Сущность БД «Time_reception_preparations» имеет следующие атрибуты:

– id_time_reception – номер назначения по порядку, целочисленного типа;

– Время_приема_лекарственных_сре – возможные варианты приема лекарственных средств по времени, символьного типа.

Сущность БД «Harmful_habits» имеет следующие атрибуты:

– id_harmful_habits – номер вредных привычек пациента по порядку, целочисленного типа;

– Вредные привычки – вредные привычки пациента на данный момент, символьного типа.

Сущность БД «Diuresis» имеет следующие атрибуты:

– id_Diuresis – номер вредных привычек пациента по порядку, целочисленного типа;

– Диурез – Состояние диуреза у пациента на данный момент, символьного типа.

На этапе логического проектирования БД информационной системы цитоморфологобио-физической диагностики была разработана физическая модель схемы данных, основанная на реляционной модели (рис. 1).

Исходя из требований к информационному обеспечению, выбрана физическая модель, ориентированная на СУБД MS SQL Server 2008, так как она показывает наивысшее быстродействие[2].

4. Выводы

БД информационной системы цитоморфологобиофизической диагностики разработана на основе реляционной модели, которая имеет простую и удобную для пользователя схему данных в виде таблиц. Она позволяет хранить необходимый набор текстовых, числовых и графических данных о пациенте, представить их в удобном, структурированном виде, с возможностью корректировки этих данных, а так же имеет возможность быстрого доступа к данным для их визуального отображения и проведения операций, связанных с их анализом и классификацией.

Литература

1. Проблема формализации текстовых данных в универсальных медицинских информационных системах / М.Ю. Болгов, Д.А. Микитенко // Украинский журнал телемедицины и медицинской телематики. – 2006. – Т.4, №2. – С.171-176.

2. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. Database Systems: A Practical Approach to Design, Implementation, and Management Third Edition 3-е изд. / Т. Коннолли, К. Берг — М.: "Вильямс" 2003. — 1436 с. — ISBN 0-201-70857-4.

УДК 001.891:65.011.56

АЛГОРИТМИ ПОШУКУ ПЛАГІАТУ

А.А. Чижова
Харківський національний університет радіоелектроніки
просп. Леніна, 14, м. Харків, Україна, 61166
Контактний тел.: 097-847-29-63
E-mail: Chijova_alla@ukr.net

Дана загальна характеристика алгоритму. Досліджено основні алгоритми пошуку плагіату тексту. Описано основні принципи роботи алгоритмів

Ключові слова: алгоритм, реалізація, програма, пошук

Дана общая характеристика алгоритма. Исследованы основные алгоритмы поиска плагиата текста. Описаны основные принципы работы алгоритмов

Ключевые слова: алгоритм, реализация, программа, поиск

A general characteristic of the algorithm. The basic algorithms for finding plagiarism of text. The basic principles of the algorithms

Keywords: algorithm, implementation, program search

1. Введення

Плагіат - навмисне привласнення авторства чужого твору науки чи думок або мистецтва чи винаходи. Плагіат може бути порушенням авторсько-правового законодавства та патентного законодавства і в їх якості може спричинити за собою юридичну відповідальність. З іншого боку, плагіат можливий і в областях, на які не поширюється дія яких-небудь видів інтелектуальної власності, наприклад, в математиці та інших фундаментальних наукових дисциплінах.

Алгоритм - це система правил виконання обчислювального процесу, що обов'язково приводить до розв'язання певного класу задач після скінченного

числа операцій. При написанні комп'ютерних програм алгоритм описує логічну послідовність операцій. Для візуального зображення алгоритмів часто використовують блок-схеми.

2. Алгоритми пошуку плагіату

Метод ідентифікаційних міток.

При пошуку плагіату потрібно знаходити копії та часткові копії файлу у текстовій базі великого обсягу. У цьому випадку безпосереднє порівняння файлів не ефективне.

Розглядаєма техніка дозволяє перевести файл у більш коротке подання: до документа зіставляється набір ідентифікаційних міток, так щоб для близьких документів ці набори перетиналися.

