УДК 004.43
DOI https://doi.org/10.32782/2521-6643-2023.1-65.7

Павлов В. Г., кандидат технічних наук, доцент,
доцент кафедри обчислювальної техніки
Національного технічного університету України
«Київський політехнічний інститут імені Ігоря Сікорського»
ORCID: 0000-0002-4299-0319

ЛЕКСИЧНА ЗГОРТКА ПРИ АНАЛІЗІ СХОЖОСТІ ТЕКСТІВ ПРОГРАМ

Стаття присвячена вирішенню проблеми захисту авторського права на тексти комп'ютерних програм. Хоча на законодавчому рівні вихідні та об'єктні коди комп'ютерних програм визнані такими, що підлягають захисту та на які розповсюджується авторське право, практична реалізація цього не є досконалою. Причина, полягає у тому, що історично спочатку постала проблема захисту авторства на літературні тексти, а потім цей підхід поширився й на тексти комп'ютерних програм. При цьому програмні коди розглядаються лише як різновид літературних текстів, тому для аналізу їх схожості пропонуються ті ж методики, які застосовуються до літературних текстів. Вони не враховують особливості текстів комп'ютерних програм, насамперед граматичні правила побудови програмних кодів. На відміну від граматики літературних текстів, синтаксис мов програмування побудований на більш жорстких правилах, які мають формалізований вигляд та описуються за допомогою метамов. Тому будь який оператор чи інструкція має у своєму складі стали вирази, які при компіляції комп'ютерної програми розглядаються як стандартні токени певної мови програмування. Їх назви та розташування не можуть бути довільними, а тому вони визначають як би лексичний каркас програми. Але під час створення програмного коду його автор має можливість вільно використовувати власні назви для певних складових комп'ютерної програми – назв змінних, міток, розроблених функцій тощо. Ці назви відносяться до користувацьких токенів та при компіляції не розглядаються як стали складові команд. Вони можуть бути легко замінені у вихідному коді без будь яких змін у послідовності стандартних токенів. Таке «клонування» програмного коду з боку недоброчесних користувачів часто залишається непомітним, бо програмні засоби для знаходження схожості текстів дають значно занижений результат, оскільки не розрізняють стандартні та користувацькі токени у текстах, які порівнюються. Той же невірний підхід до текстів комп'ютерних програм може також надати завищену оцінку при порівнянні через ті ж недоліки. Це доводиться на прикладах, які наведені у статті.

У статті запропонований підхід при якому у текстах комп'ютерних програм стандартні токени відокремлюються від користувацьких, внаслідок чого останні мають значно менший вплив на результат перевірки схожості текстів. Це перетворення, яке назване лексичною згорткою, продемонстровано на прикладі основних конструкцій мови програмування С та фрагменту програмного коду. Цей підхід може бути поширений на інші мови програмування.

Ключові слова: *авторське право, схожість програмного коду, лексична згортка, токен.*

Pavlov V. G. Lexical convolution in analyzing the similarity of program texts

The article is devoted to solving the problem of copyright protection on texts of computer programs. Although at the legislative level, the source and object codes of computer programs are recognized as subject to protection and to copyright, the practical implementation of this is not perfect. The reason is that, historically, the problem of protecting authorship of literary texts arose first, and then this approach spread to the texts of computer programs. In this case, program codes are considered only as a kind of literary texts, therefore, for the analysis of their similarity, the same techniques are proposed that apply to literary texts. They do not take into account the peculiarities of the texts of computer programs, especially the grammatical rules for constructing program codes. Unlike the grammar of literary texts, the syntax of programming languages is built on stricter rules, which have a formalized form and are described using metalanguages. Therefore, any operator or instruction has in its composition the constant expressions, which, when compiling a computer program, are considered as standard tokens of a particular programming language. Their names and locations cannot be arbitrary, and therefore they define, as it were, the lexical skeleton of the program. But when creating program code, its author has the opportunity to freely use proper names for certain components of a computer program – variable names, labels, developed functions, etc. These names refer to user tokens and are not considered as permanent command components when compiled. They can be easily exchanged in the source code without any change in the sequence of standard tokens. Such "cloning" of program code by dishonest users often remains invisible, because software tools for finding the similarity of texts give a significantly underestimated result, since they do not distinguish between standard and user tokens in the texts being compared. The same wrong approach to the texts of computer programs can also provide an overestimation when compared due to the same disadvantages. This is proved by the examples given in the article.

