International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 11 | Nov - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

String Matching Algorithm -A Comparative Survey

1st Om Mangle
Student, Department Of Electronics And Telecommunication Engineering, BRACT's Vishwakarma Institute Of Information Technology, Pune, India

2nd Gaurav Bhaltilak
Student, Department Of Electronics And Telecommunication Engineering, BRACT's Vishwakarma Institute Of Information Technology, Pune, India

3rd Sushant Pawar
Student, Department Of Electronics And Telecommunication Engineering, BRACT's Vishwakarma Institute Of Information Technology, Pune, India

4th Burhanuddin Mal
Student, Department Of Electronics And Telecommunication Engineering, BRACT's Vishwakarma Institute Of Information Technology, Pune, India

5th Dr. Jayashree Tamkhade
Assistant Professor, Department Of Electronics And Telecommunication Engineering, BRACT's Vishwakarma Institute Of Information Technology, Pune, India

Abstract—String matching is a problem with many appli-cations, ranging from simple text processing to complicated bioinformatics and plagiarism detection. This paper surveys four of the string matching algorithms most commonly in use for string matching: Naive, Rabin-Karp, Knuth-MorrisPratt (KMP), and Boyer-Moore. Each of these algorithms uses a different approach to a solution of the described problem. Some algorithms have distinct advantages and difficulties regarding time and space complexity and practical performance.

The naive algorithm is easy to understand but very inefficient for large texts since its time complexity is O(m*n). Rabin-Karp uses hashing in an attempt to speed this up and has an averagecase time complexity of O(m + n) but can be slow as O(m
* n) if hash collisions continually happen. The KMP algorithm develops this further through preprocessing of the pattern to get a constant time complexity of O(m + n) and hence make it very suitable for single-pattern searches. Further optimization is done in the Boyer-Moore algorithm; it scans the pattern from right to left, skips parts of the text, hence achieving the best cases of O(m + n) with worst cases rarely going to O(m * n).

This survey makes a deep analysis and comparative study on these algorithms, focusing on theoretical Aspects, practical implementations, and performance metrics to give an all-rounded insight into these algorithms. Conclusions from the survey will
help choose the best-fit algorithm for an application given a scenario so that the best performance could be achieved considering any string matching scenario.

I. Introduction

String matching is one of the most simple and important problems in computer science, being at the heart of numerous applications involving text manipulation and bioinformatics. In string matching algorithms, the goal is to find occurrences of a given ”pattern” string within a larger ”text” string [3]. These are very fundamental yet computationally expensive algorithms, especially with the increase in the size of the text and pattern complexity.

This survey undertakes a study and comparison of four fa-mous string matching algorithms: Naive, Rabin-Karp, Knuth-Morris-Pratt (KMP), and Boyer-Moore. Each algorithm attacks the string matching problem in a different way, with various thingsregarding time and space complexity, and practical per-formance.

The survey will provide an in-depth critical review of the comparison between algorithms, underlining the theoretical background, the practical implementation, and the perfor-mance measures. Once these relative strengths and weaknesses are well understood for each algorithm, then an informed decision could be made regarding the applicability of any of them in practical situations for optimum and effective real-world implementation.

II. Background

String matching is one of the basic operations in computer science, which may serve as the foundation for many important applications: text editors, search engines, plagiarism detection, and many others. The problem involves finding all occurrences of a pattern string within a text string [2]. That is simple to state, not necessarily easy to do, especially when the text is very long or when complex patterns are being sought.

String matching algorithms have been developed with the view of making them efficient by reducing computation time. Early simple solutions like the Naive algorithm were expensive in terms of time and space complexity for large texts. This resulted in the development of more techniques that minimized useless operations and optimized the performance. Amongst the first few proposed methods, the Naive Algorithm was one of them. This method offers simplicity and ease of implementation but definitely suffers in performance for large-scale applications due to its time complexity [4]. So we have other algorithms which are more optimised.

In the present survey, a deep consideration is given to each of these four algorithms by comparing their theoretical underpinnings, practical implementation issues, and empirical performance. A discussion about the strengths and weaknesses of each can be found here, with a view towards gaining insight

into their applicability in real-world scenarios that shall help practitioners choose among them for particular needs.

