Edelweiss Applied Science and Technology
ISSN: 2576-8484
Vol. 9, No. 1, 1129-1137
2025
Publisher: Learning Gate
DOI: 10.55214/25768484.v9i1.4349
© 2025 by the authors; licensee Learning Gate

Comparing the speed of searching for a regular expression versus classic string search functions

Edwin Insuasty[1*], Jesús Insuasti[2]
[1,2]University of Nariño, Pasto, Colombia; edwin@udenar.edu.co (E.I.) insuasti@udenar.edu.co (J.I.).

Abstract: This study investigates the comparative efficiency of regular expressions and native string search methods in object-oriented programming languages such as Java and C#. String search methods, including Index of, Last Index of, and Contains, are commonly employed in programming tasks. However, their performance often deteriorates with increased text size. By contrast, regular expressions offer a versatile and powerful approach to text search, making them an appealing alternative. The research employed an empirical methodology, evaluating execution times for both approaches on four computers with varying hardware configurations. The dataset consisted of an extended version of Miguel de Cervantes' Don Quijote de la Mancha, resulting in a text size of over 122 million characters. The experiments revealed that regular expressions significantly outperformed traditional methods, achieving speeds up to 157 times faster on lower-end hardware and maintaining consistent superiority across other configurations. The findings underscore the efficiency of regular expressions for extensive text processing tasks, particularly in computationally constrained environments. Developers are encouraged to adopt regular expressions not only for their speed but also for their robustness in handling complex string operations. This research contributes to the growing knowledge of algorithm performance and provides practical recommendations for optimizing text search in programming applications.

Keywords: *Algorithms, Expressions, Methods, Regular, Search, String.*

1. Introduction

Many object-oriented programming languages incorporating the String class [1] offer string search methods such as IndexOf and LastIndexOf. These methods locate the first occurrence of a character or substring in the main string, traversing forward and backward, respectively; if the element is found, they return the position (base 0) of its first character in the main string, and if the component is not found, they return -1. In addition, the Boolean method Contains allows checking whether a character or substring exists in the treated string. These methods are widely used in programming tasks that involve searching within a character string and work effectively in most cases.

However, the performance of these methods can suffer when searching large strings. In such cases, searching for substrings can take several seconds or even minutes, which can be problematic in applications requiring a fast response. For relatively short texts, the difference in search time is not humanly perceptible, leading programmers to preferentially use the String class methods.

For this research comparing the speed of regular expression searching versus traditional string search functions, the text of "*El ingenioso hidalgo Don Quijote de la Mancha*," a public domain work with a version of 1,020,351 characters, was used. A text string was created from this work, joined together 120 times, resulting in a total of 122,442,122 characters. The efficiency evaluation was carried out using

History: Received: 6 December 2024; Revised: 13 January 2025; Accepted: 17 January 2025; Published: 20 January 2025
* Correspondence: edwin@udenar.edu.co

several computers with different characteristics to compare the speed of regular expressions versus traditional string search methods.

2. Literature Review

The comparison of regular expressions and native string search methods in programming has garnered significant attention, particularly in the context of algorithm efficiency and text processing. Formalized in the mid-20th century, regular expressions serve as a robust tool for pattern matching and text manipulation in computational applications [2]. Their versatility extends across domains such as data validation, search-and-replace operations, and syntax checking [3].

Studies have consistently highlighted the efficiency of regular expressions over conventional string methods in specific scenarios. For example, Navarro [2] illustrates that regular expressions outperform native methods when dealing with complex patterns or large datasets, primarily due to their optimized implementation in modern programming languages. Similarly, Lovos and Gibelli [4] evaluate various algorithmic approaches, emphasizing that while traditional methods like IndexOf in C# and Java are straightforward and widely supported, they lack the computational speed of regular expressions in high-volume text processing tasks.

Balari's empirical results [3] align with theoretical analyses, showing that regular expressions leverage deterministic finite automata (DFAs) and non-deterministic finite automata (NFAs) to achieve superior performance in repetitive and rule-based searches. Practical evaluations conducted by OpenWebinars [5] further reinforce these findings, demonstrating that regular expressions retain their efficiency advantage in resource-constrained environments over traditional search functions.

