxmatch);
      }
    }
    } while (maxmatch >
MinimumMatchLength);
      return tiles;
  }
```

Табл. 1. Коментарі до псевдокоду

String P	перший рядок токенів, вхідний параметр алгоритму
String T	другий рядок токенів, вхідний параметр алгоритму
int MinimumMatchLength	довжина мінімального тайлу, вхідний параметр алгоритму
tiles	множина знайдених тайлів
maxmatch	довжина найбільшого спільного підрядку (не тайлу) на поточній ітерації зовнішнього циклу (першого з чотирьох описаних вище)
unmarked(*токен)*	певна функція, що повертає *true*, якщо відповідний токен непомічений (не належить жодному із вже знайдених тайлів), чи *false* у протилежному випадку
matches	множина найбільших (рівної довжини) підрядків, знайдених на поточній ітерації зовнішнього циклу (першого з чотирьох описаних вище)
match(p, t, j)	поточні індекси кожного із двох рядків (параметри другого і третього з описаних вище циклів), що порівнюються і знайдена довжина спільного підрядку (параметр четвертого циклу)
not occluded	жоден із токенів, які належать підрядку не був помічений на попередніх ітераціях
mark(*токен)*	помітити відповідний токен

Зрозуміло, що чим більша нормалізована сума довжин отриманих тайлів (знайдених за описаними вище правилами), тим більше схожість даних підрядків. Тому функцію подібності для цього алгоритму можна представити у вигляді:

$$sim(P,T) = \frac{2 \cdot |tiles|}{|P| + |T|}, \qquad (1)$$

де $|tiles|$ - потужність (за кількістю символів) знайденого набору тайлів, а $|P|$, $|T|$ - відповідні потужності рядків, що порівнюються.

Метод відбитків [3], або метод ідентифікаційних меток передбачає представлення токенізованого рядку у вигляді набору хешзначень (відбитків) і порівняння в подальшому цих наборів. Для цього даний рядок розбивають на $(m-(k-1))$ підрядків (де m – довжина вхідного рядку) довжини k. Мається на увазі що беруть k символів починаючи з першого, k символів починаючи з другого, k символів починаючи з третього… і так далі до кінця рядку. Такі підрядки називатимемо k -грамами.

Наприклад маємо на вході рядок:

afriendtoallisafriendtonone

Він складається з 27 символів, тобто m = 27. Припустимо будемо розбивати рядок на k-грами довжиною 5 символів (k = 5):

afrie, frien, riend, iendt, endto, ndtoa, dtoal, toall, oalli,
allis, llisa, lisaf, isafr, safri, afrie, frien, riend,
iendt, endto, ndton, dtono, tonon, onone

Тобто маємо наступну кількість k-грамів:

$$n = (m-(k-1)) = (27-(5-1)) = 23. \qquad (2)$$

До кожного з k-грамів застосовуємо певну хеш-функцію, отримуючи в результаті набір ідентифікаційних міток. Для порівнювання таких наборів міток, отриманих для різних програм, використовують алгоритм просіювання. Під час пошуку спільного підрядка за допомогою цього алгоритму, використовуються наступні умови:

Якщо довжина підрядка більша чи рівна гарантованій довжині t, то співпадіння буде виявлено.

Спів падіння, коротші за шумовий поріг k, ігноруються (цей пункт забезпечується виділенням з тексту k-грамів).

Суть алгоритму в тому, що ми просуваємо вікно розміру w вздовж нашої послідовності міток, вибираючи на кожному кроці мінімальне значення, причому значення ширини вікна знаходимо за формулою:

$$w = (t-(k-1)). \qquad (3)$$

Покажемо принцип роботи алгоритму на прикладі. Нехай для даного рядка:

afriendtoallisafriendtonone

побудовано набір ідентифікаційних міток за k-грамами довжини k = 5:

30, 12, 7, 41, 55, 3, 37, 24, 62, 15, 43, 22, 59, 6, 30, 12, 7, 41, 55, 29, 17, 49, 57.

