International Journal of Engineering Science and Applied Mathematics
Volume.13, Number 11; December-2022;
ISSN: 2836-9521| Impact Factor: 5.78
https://zapjournals.com/Journals/index.php/ijesam
Published By: Zendo Academic Publishing

ENHANCING SEARCH PROCESSES: EVOLUTION OF PATTERN SEARCHING ALGORITHMS

[1]Rawan A. Abdeen.

Article Info

Keywords: pattern searching, Pattern searching is a fundamental problem in computer science with string matching, algorithm, diverse applications ranging from text editing to bioinformatics and bioinformatics, computer vision, computer vision.

Abstract

This paper explores the significance of pattern database search, DNA sequence, searching algorithms in various domains and discusses the challenges computational cost, scalability, associated with their implementation. A pattern represents a non-empty data processing language and can be described by a string or a set of strings. String matching, the core task of pattern searching, involves finding occurrences of a given pattern within a larger string, irrespective of the alphabetic order. The problem of pattern searching arises in database searches, substring pattern matching, and numerous other applications that require efficient search processes. This study emphasizes the need for fast and efficient pattern searching algorithms, considering the exponential growth in data availability. Different algorithms possess distinct advantages and disadvantages, necessitating careful selection based on the specific application requirements. Furthermore, the paper highlights the extensive applications of pattern searching, including DNA and protein sequence analysis, spell checking, computer viruses' detection, signature matching, and language translation. Real-time detection of face masks and World Wide Web search engines are other domains where pattern searching finds utility. While several algorithms exist, their scalability and computational costs pose challenges for large databases and DNA sequences. Therefore, it is crucial to develop algorithms that overcome these limitations and improve performance. The paper discusses the need for advancements in pattern searching algorithms to address the evolving demands of data processing and storage.

Introduction

[1] Department of Informatics, Faculty of Natural Sciences and Mathematics, University of Tetovo, Tetovo, Republic of North Macedonia

Pattern searching is an important problem in computer science that is used to show search results in text editing to find unique patterns in the text editor, data compression, DNA sequence matching – of, spell checking, computer viruses, signature matching, dictionary-based language translation [1], World Wide Web search engines and other computer application systems [2], in biology [3], most of major data processing in bioinformatics involves in one way or another the recognition of certain patterns within DNA, RNA or protein sequences [4], and the detection of face masks in real time [5]. A pattern represents a nonempty language that contains strings other than the empty string. It can be described by a string, by a finite set of strings, or by other means. A string is a set of characters that can contain spaces and numbers. A string can be ordered or unordered because the main task of string matching is to find string A within string B, regardless of alphabetic order [6]. The problem of pattern searching is to search for occurrences of strings of language in other strings – or in texts that are less formal [7]. String matching can be understood as the problem of finding a pattern with some property within a given sequence of symbols. The simplest case is that of finding a certain string within the model [8]. Pattern searching are very useful when performing database search operation, they are also useful in finding patterns in substring from a larger string. We have problems that need fast and efficient algorithms for computation. There are many applications that require search process and thus we need Pattern searching algorithms [9]. One thing is certain, that each algorithm that exists depending on the environment where it is implemented has its advantages and disadvantages, which are different from another algorithm. The availability of data is increasing day by day tremendously. Therefore, a great need has arisen to store and process data. The Pattern Searching problem has various applications. The main objective of Pattern Searching is to search for a particular pattern for a position in a large piece of text (e.g. from a book, a paragraph, a sentence, etc.). The goal is to find the representation of a text within another text. For example, when we need to find a text in the text editor, it is a difficult task to find that word or text manually. If similar words are found then we will highlight all occurrences of the string we are looking for, otherwise it will show no matches if there are zero occurrences of the string [2]. To search for a pattern within a string, an algorithm is needed to find the pattern, as well as to recognize the locations where it is found in a given pattern of characters. Determining which of the algorithms is the best to use depends on the application where the algorithm will be used [10], and many current algorithms may not scale well for large databases or sequences of DNA due to high computational costs [11]. Each algorithm tries to avoid problems that have been encountered in existing algorithms.

