«Системні технології» 4 (123) 2019 «System technologies»
DOI 10.34185/1562-9945-4-123-2019-04
УДК 510.25+004.415.2+004.912

О.С. Куроп'ятник

КОНСТРУКТИВНЕ ТА ОБ'ЄКТНО-ОРІЄНТОВАНЕ МОДЕЛЮВАННЯ ТЕКСТІВ ДЛЯ ВИЯВЛЕННЯ ЗАПОЗИЧЕНЬ

Анотація. Метою даної роботи є розробка моделі тексту для виявлення запозичень та доведення її до програмної реалізації. Задачею є розробка об'єктно-орієнтованої моделі та програмної реалізації графової моделі тексту, з застосуванням до задачі виявлення запозичень. А також отримання часових показників роботи програмної реалізації для подальшої для оцінки можливості її використання у академічному середовищі.

Представлено модифікацію раніше розробленої конструктивної графової моделі тексту. На її основі розроблено об'єктно-орієнтовану модель для виявлення запозичень. Вона включає опис елементів моделі (графових та мовних конструкцій) та їх складових, а також методи їх формування й обробки. У відповідності до моделей створено комп'ютерну програму для співставлення текстів, поданих як файли. Виконано комп'ютерний експеримент для отримання часових показників роботи методів побудови графу та порівняння текстів.

Ключові слова: конструктор, об'єктно-орієнтована модель, графова модель тексту, виявлення запозичень.

Постановка проблеми. Задача виявлення запозичень, що є актуальною в академічному середовищі, потребує автоматизованих засобів розв'язання. До таких можна віднести бази матеріалів для перевірки, а також програмне забезпечення (ПЗ), яке дозволить виконувати зіставлення матеріалів та їх оцінку, надаватиме вичерпну інформацію про результати у зручній для сприйняття формі.

Робота такого ПЗ може включати етапи попередньої обробки матеріалів, виявлення запозичень, оцінки та аналізу результатів, формування звітності. Їх суть здебільшого залежить від вхідних даних та формату. У даній роботі мова йтиме про тексти, представленні файлами у форматах txt, rtf, doc, docx.

Виявлення запозичень є одним з ключових етапів. Його реалізація потребує вирішення задач розробки моделей структур даних для представлення текстів та алгоритмів їх співставлення. Дані задачі ускладняються можливістю маскування запозичень [1].

Робота присвячена розробці моделей даних та алгоритмів співставлення текстів та їх програмній реалізації.

Аналіз останніх досліджень і публікацій. Питанням плагіату присвячено чимало робіт, в тому числі у сфері розробки програм-антиплагіатів [2, 3]. Їх робота передусім передбачає виявлення запозичень, а їх реалізація – розробку методів (сукупності операцій) обробки з невисокою алгоритмічною складністю та спеціалізованих структур даних.

Сьогодні науковою спільнотою пропонується використання таких моделей та структур даних як масиви LERP-RSA (найдовший очікуваний повторюваний масив зменшеного шаблону суфікса) [4], теговий класифікатор – модель на основі Stanford NER's three-class [5], структури, основані на ДНК-послідовностях [6], графові представлення [7] тощо.

Використовуються такі алгоритми як GreedyString-Tiling [5, 8], ARPaD [5], шингли [9], статистичні методи [10], генетичні алгоритми [11] та інші. Також слід відзначити, що значна увага приділяється морфологічному аналізу та лемматизації, або стеммінгу, попередній синтаксичній обробці текстів.

Вказані моделі та алгоритми лише частково мають програмні реалізації.

Мета дослідження. Метою даної роботи є розробка моделі тексту для виявлення запозичень та доведення її до програмної реалізації. Задачею є розробка об’єктно-орієнтованої моделі та програмної реалізації графової моделі тексту (основні положення представлені у роботі [12]), з застосуванням до задачі виявлення запозичень. А також отримання часових показників роботи програмної реалізації для подальшої для оцінки можливості її використання у академічному середовищі.

Конструктивна графова модель тексту. Графова модель передбачає представлення тексту у вигляді орієнтованого навантаженого графу [12]. Для формалізації моделі використаємо апарат конструктивно-продукційного моделювання [13].

Для представлення графу визначимо конструктор і спеціалізуємо його відповідним чином:

$$C = \langle M, \Sigma, \Lambda \rangle \ {}_S\mapsto C_g = \langle M_g, \Sigma_g, \Lambda_g \rangle, \qquad (1)$$

де M_g – розширюваний носій, що включає множини конструкцій-графів, мовних конструкцій (слів, речень, абзаців) і їх елементів, Σ_g – множина операцій і відношень на елементах M_g, Λ_g – інформаційне забезпечення конструюван-

ня, що включає онтологію, мету, обмеження, правила, умови початку і завершення конструювання.

Онтологія конструктора графів. Носій включає множини термінальних і нетермінальних елементів $M_g = T_g \cup N_g$. Терміналами є мовні конструкції і їх складові (T_T), а також конструкції графів і їх складові: $T_g = \overline{\Omega} \cup \Omega_g \cup T_T \cup V \cup E$, де Ω_g – множина конструкцій-графів, V, E – множин вершин і дуг з їх атрибутами.

