https://doi.org/10.15407/csc.2024.03.045
UDC 004.04.043; 004.912; 004.62

K.K. DUKHNOVSKA, Ph.D. in Technical Sciences, Associate Professor
at the Department of Software Systems and Technologies, Faculty of Information Technologies,
Taras Shevchenko National University of Kyiv, Ukraine,
Bohdana Khmelnytskyi Street, 24, Kyiv, Ukraine, 02000,
ORCID: https://orcid.org/0000-0002-4539-159X,
kseniia.dukhnovska@knu.ua

I.L. MYSHKO, student at the Department of Software Systems and Technologies,
Faculty of Information Technologies,
Taras Shevchenko National University of Kyiv, Ukraine,
Bohdana Khmelnytskyi Street, 24, Kyiv, Ukraine, 02000,
ORCID: https://orcid.org/0009-0003-6018-6521,
ivan.mishko21@gmail.com

A COMPARATIVE ANALYSIS OF FULL-TEXT SEARCH ALGORITHMS

The exponential growth of electronically stored textual data poses a significant challenge for search engine developers. This paper is dedicated to a detailed study and comparison of three classical full-text search algorithms: Knuth-Morris-Pratt (KMP), Boyer-Moore (BM), and Rabin-Karp (RK). These algorithms are widely used in computer science for efficient substring searching in textual data. The research results allowed us to identify the strengths and weaknesses of each algorithm and to determine the conditions under which each algorithm is most efficient.

Keywords: *full-text search, substring search algorithms, Knuth-Maurice-Pratt algorithm, Boyer-Moore algorithm, Rabin-Karp algorithm.*

Introduction

While traditional search algorithms are efficient for small datasets, they often fall short when dealing with large-scale systems. In today's information-rich environment, users demand rapid and accurate retrieval of specific objects from vast collections.

Advanced search algorithms address these challenges through innovative approaches. Indexing, which involves creating a database of stored information about all objects in the system, enables swift localization of desired resources. Filtering, using various criteria, reduces the number of irrelevant results, ensuring a more accurate representation of the user's query. Contextual analysis, which considers the specific characteristics of the search to enhance result precision, is another crucial component.

Implementing advanced search algorithms offers numerous benefits. Primarily, it significantly increases search speed, enabling users to interact with the platform efficiently. Additionally, it improves search accuracy, providing users with only the relevant information.

Cite: Dukhnovska K.K., Myshko I.L. Analysis and Comparison of Full-Text Search Algorithms. Control Systems and Computers, 2024, 3, 45-52. https://doi.org/10.15407/csc.2024.03.045

The growing volume of information on platforms and the increasing demand for fast, accurate search results underscore the importance of research into advanced search algorithms.

Analysis of Existing Research and Problem Identification

Some text search algorithms, such as the Knuth-Morris-Pratt algorithm, are renowned for their speed, efficiency, and versatility. They find applications in various fields, including text processing and data analysis. Due to their power and ease of use, these algorithms remain popular among software developers.

In [1], an analysis of pattern searching algorithms is conducted, specifically focusing on the Knuth-Morris-Pratt algorithm, the Boyer-Moore algorithm, the Bitap algorithm, and the Aho-Corasick algorithm. This study delves into the fundamental principles of each algorithm and highlights key directions for future research aimed at improving their performance and reducing resource consumption.

The Knuth-Morris-Pratt algorithm finds application in various domains. In a study [2], a novel approach to detecting and preventing SQL injection and cross-site scripting (XSS) attacks is proposed. SQL injection attacks are a form of cyberattack that exploits vulnerabilities in applications that use SQL queries to interact with a database. They pose a significant threat to data-driven web applications. The paper explores various patterns of such attacks and proposes to detect them using the Knuth-Morris-Pratt algorithm. The study demonstrates that the proposed method is more effective in detecting and preventing SQL injection and XSS attacks.

In another study [3], security issues such as network intrusion and viruses are also investigated. The study introduced an intrusion detection system (IDS) and performed event classification to categorize events as either normal or intrusive. This classification process relies on one of the string matching algorithms. The paper employs three substring search algorithms: Brute-Force, Knuth-Morris-Pratt, and Boyer-Moore.

In [4], the Knuth-Morris-Pratt algorithm and its improvements are applied to multi-channel string filtering. The term “multi-channel string” refers to the use of multiple read or write paths within a string, where each path represents a separate stream or “channel” for data processing.