The article proposes an approach in which standard tokens are separated from user tokens in the texts of computer programs, As a result, the latter have much less influence on the result of checking the similarity of texts. This transformation, which is called lexical convolution, is demonstrated on the instance of the basic constructions of the C programming language and a fragment of the software code. At the same time, it is possible to expand on the other program languages.

Key words: *copyright, similarity of program code, lexical convolution, token.*

Постановка проблеми. Впровадження інформаційних технологій майже у всі сфери людської діяльності потребує розробки величезної кількості комп'ютерних програм. У більшості країн світу розробка та використання комп'ютерних програм законодавче регулюється авторським правом, порушення якого відноситься до найбільш тяжких злочинів та суворо карається. Світова практика такого підходу втілюється у законодавстві України, де визначається, що охорона авторського права «поширюється на комп'ютерні програми, виражені у вихідному або об'єктному кодах, якщо вони є оригінальними» [1, стаття 20]. З цього випливає, що об'єктами захисту авторського права є як сирцеві програми, написані на мові програмування, так й їх скомпільовані варіанти. Таким чином постає питання доведення факту порушення авторського права на тексти комп'ютерних програм шляхом виявлення їх схожості.

Аналіз останніх досліджень і публікацій. На сьогодні існує досить багато методик перевірки текстових документів, які добре відомі та знайшли втілення у багатьох алгоритмах та програмних продуктах. Наприклад, у оглядах [2–5], наданий опис таких відомих програмних продуктів, як Unicheck, Strike-Plagiarism, Plagiarisma, Edu-Birde, Advego Plagiatus, Content-watch, Copyscape, Copywritely, Etxt Antiplagiat, Like-Exactus, Quetext, Plag, Plagiarism Detector та ін. Але усі вони пропонуються для порівняння літературних текстових документів, а ніяк для текстів комп'ютерних програм.

Вони ґрунтуються або на пошуку входження одного рядку у інший (алгоритм Бойєра – Мура у різних модифікаціях [6, с. 762–772], алгоритм Кнута – Морріса – Пратта [7, с. 323–350] та ін.), або на застосуванні блоків тексту у вигляді «шинглів» (shingles), що було запропоновано А. Бродером у 1997 році [8, с. 28]. Усі ці методи демонструють досить високу ефективність та широко застосовуються на практиці, але вони розглядають текст комп'ютерної програми як звичайний літературний текст. Такий підхід, на жаль, склався історично, бо літературні твори з'явилися в якості об'єкту авторського права значно раніше ніж комп'ютерні програми, тому на них автоматично були поширені ті ж методи доведення схожості, що й до звичайних текстів. Але, як доведено автором [9], цій підхід не може бути застосований безпосередньо до текстів комп'ютерних програм, оскільки синтаксичні правила їх побудови суттєво відрізняються від синтаксичних правил літературних текстів. Причина цього у відмінностях правил, за якими будуються граматики комп'ютерних мов програмування високого рівня, та граматиками літературних мов. Тому якщо до текстів комп'ютерних програм застосовуються ті ж методики виявлення схожості, що й для звичайних текстів, то вони дають спотворені результати, оскільки не враховують специфіки побудови їх синтаксису.

Метою статті є застосування синтаксичних особливостей мов програмування на прикладі мови С та розгляд їх використання у визначенні схожості текстів програм. Для досягнення означеної мети вирішуються наступні завдання:

- аналіз особливостей синтаксису мов програмування;
- відтворення цих особливостей у вигляді лексичної згортки.