Вершина має атрибути $\overline{w}_v = \langle id, content \rangle$, *id* – ідентифікатор, приймає цілочислені значення, *content* – частина текстової конструкції. Атрибути дуги – $\overline{w}_e = \langle id, routes, start, end \rangle$, де *routes* – множина номерів шляхів, в які входить дуга (вказує на порядок обходу графа), *start, end* – вершини, які є інцедентними до дуги *e*.

Навантажений граф будемо позначати як $_{\overline{w}_g} G = \langle V, E \rangle$ ($V = \{_{\overline{w}_{vi}} v_i\}$, $E = \{_{\overline{w}_{e_j}} e_j\}$ – множини вершин і дуг, навантажених атрибутами). Кожна множина містить порожній елемент.

Граф має атрибути $\overline{w}_g = \langle start\_v, last\_v, current\_v, amount\_l \rangle$, де *start_v* – стартова вершина графа, *last_v* – остання додана вершина, *current_v* – поточна вершина при формуванні графа, *amount_l* – кількість циклів, в які входить стартова вершина.

Розглянемо сигнатуру Σ_g:

$$\Sigma_g = \langle \Xi_g, \Theta_g, \Phi, \{\rightarrow\} \rangle \cup \Psi_g, \qquad (2)$$

де $\Xi_g = \{\cdot, :\widehat{=}, :\doteq, \widetilde{\cup}, \cup\}$ – множина операцій перетворення і зв'язування, $\Theta_g = \{\Rightarrow, |\Rightarrow, ||\widetilde{\Rightarrow}\}$ – множина операцій виводу, $\Phi_g = \{\div, :=, \#, +\}$ – множина операцій над атрибутами, Ψ_g – множина правил продукцій виду $\psi_i :< s_i, g_i >$, i – номер правила, s – послідовність операцій підстановки, g – послідовність операцій над атрибутами, «$\rightarrow$» – відношення підстановки.

Операція $e :\widehat{=} (v_1, v_2, G)$ полягає у визначенні дуги e, що з'єднує вершини v_1, v_2 у графі G.

Операція $v :\doteq (x, V)$ полягає в знаходженні вершини $v \in V$ з атрибутом ваги, рівним x.

Операція $\div(c, n, L)$ полягає у виконанні n операцій зі списку L, якщо $c = true$.

Операція обчислення потужності множини $\#Q$ визначає число, яке дорівнює кількості елементів в Q.

Операція додавання двох чисел $+(a,b)$ передбачає знаходження третього числа, що є їхньою сумою

Операція об'єднання графів ${}_{\bar{w}_g}G = \tilde{\bigcup}({}_{\bar{w}_1}G_1, {}_{\bar{w}_2}G_2)$ передбачає формування нового графа ${}_{\bar{w}_g}G$, що включає об'єднані множини вершин і дуг вихідних графів ${}_{\bar{w}_g}G = \langle V, E \rangle$, де $V = V_1 \bigcup V_2$, $E = E_1 \bigcup E_2$, ${}_{\bar{w}_1}G_1 =< V_1, E_1 >$, ${}_{\bar{w}_2}G_2 =< V_2, E_2 >$, при цьому $\bigcup$ – традиційна операція об'єднання множин.

Відношення підстановки має вигляд

$$\psi_i = \langle s_i, g_i \rangle,\ s_i = \langle \bar{s}_i, \tilde{s}_i \rangle,\ g_i = \left\langle \bar{g}_i\ , \tilde{g}_i\ \right\rangle, \tag{3}$$

де $\bar{s}_i$, $\tilde{s}_i$ – відношення підстановки для розпізнавання мовної конструкції і побудови конструкції графа відповідно, $\bar{g}_i$, $\tilde{g}_i$ – операції над атрибутами мовної конструкції і графа, його вершин і дуг відповідно. У разі якщо операції над атрибутами не виконуються, відношення підстановки має вигляд $\psi = \langle s, \varepsilon \rangle$.

Операція повного виводу $\|\tilde{\Rightarrow}(\Psi, {}_{w_l}l)$ та більш детальна інформація щодо інших операцій наведена в роботі [12]. Результатом операції виведення є конструкція-граф.

Метою конструювання є побудова конструкції графу, яка відповідає заданій конструкції тексту.

Обмеження конструктору графів накладаються конструкцією тексту. Кількість графів залежить від кількості різних символів у тексті.

Початкова умова конструювання: σ – нетермінал, з якого починається вивід.

Умова завершення конструювання: форма не містить нетерміналів, кожному елементу конструкції тексту відповідає елемент конструкції графу.

Конкретизація конструктора графа. Виконаємо конкретизацію конструктора C_g:

$$C_{g\,K} \mapsto {}_K C_g = \langle M_g, \Sigma_g, \Lambda_1 \rangle, \tag{4}$$

де $\Lambda_1 \supset \Lambda_g$, $\Lambda_1 \supset \{c \in T_T,\ \ N_g = \{\alpha, \delta\}\}$, $T_g \supset \{G, G^*, G^{**}\}$, $G = \langle V, E \rangle$, $V = \{v\}$, $E = \varnothing$, $G^* =< V^*, G^* >$, $V^* = \{v_1^*, v_2^*\}$, $E^* = \{e_1^*\}$, $G^{**} =< V^{**}, G^{**} >$, $V^{**} = \{v_1^{**}, v_2^{**}\}$, $E^{**} = \{e_1^{**}\}\}$.