This can occur both in hardware (e.g., multi-threaded readers or writers) and in software that processes data from various sources or different parts of a system. In multi-channel strings, each stream can represent a separate channel for data transmission, and such strings are often used to optimize reading and writing in a parallel processing or high-performance systems.

Multi-channel strings can enable task distribution and improve data processing efficiency by utilizing multiple streams for different operations. In programming or computer architecture, when referring to multi-channel strings, it may indicate the ability to use multiple read or write streams for different parts of data or different data operations. This approach can lead to improved performance and optimized resource utilization.

The study utilized the Knuth-Morris-Pratt algorithm and its derivatives for string pattern matching.

With the COVID-19 pandemic forcing students into online learning, the reliance on devices like computers and tablets has increased. Study [5] proposes using the Knuth-Morris-Pratt algorithm to facilitate the translation of the Palembang language to Bahasa Indonesia, thereby enabling students to learn Palembang through a software application.

Study [6] performs pattern matching on compressed patterns within binary texts encoded using minimum redundancy codes. A modified Knuth-Morris-Pratt algorithm is employed to mitigate false positives, where an encoded pattern appears in the encoded text but does not correspond to the original pattern. A bitwise Knuth-Morris-Pratt algorithm is introduced, capable of shifting one additional bit upon mismatch due to the binary nature of the alphabet.

String searching algorithms with linear time complexity and constant space complexity are generally quite complex. Most of them consist of two phases: a preprocessing phase and a search phase. An exception is the one-pass Crochemore algorithm, a version of the Knuth-Morris-Pratt algorithm with "instantaneous" pattern shift calculations. Study [7] examines Crochemore's approach and builds upon it.

Therefore, the study of improved search algorithms is not only relevant but also important for improving the functionality and efficiency of using modern information technologies.

The Purpose and Objectives of the Study

The purpose of the study is to improve the efficiency of algorithms in the context of searching for text patterns in large volumes of data. This work aims to determine the optimal conditions and scenarios for the use of algorithms to increase the productivity of search engines in information systems.

To achieve the goal, the following tasks were set:

- using different algorithms to compare the quality of the search engine;
- comparison of the speed of the search mechanism using different algorithms;
- study of the impact of algorithm partitioning on parallel flows.

Analysis of Full-text Search Algorithms

The study was conducted using text files that were generated using a script that creates files with random data and search terms at the end. A total of 10.000 files in the Ukrainian language were generated, each of them having a length of about 140.000 characters.

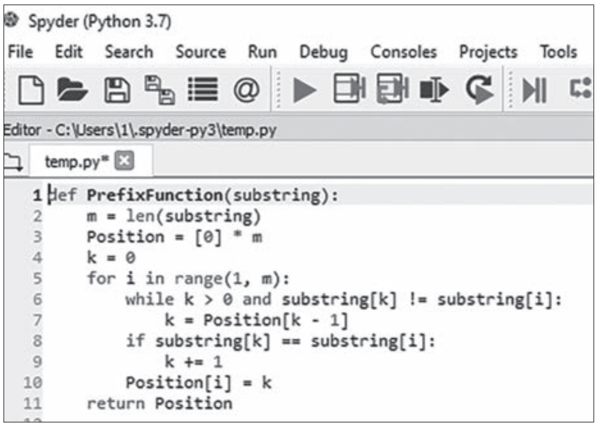
Python 3.11 and the built-in functionality of the programming language to generate random words were used to generate the files. Each file contains words 8 characters long, consisting of lowercase letters of the Ukrainian alphabet. In addition, search terms were randomly included in some files to investigate the performance of full-text search algorithms.

FastAPI, which is a framework for rapid API development, was used to implement the search engine and conduct research. The results of research on the effectiveness of the search engine using various algorithms.

The Results of Research on the Effectiveness of the Search Engine Using Various Algorithms

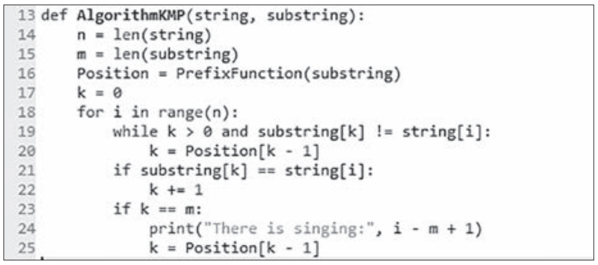
To achieve this goal, a study was conducted using various line search algorithms.