III. LITERATURE SURVEY

Reference	Objective/Focus	Methodology	Results/Findings	Limitations	Relevance to Your Work
AbdulRazzaq et al. (2013)	Review of exact string matching algorithm efficiency	Analyzed the efficiency of various algorithms by examining comparison attempts and runtime	Found suffix automata and hybrid algorithms to be the fastest with fewer attempts, while bit-parallelism algorithms have limitations	Limitations with bit-parallelism algorithms in terms of performance efficiency in large datasets	Relevant for understanding the efficiency of various exact string matching algorithms
Myers (1999)	Develop a fast bit-vector algorithm for approximate string matching	Used dynamic programming and bit vector operations to improve algorithmic performance	Achieved superior efficiency with O(nm/w) time, outperforming previous 4- Russians algorithm and bit-vector methods	Limited application when k/mk/mk/ m ratio is too small, where filter based algorithms are more efficient	Provides advanced techniques for approximate string matching with faster algorithms
Gallagher (2006)	Survey on graph-based pattern matching techniques	Reviewed techniques for graph matching, focusing on structural and semantic variations	Emphasized challenges in matching patterns in semantic graphs with millions of typed vertices and edges	NP-completeness of subgraph isomorphism problem poses scalability challenges in large graphs	Useful for applications that involve graph based matching in large scale datasets
Alqahtani et al. (2021)	Survey of text matching techniques	Explored text mining, text clustering, and NLP for matching, with emphasis on string-based approaches	Demonstrated the effectiveness of cosine similarity and deep learning-based clustering for text matching	Limitations in scalability and handling large datasets in some text clustering methods	Provides insights into text matching techniques applicable for large datasets
Someswara rao et al. (2011)	Survey and performance evaluation of parallel algorithms for string matching	Evaluated parallel algorithms for string matching on different computing models	Showed that parallel algorithms significantly improve performance in large datasets	Lack of generalization for certain algorithms across different hardware architectures	Highlight s the advantages of parallel algorithms in high-performance environments
S. I. Hakak et al. (2019)	Review of exact string matching algorithms	Survey of various single pattern matching algorithms and their classification	Proposed new taxonomy for exact string matching algorithms	Difficulty in selecting appropriate algorithms for specific applications	Helps identify suitable exact string matching techniques for specific needs
Syeda S. Hasan et al. (2015)	Comparison of approximate string matching algorithms	Comparison of Brute Force, Lipschitz, and Ball Partitioning algorithms	Lipschitz Embeddings more efficient than others	Brute Force method is inefficient in large databases	Provides insight into efficient approximate matching methods for large datasets
K. M. Alhendawi and A. S. Baharudin (2013)	Review of BoyerMoore, KnuthMorris-Pratt, Rabin-Karp algorithms	Characterbased, hashing, and suffix automata approaches	Time complexity and pattern length play major roles in performance	High time complexity for large data sets	Applicable to scenarios requiring exact string matching in large databases
G. Navarro and R. BaezaYates (2001)	Exploration of hybrid indexing in approximate string matching	Hybrid algorithms combining different search methods	Reduced time complexity for approximate matching	Complexity increases with number of approximate matches	Useful for scenarios needing efficient hybrid algorithms for matching
S. Fide and S. Jenks (2008)	Study of multiple pattern search methods	Q-Gram and Non Q-Gram approaches	Hashing based methods are fast but prone to collision	Hash collisions affect accuracy	Relevant for tasks involving multiple pattern search across large data sets

Fig. 1. Literature Survey

IV. METHODOLOGY

A. Naive Algorithm

This is the most basic and simplest string matching algo-rithm It performs checking at all positions in the text whether an occurrence of the pattern starts there or not. After each attempt, the algorithm shifts the pattern by exactly one position to the right. Time Complexity of this algorithm is O(n*m) considering the worst case , where n is length of the string in which we are searching the pattern and m is pattern length .Space complexity is O(m) [1] [5].