Побудова графа передбачає розпізнавання мовних конструкцій за допомогою правил [12]:

$$\bar{s}_1 = \langle \sigma_{d_1} \to c\sigma \rangle,\ \bar{g}_1 = \langle \div (code \lrcorner c \neq EOF, 1, d_1 := true) \rangle, \quad (5)$$

$$\bar{s}_2 = \langle \sigma_{d_2} \to c \rangle,\ \bar{g}_2 = \langle \div (code \lrcorner c = EOF, 1, d_2 := true) \rangle,$$

де *c* – символ тексту, крім *EOF* – ознака кінця тексту в його електронному поданні.

Правило для додавання першої вершини в граф має вигляд:

$$\tilde{s}_1 = \langle \sigma \tilde{\to} G\alpha \rangle,\ \tilde{g}_1 = \langle id \lrcorner v := \# V, content \lrcorner v := c,$$

$$start\_v \lrcorner G := v,\ current\_v \lrcorner G := v, last\_v \lrcorner G := v, amount\_l \lrcorner G := 0 \rangle. \quad (6)$$

Правило $\tilde{s}_2$ дозволяє додати до графу нову вершину і дугу, яка б пов'язала нову вершину з поточною в графі:

$$\tilde{s}_2 = \left\langle G\alpha \ {}_{d_1^*}\tilde{\to} \tilde{\cup}(G, G^*)\alpha, \tilde{\cup}(G, G^*)\alpha \to G\alpha \right\rangle,$$

$$\tilde{g}_2 = \langle v_1 := (c, V), e_1 := (current\_v \lrcorner G, v, G),$$

$$\div (id \lrcorner v_1 = 0 \,\&\, id \lrcorner e_1 = 0, 11, d_1^* := true,$$

$$id \lrcorner v_1^* := id \lrcorner current\_v \lrcorner G,\ content \lrcorner v_1^* := content \lrcorner current\_v \lrcorner G,$$

$$id \lrcorner v_2^* := \# V + 1,$$

$$content \lrcorner v_2^* := c, id \lrcorner e_1^* := \# E + 1,\ start \lrcorner e_1^* := v_1^*,\ end \lrcorner e_1^* := v_2^*,$$

$$routes \lrcorner e_1^* := \{amount\_l \lrcorner G\},\ last\_v \lrcorner G := v_2,\ current\_v \lrcorner G := v_2) \rangle. \quad (7)$$

Правило $\tilde{s}_3$ дозволяє додати до графу нову дугу, яка б пов'язала поточну вершину зі стартовою:

$$\tilde{s}_3 = \left\langle \alpha \ {}_{d_2^*}\to \tilde{\cup}(G, G^{**})\alpha, \tilde{\cup}(G, G^{**})\alpha \to G\alpha \right\rangle$$

$$\tilde{g}_3 = \langle v_1 := (c, V), e_1 := (current\_v \lrcorner G, v_1, G),$$

$$\div (v_1 = start\_v \lrcorner G \,\&\, id \lrcorner e_1 = 0, 11, d_2^* := true,\ id \lrcorner v_1^{**} := id \lrcorner curret\_v \lrcorner G,$$

$$content \lrcorner v_1^{**} := content \lrcorner curret\_v \lrcorner G,\ id \lrcorner v_2^{**} := id \lrcorner start\_v \lrcorner G,$$

$$content \lrcorner v_2^{**} := c, id \lrcorner e_1^{**} := \# E + 1, start \lrcorner e_1^{**} := v_1^{**}, end \lrcorner e_1^{**} := v_2^{**},$$

$$routes \lrcorner e_1^{**} := \{amount\_l \lrcorner G\},\ amount\_l \lrcorner G := amount\_l \lrcorner G + 1,$$

$$current\_v \lrcorner G = start\_v \lrcorner G) \rangle. \quad (8)$$

Правило $\tilde{s}_4$ дозволяє змінити навантаження існуючої дуги:

$$\tilde{s}_4 = \left\langle G\alpha \ {}_{d_3^*}\to G\alpha \right\rangle,\ \tilde{g}_4 = \langle v_1 := (c, V \lrcorner G), e_1 := (current\_v \lrcorner G, v_1, G),$$

$$\div (id \lrcorner v_1 \neq 0 \,\&\, id \lrcorner e_1 \neq 0, 3, d_3^* := true,$$

$$\div(start\_v \lrcorner G \neq v_1, 5, routes \lrcorner e_1 := routes \lrcorner e_1 \cup \{amount\_l \lrcorner G\},$$

$$\div(start\_v \lrcorner G = v, 6, routes \lrcorner e_1 := routes \lrcorner e_1 \cup \{amount\_l \lrcorner G\},$$

$$current \lrcorner v := start\_v \lrcorner G,\ amount\_l \lrcorner G := amount\_l \lrcorner G + 1))\rangle. \quad (9)$$

Наступне правило дозволяють завершити процес побудови конструкції-графа:

$$\widetilde{s}_5 = \langle \alpha \widetilde{\rightarrow} \varepsilon \rangle. \quad (10)$$