Побудову набору міток для документа розглянемо на довільному тексті:

abracadabra

(він складається з 11 символів; m = 11)

К-грамом називаються будь-які к символи що стоять підряд. Побудуємо всілякі к-грами для цього тексту при, наприклад, к = 3:

abr, bra, rak, aka, kad, ada, dab, abr, bra

Кількість к-грамів, які можна побудувати для тексту довжини m позначимо n,

$n = (m - (до - 1))$ (у прикладі n = 9) (2.2)

Хешуємо всі к-грами, набір хеш - значень (h1 ... hn) що вийшов характеризує вихідний документ. Для розглянутого тексту виходить така послідовність хеш - значень:

12, 35, 78, 3, 26, 48, 55, 12, 35

На практиці використовувати всі значення не доцільно, тому вибирають невелику їх кількість. Вибрані хеш - значення стають дрібними документами [9]. Разом з самою міткою зберігається інформація про те, якому файлу вона належить і в якому місці цього файлу зустрічається. Якщо хеш-функція гарантує малу ймовірність колізій, то однакова мітка у наборах двох файлів свідчить про те, що у них є спільний підрядок. За кількістю загальних міток можна судити про близькість файлів.

Вибір хеш - значень, які будуть представляти документ:

– наївний підхід - вибрати кожне і-те з n значень. Проте, такий спосіб не стійкий до вставлення та видалення символів, зміни їх порядку. Тому спиратися на позицію всередині документа не можна;

– за Хайнце слід призначити мітками p - мінімальних хеш - значень, їх кількість для всіх документів буде постійно. За допомогою цього методу можна знайти часткові копії, але він добре працює на файлах приблизно одного розміру, знаходить схожі файли, може застосовуватися для класифікації документів;

– Манбер запропонував обирати в якості міток тільки ті хеш - значення, для яких $h \equiv 0 \bmod p$, так залишиться тільки n / p позначок. Однак, у цьому випадку відстань між послідовно вибраними хеш - значеннями не обмежена і може бути велика. У цьому разі збіги, які опинилися між мітками, не будуть враховані;

– метод просіювання не має цього недоліку. Алгоритм гарантує, що якщо в двох файлах є хоч би один досить довгий загальний підрядок, то як мінімум одна позначка у їх наборах збігається.

Алгоритм просіювання для побудови позначок. При пошуку загального підрядка у файлах необхідно керуватися наступними умовами:

– якщо довжина підрядка збігається більше або дорівнює гарантованої довжині t, то збіг буде виявлено;

– збіг коротше шумового порогу к, ігнорується.

Пункт 2 забезпечений виділенням з тексту *к*-грамів. Чим більше к, тим менше ймовірність, що збіги випадкові. Але із зростанням к падає стійкість методу до перестановок. У художніх текстах зазвичай за к приймають середню довжину стійких виразів.

Щоб задовольнити пункт 1 необхідно, щоб кожний з послідовно ідучих (t-*к* +1) хеш - значень хоча б одне було обрано як позначка.

Ідея алгоритму: просуваємо вікно розміру w = =(t - к + 1) вздовж послідовності $h_1 \ldots h_n$, на кожному кроці вікно переміщається на одну позицію вправо. Призначається міткою мінімальне h_j у вікні.

Якщо у одному вікні два елементи приймають мінімальне значення, правий призначається міткою.

Алгоритм Хескела.

Нехай є дві програми, які представлені у вигляді рядків токенів *а* і *б* відповідно. Одним з критеріїв подібності рядків вважається довжина їх найбільшої загально послідовності. Завжди можна знайти такий елемент рядка а і, що НОП рядків $a' = a_{|a|}a_{|a|}\text{-}1 \ldots a_i a_1 \ldots a_{i-1}$ і буде значно менше, ніж НОП (*а, б*) (якщо НОП (*а, б*)> 1). Щоб уникнути цього явища можна скористатися алгоритмом порівняння рядків Хескела, він вимагає декількох проходів, але працює за лінійне час [13]. Розбиваються рядка *а* і *б* на к-грами. Знаходяться ті к-грами, які зустрічаються в *а* і *б* тільки по одному разу. Для кожної такої пари перевіряється чи збігаються елементи рядків, безпосередньо що лежать над ними; якщо це так, то проводиться та ж перевірка і для них і так далі, поки розбіжність не як знайдено. Аналогічно для рядків, що лежать нижче відповідних *к-грамів*. Виходить набір загальних непересічних підрядків а і б. Їхня загальна довжина може служити мірою схожості програм відповідних *а* і *б*.

Переваги і недоліки.

Переваги:

– лінійна трудомісткість алгоритму.

Недоліки:

– можливість збігу токенізірованного представлення програм, але відсутність збігу в початкових кодах програм;

– невелика кількість унікальних *к-грамів* у великих програмах, відповідно багато збіги, не містять в собі *к-грамів* будуть проігноровані;

– вставка в знайдений блок або зміна на семантично еквівалентний оператора в багатьох випадках буде призводити до ігнорування тієї частини блоку, в якій не міститься унікальний *к-грам*.

Колмогорівськая складність у задачі знаходження плагіату.