The algorithm is simple and implementation is very easy. It is best suited for searching a pattern from small text string. It is not good for larger and more complex strings as it will take higher execution time. Applications of this algorithm is simple text searches and educational purposes.

Example, consider we have to search pattern “AAB” from text “ACAABABC”. So the match will be found at second shift . Initially there is a mismatch and there is a mismatch in first shift also so every time there is mismatch the search will shift ahead by one index.

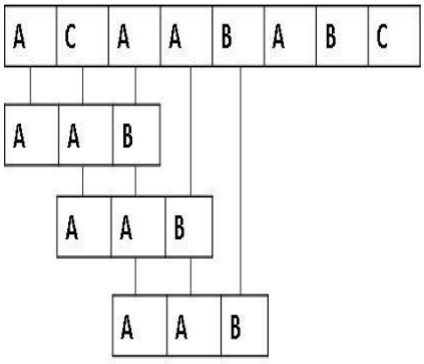

Fig. 2. Naive Algorithm

B. Knuth-Morris-Pratt (KMP) Algorithm

Knuth, Morris and Pratt introduced a linear time algorithm for string matching. Here the matching time is O(n). In KMP the execution time is lesser than that of Na¨ıve Algorithm [6]. In KMP we form suffixes and prefixes for the pattern we have to search. The idea of KMP is that is there any suffix same as prefix which means we are checking if the beginning part of pattern is appearing again in the pattern.

Here we generate pie table for a pattern which is same as the size of pattern and it is based on longest prefix that is same as suffix. Consider the following example for a pattern “ABABD”

In the below pattern suppose there is mismatch at B so the search will not backtrack to pattern index shift +1 rather it will start searching from the corresponding index number assigned

in pie table for the index where mismatch is happened. This algorithm will take n times for parsing (searching) and m for

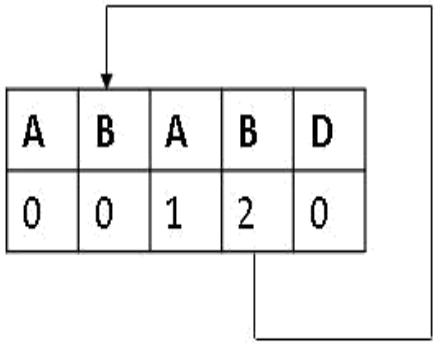

Fig. 3. KMP Algorithm

creating pie table. The algorithm has time complexity O(m*(n-m+1)) in worst case whereas is average case it is O(m +n) [10].

KMP can be used in data mining, text editors. The algorithm is faster and best suited for larger text than that of na¨ıve algorithm. Here we have additional step of creating LPS or pie table so the implementation is complex and not efficient for smaller texts.

C. Boyer Moore Algorithm

The Boyer-Moore algorithm is an efficient pattern-searching technique in a given text. The algorithm finds all occurrences of the pattern in the text using mainly two important heuristics: bad character heuristic and, sometimes, good suffix heuristic. Instead, it determines how far the pattern can shift along the text after a mismatch has taken place—faster against the naive approach or Knuth-Morris-Pratt algorithm [5]. The advantage of this algorithm consists in its ability to maximize the shift distance for mismatched characters and matched suffixes, which makes it very effective for large texts and patterns.

The average time complexity of the Boyer-Moore algorithm is O(n + m), where n is the length of the text, and m is the length of the pattern.

The Boyer-Moore algorithm performs pattern searches in text using effective heuristics, namely the bad character rule and occasionally the good suffix rule [5]. It is, therefore, applicable in many fields such as in text editors, data mining, bioinformatics, and network security. While it does entail a preprocessing stage, on occasion, it is not faster than other algorithms

D. Rabin Karp Algorithm

Rabin-Karp is the string matching algorithm which uses the rolling hashing function for searching the pattern in a text [1]. The hashing approach based on hash technique for matching. Both pattern and text values comparison based on hash value. The comparison is from left to right comparing the hash of text and pattern. The hash technique has best performance because in these technique we use integer numbers which decrease the computation time [7].