The impact of hardware architecture on string search efficiency has also been a focal point. Studies by Intel Corporation [6] and Advanced Micro Devices Inc [7] reveal that processor design and clock speeds significantly influence the performance of both search methods. Despite these dependencies, regular expressions consistently exhibit lower execution times than traditional approaches, as reported by Microsoft [8].

While traditional string methods remain integral to programming due to their simplicity and wide availability, the advantages of regular expressions in computational speed and versatility make them indispensable for modern text processing. Considering this review, it is essential to highlight the need for further research to optimize regular expression engines and explore their integration with emerging programming paradigms.

3. Theoretical Background

A Turing machine is a fundamental theoretical model in the theory of computation. It consists of an ideal device with an infinite tape in both directions, on which symbols can be written and read using a head, following a predefined set of rules. This machine allows the simulation of the operation of computational algorithms, facilitating the study of their characteristics and the problems associated with their implementation on real machines.

Alan Turing defined the machine concept automatic as: "If at each stage the motion of a machine (in the sense of § 1) is completely determined by the configuration, we shall call the machine an 'automatic machine' (or a-machine)" [9]. According to this definition and the additional ideas proposed by Turing, the automatic operation of the machine implies that the process develops without human intervention during its execution. "What is essential for such a model to work is that the operations can be carried out mechanically and autonomously, that is, once a sequence has been started, it can continue until its conclusion without the need for future interventions by the user" [10].

Turing machines are valuable tools for analyzing the computational complexity of algorithms by evaluating the resources required for their execution, such as time and memory. Time complexity refers to the number of steps a Turing machine needs to complete a task, although it is measured in clock cycles in practice.

Measuring an algorithm's complexity is a complex process that requires advanced knowledge of Computational Complexity. When comparing two algorithms, such as regular expressions and the IndexOf method, alternative methods can evaluate their efficiency in terms of time without measuring the full computational complexity. Since the internal details of string search algorithms are not entirely known and because Turing machines are not directly applied in this context, evaluation methods based on empirical measurements are chosen.

4. Runtime of an Algorithm

The measurement of an algorithm's execution time is deeply influenced by the architecture of the computer on which it is executed. The two main computer architectures are the von Neumann and Harvard Architectures. In the former, a program's data and instructions are stored in a single memory, while in Harvard Architecture, they are stored in separate memories [11]. Currently, Harvard Architecture is more predominant since it allows the simultaneous and overlapping execution of instructions, which increases processing speed.

Another critical factor affecting processing efficiency is the computer's clock speed or frequency, measured in gigahertz (GHz), representing the number of cycles per second. Intel defines a cycle as "a synchronized pulse generated by an internal oscillator, which helps to understand the speed of a CPU. During each cycle, billions of transistors inside the processor open and close" [12]. The ability to execute multiple instructions in a single clock cycle or the need for multiple cycles to execute a single instruction varies by processor design.

Newer generations of processors seek to improve the efficiency of executing instructions per clock cycle. Therefore, a processor with a higher clock speed but several years of age may perform worse than a newer model with a lower clock speed due to its more advanced and efficient architecture for handling instructions [12].

CPUs have multiple cores and automatically adjust the clock frequency from a base value to a maximum value (turbo mode), considering factors such as core temperature, workload, number of active cores, and power consumption [13].

Several programming languages can be used to measure time in milliseconds to estimate the execution time of an algorithm. However, this unit may be insufficient due to the precision required, especially for algorithms that run in nanoseconds, as it may report 0 milliseconds for extremely short times. Therefore, it is recommended to measure in ticks, where one tick equals 100 nanoseconds or one ten-millionth of a second [14]. If a measurement reports zero ticks, the algorithm can be executed repeatedly several times, and the total measured is divided by the number of executions to obtain an estimate in nanoseconds.