1 Background

There are many applications that have search functionality such as performing database search operations, and that is why Pattern Searching algorithms are necessary. Each algorithm has its advantages and disadvantages. In the context of our research, Pattern searching algorithms will be experimented on computers with different performances, with different inputs, with the sole purpose of having accurate conclusions about their speed and ranking. Although there are a large number of research where various analyses and comparisons have been made between the algorithms that currently exist, there is still a dilemma as to how accurate such research are since it must be taken into account that we are dealing with analyses, tests that are carried out in computers and such analyses besides depending on the complexity of the algorithm that is executed, also depend on the performance of the computer, the active processes that are running in the operating system, the operating system itself etc. Based on these circumstances, we will elaborate our analyses several times in order to reach the most approximate and reliable results.

We will test the Pattern searching algorithms in C#, Java and Python programming languages, with inputs from various sources. However, to be as accurate as possible in the analysis, we will use the same inputs to all pattern searching algorithms, and based on the results that will come out from the selected programming languages, we will make their comparisons. The accuracy of these results also depends on the code sequence that will measure the execution of the algorithms in question. After the analyses and comparisons, we will elaborate the results together with the relevant clarifications through tables, where in the tables we will have the time of the speed of the algorithms. Our research is firstly related to the comparison of pattern searching algorithms. Comparison of algorithms between different CPUs using strings of different sizes, and comparison of programming languages implemented on different CPUs. The results of the execution time of the Pattern Searching algorithms will be displayed in tables where we can then make comparisons between programming languages and comparisons between CPUs.

1.1 Analysis of algorithms

In order to find which algorithm is better than another algorithm, analyses and comparisons between them should be done. To compare algorithms, their complexity should be calculated. There are two types of complexity as well [12] [13]: • Space Complexity – which actually represents the necessary memory or space required by the algorithm to correctly execute the inputs, and

- Time Complexity – which actually represents the time required for the algorithm to correctly execute the inputs.

Nowadays, temporal complexity is more important than spatial complexity. We say such a thing based on the fact that always when we talk about the complexity of algorithms, it is meant how fast the algorithm manages to execute a certain code in proportion to the memory it uses. To make the time comparison of the algorithms is actually a very difficult task, a task that implies that the running time of the algorithm must be calculated, and such a calculation always depends on the processor, the programming language where it is executed and many other factors. Even if the processor and the programming language are the same, it is still difficult to determine the exact time interval of the execution of the algorithms, since they cannot be the same, the use of the processor in the same way by different processes within the operating system. However, we will talk about the complexity of the algorithms and their calculation after the description of the most popular Pattern Searching algorithms today and the description of their code. Algorithm analysis defines the estimation of the resources needed for an algorithm to solve a given problem. Sometimes the resources include memory, time and communication spaces. Obviously, an algorithm that takes months or years to solve a given problem is not useful. In addition, the algorithm that requires gigabytes (GB) of main memory to solve certain problems is not efficient. In general, the time required by an algorithm increases with the size of the input, so it is normal to describe the execution time of a program as a function of the size of its input.

2 Comparison of each pattern searching algorithm in C#, Java, and Python programming languages In order to make a comparison with the algorithms used to find the model, in this research, we used three computers with different processors:

I. Processor: AMD A9-9410 RADEON R5, 5 COMPUTE CORES 2C +3G 2.90GHz

Installed RAM: 8.00 GB System type 64-bit operating system, x64-based processor, II. Processor: Intel ® Core ™ i7-2620m CPU @ 2.70 GHz Installed memory (RAM): 8.00 GB System Type: 64-bit,

III. Processor Intel ® Core ™ i5-6200U CPU @ 2.30GHz 2.40GHz Installed Ram 4.00GB System type 64 bit operating system, x64 based processor.

We implemented the algorithms in the C# programming language in Visual Studio 2017, in the Java programming language in Eclipse, and in Python 3.10 in PyCharm 2022.2. The program measures the execution time of the algorithms, while in the experiment we will use text of different sizes, where we will see more clearly the changes in the execution time of the algorithms. Below are the tables with the data obtained from the experiment performed comparing the speed of algorithms in C#, Java and Python programming languages on all three CPUs.