Виклад основного матеріалу дослідження. Згадаємо, яка послідовність вивчення будь-якої мови спілкування:

абетка мови, як певна сукупність знаків, що використовуються у даній мові;

словник мови, як множина поєднань цих знаків у певні групи – слова;

правила поєднання слів у речення та фрази – синтаксис мови.

У мов програмування перші два етапі схожі, а третій відрізняється тим, що опис правил відбувається за допомогою метамови, яка схожа на мову формул та визначає сувору послідовність складових у виразі команди. Метамова для опису мов програмування була запропонована Д. Бєкусом та П. Науром та після декількох удосконалень знайшла своє втілення у стандарті [10], який є універсальним та застосовується при опису граматик більшості мов програмування. Ці мови відносяться до так званих формальних мов, через те, що будь яке відхилення від правил граматики вважається синтаксичною помилкою та підлягає виправленню. Тому кількість сполучень, які застосовуються у текстах комп'ютерних програм, завжди є обмеженою множиною, яка повинна бути розпізнана при компіляції цих програм.

Через це певні сполучення можуть багаторазово зустрічатися як у межах тексту однієї комп'ютерної програми, так й у текстах різних комп'ютерних програм. Якщо вважати повторення цих сполучень ознаками схожості текстів, то це буде невірно, бо ці повторення обумовлені особливостями побудови текстів комп'ютерних програм за певними алгоритмами, які реалізуються за допомогою певних команд даної мови. Тому, якщо ці команди будуються виключно зі сталих конструкцій комп'ютерної мови, то вони, природно, будуть мати **однаковий текстовий зміст**, бо відхилення від послідовності цих конструкції за правилами граматики **неможливо**, бо будете розглядатися як синтаксична помилка.

Це – **перша** головна відмінність текстів комп'ютерних програм від звичайних літературних текстів, у яких кількість варіантів сполучень слів у реченнях може буди набагато більшою, тому їх повторення у текстах, які порівнюються, можна вважати ознакою схожості. Оскільки програми порівняння текстів не враховують цю особливість текстів комп'ютерних програм, вони будуть вважати «схожістю» кожну однакову послідовність або команду, яка міститься у текстах, що порівнюються. Через це оцінка схожості текстів, яка у більшості методик базується на підрахунку відсотка фрагментів текстів, що співпали, по відношенню до усього тексту, буде **завищеною**.

Наведемо приклад порівняння двох фрагментів текстів комп'ютерних програм, написаних на мові С. Цей вибір пояснюється тим, що дана мова програмування є базовою для багатьох інших мов програмування та містить багато схожих команд.

Нехай треба обчислити суму $S = \sum_{i=1}^{10} \frac{1}{i^2}$. У обох фрагментах для цього використовується цикл, у якому змінна ***i*** послідовно змінюється від 1 до 10. Але у першому фрагменті для цього побудований цикл з передумовою:

```
i=1;
s=0;
while (i<=10)
 {
   s=s+1/(i*i);
   i=i+1;
 }
```

А у другому фрагменті обрахування виконується у циклі де умова після:

```
i=1;
 s=0;
do
 {
   s=s+1/(i*i);
   i=i+1;
 }
while (i<=10);
```

З точки зору порівняння текстів, як комп'ютерних програм, обидва фрагменти не мають схожості, бо мають ***різні конструкції*** та використовують ***різні за синтаксичною побудовою цикли***. Але якщо порівнювати їх як звичайні літературні тексти, то вони мають ***однакові рядки***:

```
i=1;
s=0;
 {
   s=s+1/(i*i);
   i=i+1;
 }
while (i<=10);
```

Тому програмні засоби, які порівнюють ці фрагменти, як звичайний текст, надають результат від **46%** до **86%** схожості [9], що не відповідає дійсності.