5. Regular Expressions

In Algebra within Mathematics, structures known as formal languages are studied. These languages are sets of finite or infinite strings constructed from symbols of a finite alphabet. The strings of symbols that form part of the language are called words and can vary in length, including the empty string, denoted by ε, which has a zero length.

In studying formal languages, it is common to work with alphabets that consist only of letters, digits, or an alphanumeric combination. For example, some examples of alphabets are:

$$\Sigma_0 = \{a, b, c, d\}\ \Sigma_1 = \{0,1\}. \quad (1)$$

According to these alphabets, the following sets are formal languages:

$$L_0 = \{\varepsilon, aa, aab, dab, ccccbaadd\} \quad (2)$$

$$L_1 = \{0,1,01,101,100110,11111\} \quad (3)$$

Given an alphabet, infinitely many languages can be generated, but not all are relevant to theoretical study. Only those whose words follow a specific structure defined by certain rules are of interest. The Theory of Formal Languages focuses on analyzing these languages, investigating their structural properties, defining classes of structural complexity, and establishing relationships [15].

In the 1950s and 1960s, Noam Chomsky introduced a hierarchy for classifying formal languages, as illustrated in Figure 1. This hierarchy comprises the following sets of languages:
T3 = Regular languages.
T2 = Context-free languages.
T1 = Context-sensitive languages.
T0r = Recursive languages.
T0re = Recursively enumerable languages [15].

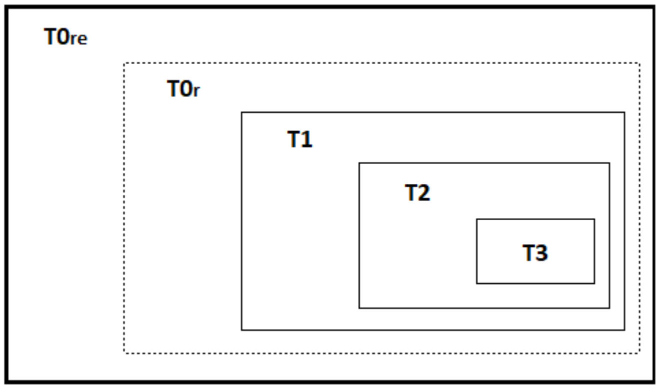


Figure 1.
Noam Chomsky's hierarchy.

Regular languages, represented as T3 in the Chomsky hierarchy, are the most basic among formal languages and are found within the set of other types of formal languages. Their structure is comparable to that of natural languages in terms of word formation, where there is a dependency between one character and the previous one. "Regular languages can be described by three different mechanisms: regular expressions (REX), deterministic finite automata (DFAs), and non-deterministic finite automata (NFAs). While some of these mechanisms are more suitable for describing languages, others are more effective for implementing efficient recognizers" [2].

In the context of this study, regular expressions will be used as an alternative to the string search methods present in modern programming languages. Regular expressions are logical formulas built from specific syntactical rules, used to perform operations on character strings. These expressions generate a pattern of character sequences that is used to search for matches within a string, performing a given operation upon finding a match. A regular expression defines a regular language and generates a finite automaton capable of processing it.

To build a regular expression, regular characters, which are common in human language texts, and metacharacters , which indicate specific rules for building the desired pattern in a string, are used. As mentioned below: "A regular expression can consist exclusively of regular characters (such as abc) or a combination of regular characters and metacharacters (such as ab*c). Metacharacters describe certain character constructions or arrangements, for example, whether a character should be at the beginning of the line or whether a character must or can occur exactly once, more times, or less often" [16].

Table 1 presents examples of regular expressions, indicating the purpose of each and the corresponding regular expression that the regular expression engine would use to search within a base string.

Table 1.
Examples of regular expressions.