In the tables below, we can see the comparison of the execution time of the programming languages C#, Java and Python using three different CPUs AMD A9-9410 RADEON R5, CPUs i7-2620m and i5-6200U which shows which of the programming languages performs better depending on which CPU they are implemented on. From the tables it is clear that text sizes (characters) ranging from 100 characters to 1 million were used. Whereas, the figures are given as execution time in milliseconds. According to the results shown in the tables below Naive algorithm, KMP algorithm, Rabin-Karp algorithm, Finite Automata algorithm, Boyer-Moore algorithm, AhoCora sick algorithm, Z algorithm, Java programming language is faster in time complexity than C# programming language and Python. While the Python programming language is the language that takes the most time during the execution time of the algorithm.

Table 1. Running time of Naive algorithm in C#, Java and Python programming languages

Naïve Algorithm									
	C#			**Java**			**Python**		
Text Size (Characters)	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**
100.00	0.1095	0.0024	0.1131	0.0027	0.0011	0	0.0989	0	0.0998
500.00	0.2403	0.004	0.0993	0.0032	0.0022	0.0002	0.3059	0.09984	0.1004
1,000.00	0.2998	0.0032	0.2292	0.0033	0.0009	0.0002	0.7999	0.19991	0.8761
5,000.00	0.3727	0.0126	1.207	0.0126	0.0018	0.0018	1.6972	1.49795	2.5211
10,000.00	0.3782	0.0169	2.078	0.0197	0.003	0.0025	3.4681	2.39724	6.5556
50,000.00	1.2706	0.1486	5.9197	0.0415	0.0137	0.0175	12.7561	12.3919	28.7274
100,000.00	2.151	1.7691	12.1105	0.2254	0.0301	0.0448	28.8799	25.3835	87.7738
1,000,000.00	9.3829	2.067	30.5555	0.4168	0.1108	0.1291	509.911	305.6094	1202.025

In Table 1, at text size 500 to 1 million on the three CPUs, the Java programming language is faster in time complexity. According to data on CPU i7-2620m at text size 100, programming language Python (0 ms) performs better than Java (0.0011 ms) and C# (0.0024 ms).

Table 2. Running time of KMP algorithm in C#, Java and Python programming languages

KMP Algorithm									
	C#			**Java**			**Python**		
Text Size (Characters)	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**

100.00	0.0597	0.0047	0.0028	0.0018	0.0007	0	0.2	0	0.1999
500.00	0.1435	0.0064	0.0052	0.0029	0.0011	0.0001	0.3185	0.0998	0.7235
1,000.00	0.1792	0.0048	0.0695	0.0033	0.0011	0.0003	0.6993	0.3996	1.1483
5,000.00	0.409	0.014	0.1338	0.0104	0.0028	0.0018	7.4838	2.0979	4.6741
10,000.00	0.4406	0.0029	0.1263	0.0135	0.0045	0.0044	6.6664	4.6961	10.2164
50,000.00	0.4108	0.0471	0.4665	0.0516	0.0152	0.0175	31.273	22.3855	50.278
100,000.00	0.7288	0.025	0.7814	0.1147	0.0312	0.0413	86.97	46.0711	129.7641
1,000,000.00	2.9468	0.53772	2.4765	0.3946	0.0943	0.3367	876.39	522.8753	1680.35

According to Table 2 in the KMP algorithm, on the data on the CPU i7-2620m, at the text size of 100, the programming language Python (0 ms) is faster than Java (0.0007 ms) and C# (0.0047) and at the text size of 100,000.00, the programming language C# (0.025 ms) outperforms Java (0.0312 ms) and Python (46.0711 ms).