Припустимо нас цікавлять спів падання довжини 7 і більше, тоді:

$$w = (t-(k-1)) = (7-(5-1)) = 3, \qquad (3a)$$

і отримуємо наступну послідовність вікон:

*(30, 12, **7**), (12, 7, 41), (7, 41, 55),*
*(41, 55, **3**), (55, 3, 37), (3, 37, 24),(37, **24**, 62),*
*(24, 62, **15**), (62, 15, 43), (15, 43, 22),*
*(43, **22**, 59), (22, 59, **6**),(59, 6, 30), (6, 30, 12),*
*(30, 12, **7**), (12, 7, 41), (7, 41, 55),*
*(41, 55, **29**),(55, 29, **17**), (29, 17, 49),*
(17, 49, 57),

причому жирним виділено значення, що назначено мітками. Кінцевий набір міток буде наступним:

7, 3, 24, 15, 22, 6, 7, 29, 17.

Далі цей набір порівнюється з аналогічним набором іншого програмного коду для пошуку схожості між ними.

Проведення експерименту
Засоби розробки

Для реалізації поставленої задачі обрано мову програмування Java, оскільки вона досить проста для розуміння, безкоштовна і надає необхідні для більш простої реалізації інструменти (значна бібліотека класів, зручні засоби створення інтерфейсу, незалежність від платформи). Також для розробки використовується середовище програмування NetBeans 6.5, оскільки її розробники найбільш тісно пов'язані з розробниками Java, що робить її дуже зручною і, крім того, вона безкоштовна. Для обробки отриманих даних використовується Microsoft Excel, загальновизнаний стандарт у цій сфері.

Вимоги до апаратного забезпечення

Для тестування використовувались двоядерний процесор Intel Core 2 Duo 2 ГГц, оперативна пам'ять 2 Гб, Unix-подібна операційна система Mac OS X 10.5 Leopard. Але, оскільки програма є платформонезалежною (реалізована на Java), то вибір операційної системи залежить лише від уподобань і можливостей користувача.

Результати експерименту

У процесі розробки і тестуванні алгоритмів на реальних даних, було досягнуто їх стабільної роботи, що дозволило оцінювати їх ефективність і швидкодію.

Було обрано 2 групи файлів: першу – на 82 файли для оцінки швидкодії алгоритмів, другу – на 10 (найбільш цікавих) програмних кодів для оцінки ефективності пошуку плагіату. Причому останні 10 були попарно оцінені людиною на наявність в них плагіату. В результаті отримали наступну кількість пар:

$$\frac{n \cdot n - n}{2} = \frac{10 \cdot 10 - 10}{2} = 45. \qquad (4)$$

Шкала оцінювання лежить на відрізку $[0,1]$, проте людські оцінки знаходяться лише у межах кількох значень на цьому відрізку, оскільки важко досить точно визначити наскільки схожі різні програми. Тому взято 5 можливих оцінок – дві для мало схожих файлів, дві для дуже схожих і одна "середня".

Табл. 2. Шкала оцінювання

0.00	Якщо подібність є, то дуже незначна, щоб приймати до уваги - повністю ігноруємо.
0.10	Це не плагіат, файли зовсім не схожі за виглядом, є певна структурна схожість, але файли досить великі.
0.50	Це не плагіат, але це різні програмні коди однією людини, тому є досить велика схожість.
0.90	Це плагіат, файли дуже схожі. Були змінені коментарі, назви деяких змінних, але все одно це очевидне використання чужого коду.
0.95	Це плагіат, файли практично ідентичні, використані різні вхідні дані.

Оскільки при порівнянні порядок файлів неважливий (порівняння першого файлу з другим, а потім навпаки другого файлу з першим дасть одну й ту саму оцінку) і порівняння двох однакових файлів не потрібно, то отримаємо наступну трикутну матрицю оцінок:

Табл. 3. Попарні оцінки файлів вихідних кодів

Номера файлів, що порівнюються	1	2	3	4	5	6	7	8	9	10
1	–	0.00	0.00	0.90	0.00	0.00	0.00	0.00	0.00	0.00
2	–	–	0.10	0.00	0.90	0.10	0.10	0.10	0.10	0.10
3	–	–	–	0.00	0.10	0.10	0.95	0.10	0.10	0.10
4	–	–	–	–	0.00	0.00	0.00	0.00	0.00	0.00
5	–	–	–	–	–	0.10	0.10	0.10	0.10	0.10
6	–	–	–	–	–	–	0.10	0.10	0.50	0.10
7	–	–	–	–	–	–	–	0.10	0.10	0.10
8	–	–	–	–	–	–	–	–	0.10	0.90
9	–	–	–	–	–	–	–	–	–	0.10
10	–	–	–	–	–	–	–	–	–	–

Потім ту ж саму роботу необхідно було виконати за допомогою реалізованих алгоритмів при різних значеннях вхідних параметрів. Для цього, перш за все, необхідно було отримати токенізовану форму і відбит-

ки кожного з файлів, причому було взято 3 набори відбитків для трьох значень параметра k (див. опис алгоритму) – $k = 3$, $k = 4$, $k = 5$.