Інтерпретація конструктора графа. Інтерпретуємо конструктор:

$$\langle C_g, C_{A,G} = \langle M_A, V_A, \Sigma_A, \Lambda_A \rangle \rangle_I \mapsto \langle {}_A C_g,\ {}_A C_g = \langle M_g, \Sigma_g, \Lambda_2 \rangle \rangle, \quad (11)$$

де $\Lambda_2 \supset \Lambda_1$, $V_A = \{A_i^0 \mid_{X_i}^{Y_i}\}$ – множина базових алгоритмів [13], X_i, Y_i – множини визначень та значень алгоритму $A_i^0 \mid_{Y_i}^{X_i}$. $\Lambda_2 = \{M_A = \bigcup_{A_i^i \in V_A} (X(A_i^0) \cup Y(A_i^0)) \cup \Omega(C_l) \cup \Omega(C_g)\}$ – неоднорідний носій, $\Omega(C_l)$, $\Omega(C_g)$ – множини мовних і графових конструкцій; $\Lambda_2 \supset \{(A_3 \mid_{l_1,l_2}^{l} \lrcorner'' \cdot'')$; $(A_4 \mid_{l_h,l_q,f_i}^{f_i} \lrcorner''\Rightarrow'')$; $(A_5 \mid_{f_i,\Psi}^{f_j} \lrcorner''|\Rightarrow'')$; $(A_6 \mid_{\sigma,\Psi}^{\overline{\Omega}} \lrcorner''||\Rightarrow'')$; $(A_7 \mid_{a,b}^{a} \lrcorner'':='')$; $(A_8 \mid_{a,b}^{c} \lrcorner''='')$; $(A_9 \mid_{v_1,v_2,G}^{e} \lrcorner'':\widehat{=}'')$; $(A_{10} \mid_{x,V}^{v} \lrcorner'':\doteq'')$; $(A_{11} \mid_{c,n,L}^{L} \lrcorner''\div'')$; $(A_{12} \mid_{Q}^{x} \lrcorner''\#'')$; $(A_{13} \mid_{a,b}^{c} \lrcorner''+'')$; $(A_{14} \mid_{G_1,G_2}^{G} \lrcorner''\widetilde{\cup}'')$, $(A_{15} \mid_{Q_1,Q_2}^{Q} \lrcorner''\cup'')$.

Дана графова модель може бути використана для пошуку підрядка в рядку та порівняння текстів. Для прискорення цього процесу пропонується застосувати стиснення графа [12]. Воно полягає у заміні вершин графа, які поєднанні дугами з однаковими навантаженнями, на одну вершину. Навантаження цієї вершини визначається як результат конкатенації вмісту всіх розглянутих вершин.

Об'єктно-орієнтована модель графового представлення тексту. Для використання графів для співставлення текстів представимо останній як набір орієнтованих графів. Стартовими вершинами є унікальні символи, з яких починаються слова. Під словом розуміємо послідовність кириличних чи латинських символів або цифр.

Для програмної реалізації графової моделі виконаємо її об'єктно-орієнтоване (ОО) моделювання з використанням UML. Реалізацію графової моделі покладено на три класи: Work, Graph, Vertex (рис. 1).

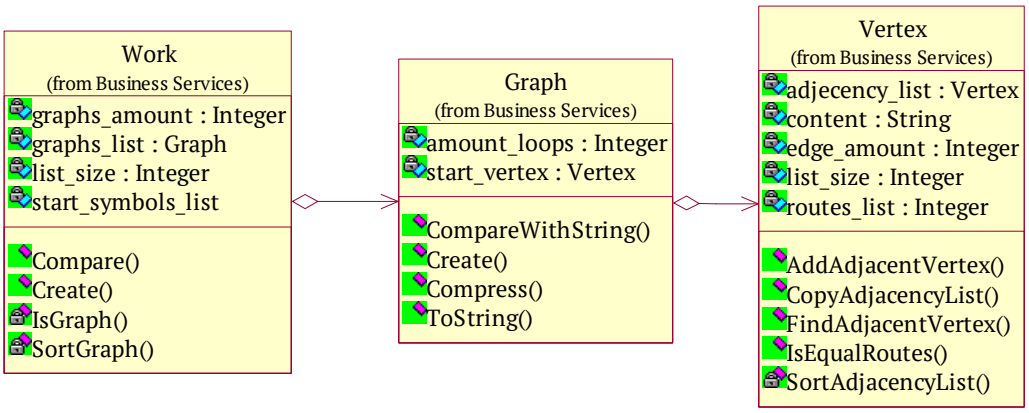

Рисунок 1 - Класове представлення графової моделі

На клас Work покладено відповідальність за створення набору графів, а також порівняння тексту (рядка) з текстом у вигляді набору графу. Відповідальність класу Graph – створення графу, що відтворює текст як сукупність підрядків, що починаються з однакового символу, та порівняння рядка (його частини) з текстом (або його частиною) у вигляді графу. Клас Vertex відповідає за створення вершини графу, яка навантажена одним або декількома символами, а також відтворення зв'язків між вершинами графу за допомогою списку суміжних вершин та забезпечення переходу між останніми при обході графу.

Програмна реалізація моделі тексту. Розглянемо відповідність алгоритмів моделі (11) методам класів (рис. 1).