Table 3. Running time of Rabin-Karp algorithm in C#, Java and Python programming languages

Rabin-Karp Algorithm									
	C#			**Java**			**Python**		
Text Size (Characters)	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**
100.00	0.1676	0.003	0.0036	0.004	0.0003	0.0002	0.3	0.2	0.777
500.00	0.1655	0.0044	0.0055	0.0029	0.001	0.0002	0.5987	0.0999	1.011556
1,000.00	0.343	0.0144	0.0143	0.0032	0.001	0.0002	1.2244	0.4995	2.477222
5,000.00	0.4669	0.0061	0.1688	0.0108	0.0022	0.0025	3.275	2.9973	6.882556
10,000.00	0.811	0.0351	0.1413	0.0143	0.0042	0.0049	9.17	6.7946	15.62733
50,000.00	1.3802	0.0675	0.4631	0.0744	0.0163	0.0254	47.693	30.4802	82.46944
100,000.00	2.0835	0.115	0.611	0.0976	0.0299	0.0375	191.45	61.7618	154.3688
1,000,000.00	9.416	1.3168	4.6868	0.5017	0.1015	0.2278	1710.356	696.1685	2313.46

In the Rabin Karp algorithm (Table 3), for all text sizes, the Java programming language performs faster in time complexity than the other two programming languages.

Table 4. Running time of Finite Automata algorithm in C#, Java and Python programming languages

Finite Automata Algorithm			
Text Size	**C#**	**Java**	**Python**

(Characters)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)
100.00	0.0253	0.0019	0.018	0.0273	0.001	0.0746	1.021	1.0986	2.466333
500.00	0.164	0.0025	0.0075	0.0285	0.001	0.1291	1.658	1.0986	2.955556
1,000.00	0.1671	0.0023	0.0613	0.0305	0.001	0.2024	1.606	1.2982	3.232889
5,000.00	0.3861	0.4484	0.4591	0.0364	0.1122	0.1873	53.01	36.6744	91.31444
10,000.00	0.5908	0.5362	0.5632	0.0337	0.1094	0.1602	53.16	37.1761	101.8878
50,000.00	1.0932	0.5741	0.6388	0.2718	0.1142	0.1421	55.167	44.4715	114.8871
100,000.00	1.2509	0.5197	0.8178	0.2873	0.1192	0.1988	79.954	57.1635	203.288
1,000,000.00	5.9415	0.9367	3.8872	0.5108	0.1567	0.3465	575.92	258.8382	1001.616

On the AMD A9-9410 RADEON R5 CPU (Table 4) we can see that at text size 100, the programming language C# (0.0253 ms) is faster than Java (0.0273 ms) and Python (1.021 ms).

Table 5. Running time of Boyer Moore Bad Suffix algorithm in C#, Java and Python programming languages

Boyer Moore Bad Suffix Algorithm									
	C#			Java			Python		
Tezt Size (Characters)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)
100.00	0.1197	0.0011	0.0166	0.0021	0.0003	0	0.199	0	0.199
500.00	0.1758	0.0016	0.0694	0.0036	0.0006	0.0012	0.114	0.2997	0.659
1,000.00	0.2448	0.0017	0.1082	0.0037	0.001	0.0047	0.5325	0.2996	0.811
5,000.00	0.4558	0.0058	0.0746	0.0042	0.0011	0.0014	1.6166	1.3987	3.848
10,000.00	0.336	0.0031	0.3034	0.0093	0.002	0.0031	3.3007	2.9976	7.222
50,000.00	0.936	0.0545	0.6545	0.0443	0.0087	0.0142	17.949	13.3904	33.349
100,000.00	1.7607	0.1523	0.5918	0.0794	0.0167	0.0263	33.831	25.5835	69.011
1,000,000.00	10.2255	1.4839	3.8882	0.385	0.0815	0.1262	534.418	288.8207	732.8062

Whilst, on CPU i7-2620m (Table 5), at text size 100, the programming language Python (0 ms) is faster than Java (0.0003 ms) and C# (0.0011 ms).