Значення взято саме такі, для того щоб забезпечити «відсіювання» надто малих співпадінь (< 3 токенів), але в той же час не втратити у точності роботи алгоритму.

Слід зазначити, що процес побудови відбитків виконується після токенізації, що веде за собою певний програш у часі методу ідентифікаційних міток перед алгоритмом жадібного рядкового замощення. Наведемо для порівняння діаграму середнього витраченого часу (у наносекундах) на обробку 82 файлів (для більшої достовірності результатів для кожного з алгоритмів викинуто максимальне і мінімальне значення з вибірки):

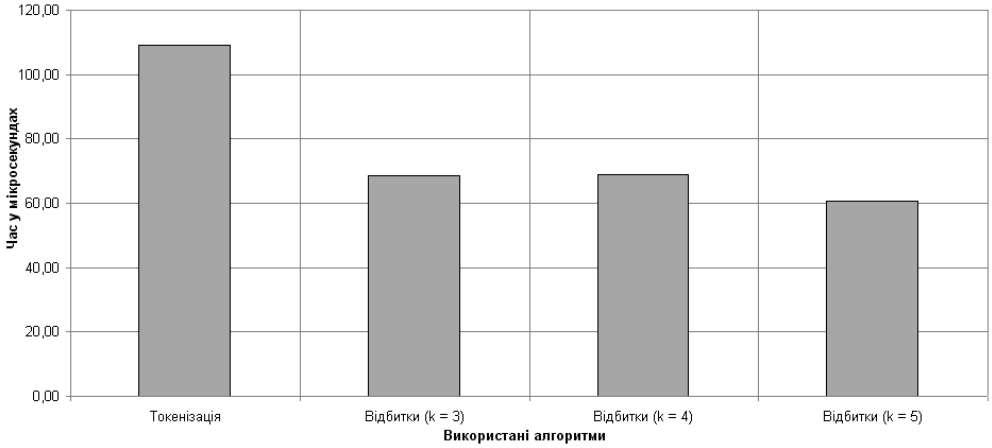

Рис. 1. Порівняння середнього часу обробки програмних кодів алгоритмів

Оскільки цей програш у часі одноразовий і, як видно з діаграми, досить невеликий, то він не є суттєвим для оцінки швидкодії цих алгоритмів і надалі ми не будемо до цього повертатись.

Далі оброблені 10 програмних кодів були порівняні між собою використовуючи реалізовані алгоритми, в результаті чого отримали досить значну вибірку даних.

Спочатку наведемо графік порівняння людських оцінок і оцінок базового жадібного алгоритму:

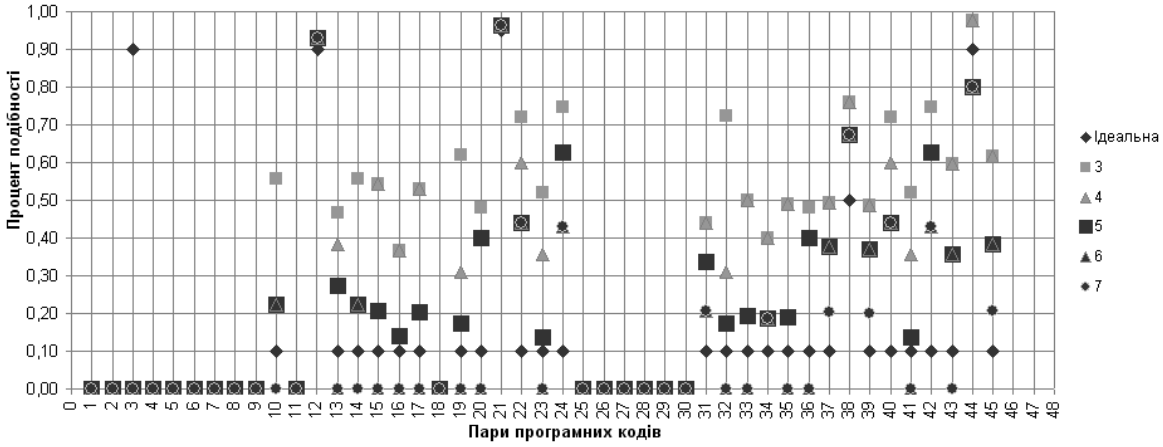

Рис. 2. Порівняння роботи базового Жадібного алгоритму при різних мінімальних довжинах найбільшого префіксу

Людські оцінки зазначено як «ідеальні», а всі інші (при різних значення мінімальної довжини максимального покриття – цими значеннями і помічені графіки у легенді). Перший «пік» ідеальної оцінки не помітила жодна з реалізацій (навіть з подальших) – це пояснюється дуже малою величиною кожного з цієї пари програмних кодів (токенізоване представлення складало 2 і 3 символи відповідно), тому ці співпадіння відсіювались як