A_1^0 – композиція алгоритмів – присутній у всіх методах, оскільки кожен з них складається з декількох підалгоритмів, таких як присвоєння, порівняння, об'єднання та інші.

A_2^0 – умовне виконання алгоритму – присутній у методах класу Vertex (FindAdjacentVertex, AddAdjacentVertex, SortAdjacencyList) та Graph (Create, CompareWithString, Compress), оскільки у програмній реалізації методів присутній оператор умовного виконання, що визначає використання тих чи інших алгоритмів.

A_3 – конкатенація – можна розглядати як додавання вершин у граф, що безпосередньо реалізовано методами Graph::Create та Vertex::AddAdjacentVertex.

$A_4 - A_6$ – реалізовано методом Graph::Create та Work::Create, що дозволяють послідовно побудувати графи.

A_7 – присвоєння операндів – присутнє у всіх методах.

A_8 – порівняння атрибутів – аналогічно застосуванню A_2^0, атрибутами є атрибути відповідних класів та вхідні параметри методів.

A_9, A_{10} – визначення дуги та знаходження вершини відповідно – необхідні для побудови графу, реалізовані у методі Vertex::FindAdjacentVertex.

$A_{11}|_{c,n,L}^{L}$ – виконання n операцій із списку L, якщо $c = true$, L – список з n операцій – реалізовано у методах Graph::Create, Vertex::AddAdjacentVertex.

A_{12} – обчислення потужності множини – реалізовано у вигляді атрибутів Work::graph_amount, Work::list_size, Graph::amount_loops, Vertex::edge_amount, Vertex::list_size, які змінюються методами Create відповідних класів.

A_{13} – додавання двох чисел – реалізовано у методах класів Work (Create, SortGraph, IsGraph, Compare), Graph (Create, CompareWithString) та Vertex (FindAdjacentVertex, AddAdjacentVertex, SortAdjacencyList).

A_{14}, A_5 – об'єднання графів та множин відповідно – у методах Create класів Work, Graph.

Правила конструктору графу, представлені у (5) – (10), реалізуються методом Graph::Create, який у свою чергу викликає методи класу Vertex: AddAdjacentVertex та FindAdjacentVertex, що разом забезпечують конструювання графу за текстом.

Стиснення графу виконується методом Graph::Compress за допомогою методів класу Vertex: IsEqualRoutes та CopyAdjacencyList.

Безпосереднє порівняння текстів виконує метод Graph::CompareWithString, який приймає на вхід два тексти: один – у вигляді графу (він вважається оригінальним), а інший – у вигляді рядку (він вважається копією, для якої визначається відсоток запозичень). На виході метод має: позиції запозиченого фрагменту та його вміст. Загальні результати накопичує метод Work::Compare і повертає як об'єкт класу Result, що містить усі запозичені фрагменти.

Для використання графової моделі побудована ОО-модель додатку. Вона включає 13 класів, що розподілені за рівнями відповідно до принципів трирівневої архітектури:

– рівень представлення: Form, ParamForm – форми інтерфейсу користувача, DataTransfer – клас, відповідальний за передачу даних між формами;

– рівень логіки: Controller, ControllerMN (відповідальність – передача даних між рівнями та керування роботою об'єктів інших класів), CheckParams

(збереження параметрів для відбору фрагментів), SourceText (збереження тексту), ProcessingText (попередня обробка тексту), Work, Graph, Vertex, Result;

– рівень даних: FileWorker, що відповідає за зчитування файлів, які будуть порівнюватися, та збереження результатів.

Текст представлено об'єктами класу SourceText, для його ініціалізації використовуються об'єкти та методи класу FileWorker, який утворює конструкцію тексту.

Для роботи з програмою розроблено інтерфейс користувача (рис. 2). Передбачено порівняння файлів один з одним та багато з багатьма. Програма виконує такі дії: зчитування тексту документу, попередня обробка тексту, побудова та співставлення текстів у відповідності до розроблених моделей і представлення результатів у вигляді загального відсотку запозичень та фрагментарного складу у форматі «фрагмент – відсоток», виділення запозичень кольором.

При порівнянні один з одним передбачено відбір (фільтрацію) результатів за такими параметрами, як мінімальна допустима довжина фрагменту (впливає на загальну оцінку) та максимальна допустима довжина розриву між фрагментами (не впливає на загальну оцінку, змінює якісний та кількісний склад фрагментів). Дана функція допомагає оцінити роботу зі зміни тексту: наявність коротких нечастих збігів може свідчити про їх випадковість; довгі та часті збіги свідчать скоріш про незначну переробку документу.