Search for any email address from the gmail.com and hotmail.com domains	(\W|^)[\w.\-]{0.25} @(gmail|hotmail)\.com(\W|$)
Search for the word "Dulcinea" exactly	Dulcinea
Find any IP address that is in the range from 170.121.1.0 to 170.121.1.255.	170\.121\.1\.\d{1,3}
Search for product references that begin with the letters AC and that can be written in different ways such as: AC 23-98745, AC-17-6547, AC# 56 6541, AC#78-3215, AC 65432	(\W|^)ac[#\-]{0,1} \s{0,1} \d{2} [\s-]{0,1} \d{4} (\W|$)

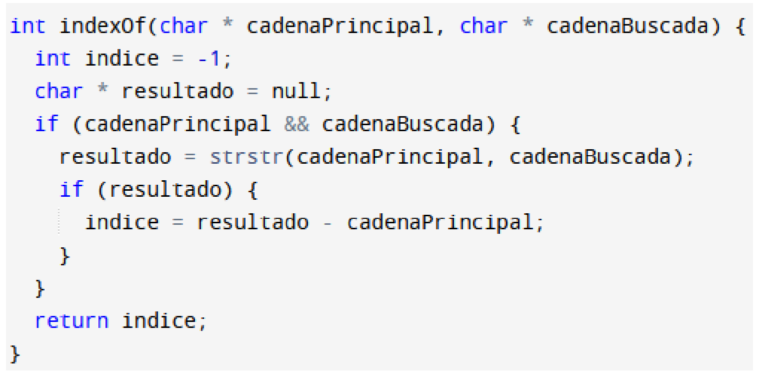
When a regular expression is used in application code, it is processed by a regular expression engine, software specialized in interpreting and executing these logical formulas. Each application development technology has its regular expression engine, which is generally incompatible.

6. Search Methods in Programming Languages

No standard function is equivalent to C#'s IndexOf or Java's indexOf in the C language. These functions calculate the index (zero-based) of the first occurrence of a substring or character within a string and return -1 if the search fails. Since these methods are frequently used in programming, they have been incorporated into languages such as Java and C#. Conversely, a custom function must be implemented in C to perform this task, as shown in Figure 2.

The indexOf method in Java has three overloads [17] while IndexOf in C# has nine [18]. This indicates greater flexibility in C# to handle different search variants. In addition, both languages provide methods for reverse lookups: lastIndexOf in Java and lastIndexOf in C#.

In C#, string search methods include overloads that accept a StringComparison parameter, an enumerator that adjusts the search based on the cultural settings of the Windows operating system. This allows you to specify different cultures, such as US English, UK English, Swedish Sami, Latin American Spanish, or Invariant Culture.



```c
int indexOf(char * cadenaPrincipal, char * cadenaBuscada) {
  int indice = -1;
  char * resultado = null;
  if (cadenaPrincipal && cadenaBuscada) {
    resultado = strstr(cadenaPrincipal, cadenaBuscada);
    if (resultado) {
      indice = resultado - cadenaPrincipal;
    }
  }
  return indice;
}
```

Figure 2.
IndexOf function in C language.

7. Previous Works

No precedents have been found comparing the efficiency of regular expressions with traditional string search methods implemented in languages such as C# and Java. Although there is abundant theoretical and empirical research on the efficiency of algorithms in general and comparative studies on the efficiency of specific algorithms, such as those used in graphs or calculating distances between words, a direct comparison between regular expressions and traditional search methods has not been documented.

Furthermore, simulators have been developed to measure the efficiency of algorithms, allowing comparisons between them. One example is the SIMULA-ALGO simulator from Argentina's National University of La Plata. This simulator uses a theoretical analysis of the algorithm [19]. However, for this study, SIMULA-ALGO is not suitable, as it requires detailed knowledge of the specific steps of each algorithm (both for regular expressions and for IndexOf), which are not publicly available.

8. Methodology

This work focuses on the concept of time complexity. As discussed above, it is not feasible to calculate the exact number of operations that each algorithm would require to generate a result. This precludes a direct theoretical comparison, making it necessary to empirically evaluate the efficiency of string search algorithms under controlled computational conditions.

Four machines with different hardware characteristics and Windows operating systems have been selected for this empirical comparison. Each machine will be started safely without networking to minimize interference from non-essential applications and processes that could affect performance during the experiment. This will ensure that the software comparing the two algorithms can run without interference, thus allowing optimal processor utilization.