The algorithm has two steps pre-processing and searching. In preprocessing string is converted into decimal numbers. And computing the hash value for text and pattern. And in searching phase comparison of hash value is done if they match then compare the string character by character [9]. Time complexity of the algorithm for Average case O(n + m) , Worst Case it is : O ((n-m+1).m) where n is the length of the text and m is the length of the pattern [9].

Advantages are Better when searching the multiple patterns in the text, The average case time complexity is fast and linear, Implementation is easy compared to other string matching algorithm [1].

Disadvantages are Hash collisions when different substrings generate same hash value, False positive where hash value match but string might not requires the multiple string com-parison, The algorithm requires extra memory to store hash values for the pattern and substrings of the text [1].

Eg:

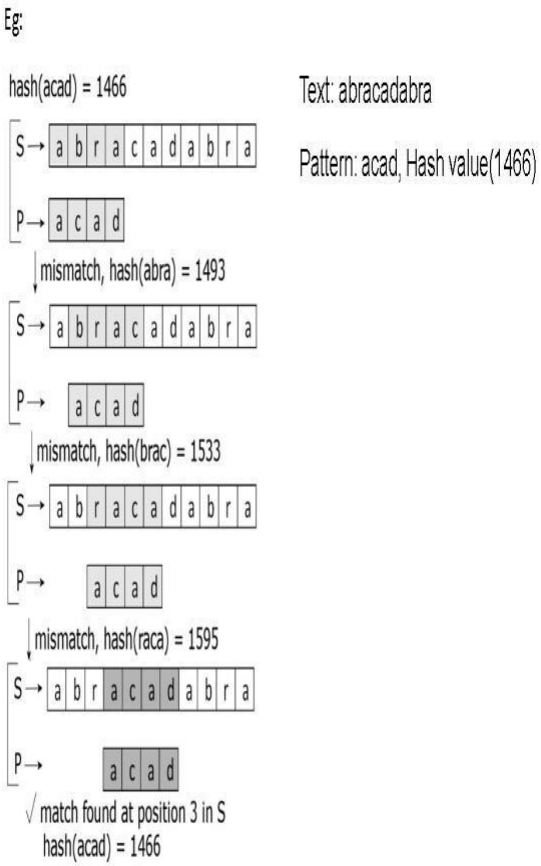

Fig. 4. Robin Karp Algorithm

E. Comparison Table

Algorithm	Preprocessing time	Time Complexity	Key Idea	Best Use Case
Naive (BruteForce)	O(1)	O(m * n)	Checks for the pattern at every position in the text	Small patterns, simple implementations
Knuth-Morris-Pratt	O(m)	O(m*(nm+1))	Uses prefix function to skip comparisons	Patterns with repeating prefixes
Boyer-Moore	O(m)	O(n + m)	Uses bad character and good suffix heuristics	Large alphabets, long patterns
Rabin-Karp	O(m)	O ((n-m+1) ·m)	Uses hashing to find any one of a set of pattern strings	Multiple pattern searches

Fig. 5. Comparison Table

V. Applications

A. Plagiarism Detection:

Plagiarism detection relies importantly on string match-ing algorithms, representing a process of finding copied or paraphrased parts of documents. Generally speaking, these algorithms compare word and character sequences in order to find similar ones. Several methods, such as exact matching, shingling, and others, allow for the direct detection of identical or almost identical sequences. Also, there is a fingerprinting technique that generates special hash values of sequences, allowing one to compare documents effectively and find suspi-cious matches. Apart from that, contextual similarity analysis also detects semantic and syntactic aspects to discover para-phrased or reworded text. Using these integrated approaches, the plagiarism detection system will identify material whether duplicated verbatim or paraphrased, hence protecting academic integrity from intellectual property theft.

B. Data Mining:

String matching algorithms play a significant role in data mining, relating to large text datasets for finding patterns or extracting relevant information. These algorithms include the determination of occurrences of any given substring (pattern) within a larger string (text), and assessment for similarity between two different strings. Basically, algorithms related to strings are of much importance for data mining, including tasks that pertain to keyword identification, detection of du-plicate entries, clustering like texts, or matching a user query against the most appropriate document. Efficient algorithms, such as Knuth-Morris-Pratt (KMP) and Boyer-Moore, achieve this by reducing the number of comparisons required to find patterns. These techniques amply enhance the speed and accu-racy of text mining, thereby helping in information retrieval, text classification, and natural language processing through locating relevant patterns in a large volume of unstructured data with speed.