випадкові і не приймались до уваги взагалі. Всі інші випадки плагіату були добре помічені усіма реалізаціями, навіть той випадок коли два файли вихідних кодів різних програм писала одна й та сама людина, внаслідок чого у коді присутній деякий свій виразний стиль, який програма розпізнала плагіат. Але, оскільки плагіат в межах однією людини таким не є, то з цією неточністю варто боротись не шляхом зміни алгоритму, а шляхом перевірки хто є номінальним автором програмного коду іншими шляхами.

З графіку очевидно, що найближчим до ідеального є графік 5, оскільки він близький до нього не тільки у випадках явного плагіату, але й на усій вибірці також. Це означає, що при токенізації для отримання вірогідних результатів варто вибирати саме таку мінімальну довжину максимального покриття.

Тепер поглянемо наскільки впливає на результат оцінювання введення коефіцієнту помилки (розглянемо для обраної мінімальної довжини максимального покриття рівній 5):

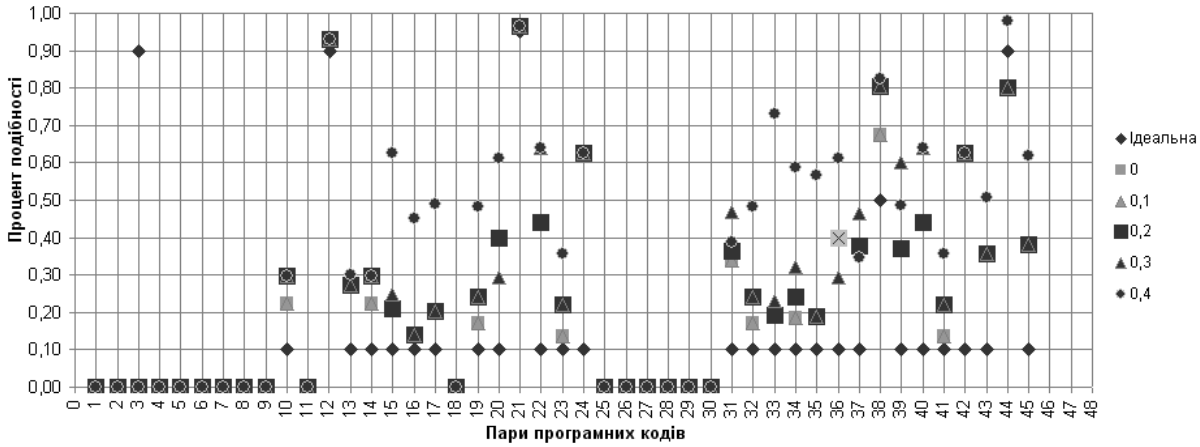

Рис. 3. Порівняння роботи Жадібного алгоритму при мінімальній довжині найбільшого префіксу = 5 і різних відносних похибках

Порівнюючи побудовані графіки з ідеальним, можна сказати що в межах від 0 до 0,3 цей коефіцієнт слабко впливає на результат, а при більшому вже з'являються «паразитні» випадки плагіату, що є негативним ефектом для оцінювання.

Отже з усіх перевірених варіантів застосування жадібного алгоритму слід зупинитися на його базовій реалізації з мінімальною довжиною максимального покриття рівній 5.