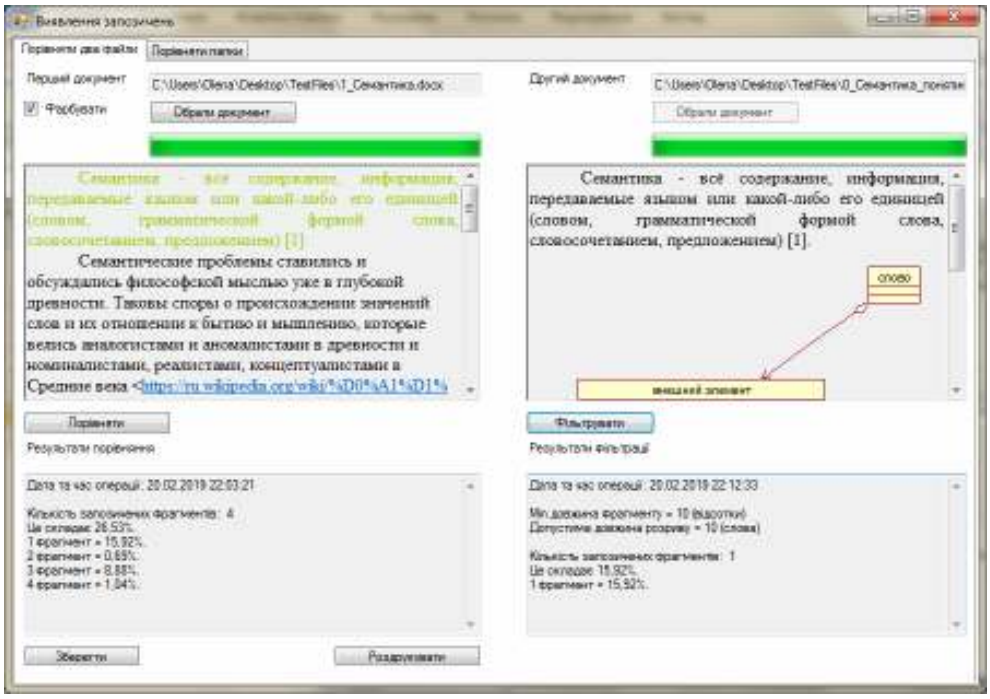

Рисунок 2 - Порівняння файлів один з одним

 ISSN 1562-9945

Результати розробки ПЗ на основі створених моделей. Для визначення часової характеристик алгоритмів побудови графу і порівняння текстових файлів виконано комп'ютерний експеримент. Вимірювався час для обробки 32 відібраних текстових файлів за тематикою «Розробка ПЗ» з Wikipedia (розміром від 16 – 24 Кб, від 2 –14 тис. символів) кожен з кожним. Кількість виконання операції порівняння – 512 разів. Експеримент виконано на ПК з такими характеристиками: процесор Intel Pentium(R) Dual Core CPU, кеш L1 коду/ L1 даних/ L2 – 2*32/2*32/1024 Кб, тактова частота/частота системної шини/частота пам'яті – 2,3 ГГц /400 МГц/400 МГц, час доступу до ОП (читання/запис) 5751/4253 Мб/с, операційна система – MS Windows 7 Ultimate SP1.

На основі отриманих даних побудовано графіки залежності часу виконання операцій побудови набору графів (у секундах) та порівняння тексту (мс) від його розміру (рис. 3, 4). На графіках визначено тренд, що є лінійною функцією.

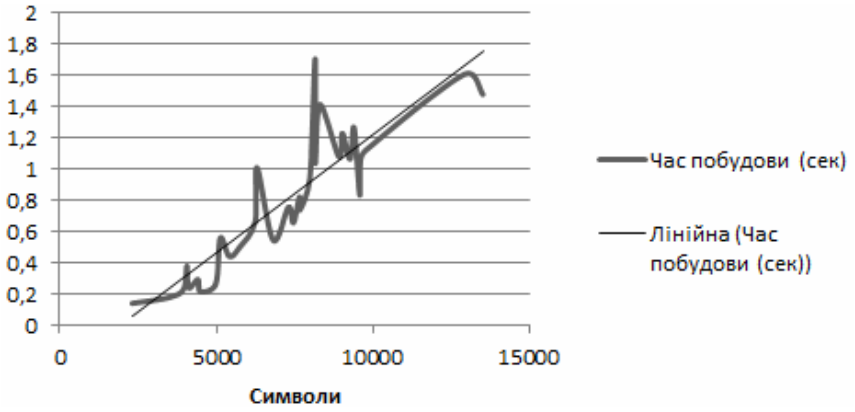

Рисунок 3 - Час виконання операції побудови набору графів

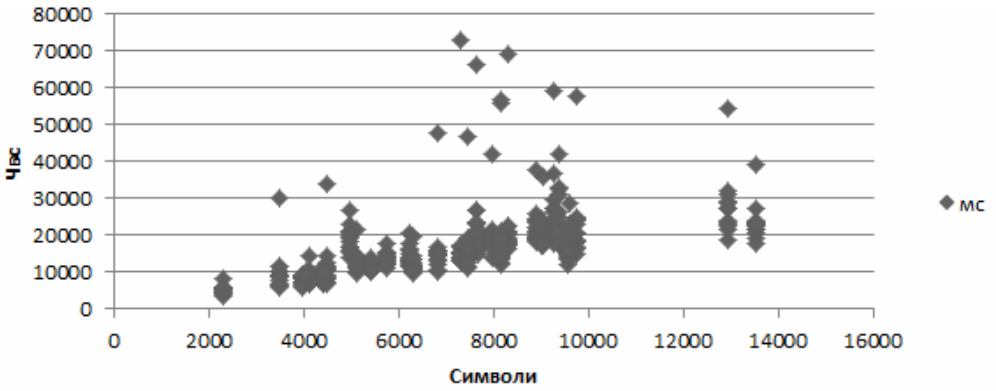

Рисунок 4 - Час виконання операції співставлення

Висновки. Модифікація конструктивної графової моделі тексту дозволила побудувати об'єктно-орієнтовану модель текстового документа як набору графів. Модель доведено до програмної реалізації, в рамках якої для зменшення впливу маскувань на виявлення запозичень було розроблено спеціальні методи зчитування та попередньої обробки даних.