C. Web Searches:

String matching algorithms like Knuth-Morris-Pratt (KMP), Rabin-Karp, and Boyer-Moore are essential in optimizing web searches for efficient query matching. KMP improves search efficiency by avoiding redundant comparisons, particularly useful for exact string matching in large datasets. Rabin-Karp uses hashing to search for patterns in a document, which allows it to handle multiple pattern searches, though it can face hash collisions. Boyer-Moore optimizes the search process by skipping sections of the text where mismatches occur, making it faster, especially for longer patterns. These algorithms are vital in ensuring quick and accurate retrieval of relevant documents in web searches.

VI. CONCLUSION

In other words, the string matching algorithm depends basically upon the nature of the problem arising and the other parameters involved. While the Naive algorithm is inefficient for big texts and big patterns, it is easy to implement and therefore sometimes appropriate for little texts and/or little frequency search. The KMP algorithm, while it guarantees linear time, presents a good alternative for when there are multiple searches of a fixed pattern, for the fact that it requires a preprocessing overhead in the beginning, in addition it can be utilized for all the subsequent searches. Rabin-Karp is advantageous for searching multiple patterns at once, though hash collisions can occasionally reduce its efficiency. Finally, the Boyer-Moore algorithm excels in practice, especially with larger alphabets and when the pattern is much shorter than the text, making it an excellent choice for real-world applications where speed is crucial. Each algorithm presents different tradeoffs, so understanding their benefits and limitations is key to choosing the right one for any given task.

References

[1] Saqib Iqbal Hakak, Amiruddin Kamsin, Palainahnakote Shivakuma, Gul-shan Amin Gilkar, Wazir Zada Khan, Muhammad Imran, ”Exact String Matching Algorithms: Survey, Issues, and Future Research Directions” 2019, IEEE Access Special Section On New Trends In Brain Signal Processing And Analysis.

[2] Syeda Shabnam Hasan, Fareal Ahmed, Rosina Surovi Khan ”Approxi-mate String Matching Algorithms: A Brief Survey and Comparison”, International Journal of Computer Applications, 2015, vol-120, pp. 0975-8887.

[3] Chinta Someswararao, K. Butchi Raju, S.V. Appaji, S. Viswanadha Raju and K.K.V.V.V.S. Reddy, ”Recent Advancements in Parallel Algorithms for String Matching on Computing Models – A Survey and Experimental Results” ADCONS 2011, 2012, pp. 270-278.

[4] Awatif Alqahtani, Hosam Alhakami, Tahani Alsubait, Abdullah Baz, ”A Survey of Text Matching Techniques ” 2021 Engineering, Technology & Applied Science Research , vol-11 pp. 6656-6661.
[5] Brian Gallagher, ”Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching”, American Association for Artificial Intelligence, 2006.
[6] Gene Myers, ”A Fast Bit-Vector Algorithm for Approximate String Matching Based on Dynamic Programming” 1999, Journal of the ACM, vol-46, pp. 395-415.
[7] Akram Abdulrazzaq, Atheer & Abdul Rashid, Nur’Aini & Hasan, Awsan & Abu-Hashem, Muhannad, ”The exact string matching algorithms efficiency review,” 2013, Global Journal on Technology
[8] Dany Breslauer, Zvi Galil, ”Efficient comparison Based String Matching” 1993, Journal of Complexity, pp. 339-365.
[9] Diwate, Rahul., ”Study of Different Algorithms for Pattern Matching,” 2013, International Journal of Advanced Research in Computer Science and Software Engineering.
[10] Vidya SaiKrishna, Prof. Akhtar Rasool and Dr. Nilay Khare, ”String Matching and its Applications in Diversified Fields” 2012, IJCSI International Journal of Computer Science, vol-9.