Тепер перевіримо ефективність методу ідентифікаційних міток. Побудуємо три графіки, розділивши всі варіанти використання на три групи за коефіцієнтом k:

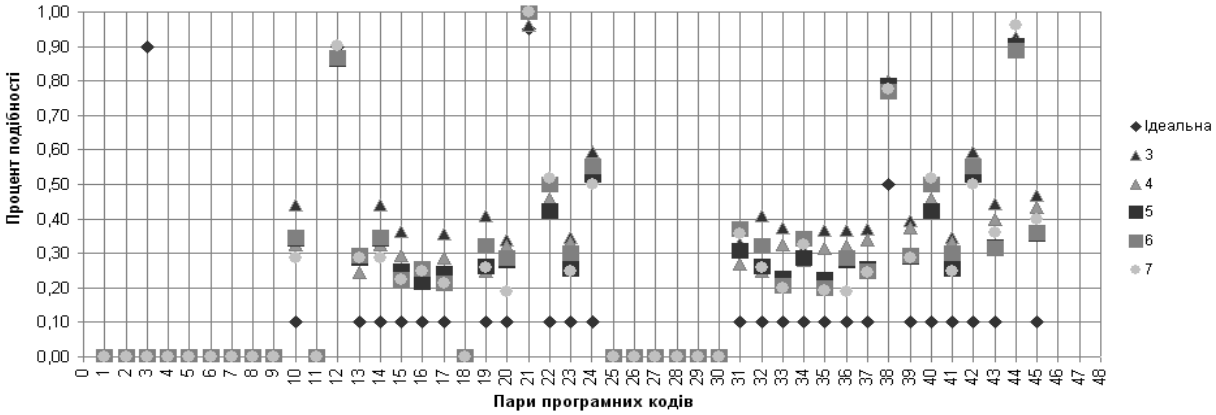

Рис. 4. Порівняння роботи методу відбитків при сталому значенні довжини одного k-граму (k = 3) і різних значеннях ширини вікна

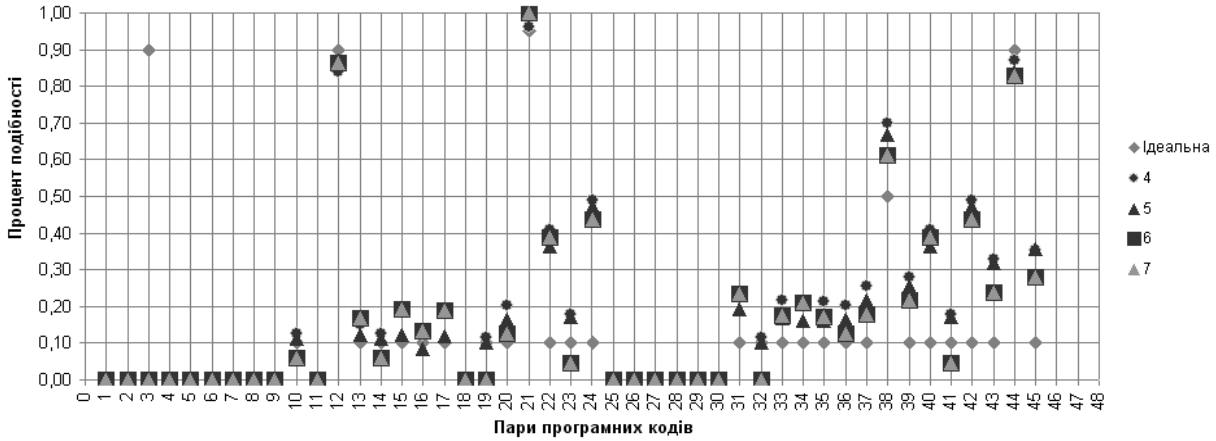

Рис. 5. Порівняння роботи методу відбитків при сталому значенні довжини одного k-граму (k = 4) і різних значеннях ширини вікна

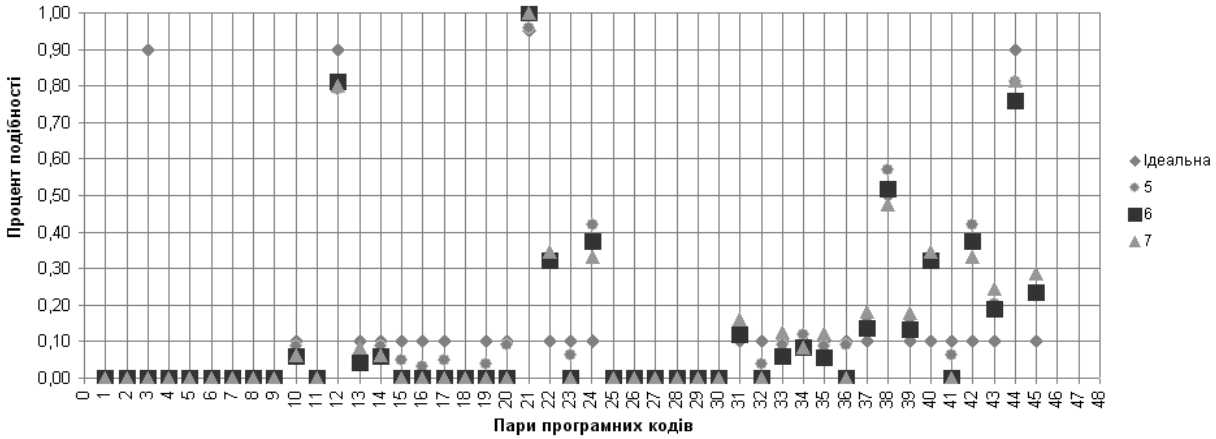

Рис. 6. Порівняння роботи методу відбитків при сталому значенні довжини одного k-граму (k = 5) і різних значеннях ширини вікна

Можна побачити, що для значення $k = 3$ оцінки імовірності плагіату між піками дуже великі, що призводить до помилки другого роду (знаходимо плагіат там, де його насправді немає), а для значення $k = 5$ навпаки маємо дуже малу ймовірність плагіату, що може призвести до помилки першого роду (плагіат не буде знайдено, хоча насправді він має місце). Тому досить розумним буде