Передбачається подальша робота з удосконалення моделі та ПЗ:

– тестування в умовах ЗВО з метою отримання оцінок часової ефективності та якості виявлення запозичень;

– врахування у моделі правомірних запозичень;

– лематизація тексту;

– покращення часових характеристик модулю зчитування даних з файлів.

ЛИТЕРАТУРА / ЛІТЕРАТУРА

1. Шинкаренко В. Формирование тестов для проверки способности демаскировки заимствований в программах выявления плагиата. / В. Шинкаренко, Е. Куропятник // Information Technologies & Knowledge. – 2018. – Vol. 12, No. 1. – P. 84 – 100.

2. Gulis I., Chudá D., Petrík J. Plagiarism Detection in Students' Assignments Written in Natural Language //International Conference on e-Learning. – 2016. – Т. 16. – P. 141.

3. Михайловский Ю. Б. Система Anti-Plagiarism як інструмент запобігання плагіату в навчальній та науковій діяльності / Ю. Б. Михайловский, Н. А. Длугунович. – Вісн. Хмельниц. націон. ун-ту. Технічні науки. – 2013. – № 3. – С. 162–168.

4. Xylogiannopoulos K. Text Mining for Plagiarism Detection: Multivariate Pattern Detection for Recognition of Text Similarities / . Xylogiannopoulos K, P. Karampelas, R. Alhajj // 2018 IEEE/ACM ASONAM, Barcelona. – 2018. – P. 938-945. – doi: 10.1109/ASONAM.2018.8508265

5. Lee E. Identifying text reuse using word net-based extended named entity recognition / E. Lee, P. Kim // Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems. – ACM, 2018. – P. 199 – 202.

6. Ho P.H. DNA Sequences Representation Derived from Discrete Wavelet Transformation for Text Similarity Recognition / P. H. Ho, N. A. T. Nguyen, T. H. Vo // Sieminski A., Kozierkiewicz A., Nunez M., Ha Q. (eds) Modern Approaches for Intelligent Information and Database Systems. Studies in Computational Intelligence. – Springer, Cham. – 2018. – Vol 769. – P. 75 – 85.

7. Osman A. H. Conceptual Similarity and graph-based method for plagiarism detection / A. H. Osman, N. Salim, M. S. Binwahlan, H. Hentably, A. M. Ali // Journal of Theoretical and Applied Information Technology. – 2011.–Vol. 32, No.2. – P. 135 – 145.

8. kumar Jayapal A. Similarity Overlap Metric and Greedy String Tiling at PAN 2012: Plagiarism Detection // Conference: CLEF. – At http://www.clef-initiative.eu/documents/71612/da184f72-1a8e-43a8-80e4-dd1b2b6fdb09

9. Зиберт А. О. Разработка системы определения наличия заимствований в работах студентов высших учебных заведений. Алгоритмы поиска нечетких дублика-

тов / А. О. Зиберт, В. В. Хрусталев // Universum: технические науки. – 2014. – №. 3 (4). – Режим доступа: http://7universum.com/ru/tech/archive/item/1139
10. Meuschke N. State-of-the-art in detecting academic plagiarism / N. Meuschke, B. Gipp // International Journal for Educational Integrity. – 2013. – Vol. 9 No. 1. – pp. 50–71.
11. Vani K. Detection of idea plagiarism using syntax–semantic concept extractions with genetic algorithm / K. Vani K., D. Gupta //Expert Systems with Applications. – 2017. – Vol. 73. – P. 11-26.
12. Shynkarenko V. Constructive-synthesizing model of text graph representation // V. Shynkarenko, O. Kuropiatnyk – CEUR Workshop Proceedings. – 2016. – Vol. 1631. – P. 63 – 72.
13. Shynkarenko V. I. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure / V. I. Shynkarenko, V. M. Ilman. // Cybernetics and Systems Analysis. – 2014. – Vol. 50. – Issue 5. – P. 655-662.

