3. Креневич А. П. Алгоритми і структури даних: підручник. Київ: ВПЦ «Київський університет», 2021. 200 с.

4. Goldberg A. V., Tarjan R. E. A new approach to the maximum-flow problem. *Journal of the ACM*. 1988. Vol. 35, № 4. P. 921–940.

**УДК 004.422.5**

*Клименко Ю. Ю., здобувач 2 курсу спеціальності 122 Комп'ютерні науки,*
*Потапова Н. А., канд. екон. наук, доцент, доцент кафедри інформаційних технологій*

## АЛГОРИТМІЧНІ АСПЕКТИ РЕАЛІЗАЦІЇ СИСТЕМ АНТИПЛАГІАТУ МЕТОДАМИ SHINGLING ТА RABIN-KARP

*Донецький національний університет імені Василя Стуса, м. Вінниця*

Сучасний інформаційний простір характеризується експоненційним зростанням обсягів текстового контенту. Це породжує проблему забезпечення академічної доброчесності та захисту авторських прав. Актуальність розробки ефективних алгоритмів антиплагіату зумовлена необхідністю автоматизованої перевірки робіт студентів, наукових статей, а також SEO-аналізу вебресурсів, де унікальність тексту є критичним фактором ранжування.

Класичні методи порівняння рядків, як-от відстань Левенштейна, мають квадратичну складність $O(n * m)$, що робить їх непридатними для обробки великих баз даних у реальному часі. Сучасні підходи базуються на хешуванні та теорії множин, що дає змогу досягти лінійної складності та значно підвищити швидкодію систем [1].

Мета роботи полягає у розробці та програмній реалізації гібридного алгоритму мовою C#, що поєднує високу швидкість пошуку точних підрядків та стійкість до нечітких дублікатів.

В основі запропонованого підходу лежить послідовне застосування алгоритмів Рабіна–Карпа та методу шинглів (Shingling). Алгоритм Рабіна–Карпа використовується для миттєвого виявлення прямих запозичень (цитат). Його ключовою особливістю є використання поліноміального плинного хешу (rolling hash), що дає змогу перераховувати хеш-суму вікна тексту під час зсуву на один символ за константний час $O(1)$ за формулою [1]:

$$H_{i+1} = (d * (H_i - T[i] * h) + T[i + m])(mod q), \quad (1)$$

де $H_i$ – поточний хеш;
$d$ – розмір алфавіту (множник зсуву);
$T[i]$ – символ, що виходить;
$h = d^{m-1}(mod\ q)$ – вага старшого розряду;
$q$ – просте число.

Для виявлення модифікованого тексту (рерайту), де прямий пошук не дає результатів, застосовується метод шинглів. Текст піддається канонізації та розбивається на $n$-грами (шингли). Ступінь схожості документів A та B визначається через коефіцієнт Жаккара:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}, \tag{2}$$

де $|A \cap B|$ – кількість спільних шинглів (перетин множин),

$|A \cup B|$ – загальна кількість унікальних шинглів (об'єднання множин).

Результат $J(A,B)$ належить проміжку [0,1], де 1 означає повний збіг [2].

Програмна реалізація (рис. 1) виконана мовою C# з використанням оптимізованих структур даних. Результати тестування показали, що за значної модифікації тексту (рерайтинг) алгоритм Рабіна–Карпа фіксує наявність ключового патерну, а метод шинглів визначає залишкову схожість на рівні 20,00 %.

```
// Частина алгоритму Рабіна-Карпа (оновлення хешу)
if (i < text.Length - M)
{
    t = (d * (t - text[i] * h) + text[i + M]) % q;

    if (t < 0)
        t = t + q;
}

// Частина методу Шинглів (пошук спільних елементів)
double intersection = 0;
foreach (int hash in set1)
{
    if (set2.Contains(hash))
    {
        intersection++;
    }
}

// Розрахунок коефіцієнта схожості
double union = set1.Count + set2.Count - intersection;
double result = intersection / union;
```

*Рисунок 1 – Фрагмент програмної реалізації гібридного методу*

```
Текст 1 (Оригінал): ...Алгоритми пошуку плагіату є важливою час...
Текст 2 (Рерайт):   ...Алгоритми пошуку плагіату складають важл...
Патерн пошуку: систем освіти

Шинглів у T1: 12
Шинглів у T2: 12
Перетин: 4
Об'єднання: 20

Точний збіг (Rabin-Karp): True
Схожість (Shingling):     20,00%
```

*Рисунок 2 – Результат роботи коду*

**Висновки.** Розроблений гібридний алгоритм дає змогу нівелювати обмеження класичних методів рядкового зіставлення. Алгоритм Рабіна–Карпа забезпечує лінійну складність пошуку цитувань, тоді як метод шинглів гарантує стійкість системи до лексичних модифікацій.

## Список використаних джерел

1. Introduction to algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. 4th ed. Cambridge, MA: The MIT Press, 2022. 1312 p. URL: https://books.google. com.ua/books?id=NLngYyWFl_YC&printsec=copyright&hl=uk#v=onepage&q&f=false (дата звернення: 27.11.2025).

2. Leskovec J., Rajaraman A., Ullman J. D. Mining of Massive Datasets. 3rd ed. Cambridge: Cambridge University Press, 2020. P. 75–88. URL: http://www.mmds.org/ (дата звернення: 27.11.2025).

3. Meuschke N., Gipp B. State-of-the-art in detection of academic plagiarism. *Frontiers in Computer Science*. 2024. URL: https://www.frontiersin.org (дата звернення: 27.11.2025).

**УДК 004.422.5**

*Вершигора Р. О., здобувач*
*2 курсу спеціальності 122 Комп'ютерні науки,*
*Потапова Н. А., канд. екон. наук, доцент,*
*доцент кафедри інформаційних технологій*

# ОПТИМІЗАЦІЯ ОБРОБКИ ДАНИХ ЗА ДОПОМОГОЮ ХЕШ-СТРУКТУР У C#

*Донецький національний університет імені Василя Стуса, м. Вінниця*

Оптимізація обробки даних за допомогою хеш-структур у C# набуває все більшої значущості у сучасному програмуванні, враховуючи зростаючі обсяги інформації та потребу у швидкому доступі до неї. Використання хеш-структур дає змогу зменшити час пошуку, вставки та видалення елементів, що суттєво підвищує продуктивність програмних систем [1]. Ця тема є актуальною для дослідників та розробників, які прагнуть створити масштабовані та ефективні додатки.

Хеш-структури – це типи даних, які реалізують швидкий доступ до об'єктів за допомогою обчислення індексів через хеш-функції. Основний елемент – пара ключ-значення, де ключ перетворюється у числовий індекс. У C# широко використовуються три основні хеш-структури: Hashtable, Dictionary і HashSet. Hashtable підтримує зберігання необмеженого обсягу пар без суворої типізації, Dictionary є більш сучасною типізованою колекцією, що підвищує швидкість і безпеку, а HashSet [2] зберігає унікальні елементи, що зручно для фільтрації та перевірки належності даних.

Основний принцип роботи хеш-структур полягає у конвертації ключа в хеш-код за допомогою хеш-функції, який потім відображається у позицію зберігання