обрати середній варіант, особливо для значень коефіцієнтів $k = 4$, $t = 5$, оскільки вони найбільш близькі до ідеального у піках. Проте цей вибір ще не є остаточним.

Для порівняння точності роботи алгоритмів (відносно людської оцінки подібності програмних кодів) використаємо метод середньоквадратичної похибки:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (sa_i - sh_i)^2}{n}}, \qquad (5)$$

де *sa* – коефіцієнт подібності, знайдений поточним алгоритмом (з поточними параметрами); *sh* – заданий людиною коефіцієнт подібності (ідеальний); *n* - кількість пар програмних кодів, що порівнюються.

Знайдемо і порівняємо середньоквадратичні похибки для всіх наведених алгоритмів:

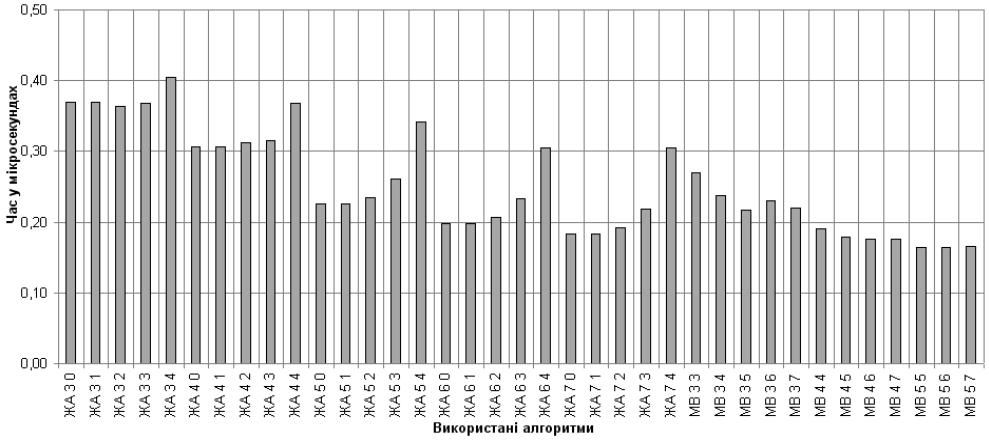

Рис. 7. Порівняння середньоквадратичної похибки алгоритмів

Тепер порівняємо час роботи алгоритмів для різних параметрів на виборці у 82 файли програмних коди. З графіку очевидно, що при будь-яких параметрах метод відбитків працює значно швидше (приблизно вдвічі) ніж алгоритм жадібного рядкового замощення.