Table 6. Running time of Boyer Moore Good Suffix algorithm in C#, Java and Python programming languages

Boyer Moore Good Suffix Algorithm									
Text size (characters)	C#			Java			Python		
	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)
100.00	0.0147	0.0013	0.0075	0.0502	0.045	0.0038	0.333	0	0.3
500.00	0.1087	0.0047	0.0083	0.0966	0.0501	0.0045	0.0999	0.2065	0.8135
1,000.00	0.1057	0.0049	0.0095	0.1009	0.0495	0.0047	0.6095	2.4231	0.7977
5,000.00	0.106	0.0042	0.0179	0.1137	0.0483	0.0054	2.4358	4.7497	3.363
10,000.00	0.434	0.0446	0.0287	0.1643	0.0568	0.0064	5.317	5.2542	6.2557
50,000.00	1.4238	0.1477	0.1124	0.3022	0.0762	0.0167	23.728	25.4049	33.4562
100,000.00	1.709	0.2293	0.2435	0.3776	0.0965	0.0374	44.537	48.4697	68.965
1,000,000.00	9.4931	1.7069	1.8788	0.7122	0.3052	0.1561	638.524	500.1895	1044.973

According to the figures given in Table 6, we see that Java is faster in the time complexity starting from text size 500, 5000, 10000, 50000, 100000, 1 million on CPUs i5-6200U, AMD A9-9410 RADEON R5 and i7-2620m. At text size 100, on CPU AMD A9-9410 RADEON R5, programming language C# (0.0147 ms) is faster than Java (0.0502 ms) and Python (0.333 ms) and on CPU i7-2620m at text size 1000 languages C# programmer (0.0049 ms) is faster than Java (0.0495 ms) and Python (2.4231 ms).

Table 7. Running time of Aho-Corasick algorithm in C#, Java and Python programming languages

Aho-Corasick Algorithm									
Text Size (Characters)	C#			Java			Python		
	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)	AMD A9-9410 RADEON R5 (ms)	i7-2620m (ms)	i5-6200U (ms)
100.00	0.1894	0.0311	0.0341	0.1776	0.0054	0.0805	1.5142	0.9987	2.6653
500.00	2.4155	0.3886	1.149	0.2791	0.6107	0.1034	10.683	5.995	16.761
1,000.00	5.8538	1.057	3.6734	0.3221	0.0828	0.1373	24.279	11.2952	22.358

5,000.00	27.1367	5.7329	24.0303	0.8081	0.3138	0.636	188.476	50.1705	126.885
10,000.00	32.6798	7.7263	32.2051	1.3178	0.5649	1.9564	241.673	98.7371	403.9206
50,000.00	95.4286	21.8297	63.0902	3.5016	2.0498	7.7035	1402.934	865.183	2953.963
100,000.00	122.1663	38.6392	95.8691	7.1604	3.8423	13.4349	3611.075	2231.214	6224.795
1,000,000.00	710.9564	356.3263	738.5208	51.5319	33.8784	122.0073	111579.2	114310.6	293299.4

In programming languages C#, Java and Python in the Aho-Corasick algorithm (Table 7), it is obvious that Java is faster in time complexity starting from text size 1000, 5000, 10000, 50000, 100000, 1 million in CPUs i56200U, AMD A9-9410 RADEON R5 and i7-2620m. According to data on CPU i5-6200U, at text size 100, programming language C# (0.0341 ms) is faster than Java (0.0805 ms) and Python (2.6653 ms) and on CPU i72620m at text size 500 languages programming C# (0.3886 ms) is faster than Java (0.6107 ms) and Python (5.995 ms).

Table 8. Running time of Z algorithm in C#, Java and Python programming languages

Z Algorithm									
	C#			**Java**			**Python**		
Text Size (Characters)	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**	**AMD A9-9410 RADEON R5 (ms)**	**i7-2620m (ms)**	**i5-6200U (ms)**
100.00	0.2526	0.001	0.011	0.0031	0.0001	0	0.2	0	0.3
500.00	0.3528	0.0022	0.0264	0.0045	0.0011	0.0002	0.2	0	0.799
1,000.00	0.3561	0.0018	0.0788	0.0037	0.0011	0.0004	0.609	0.6995	1.3095
5,000.00	0.5011	0.0027	0.1892	0.0139	0.0026	0.0045	3.558	2.7977	6.4652
10,000.00	0.4571	0.0129	0.1486	0.0212	0.004	0.0057	7.2608	6.0946	11.288
50,000.00	1.3906	0.0768	0.4773	0.0433	0.0147	0.0309	32.982	28.8819	58.439
100,000.00	2.3978	0.1101	0.7666	0.1119	0.029	0.0438	68.101	64.0594	116.506
1,000,000.00	6.6865	1.5972	4.3924	0.4333	0.2021	0.1767	531.993	603.0241	1151.778