The Knuth-Morris-Pratt (KMP) algorithm employs a precomputed table to expedite substring searching. By leveraging information about the pattern's internal structure, the algorithm can avoid unnecessary character comparisons, significantly enhancing search efficiency. The Knuth-Morris-Pratt algorithm is an efficient algorithm for searching for a substring within a string. It employs preprocessing of the search pattern to avoid unnecessary character comparisons, significantly accelerating the search process. Before starting the search, the algorithm constructs a shift table (prefix function) (Fig. 1). For each character in the pattern, the shift table indicates how far to shift the pattern if this character does not match the corresponding character in the text. The values in the table are calculated based on the prefixes and suffixes of the substring of the pattern preceding the current character. This table contains information about how many characters the pattern can be shifted in case of a mismatch during comparison with the text. Next, the algorithm compares the characters of the pattern with the characters of the


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools
Editor - C:\Users\1\.spyder-py3\temp.py
temp.py*

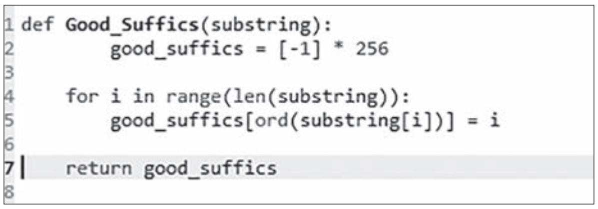
```
def PrefixFunction(substring):
    m = len(substring)
    Position = [0] * m
    k = 0
    for i in range(1, m):
        while k > 0 and substring[k] != substring[i]:
            k = Position[k - 1]
        if substring[k] == substring[i]:
            k += 1
        Position[i] = k
    return Position
```

Fig. 1. Prefix functions



```
def AlgorithmKMP(string, substring):
    n = len(string)
    m = len(substring)
    Position = PrefixFunction(substring)
    k = 0
    for i in range(n):
        while k > 0 and substring[k] != string[i]:
            k = Position[k - 1]
        if substring[k] == string[i]:
            k += 1
        if k == m:
            print("There is singing:", i - m + 1)
            k = Position[k - 1]
```

Fig. 2. Substring search by KMP algorithm

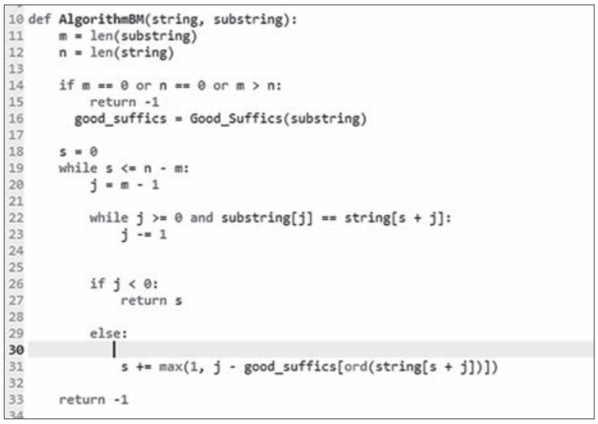


```
def Good_Suffics(substring):
        good_suffics = [-1] * 256

    for i in range(len(substring)):
        good_suffics[ord(substring[i])] = i

    return good_suffics
```

Fig. 3. Creating a table of good suffixes



```
def AlgorithmBM(string, substring):
    m = len(substring)
    n = len(string)

    if m == 0 or n == 0 or m > n:
        return -1
      good_suffics = Good_Suffics(substring)

    s = 0
    while s <= n - m:
        j = m - 1

        while j >= 0 and substring[j] == string[s + j]:
            j -= 1


        if j < 0:
            return s

        else:

            s += max(1, j - good_suffics[ord(string[s + j])])

    return -1
```