З іншого боку у всіх мовах програмування присутні складові конструкцій команд, які не є сталими. Ці складові позначають змінні, мітки, функції, які створив автор програмного коду та яким надав назви за власним розсудом. Граматика мов програмування лише надає певні обмеження, але в цілому ці назви є результатом суто власної фантазії автора програми. Якщо змінити лише назви змінних, але не змінювати жодної команди та їх послідовності, то програма по структурі та по синтаксичній побудові залишиться той же само, але з точки зору порівняння символів тексту, вона буде мати відмінності, які зафіксують програмні засоби, які розглядають текст, як літературний, тобто в цьому разі маємо **занижену** оцінку схожості.

В якості приклада наведемо функцію обрахування ступені числа:

```
power(base, n)
int base, n;
  {
int i, p;
p = 1;
for (i = 1; i <= n; ++i)
p = p * base;
return p;
  }
```

У цьому фрагменті програмного коду застосовуються назви самої функції та змінних: ***power, base, n, i, p***. Якщо їх змінити на інші, то це не вплине ні на послідовності команд, ні на структуру коду, тобто він залишиться незмінним, тобто має повну схожість з точки зору програмного коду. Але при порівняння за допомогою різних програмних засобів порівняння літературних текстів цього фрагменту з іншим, де усього лише змінені назви змінних, а саме:

```
stupen(number, k)
int number, k;
```

```
{
int j, s;
s = 1;
for (j = 1; j <= k; ++j)
s = s * number;
return s;
}
```

отримані оцінки схожості набагато менше 100%:

0% – **Copyleaks** (https://app.copyleaks.com/text-compare);

50.72% – **Anytexteditor** (https://anytexteditor.com/text-compare);

87% – **Smodin** (https://smodin.io/uk/;

тобто знову результат порівняння буде ***невірний***.

Таким чином, **друга** головна відмінність текстів комп'ютерних програм від звичайних літературних текстів у тому, що заміна назв змінних та функцій не є суттєвою для порівняння схожості текстів комп'ютерних програм, але у той же час вважається достатньою для виявлення відмінностей у разі, якщо тексти порівнюються як звичайні літературні. Якщо не враховувати цю особливість текстів комп'ютерних програм, то оцінка схожості у разі застосування по відношенню до них тих же методів, що і для літературних текстів буде **заниженою**.

Зазначимо, що через синтаксичні особливості мов програмування, жодний програмний продукт для порівняння літературних текстів на схожість не надає правдивої інформації при порівнянні текстів комп'ютерних програм, а тому програми порівняння текстів не підходять для програмних кодів.

У роботі [9] зазначено, що насамперед причина цього у тому, що у цих програмних засобах увесь текст при порівнянні розглядається однорідним, у той час, як при порівнянні текстів комп'ютерних програм повинні розрізнятися стандартні та користувацькі токени. Перші можуть зустрічатися у текстах комп'ютерних програм багаторазово, тому тільки їх наявність не може бути ознакою схожості. При цьому послідовність та взаємне розташування стандартних токенів може виступати, як така ознака. Схожість користувацьких токенів, навпаки, може бути суттєвою ознакою схожості текстів комп'ютерних програм. Тому спочатку у тексті сирцевої комп'ютерної програми повинні бути знайдені та позначені токени першого та другого типу, тобто виконана **лексична згортка**.

Розглянемо побудову цієї згортки на прикладі деяких конструкції мови С [11, с. 35–92].

Змінні. Для того, щоб при побудові згортки уникнути «прив'язки» до користувацьких токенів введемо для них позначення великою літерою "**V**", за якою буде йти цифра, яка буде позначати тип змінної, а саме:

1 – змінна розміром 1 байт;

2 – змінна розміром 2 байти;

4 – змінна розміром 4 байти;

8 – змінна розміром 8 байтів.

Константи. У граматичної згортці буде позначатися великою літерою "**K**", цифра за якою буде позначати тип константи:

1 – ціла число;

2 – число з дробовою частиною;

3 – символьна константа;

4 – строкова константа;

5 – константа перерахування.