Відстань між послідовностями, засноване на теорії інформації:

$$d(x,y) = 1 - \frac{K(x) - K(x|y)}{K(xy)} \quad (2.5)$$

де *K(x)* - Колмогоровская складність послідовності x. Вона показує скільки інформації містить послідовність *x*. За визначенням, *K(x)* - довжина найкоротшої

програми, яка на порожній введення друкує x, K(x|y) - кількість інформації, отриманої x від y, якщо пусто, то воно дорівнює K(x); (K(x) - K(x|y)) - скільки у «знає» про x. За визначенням, K(x|y) - це довжина найкоротшої програми, яка на введення у друкує x.

Основна відмінність наведеної метрики від інших, що використовуються в задачах визначення плагіату, полягає в її універсальності: дві програми, близькі щодо будь-якої іншої метрики, будуть близькими і щодо даної. Теоретично, детектор, заснований на такий метриці, неможливо обдурити [15].

Процес порівняння двох програм.

Спочатку проводиться токенізація, далі запускається алгоритм TokenCompress, заснований на алгоритмі стиснення LZ. Першим ділом він знаходить довжелезнмий неточно повторюваний підрядок, який закінчується в поточному символі; кодує її указником на попереднє розміщення і зберігає інформацію про внесені поправки.

Реалізація відрізняється від класичного LZ-стиснення деякими особливостями, пов'язаними зі специфікою завдання: класичний алгоритм сімейства LZ з обмеженим буфером може пропустити деякі довгі повторювані підрядків через обмеження на розмір словника під час кодування.

Це не страшно для кодування звичайних текстів, де потрібно перш за все економити пам'ять і зменшувати час виконання, адже втрати якості стиснення будуть украй малі.

Але в силу специфіки завдання і вибраного алгоритму її рішення, який використовує неточні збіги, втрата навіть невеликої кількості довгих повторюваних підрядків неприйнятна.

Псевдокод алгоритму:

```
Algorithm TokenCompress
Input: A token sequences
Output: Comp(s) and matched repeat pairs
i=0
An empty buffer B;
while (i < s) {
p = FindRepeatPair (i) {
if (p.compressProfit > 0) {
EncodeRepeatPair (p, compFile);
i=i  p.length;
OutputRepeatPair (p, repFile);
} else {
AppendCharToBuffer (s i , B);
i++;
}
}
EnodeLiteralZone (B, compFile);
return Comp (s) = compFile.file_size;
and repeat pairs stored in repFile.
```

Передбачається, що спочатку алгоритму ініціалізуємо словник усіма підстроками рядка y, а далі алгоритм залишається без змін.

Класичні алгоритми сімейства LZ НЕ підтримують неточних збігів. Цей же алгоритм шукає неточно повторювані підрядка та кодує їх збігу. TokenCompress використовує порогову функцію, щоб визначити чи вигідніше кодувати невідповідності або краще скористатися альтернативами.

Відповідно, деякі неточні повтори можуть бути об'єднані в один з невеликою кількістю помилок. Неточно збіглися пари підрядків записуються у файл і пізніше можуть бути переглянуті людиною.

На основі роботи цього алгоритму вважається *d(x, y)*, яка потім використовується для визначення наскільки ймовірно те, що одна з цих програм є плагіатом інший.

Переваги і недоліки.

Переваги:

– переваги токенізірованного подання;

– використовується евристичне наближення метрики, такий що, якщо дві програми, близькі щодо будь-якої іншої функції відстані, будуть близькими і щодо даної;

– загальні підрядки, менше порогової довжини ігноруються, тому алгоритм не візьме до уваги малі випадково збіглися ділянки коду;

– при розбитті співпала ділянки коду на дві і більше частини вставкою одного - декількох блоків або одиночних операторів, а також перестановкою найбільшої кількості незалежних операторів, функція схожості слабо змінюється;

– вкрай висока стійкість до вставці операторів;

– алгоритм нечутливий до перестановок великих фрагментів коду.

Недолік складається у тому що можливість збігу токенізірованного представлення програм, але відсутні збіги в початкових кодах програм.

Метод пошуку на XML уявлень.

Представлення програми у вигляді дерева відображає її корисні для пошуку плагіату властивості, і не враховує даремні.

Метод пошуку плагіату, заснований на представленні програми у вигляді дерева, опис якого зберігається у форматі XML. Використання стандартних інструментів для роботи з XML значно спрощує архітектуру детектора плагіату.

Програми, написані на процедурних мовах, таких як Pascal і C, добре структуровані, тому отримати їх у XML поданні легко. Для оцінки близькості двох програм використовуються числові матриці, побудовані на основі XML описів [16].

Побудова XML-уявлень.