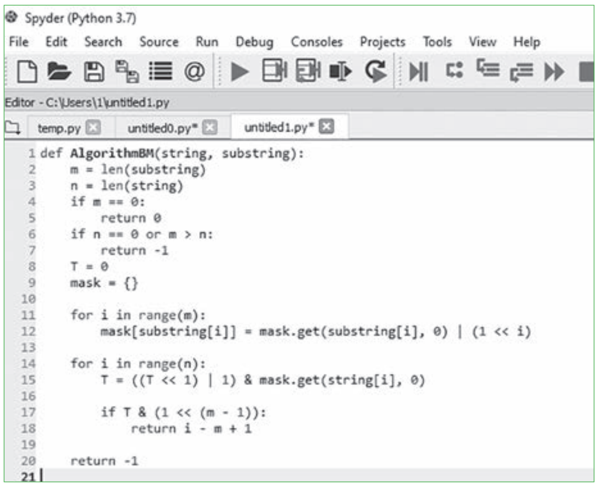
Fig. 4. Substring search by BM algorithm

Table 1. **"Good suffixes" for the word integral**

i	*n*	*t*	*e*	*g*	*r*	*A*	*l*	*
1	2	3	4	5	6	7	8	8

text one by one (Fig.2). If a mismatch occurs, it uses the shift table to determine the new position from which to continue the comparison, thus avoiding repeated checks of already checked characters.

The algorithm has linear time complexity $O(n+m)$ — where n is the string length and m is the substring length.


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\1\untitled1.py
temp.py
untitled0.py*
untitled1.py*

```
def AlgorithmBM(string, substring):
    m = len(substring)
    n = len(string)
    if m == 0:
        return 0
    if n == 0 or m > n:
        return -1
    T = 0
    mask = {}

    for i in range(m):
        mask[substring[i]] = mask.get(substring[i], 0) | (1 << i)

    for i in range(n):
        T = ((T << 1) | 1) & mask.get(string[i], 0)

        if T & (1 << (m - 1)):
            return i - m + 1

    return -1
```

Fig. 5. Substring search by BP algorithm

The BM algorithm operates by comparing a pattern (substring) to a text starting from the end rather than the beginning, allowing it to quickly skip over sections of the text where the pattern cannot possibly occur. This makes it one of the fastest general-purpose substring search algorithms. Let's consider an example. Suppose we have the pattern "integral". First, we create a "good suffix" table. We start from the end of the given pattern (Table 1).

The value in the second row of the table represents the index of the first occurrence of the letter within the substring. The column marked with an asterisk indicates the length of the substring. The script for creating a table of good suffixes is presented in Fig. 3.

To find the substring "integral" within the string "indefinite integral", we compare the 8th character of the string with the 8th character of the substring. In this case, we compare '*i*' and '*l*', which don't match. Since the letter '*i*' is present in the good suffix table, we shift the pattern to the right by the difference between the positions of '*i*' and '*l*' in the pattern, which is 7 in this case. This aligns the '*r*' of the pattern with the '*r*' of the text. We then repeat the process (Fig. 4).

The algorithm has linear time complexity $O(n+m)$ – where n is the string length and m is the substring length. However, it is important to note that this worst case scenario is rare.

The BP algorithm is frequently employed in tasks such as word searching in texts, spell checking, and bioinformatics analyses. The Bitap algorithm relies on the use of bitmasks to represent information about character matches (Fig.5). The algorithm implements fuzzy search.

The algorithm has linear time complexity $O(n+m)$ — where n is the string length and m is the substring length.

The RK algorithm employs a hashing scheme to efficiently search for a pattern within a text. The algorithm can be broken down into three main steps: computing a hash value for each substring, comparing these hash values, and efficiently calculating the hash value for the next substring (as illustrated in Fig. 6—8).

All algorithms were optimized for parallel processing, running in 8 parallel threads to maximize performance.

The advantages and disadvantages of the presented algorithms are summarized in Table 2.

A comparative analysis of the algorithms was conducted using several key performance indicators, which were classified into two main categories: search result quality and search speed.

Quality of search results:

• accuracy: the ratio of the number of relevant documents to the total number of found documents;

• completeness:

• the ratio of the number of found relevant documents to the total number of relevant documents in the collection;

• relevance: the degree of relevance of the found documents to the user's request.