The text of the literary work *El ingenioso hidalgo Don Quijote de la Mancha* by Miguel de Cervantes Saavedra was chosen for testing. Because this work is in the public domain, according to the Berne Convention [20] it can be used freely. The original text contains 1,020,351 characters. Since the search in a string of this length does not show significant differences between the algorithms, the original text has been extended by repeating it 120 times, resulting in a string of just over 122 million characters (122,442,122).

To provide a more representative evaluation, the test software will use both algorithms to search for all occurrences of a word or phrase within this expanded string. This will allow for a more accurate comparison of each method's efficiency under intensive search conditions.

9. Building the Application for the Experiment

Today, two popular general-purpose object-oriented programming languages are C# and Java. Both languages offer string-searching methods such as IndexOf and support regular expression implementation, making them suitable for developing applications that compare different string-search methodologies. While C# compiles applications directly to native code, Java is characterized by its semi-compiled approach. Its executables must be interpreted by the Java Virtual Machine (JVM), software available for multiple operating systems. These differences in the architecture and execution of the languages can influence their performance, as illustrated in Table 2, where the performance of C# and Java has been evaluated using three classical algorithms.

Table 2.
C# *Performance* vs. Java. Data based on [21].

k-nucleotide					
Source	**Secs**	**Mem**	**GZ**	**CPU**	**CPU load**
C# Net core	5.58	187,032	2574	18.81	70% 93% 81% 95%
Java	8.74	470,116	1812	27.12	71% 73% 93% 74%
Mandelbrot					
Source	Secs	Mem	GZ	CPU	CPU load
C# Net core	5.86	66,856	794	22.99	99% 98% 98% 99%
Java	6.96	76,316	796	27.06	97% 97% 97 %97%
Source	Secs	Mem	GZ	CPU	CPU load
C# Net core	2.98	1,035,064	1621	7.18	74% 77% 36% 56%
Java	3.323	609,712	2183	7.54	64% 44% 44% 76%

The C# programming language has been selected due to its faster execution speed than Java. The application developed to compare the two string search methodologies has been set up to perform 100 tests. In each test, both algorithms are executed in the following order: first the algorithm that uses the IndexOf method and then the one that uses Regular Expressions. The execution times of each algorithm are measured in clock ticks and are compared in terms of how many times faster one is than the other. One tick is equal to 100 nanoseconds.

The .NET Stopwatch class, which is in the System.Diagnostics namespace [8] has been used to measure execution times. This class provides methods and properties that allow the time interval between a start time and an end time to be measured with high precision, which is ideal for evaluating the application's performance or the execution time of a specific process within it.

In these tests, the string to be searched for in the text composed of 120 copies of the work "*Don Quijote de la Mancha*" is "*Dulcinea del Toboso*", which appears 4200 times in the text used for the experiment.

10. Running the Comparison Application

Table 3 shows the characteristics of the machines where the tests were performed. Regarding hardware, machines 1 and 2 have an exemplary configuration, machine 3 would be classified as having an intermediate configuration, and machine 4 corresponds to a low configuration.

Table 3.
Machines for the experiment.

Machine	**1**	**2**	**3**	**4**
Processor	**Intel Core i9**	**Intel Core i7**	**Intel Core i7**	**AMD E-350**
Generation	11	10	2	
Speed	2.4 GHz	1.3 GHz	2 GHz	1.6 GHz
Memory	32 Gb	16 Gb	16 Gb	2 Gb
Hard disk technology	M2	M2	SSD	HDD
Operating system	Windows 11	Windows 11	Windows 10	Windows 7

Each machine was booted in Safe Mode without Networking to run the application designed for this experiment. The application loads the base text for the 122,442,122 character searches into memory, and the loading times on each machine are shown in Table 4 in terms of minutes, seconds, and milliseconds. There is no significant difference between machines 1 and 2, but with machine 3, there is a noticeable delay in loading the text. With machine 4 with low hardware configuration, the difference is vast,

Table 4.
Base text loading times.