Декларація змінних. Позначає тип змінних перед їх використанням у програмі [11, с. 36–37]. При декларації використовуються наступні стандартні токени: **char**, **double**, **float**, **int, short, long, signed, unsigned, const** за якими за синтаксичними правилами йдуть назви змінних (користувацькі токени). Завершується будь-яка операція у мові С стандартним токеном «**;**». За підсумками згортання отримаємо:

Таблиця 1

Лексичне згортання операторів декларування змінних

Оригінальний оператор	Лексична згортка оператора
int **alpha, beta, gamma**;	int **V1, V1, V1**;
char **lambda**;	char **V1**;
float **delta** = **1.0e-5**;	float **V4** = **K2**;
int **teta** = **length**+**3**;	int **V1** = **V1**+**K1**;
const char **row** [ ] = **"example"**;	const char **V1** [ ] = **K4**;

Арифметичні та логічні оператори (бінарні та унарні), оператори відношення, дужки [11, с. 41–42]. У наведених вище прикладах декларації змінних вже використовувалися деякі арифметичні оператори та дужки. Усі ці токени є стандартними, тому у виразах залишаються незмінними.

Інкрементні та декрементні оператори [11 с. 46]. Як префіксні, та й постфіксні варіанти цих операторів відносяться до множини стандартних токенів, тому залишаються незмінними, однак оскільки вони додаються до змінної, яка має ціле значення, то вона є користувацьким токеном. Тому, наприклад вирази ++**n** та **k**-- після виконання лексичного згортання будуть відповідно перетворені у ++**V1** та **V1**--, якщо змінні **n** та **k** – однобайтні.

Конструкція IF-ELSE [11, с. 55]. Має синтаксичну побудову вигляду:

if (вираз)
  інструкція$_1$
else
  інструкція$_2$

Токени **if** та **else** відносяться до стандартних, а у складі виразу та інструкцій присутні як стандартні, так й користувацькі токени.

Конструкція ELSE-IF [11, с. 57]. Має вигляд:

if (вираз)
  інструкція$_1$
else if (вираз)
  інструкція$_2$
else if (вираз)
  інструкція$_3$
  . . .
else
  інструкція$_n$

Її синтаксичний вигляд схожий з попередньо та є її розвитком, тому лексична згортка будується аналогічно попередній.

Перемикач [11, с. 58]. Синтаксичний вигляд наступний:

switch (вираз) {
  case константа або вираз: інструкції
  case константа або вираз: інструкції
    . . .
 default: інструкції
}

У цій конструкції присутні стандартні токени **switch**, **case**, **default**, **{**, **}**. Константа є користувацьким токеном, а у складі виразів та інструкції можуть бути присутні токени обох типів.

Цикли WHILE та FOR [11, с. 60]. Мають відповідні синтаксичні конструкції:

while (вираз)
інструкція
та
for (вираз$_1$; вираз$_2$; вираз$_3$)
  інструкція

У цих конструкціях присутні стандартні токени **while** та **for**. У складі виразів та інструкцій можуть бути присутні, як стандартні, так й користувацькі токени.

Цикл DO-WHILE [11, с. 63]. Має схожу конструкцію з попереднім:

do
  інструкція
while (вираз);

У складі присутні стандартні токени **do** та **for**, а у складі інструкції та виразу токени обох типів.

Зазначимо, що всі конструкції циклів, а також перемикача можуть бути доповнені операторами **break** та **continue** [11, с. 64 – 65], які є стандартними токенами. Також у випадку, коли у складі конструкцій з умовами або циклів замість однієї інструкції використовуються декілька, вони обмежуються операторними дужками {…}, які також є стандартними токенами.