REFERENCES

1. Shynkarenkko V. Creation tests for checking plagiarism detection programs' ability of unmasking borrowings / V. Shynkarenko, O. Kutopiatnyk // Information Technologies & Knowledge. – 2018. – Vol. 12, No. 1. – P. 84 – 100.
2. Gulis I., Chudá D., Petrík J. Plagiarism Detection in Students' Assignments Written in Natural Language //International Conference on e-Learning. – 2016. – T. 16. – P. 141.
3. Mykhailovskyi Yu. B. Anti-Plagiarism System as a Tool to Prevent Plagiarism in Educational and Scientific Activities / Yu. B. Mykhailovskyi, H. A. Длугунович. – Bulletin of Khmelnytsky National University. Technical sciences. – 2013. – № 3. – С. 162–168.
4. Xylogiannopoulos K. Text Mining for Plagiarism Detection: Multivariate Pattern Detection for Recognition of Text Similarities / . Xylogiannopoulos K, P. Karampelas, R. Alhajj // 2018 IEEE/ACM ASONAM, Barcelona. – 2018. – P. 938-945. – doi: 10.1109/ASONAM.2018.8508265
5. Lee E. Identifying text reuse using word net-based extended named entity recognition / E. Lee, P. Kim // Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems. – ACM, 2018. – P. 199 – 202.
6. Ho P.H. DNA Sequences Representation Derived from Discrete Wavelet Transformation for Text Similarity Recognition / P. H. Ho, N. A. T. Nguyen, T. H. Vo // Sieminski A., Kozierkiewicz A., Nunez M., Ha Q. (eds) Modern Approaches for Intelligent Information and Database Systems. Studies in Computational Intelligence. – Springer, Cham. – 2018. – Vol 769. – P. 75 – 85.
7. Osman A. H. Conceptual Similarity and graph-based method for plagiarism detection / A. H. Osman, N. Salim, M. S. Binwahlan, H. Hentably, A. M. Ali // Journal of Theoretical and Applied Information Technology.–2011. –Vol. 32, No. 2. – P. 135 – 145.
8. kumar Jayapal A. Similarity Overlap Metric and Greedy String Tiling at PAN 2012: Plagiarism Detection // Conference: CLEF. – At http://www.clef-initiative.eu/documents/71612/da184f72-1a8e-43a8-80e4-dd1b2b6fdb09

9. Zibert A. O. Development of a system for determining the existence of adoption in the works of the students. The search algorithms of indistinct duplicates / A. O. Zibert, V. B. Hrustalev // Universum: технические науки. – 2014. – №. 3 (4). – URL: http://7universum.com/ru/tech/archive/item/1139
10. Meuschke N. State-of-the-art in detecting academic plagiarism / N. Meuschke, B. Gipp // International Journal for Educational Integrity. – 2013. – Vol. 9 No. 1. – pp. 50–71.
11. Vani K. Detection of idea plagiarism using syntax–semantic concept extractions with genetic algorithm / K. Vani K., D. Gupta //Expert Systems with Applications. – 2017. – Vol. 73. – P. 11-26.
12. Shynkarenko V. Constructive-synthesizing model of text graph representation // V. Shynkarenko, O. Kuropiatnyk – CEUR Workshop Proceedings. – 2016. – Vol. 1631. – P. 63 – 72.
13. Shynkarenko V. I. Constructive-Synthesizing Structures and Their Grammatical Interpretations. I. Generalized Formal Constructive-Synthesizing Structure / V. I. Shynkarenko, V. M. Ilman. // Cybernetics and Systems Analysis. – 2014. – Vol. 50. – Issue 5. – P. 655-662.

Received 19.02.2019.
Accepted 25.02.2019.

Конструктивное и объектно-ориентированное моделирование текстов для обнаружения заимствований

Рассмотрены некоторые алгоритмы и структуры данных, которые могут быть использованы при решении задачи обнаружения заимствований. Представлена модификация конструктивной графовой модели текста. На ее основе построена объектно-ориентированная модель для сопоставления текстов. Модель доведено до программной реализации. Установлено соответствие между элементами моделей и программными компонентами. Получены временные показатели работы программы.

Constructive and object-oriented modeling text for detection of text borrowings

The scientific community is encouraged to use such models and data structures as arrays of LERP-RSA (the longest expected duplicate array of reduced suffix templates), tag classifier-a model based on Stanford NER's three-class, structures based on DN-sequences, graph representations, etc. The following algorithms are used: GreedyString-Tiling, ARPAD, shingle, statistical methods, genetic algorithms, and others. It should also be noted that much attention is paid to morphological analysis and lemmatization, pre-processing of texts. Models and algorithms only partly have program realization.

The purpose of this work is to develop a text model to identify borrowings and bring it to program implementation. The task is to develop the object-oriented model and program implementation of a graph text model, with the application of the problem of detection of borrowing. As well as obtaining timeframes for program implementation work for further evaluation of the possibility of its use in the academic environment.

The main idea of the graph model is to present the text as a weighted oriented graph. The vertex weight is a character or sequence of characters. Edge weight is the set of numbers of paths into which the edge enters. To formalize the model will use the apparatus of constructive-synthesizing modeling. To create graphs, a constructor and its components are defined: carrier, signature, multiple statements of information support for design. Transformations are made for the constructor: specialization, interpretation and concretization.

On the basis of this model, the object-oriented model is constructed. it includes three classes: vertex, graph and work .

The object of class Work presents the text as a set of objects of class Graph. The correspondences between the components of the presented models are established.

The object-oriented model is implemented by software. Data are given about the execution time of graph construction and texts comparison.

At this stage, software implementation of the model has shown acceptable time performance. Further research in this direction is promising. Directions for improving the model and program are proposed.

Куропятник Е.С. - ассистент кафедры компьютерных информационных технологий, Днепровский национальный университет железнодорожного транспорта им. акад. В. Лазаряна.

Куроп'ятник О.С. - асистент кафедри комп'ютерних інформаційних технологій, Дніпровський національний університет залізничного транспорту ім. акад. В. Лазаря на.

Kuropiatnyk O. - Lecturer of Computer Information Technologies Department, Dnipro National University of Railway Transport named after academician V. Lazaryan.