Machine	**1**	**2**	**3**	**4**
Load time (Minutes: Seconds)	1:10.284	1:26.355	3:11.506	38:34.160

11. Results

Table 5 shows the averages of the 100 tests performed on each of the four machines. The averages of the execution times were calculated using the *IndexOf method* and Regular Expressions (RegEx). In each test, the number of times that the algorithm with *RegEx* was faster than the algorithm that used IndexOf was also calculated.

Table 5.
Results in averages.

Machine	**1**	**2**	**3**	**4**
Index of (Average ticks)	40,061,760	41,996,368	98,954,396	343,350,024
Regex (Average ticks)	604,121	596,172	1,387,242	2,261,146

Number of times regex beats index of	66.36X	70.43X	71.36X	157.285X

To understand the data obtained, we will take the data collected from Machine 1 and convert the *ticks* to seconds and fractions of a second; 1) The algorithm with *IndexOf* that searched for the 4200 times that the phrase "*Dulcinea del Toboso*" appears in the chain of 120 books together of Don Quixote de la Mancha, took 40,061,760 ticks which corresponds to 4,006,176,000 nanoseconds and this is in seconds: 4.006176 (a little more than 4 seconds); 2) The algorithm with Regular Expressions that searched the exact 4200 times that the phrase "*Dulcinea del Toboso*" appears in the chain of 120 books together of *Don Quixote de la Mancha*, took 604,121 *ticks* which corresponds to 60,412,100 nanoseconds and this is in seconds: 0.0604121 (6 hundredths of a second)

It is easy to understand in these terms that the difference between 4 seconds and 6 hundredths of a second is abysmal, and this explains why the speed of Regular Expressions is approximately 66 times faster than when using the *IndexOf method* of the C# language, taking as a reference the data obtained with Machine 1.

Considering that 2X means doubling the speed of something, the data obtained in the experiment allows us to form an idea of the enormous speed with which Regular Expressions work.

12. Conclusions

As already noted in the previous section, Regular Expressions significantly outperform the methods built into the programming language for searching character strings, regardless of the hardware configuration and operating system of the machine where they are used.

The difference between using these search techniques on machines with newer and near-generation processors is not very large, but they show cross-over results. For example, the i9 processor of machine one used in this experiment improves the performance of the i7 processor of machine two by 4.6% when using IndexOf but loses 1.3% in performance when using Regular Expressions.

Two i7 processors whose generations are far apart (10th and 2nd) have a significant performance difference when applying these two search techniques; the 10th-generation i7 processor is faster than the 2nd-generation i7, even though the slower processor's clock speed is higher than that of, the faster processor. However, the speed ratio between RegEx and IndexOf on machines 2 and 3 is almost the same.

Regular expressions are much more efficient on a low-hardware-configuration machine with an old operating system like number 4 than on very well-configured machines. This reinforces the suggestion of using regular expressions instead of the String class methods in these tasks.

An application developer should use regular expressions in strings-related tasks, such as searching, comparing, replacing, and checking syntax conditions, such as checking whether an email address is spelled correctly.

Initially, regular expressions may seem to have highly complex syntax constructions. However, much material on the Internet allows one to understand their elaboration, with very well-selected examples in a didactic sequence suitable for beginners [22, 23] and documentation for advanced programmers [24].

Transparency:

The authors confirm that the manuscript is an honest, accurate, and transparent account of the study; that no vital features of the study have been omitted; and that any discrepancies from the study as planned have been explained. This study followed all ethical practices during writing.

Acknowledgements:

The authors declare no conflict of interest. The authors wish to extend their appreciation to the Galeras.NET research group at the University of Nariño. This research was self-funded by the research group, and the computational resources provided by the Galeras.NET research group at the University of Nariño were utilized. Furthermore, the authors would like to thank the undergraduate students in Systems Engineering Department at the University of Nariño, who are active members of the Galeras.NET Research group, for their involvement in conducting the experimentation phases. The participants who contributed to this study signed the confidentiality agreement and the research ethics rules proposed by the Institutional Review Board Statement of the University of Nariño, available at https://viis.udenar.edu.co/wp-content/uploads/2022/01/Reglamento-CEI-2022-.pdf.