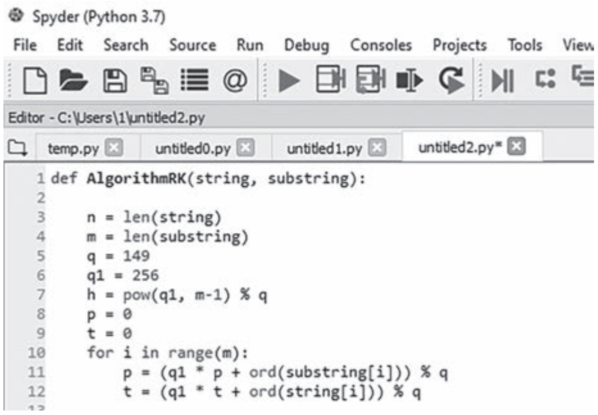
Search speed:

• search time: time required to find all relevant documents;

• resource usage: the amount of memory and processor time required for searching.

The research used the empirical testing method. This method involves conducting experiments with various search algorithms and evaluating their effectiveness using certain indicators.

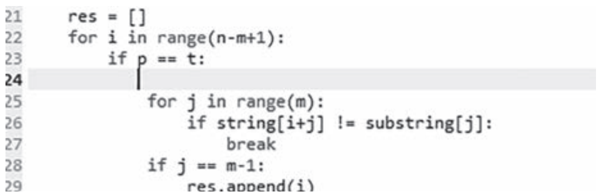
The method of empirical testing allows you to obtain practical data about the behavior of algorithms in real conditions.


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View
Editor - C:\Users\1\untitled2.py
temp.py
untitled0.py
untitled1.py
untitled2.py*

```python
def AlgorithmRK(string, substring):

    n = len(string)
    m = len(substring)
    q = 149
    q1 = 256
    h = pow(q1, m-1) % q
    p = 0
    t = 0
    for i in range(m):
        p = (q1 * p + ord(substring[i])) % q
        t = (q1 * t + ord(string[i])) % q
```

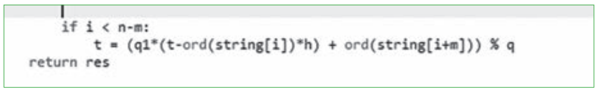
Fig. 6. Substring hashing



```python
    res = []
    for i in range(n-m+1):
        if p == t:

            for j in range(m):
                if string[i+j] != substring[j]:
                    break
            if j == m-1:
                res.append(i)
```

Fig. 7. Hash comparison



```python
        if i < n-m:
            t = (q1*(t-ord(string[i])*h) + ord(string[i+m])) % q
    return res
```

Fig. 8. Calculation of the hash for the next substring

Table 2. **Advantages and disadvantages of full-text search algorithms**

Algorithm	Advantages	Disadvantages
KMP	Fast for long patterns	More difficult to implement
BM	Fast for short patterns	Not as effective as KMP for long patterns
BP	May find inexact matches	Slower than other algorithms
RK	Very fast for long texts	It can cause false positives

The research involves the development of a test environment that includes a collection of text documents, a set of queries, software for implementing a search mechanism, and software for evaluating the effectiveness of algorithms.

Conducting experiments with different types of text documents and with different queries is an integral part of the research.

Analysis of the results and formulation of conclusions about the effectiveness of search algorithms will be an important stage of the research.

The expected results of the research include the determination of optimal search algorithms for different types of text documents, the development of recommendations for the use of search algorithms, and the acquisition of new knowledge about the behavior of search algorithms in different conditions.

The obtained data will have significant practical value for developers of information systems and search engines.

The results of the research will help determine the optimal search algorithms for various types of text documents, develop recommendations for the use of search algorithms, and improve existing and create new information systems and search mechanisms.

This, in turn, will lead to an improvement in the quality of information search, an increase in the efficiency of users' work, and an expansion of the capabilities of information systems.

Testing was conducted using various algorithms, the results of which are presented in the form of a table (Table 3).

Table 3. **Testing of full-text search algorithms**

Search request	Algorithm	Number of matches
"term 1, term 2" (in 74 texts)	KMP	74
	BM	74
	BP	74
	RK	74
"term 1, term 2, term 3" (in 25 texts)	KMP	25
	BM	25
	BP	25
	RK	25

Table 4. **Average search time**

Search request	Algorithm	Average search time
"term 1, term 2" (in 74 texts)	KMP	4.39
	BM	1.05
	BP	7.22
	RK	8.25
"term 1, term 2, term 3" (in 25 texts)	KMP	4.32
	BM	0.75
	BP	7.25
	RK	8.15