Мітки. Мітки позначають певні місця у програмі та мають назву, схожу на назви змінних, за якою йде двокрапка [11, с. 66]. Назва мітки відноситься до користувацьких токенів, тому введемо для них позначення великою літерою L. Двокрапка залишиться незмінною, бо вона – стандартний токен. Для пересування до певної мітки використовується команда **goto** за якою вказується назва мітки. Команда відноситься до стандартних токенів, а мітка – до користувацьких.

Розглянуті оператори та конструкції складають лише підмножину усіх можливих сполучень у мові С, але цього цілком достатньо, щоб продемонструвати на прикладі процес побудови лексичної згортки.

Нехай маємо наступний фрагмент програми на мові С, у якої рахується число Фибоначчі за номером N:

```
int Fun(int N) {
   int A = 1;
   int B = 1;
   int C = 0;
   if (N <= 2)
     return 1;
   else
   {
      for (int J = 3; J <= N; J++)
      {
         C = A + B;
         A = B;
         B = A;
      }
   }
   return A;
}
```

У цьому фрагменті присутні цілі змінні **A, D, C, N, J** та назва функції **Fun** Усі вони належать до користувацьких токенів, а саме **V1**. Також присутні константи у вигляді цілих чисел – користувацькі токени **K1**. Решта – стандартні токени мови С. Лексична згортка цього фрагменту має вигляд:

```
int V1(int V1) {
   int V1 = K1;
   int V1 = K1;
   int V1 = K1;
   if (V1 <= K1)
     return K1;
   else
   {
      for (int V1 = K1; V1 <= V1; V1++)
      {
         V1 = V1 + V1;
         V1 = V1;
         V1 = V1;
      }
   }
   return V1;
}
```

Зазначимо, що цю згортку можна надати у вигляді рядка, оскільки у даному разі зберігається послідовність та відносне розташування токенів.

int V1(int V1) { int V1 =K1; int V1 = K1; int V1 = K1; if (V1 <= K1) return K1; else {for (int V1 = K1; V1 <= V1; V1++) {V1 = V1 + V1;V1 = V1;V1 = V1; }} returnV1;}

У такому вигляді він більш компактний для розгляду, хоча як текст він абсолютно тотожний. Тому, якщо другий текст, схожість з яким перевіряється, також буде приведений до своєї лексичної згортки, то їх порівняння зводиться до тривіальної задачі, а саме пошуку входження одного фрагменту у інший. Тобто, якщо буде використана саме така послідовність операторів мови програмування, то схожість буде знайдена, навіть якщо назви змінних будуть іншими. Звичайно, для прикладу вибраний короткий фрагмент коду, де лише демонструються принципи лексичної згортки. Якщо фрагмент буде довшим, то алгоритм порівняння буде більш складним, але це буде розглянуто окремо у майбутньому дослідженні.

Висновки. Аналіз підходів до оцінювання схожості текстів комп'ютерних програм та результати експериментальних досліджень дають підстави для наступних висновків:

Усі програмні засоби для аналізу схожості текстів демонструють спотворений результат, оскільки призначені для порівняння літературних текстів та не враховують синтаксичних особливостей текстів комп'ютерних програм.

Запропонована методика отримання лексичної згортки текстів комп'ютерних програм, з якої штучно прибираються користувацькі токени, але залишаються стандартні токени.

Продемонстровано формування лексичної згортки на прикладі основних конструкцій мови програмування С та фрагменту програмного коду. Цей підхід може бути поширений на інші мови програмування.

Список використаних джерел:

1. Про авторське право і суміжні права: Закон України від 23.12.93 № 3793-XII. *Відомості Верховної Ради України*, 1994. № 13. Ст. 64. Дата оновлення: 20.03.2023. URL: https://zakon.rada.gov.ua/laws/show/2811-20#n855 (дата звернення 15.04.2023).
2. Онлайн-платформи та програми для перевірки тексту на плагіат. URL: https://osvita.ua/vnz/76907/ (дата звернення 17.04.2023).
3. Антиплагіатні системи, перевірка на плагіат. URL: http://library.chnu.edu.ua /index.php?page=ua/07services/06acad_int/02anti_plag_sys (дата звернення 17.04.2023).
4. 6 сервісів перевірки унікальності для українських копірайтерів. URL: https://wordfactory.ua/plagiarism-checker/ (дата звернення 17.04.2023).
5. Інструменти перевірки текстів на плагіат. URL: https://lib.zsmu.edu.ua/p_82.html
6. (дата звернення 17.04.2023).
7. Boyer R. S., Moore J. S. A fast string searching algorithm. *Communication of the ACM*. 1977. V. 20. № 10. P. 762–772.
8. Knuth D. E., Morris J. H., Jr., Pratt V. R. Fast pattern matching in strings. *SIAM Journal on Computing*. 1977. V. 6. № 2. P. 323–350. DOI: https://doi.org/10.1137/0206024.
9. Broder A. Z. On the resemblance and containment of documents. *Proceedings. Compression and Complexity of SEQUENCES 1997 (Salerno, Italy13-13 June 1997)*. IEEE Computer Society, 1998. P. 21–29. DOI: https://doi.org/ 10.1109/SEQUEN.1997.666900.
10. Павлов В. Г. Контекстний підхід у аналізі схожості текстів програм. *Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки.* 2023. Том 34 (73). № 2.
11. ISO/IEC 14977:1996 Information technology – Syntactic metalanguage – Extended BNF, New York : American National Standards Institute, 1996. 10 p.
12. Kernighan B. W., Ritchie D.M. The C Programming Language / Copyright 1978, Second Edition, Ney Jersey, Prentice-Hall, 1988. 272 p.

References:

1. Pro avtorske pravo i sumizhni prava [*On copyright and related rights]* (Law of Ukraine) № 3793-XII. (1993). URL: https://zakon.rada.gov.ua/laws/show/2811-20#n855.
2. Onlain-platformy ta prohramy dlia perevirky tekstu na plahiat [*Online platforms and programs for checking text for plagiarism]*. URL: https://osvita.ua/vnz/76907/.
3. Antyplahiatni systemy, perevirka na plahiat [*Anti-plagiarism systems, plagiarism check]*. URL: http://library.chnu.edu.ua /index.php?page= ua/07services/06acad_int/02anti_plag_sys.
4. 6 servisiv perevirky unikalnosti dlia ukrainskykh kopiraiteriv [*6 uniqueness verification services for Ukrainian copywriters]*. URL: https://wordfactory.ua/plagiarism-checker/.
5. Instrumenty perevirky tekstiv na plahiat [*Tools for checking texts for plagiarism]*. URL: https://lib.zsmu.edu.ua/p_82.html.
6. Boyer R. S., Moore J. S. (1977). A fast string searching algorithm. *Communication of the ACM*, (V. 20. No 10), P. 762–772.
7. Knuth D. E., Morris J. H., Jr., Pratt V. R. (1977). Fast pattern matching in strings. *SIAM Journal on Computing,* (V. 6. No 2), P. 323–350. https://doi.org/10.1137/0206024.
8. Broder A. Z. (1997). On the resemblance and containment of documents. *Proceedings. Compression and Complexity of SEQUENCES 1997*. 13–13 June 1997, Salerno,Italy. IEEE Computer Society, 1998. P. 21–29. DOI: https://doi.org/10.1109/SEQUEN.1997.666900.
9. Pavlov V. G. (2023). Kontekstnyi pidkhid u analizi skhozhosti tekstiv prohram [Contextual approach in analyzing the similarity of program codes]. *Scientific Notes of Taurida V.I. Vernadsky University. Series: Technical sciences.* (Volume 34 (73) No 2).
10. *Syntactic metalanguage – Extended BNF: ISO/IEC 14977:1996 Information technology* (1996), New York: American National Standards Institute.
11. Kernighan B. W., Ritchie D.M. (1988). *The C Programming Language*. (272 p.) Ney Jersey: Prentice-Hall.