In Table 8, we see that Java is faster in time complexity starting from text size 1000, 5000, 10000, 50000, 100000, 1 million on AMD A9-9410 RADEON R5, i7-2620m and i5-6200U on this CPU even at text size 100, Java programming language is faster. Python programming language (0 ms) is faster than Java (0.0001 ms) and C# (0.001 ms) and at text size 500 Python programming language (0 ms) is faster than Java (0.0011 ms) and C# (0.0022 ms) on the i7-2620m CPU, at text size 100.

3 Comparison of pattern searching algorithms in different processors

To compare the CPUs speed among themselves, only the font size of 1 million was used.

According to Tables 1–8 shown in Section 3, if we look at the figures for the C# programming language, we see that the CPU i7-2620m is faster in executing all algorithms. Then, CPU i5-6200U is ranked second in Z algorithm, Boyer Moore Bad Suffix, Boyer Moore Good Suffix, Finite Automata, Rabin Karp, KMP for execution speed. While, in the Aho-Corasick algorithms and the Naive algorithm, the CPU i5-6200U ranks last in speed. In general, it is clear that the AMD A9-9410 RADEON R5 CPU takes more time when running the Z algorithm, Boyer Moore Bad Suffix, Boyer Moore Good Suffix, Finite Automata, Rabin Karp, KMP which made it third in ranking. Referring to Tables 1–8 in Section 3, in the Java programming language they show that the CPU i7-2620m ranks first for speed in executing the Aho-Corasick, Boyer Moore Bad Suffix, Finite Automata, Rabin Karp, KMP, Naive algorithms. Whereas, in the Boyer Moore Good Suffix and Z algorithms, the CPU i5-6200U is ranked first and the CPU i7-2620m is second for execution speed. Whereas, in the Aho Corasick algorithm CPU AMD A99410 RADEON R5 is the second in the ranking for speed, while in the other algorithms it is in the third ranking for execution speed.

In Section 3, if we look at Tables 1–8, to the figures given in the Python programming language, we notice that the CPU i7-2620m is faster in executing the algorithms Boyer Moore Bad Suffix, Boyer Moore Good Suffix, Finite Automata, Rabin Karp, KMP, Naive while the AMD A9-9410 RADEON R5 CPU ranks second among these algorithms for execution speed. Meanwhile, the AMD A9-9410 RADEON R5 CPU in the Aho-Corasick and Z algorithms ranks first for execution speed, while the i7-2620m CPU in these algorithms ranks second for execution speed.

Across all algorithms, the i5-6200U CPU is ranked last for execution speed.

4 Conclusion

In this paper, a comparative study was conducted between Pattern Searching algorithms and between different CPUs using text of different sizes. What was gained as a result was that the speed of the algorithms depends on the memory of the laptop and in which programming language it is implemented. Implementation of algorithms Naive, KMP, Rabin-Karp, Finite Automata, Boyer-Moore Bad Suffix, Boyer-Moore Good Suffix, Aho-Corasick, Z Algorithm in Java, C# and Python programming language, CPU i7-2620m is faster than the other two CPUs. The result differs for the Boyer-Moore Good Suffix algorithm and the Z algorithm implemented in the Java programming language, where the i5-6200U CPU is faster than the other two CPUs. And the result differs in the Aho-Corasick and Z algorithms implemented in the Python programming language, where the AMD A9-9410 RADEON R5 CPU is faster than the other two CPUs. All algorithms implemented in Java programming language when there is more text are faster than their implementation in C# and Python programming languages using AMD A9-9410 RADEON R5 CPU, i7-2620m CPU, i5-6200U CPU. In cases where, text has much less programming languages C# and Python perform better.