У процедурних мовах програмування програма ділиться на основні структурні блоки.

Елемент Block містить відомості про логіку управління у функції. Усередині нього Contents несе інформацію про операції присвоєння, виклики функцій та інших незалежних операторах. Елемент Control Type зберігає тип управління, внутрішній Block описує наступний рівень вкладеності.

Відповідне наведеній схемі XML представлення програми виходить очевидним способом. За цим поданням будуються числові матриці, що відображають структуру програми.

Матриця будови програми. Кількість *N* стовпців в матриці залежить від конкретної мови програмування, передбачається, що воно достатньо велике.

Перший рядок цієї матриці зберігає інформацію про підключаються в програмі заголовних файлах. Всі можливі для даної мови стандартні заголовки нумеруються числами. Якщо в конкретній програмі використовується заголовок з номером i, то в першому

рядку структурної матриці на і - тому місці ставимо 1, у противному випадку - 0.

Другий рядок відповідає за глобальні змінні. Можливі типи змінних занумеровані, на і - тому місці у другому рядку матриці варто кількість глобальних змінних даного типу в програмі.

Решта рядка в матриці описують функції, для кожної з функцій відводяться 2 рядки. У першій з них перші к стовпців відповідають за тип, що повертається значення, стовпчики з *(к + 1)*-ого ставлять у відповідність можливим типам кількість аргументів даного типу.

Аналогічно, у другому рядку, що описує функцію, в перші к стовпцях зберігається інформація про кількість локальних змінних даного типу, у стовпцях с *(к + 1)* на і-тому місці стоїть кількість керуючих операторів даного типу.

Матриця управління програми. Ця матриця відображає логіку роботи програми, вона зберігає послідовності керуючих операторів для кожної з функцій.

Ці послідовності можна витягти з XML представлення програми за допомогою стандартного інструменту XQuery.

Він аналізує XML і рекурсивно, подібно алгоритмом пошуку в глибину, витягує послідовність типів операторів управління.

Для кожної функції будується масив, в якому містяться номери операторів управління в порядку їх слідування всередині функції.

Оцінка близькості програм.

На основі отриманих числових матриць, будується розширена матриця, яка містить інформацію про структуру програми.

h_1	h_2	...	h_k	h_{k+1}	h_{k+2}	...	hn
g_1	g_2	...	g_k	g_{k+1}	g_{k+2}	...	gn
r_{11}	r_{12}	...	r_{1k}	a_{11}	a_{12}	...	a_{1n-k}
l_{11}	l_{12}	...	l_{1k}	c_{11}	c_{12}	...	c_{1n-k}
cd_{11}	cd_{12}	...	cd_{1k}	cd_{1k+1}	cd_{1k+2}	...	cd_{1n}
r_{21}	r_{22}	...	r_{2k}	a_{21}	a_{22}	...	a_{2n-k}
l_{21}	l_{22}	...	l_{2k}	c_{21}	c_{22}	...	c_{2n-k}
cd_{21}	cd_{12}	...	cd_{2k}	cd_{2k+1}	cd_{2k+2}	...	cd_{2n}
...	...	...	...	...	...	...	...
r_{m1}	r_{m2}	...	r_{mk}	a_{m1}	a_{m2}	...	a_{mn-k}
l_{m1}	l_{m2}	...	l_{mk}	c_{m1}	c_{m2}	...	c_{mn-k}
cd_{m1}	cd_{m2}	...	cd_{mk}	cd_{mk+1}	cd_{mk+2}	...	cd_{mn}

Де *h* відповідає заголовками, *g* - глобальним змінним, r_k - повертає значення, *a* - аргументів, *l* - локальним змінним, *c* - типами управління, cd_k представляє послідовність типів операторів управління *к* - тієї функції [17].

Передбачається оцінити близькість двох програм.

Вони представлені матрицями А і В.

Порівнюються матриці по частинам, враховуючи їхній зміст.

«Ступінь схожості» визначається кількістю збігів, наприклад, для розділів заголовків:

$$HeaderVal = A[1][1..n] \times B^T[1][1..n] = (h_1\ h_2\ ...h_{n-1}\ h_n) \times (h_1'\ h_2'\ ...h'_{n-1}\ h'_n)^T = \Sigma(h_i * h_i') = \Sigma\, b_i \quad (2.6)$$

Литература

1. Шаньгин, В.Ф. Защита компьютерной информации. Эффективные методы и средства // Издательский дом «Вильямс», 2008. – 386 с.
2. Воронівка, Г.К. «Генетичні алгоритми, штучні нейронні мережі і проблеми віртуальної реальності» // ОСНОВА, 1997. - 112 с.