Copyright:

References

[1] M. D.-S. C. Microsoft, "Microsoft, microsoft docs - string class," Retrieved: https://docs.microsoft.com/en-us/dotnet/api/system.string?view=net-6.0 2021.
[2] G. Navarro, *Theory of computation: Formal languages, computability and complexity*. University of Chile, Editorial UC, 2017.
[3] S. Balari, *Formal language theory. An introduction for linguists*. Autonomous University of Barcelona, 2014.
[4] E. Lovos and T. Gibelli, "Analysis of algorithm efficiency: A comparative study," *Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología*, vol. 7, pp. 36-41, 2012.
[5] OpenWebinars, "Performance comparison: C# vs Java," Retrieved: https://openwebinars.net/blog/performance-c-vs-java. [Accessed Dec. 7, 2024], 2019.
[6] Intel Corporation, "CPU clock speed: An overview," Retrieved: https://www.intel.la/content/www/xl/es/gaming/resources/cpu-clock-speed.html. [Accessed Jan. 1, 2025], 2022.
[7] Advanced Micro Devices Inc, "Ryzen technology performance enhancement," Retrieved: https://www.amd.com/es/support/kb/faq/cpu-pb2. [Accessed Jan. 12, 2025], 2022.
[8] Microsoft, "Stopwatch class," Retrieved: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-6.0. [Accessed Jan. 17, 2025], 2020.
[9] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Journal of Mathematics*, vol. 58, no. 345-363, p. 5, 1936.
[10] A. Ilcic and P. García, "Modeling strategies in Alan Turing: Machine terms and concepts," *Topics*, no. 58, pp. 135-155, 2020. https://doi.org/10.21555/top.v0i58.1090
[11] S. Barrachina Mir *et al.*, "Introduction to computer architecture with QtARMSim and Arduino," 2018.
[12] Intel Corporation, "What is clock speed," Retrieved: https://www.intel.la/content/www/xl/es/gaming/resources/cpu-clock-speed.html 2022.
[13] Advanced Micro Devices Inc, "AMD Ryzen™ technology: Performance boost with precision boost 2," Retrieved: https://www.amd.com/es/support/kb/faq/cpu-pb2 2022.
[14] Microsoft, "TimeSpan struct," Retrieved: https://docs.microsoft.com/en-us/dotnet/api/system.timespan?view=net-6.0. [Accessed 2022.
[15] S. Balari, *Theory of formal languages. An introduction for linguists*. Spain: Autonomous University of Barcelona / Centre for Theoretical Linguistics, 2014.
[16] Digital Guide IONOS, "RegEx or regular expressions: The easiest way to describe character sequences," 2022.
[17] W3Api, "String.indexOf() | Java," Retrieved: http://www.w3api.com/Java/String/indexOf/ 2022.
[18] Microsoft, "String.Indexof method," 2018.
[19] E. Lovos and T. I. Gibelli, "Algorithm efficiency analysis. Simulation as a teaching-learning resource," *Revista Iberoamericana de Tecnología en Educación y Educación en Tecnología*, no. 7, pp. 36-41, 2012.
[20] World Intellectual Property Organization, *Berne convention for the protection of literary and artistic works*. Paris, France: WIPO, 1979.
[21] OpenWebinars, "Performance C# VS JAVA," Retrieved: https://openwebinars.net/blog/performance-c-vs-java/ 2019.
[22] MakeltReal Camp, "Regular expressions," Retrieved: https://blog.makeitreal.camp/expresiones-regulares/. [Accessed 2019.
[23] Microsoft, "Regular expressions language - quick reference," Retrieved: https://docs.microsoft.com/es-es/dotnet/standard/base-types/regular-expression-language-quick-reference 2019.
[24] Regular-Expressions.Info, "Regex tutorial," Retrieved: https://www.regular-expressions.info/tutorial.html 2022.