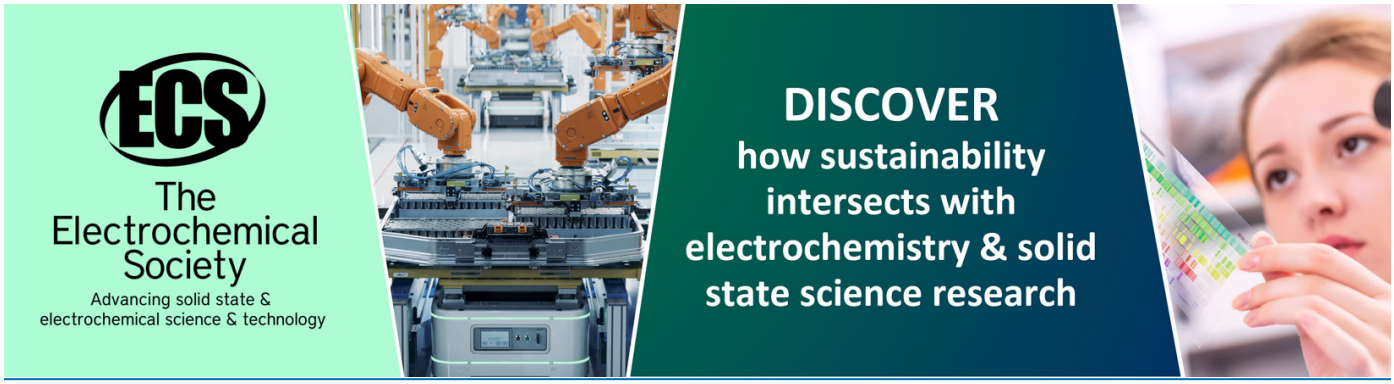
Prevalent Exact String-Matching Algorithms in Natural Language Processing: A Review



Prevalent Exact String-Matching Algorithms in Natural Language Processing: A Review

R K Pandey[1] and S Taruna[2]

Department of Computer Science, JK Lakshmipat University, Jaipur, India.

[1] pandeyravik@gmail.com, [2] staruna@jklu.edu.in

Abstract. String matching and searching problems are one of the classical hitches to the domain of computer engineering. There exist numerous variants of such algorithms, which can be categorized into dual classes; i.e. an approximate match and exact match algorithms. This paper engrossed on comparing prevalent algorithms of exact matching category in terms of their functionality and complexity along with critical opinions for a better understanding of the differences among them. The application of string matching is used in various fields like for intrusion detection in the networks, DNA matching in the field of bioinformatics, plagiarism checking for fraud detection, in the field of information security, pattern recognition, text mining, web searching, recommendation system, the document comprising, authentication system and web scraping. The uses are not only restricted to the fields termed above but also the notion has copious advantages for ongoing and forthcoming research work.

1. Introduction

The string matching is the key to several valuable algorithms used in a tangible situation. Many times, string matching and searching algorithms are considered to be the vital base algorithms for high graded research works, and to get its privilege researchers changed it according to their needs. Like if you have to perform a search in a big data environment, the parallel search algorithm is used whereas if the same algorithm used with other scenarios with the small data set, the algorithm may show low efficiency. If we require an approximate match to some string then we have to use an algorithm from the bunch of approximate match algorithm. This paper is intensively focused on a few prevalent exact match algorithms. The readings are not limited to the algorithms reviewed here only but several uncommon algorithms may perform better in your situation. Among the algorithms discussed in the paper, the Naive Algorithm is considered to be the easiest and the oldest one; but in terms of the performance, it lacks behind to the other presented algorithms.

2. Exact_String_Matching Algorithms

2.1. Brute_Force (or Naïve) Algorithm [1][13]

This algorithm is easy and coherent in technique. The algorithm has the absence of the pre-processing segments and it also doesn't require additional space while the searching process. In the algorithm, the window slides the pattern over text one by one to the right after each comparison and check for a match. If a match found, then slides by once more to check for successive matches. Naive Algorithm is good for short text inputs. The overall comparisons of character expected to be 2*n in numbers

deprived of any fixed directive. The searching time complexity is in the order of O(n*m) for the Brute_Force (Naïve) algorithm.