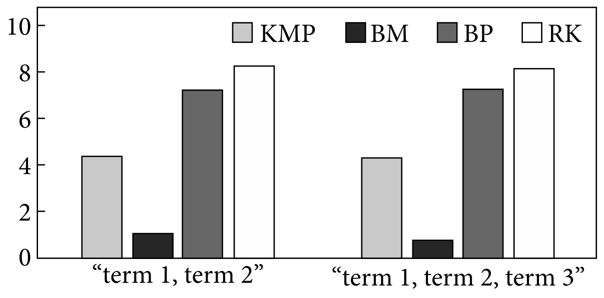


Fig. 9. Graph of the average search time (ms) for different

Table 5. **Influence of parallel streams**

Search request	Algorithm	Average search time with parallel processes	Impact on search time, %
"term 1, term 2" (in 74 texts)	KMP	0.91	–79.28
	BM	0.28	–73.33
	BP	1.38	–80.87
	RK	1.56	–81.11
"term 1, term 2, term 3" (in 25 texts)	KMP	0.96	–77.78
	BM	0.22	–70.67
	BP	1.37	–81.10
	RK	1.57	–80.76

On the basis of the conducted tests, data on the effectiveness of the search mechanism were obtained (Table 4).

Fig. 9 provides a graphical representation of the differences in average search time.

The results, as depicted in the diagram, indicate that the BM algorithm is the fastest.

Influence of Parallel Streams

An illustration of the effect of using parallel threads on the search time for two test queries is given in Table 5.

Analyzing the results of the study, we can conclude that the Boyer-Moore algorithm was the most effective in terms of search speed for both test queries (Table 4). Its advanced "bad symbol" heuristics greatly speed up the process by skipping unnecessary symbol comparisons. Despite the fact that the KMP algorithm was considered the best in previous studies, in this case, it took second place in terms of speed.

However, for short search patterns, KMP may still have an advantage due to its optimized structure and ability to avoid repeated character comparisons. Bitap and Rabin-Karp algorithms showed worse results in terms of search speed compared to BM and KMP.

Regarding search quality, all algorithms showed the same number of found matches for both queries (Table 3), which indicates their equal ability to find relevant documents.

The use of parallel streams significantly improved the search speed for all algorithms (Table 5), especially for longer texts. The speedup ranged from 70% to 81% depending on the algorithm and the query. Although parallelism required the overhead of thread synchronization, the overall time gain was worth the effort.

Conclusions

This study was aimed at comparing the effectiveness of different full-text search algorithms in the Ukrainian language. The main tasks included determining the most efficient algorithm in terms of search time and quality, analyzing the impact of parallel streams on search speed, and evaluating the possibilities and limitations of the study.

According to the test results, it was found that the Boyer-Moore algorithm demonstrated the best search speed for both test queries. The KMP algorithm ranked second for speed but may be an advantage for short search patterns. All algorithms showed the same search quality in terms of the number of matches found.

The use of parallel threads led to a significant acceleration of the search for all algorithms, especially for long texts. Although additional synchronization overhead was required, the overall speed gain justified the use of parallelism.

Despite the achieved results, the study has certain limitations, such as the small size of the test data set and the use of only the Ukrainian language. The details of the implementation of the algorithms could also affect the accuracy of the results.

In general, these findings allow us to recommend the use of the BM algorithm as an effective solution for full-text search, especially for long text documents. Parallel computing should also be used to speed up searches in large text collections.

Promising directions for further research include expanding the test data set, studying the impact of various algorithm parameters, and developing adaptive and intelligent search methods using machine learning. Integration with other information processing systems and the development of specialized search solutions are also important tasks.

REFERENCES

1. Zhang, Z. (2022). "Review on String-Matching Algorithm". In *SHS Web of Conferences,* Vol. 144, p. 03018. EDP Sciences. https://doaj.org/article/c78bdfe402274e139c26295b12694388.
2. Abikoye, O.C., Abubakar, A., Dokoro, A.H., Akande, O.N., & Kayode, A.A. (2020). "A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm". *EURASIP Journal on Information Security*, pp. 1–14. https://doaj.org/article/27372f01abe34aea9ad7f516f9739556.