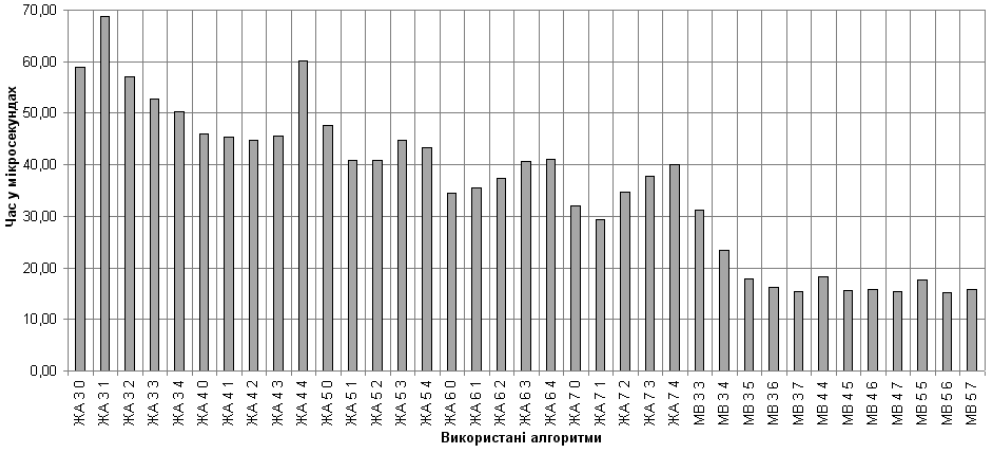

Рис. 8. Порівняння середнього часу роботи алгоритмів

Висновки

У цій роботі ми порівняли два алгоритми попарного порівняння програмних кодів програм для пошуку плагіату в них – це алгоритм жадібного рядкового замощення і метод відбитків (ідентифікаційних міток).

Алгоритм жадібного рядкового замощення у якості вхідних данних використовує токенізоване представлення програмного коду, а метод відбитків використовує набір числових значень (відбитків), що будується з токенізованого представлення. Очевидно, що на цьому етапі другий метод має певний програш у часі, але він є незначним, оскільки отримані один раз мітки можна зберігати для повторного використання при наступних порівняннях.

Порівняння алгоритмів проводилося при різних значеннях основних параметрів, тому важливо було не тільки обрати оптимальний алгоритм, але й оптимальні значення цих параметрів. Для дослідження було використано попарно оцінений людиною набір програмних кодів і всі результати роботи алгоритмів

порівнювалися перш за все з оцінками людини.

Дослідивши час роботи і ефективність реалізованих алгоритмів, ми зробили висновок, що метод ідентифікаційних міток є більш перспективним для використання у потужній системі ніж алгоритм жадібного рядкового замощення, оскільки він працює майже у два рази швидше, не поступаючись при цьому жадібному алгоритму (а часто і випереджаючи його) за якістю знаходження плагіату у програмних кодах. Серед можливих значень параметрів для цього алгоритму варто виділити алгоритм з параметрами k = 4, або k = 5 та t = 5, оскільки при цих значеннях спостерігається найкраще співвідношення швидкодії та ефективності. Але якщо взяти k і t рівні, то це дозволить не використовувати алгоритм просіювання, а одразу шукати множину спільних міток, тим самим трохи скоротивши процес порівняння кожної з пар кодів.

Результати цих досліджень доводять значну ефективність пошуку плагіату цими алгоритмами, показуючи при цьому очевидну перевагу у швидкодії одного алгоритму над іншим. Це дає змогу використовувати у системі пошуку плагіату найбільш оптимальний алгоритм, що критично для великої бази вихідних кодів, яка лежить в основі такої системи.

Список літератури

1. Огляд автоматичних детекторів плагіату в програмах, 2006 http://detector.spb.su/pub/Sandbox/ReviewAlgorithms/survey.pdf

2. Michael J. Wise. String similarity via greedy string tiling and running Karp-Rabin matching. Dept. of CS, University of Sydney, ftp://ftp.cs.su.oz.au/michaelw/doc/RKR.

GST.ps, December 1993

3. Aiken A., Schleimer S., Wikerson D., Winnowing: local algorithms for document fingerprinting. In *Proc 2003 ACMSIGMOD Int. Conf. on Management of Data,* San Diego, CA, June 9-12, pp. 76-85. ACM Press, New York, USA, 2003.

Поступила в редакцию 2.12.2009