In general, with the increase in characters, the execution time of the algorithms also increases in all three CPUs and programming languages, except for some cases where even though the text size increases there is better execution performance than when the text size is smaller. Pattern searching has an incredibly important role in many different fields. As it enables searching for pattern within text to be as easy as possible in so much data floating around the internet.

REFERENCES

Wirawan, I. M. A., & Paryatna, I. B. M. L. (2020). Implementation of the String-Matching Method on AnggahUngguhing Balinese Language Dictionary. *International Journal of Interactive Mobile*

Technologies (iJIM), *14*(1), pp. 15–30. [Online]. Available: https:// online-journals.org/ [Accessed: October 22, 2022]. https://doi.org/10.3991/ijim.v14i01. 11109

Prof. I.V. Srinivas, Moez Samnani, & Mohammed Shafaat Shaikh, "Study of String-Matching Algorithm", *IOSR Journal of Computer Engineering (IOSR-JCE*), *1*(7), pp. 32–35. [Online]. Available: https://www.iosrjournals.org/iosr-jce/papers/Conf.17025-2017/Volume-1/7.%2032-35.pdf [Accessed: October 4, 2021]. Mourad Elloumi, *Algorithms for Next-Generation Sequencing Data: Techniques, Approaches, and Applications*,

Springer International Publishing AG 2017. (eBook) Available: Springer. https://doi.org/10.1007/978-3319-59826-0

Akhtar Rasool, Amrita Tiwari, Gunjan Singla, & Nilay Khare. (2012). String Matching Methodologies: A Comparative Analysis. *International Journal of Computer Science and Information Technologies*, *3*(2), pp. 3394–3397. [Online]. Available: https://www.studocu.com [Accessed: October 4, 2021].

Santhosh, C., Ravi Kumar, M., Lakshmi Prasanna, J., Ram Kumar, I., Vinay Kumar, U., & Navya Sri, S. (2021). Face Mask Detection Using LabView. *International Journal of Online and Biomedical Engineering (iJOE)*, *17*(6), pp. 49–57. [Online]. Available: https:// online-journals.org/ [Accessed: October 21, 2022]. https://doi.org/10.3991/ijoe.v17i06. 21995 Saqib Iqbal Hakak, Amirrudin Kamsin, Palaiahnakote shivakumara, Gulsham Amin Gilkar,

Wazir Zada Khan, (Senior Member, IEEE), & Muhammad Imran. (2019). Exact String Matching Algorithms:

Survey, Issues, and Future Research Directions. Vol. 7, https://ieeexplore.ieee.org/ [Accessed: April 7, 2022]. https://doi.org/10.1109/ACCESS.2019.2914071

Maxime Crochemore, Christophe Hancart, & Thierry Lecroq. (2007). *Algorithms on String*, Cambridge University Press. (eBook) Available: z-lib.org

Gonzalo Navarro & Mathieu Raffinot. (2002). *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*, Cambridge University Press. (eBook) Available: z-lib.org

Dan Gusfield. (1997). *Algorithms on Strings, Trees, and Sequences Computer Science And Computational Biology*, Cambridge University Press, Available: z-lib.org

Ababneh Mohammd, Oqeili Saleh, & Rawan A. Abdeen. (2006). Occurrences Algorithm for String Searching Based on Brute-force Algorithm. *Journal of Computer Science*, *2*(1), pp. 82–85. Available: https://www.semanticscholar.org/; https://doi.org/10.3844/jcssp.2006.82.85

Peyman Neamatollahi, Montassir Hadi, & Mahmoud Naghibzadeh. (2020). Simple and Efficient Pattern Matching Algorithms for Biological Sequences. *IEEE Access*, Vol. XX, pp. 1–1. [Online]. Available: https://www.researchgate.net/ [Accessed: April 12, 2022].

Antti Laaksonen. (2019). *Competitive Programmer's Handbook*, Available: z-lib.org [13] Borivoj Melichar, Jan Holub, & Tomas Polcar. (2005). *Text Searching Algorithms Volume I: Forward String Matching*, Czech Technical University in Prague Faculty of Electrical Engineering Department of Computer Science and Engineering, Available: docslib.org