3. Sulaiman, H.E. (2019). “Use the Brute_Force Pattern Matching Algorithm for Misuse Intrusion Detection System”. *AL-Rafidain Journal of Computer Sciences and Mathematics*, 13 (1), pp. 68–85. https://doaj.org/article/6c40c705b1fd4263955b4fc066b9155d.
4. Hendrian, D., Ueki, Y., Narisawa, K., Yoshinaka, R., & Shinohara, A. (2019). “Permuted pattern matching algorithms on multi-track strings. *Algorithms*, 12(4), pp. 73. https://doaj.org/article/4a62dbb11cc14fd0ba9e2be7fa549cc2.
5. Shapira, D., & Daptardar, A. (2006). “Adapting the Knuth–Morris–Pratt algorithm for pattern matching in Huffman encoded texts”. *Information processing & management*, 42(2), pp. 429–439. https://www.sciencedirect.com/science/article/abs/pii/S0306457305000191.
6. Rytter, W. (2003). “On maximal suffixes and constant-space linear-time versions of KMP algorithm”. *Theoretical computer science*, 299 (1–3), pp. 763–774. https://www.sciencedirect.com/science/article/pii/S030439750200590X.

Received 15.04.2024

К.К. Духновська, кандидат технічних наук, доцент,
кафедра програмних систем і технологій, факультет інформаційних технологій,
Київський національний університет імені Тараса Шевченка,
вул. Богдана Гаврилишина, 24, м. Київ, Україна, 02000,
ORCID: https://orcid.org/0000-0002-4539-159X,
kseniia.dukhnovska@knu.ua

І.Л. Мишко, студент, кафедра програмних систем і технологій,
факультет інформаційних технологій,
Київський національний університет імені Тараса Шевченка,
вул. Богдана Гаврилишина, 24, м. Київ, Україна, 02000,
ORCID: https://orcid.org/0009-0003-6018-6521,
ivan.mishko21@gmail.com

ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ПОВНОТЕКСТОВОГО ПОШУКУ

Вступ. Ця робота була спрямована на всебічне дослідження та порівняння ефективності різних алгоритмів повнотекстового пошуку, зокрема алгоритмів Кнута-Морріса-Пратта та Боєра-Мура, з метою визначення оптимального алгоритму для повнотекстового пошуку. Крім того, дослідження оцінило вплив паралельного обчислення на швидкість пошуку та виявило обмеження наявних підходів.

Мета. Метою дослідження є підвищення ефективності алгоритмів у контексті пошуку текстових шаблонів у великих обсягах даних.

Методи.Алгоритми повнотекстового пошуку.

Результати. Аналізуючи результати дослідження, можна зробити висновок, що алгоритм Бойєра-Мура виявився найефективнішим за швидкістю пошуку для обох тестових запитів. Його удосконалена евристика “поганого символу” дає змогу значно прискорити процес, пропускаючи непотрібні порівняння символів. Попри те, що алгоритм Кнута-Моріса-Пратта вважався найкращим у попередніх дослідженнях, в даному випадку він посів друге місце за швидкістю роботи.

Проте для коротких пошукових шаблонів алгоритм Кнута-Моріса-Пратта все ще може мати перевагу завдяки своїй оптимізованій структурі та здатності уникати повторних порівнянь символів. Алгоритми Бітап та Рабіна-Карпа продемонстрували гірші результати щодо швидкості пошуку порівняно з алгоритмами Бойєра-Мура та Кнута-Моріса-Пратта.

Щодо якості пошуку, всі алгоритми показали однакову кількість знайдених збігів для обох запитів, що свідчить про їх рівноцінну здатність знаходити релевантні документи.

Використання паралельних потоків значно покращило швидкість пошуку для всіх алгоритмів, особливо для довших текстів. Прискорення сягало від 70% до 81% залежно від алгоритму та запиту. Хоча паралелізм вимагав додаткових витрат на синхронізацію потоків, загальний виграш часу виявився вартим цих зусиль.

Висновки. Перспективними напрямами подальших досліджень є розширення набору тестових даних, детальний аналіз впливу різних параметрів алгоритмів на їхню ефективність, а також розробка адаптивних та інтелектуальних методів пошуку з використанням машинного навчання.

Результати цього дослідження мають практичне значення для розробки ефективних пошукових систем, систем керування базами даних та інших програмних продуктів, які використовують повнотекстовий пошук.

Ключові слова: *повнотекстовий пошук, алгоритми пошуку підрядків, алгоритм Кнута-Моріса-Пратта, алгоритм Боєра-Мура, алгоритм Рабіна-Карпа.*