2.2. Searching with Automata [2]

The algorithm creates a minimal Deterministic Finite Automata, which recognizes the language generated by expression Σ*x. The extra space of order of O(σ*m) is being obligatory If the automata are stored in the direct access table. Order of O(n) and O(σ*m) are the time complexities for the searching and pre-processing segment for the algorithm respectively.

2.3. Rabin_Karp Algorithm [3] [4]

The algorithm had usages rolling hash function as the fundamental technique in processing. The algorithm is widespread to work with multiple pattern searches in contrast to a single pattern search. The Rabin_Karp algorithms accomplish its rapidity by using a hash function, as hash function converts each word to a definite integer employing a distinct procedure. So, in searching, if two corresponding integers are equals it indicates the match to the searched word. In the pre-processing segment, the algorithm requires a fixed space, its time complexity will be in the order of O(m), the searching time and run-time complexities are in order of O(n*m) and O(n+m)respectively.

2.4. Shift_OR Algorithm [5]

The effectiveness of the Shift_OR algorithm is high for the patterns with a shorter length in reverence to the machine word. The algorithm searches string by performing normal plotting using modest bitwise operation, due to this algorithm even popular with the name of the Bitap Algorithm. The algorithm could quickly adapt to solve even the approximate string-matching problems. O(σ+m) is the space and time complexities for the pre-processing segment of the algorithm, whereas the search time complexity is in order of O(n) for the Shift_OR algorithm.

2.5. Morris_Pratt Algorithm [6]

This algorithm is independent of the size of the alphabet to be searched. The comparisons while searching are performed in the left to the right direction. In the searching segment, the maximum number of the character comparison cannot exceed the 2*n-1 with the delay constrained as the limit of m. Space and time complexities for the pre-processing segment are in order of O(m) and for the searching, it is in order of O(n+m).

2.6. Knuth_Morris_Pratt Algorithm [6]

The size of the alphabet feeds to search did not affect the effectiveness of the algorithm. Here also the comparisons while searching are performed in the left in the right direction. While searching the segment, the maximum number of the character comparison are cannot exceed from 2*n-1. The delay in algorithm bounded within the limit of logΦ(m), where the Φ symbol is the golden ratio having the value of (1+√5)/2.Space and time complexities of the pre-processing segment are in order of O(m) and the searching segment is in order of O(n+m). This procedure is measured to be the initial algorithm performing in a linear period for string comparing.

2.7. Colussi Algorithm [7] [8]

The Colussi Algorithm is an improved variation of the algorithm discussed in 2.6 section of this paper. In the course of the algorithm, the pattern set is separated into two groups. After which the first portion is perused in the left to the right direction and if not matched then in the remaining portion is perused in right to the left direction for finding the positions in both disjoint sets. Both the basic complexities for the pre-processing segment are in-order of O(m)whereas, the order of O(n) is the searching time complexity. In the nastiest situation, the maximum of n*(3/2) text character assessments are accomplished while searching.

2.8. Forward_DAWG_Matching Algorithm [9]

This algorithm works on the notion of the Kunth_Morris_Pratt algorithm, the comparisons while searching are performed in the left to the right direction as same as done in Kunth_Morris_Pratt algorithm. In this algorithm, the inspection of exactly n characters is performed. The suffix automata of x are practiced in the algorithm, which is a DFA (Deterministic Finite Automata- M(w) = (Q,∑, δ,q0, F)). It shows the linearity for space and time and in the nastiest situation, the time complexity is in the order of O(n).

2.9. Boyer_Moore Algorithm [10]

The comparisons are being accomplished from right to left direction. In the nastiest situation for non-periodic patterns n comparisons had done while searching. This algorithm calculates the shift in core by malicious character heuristic coexistence with the virtuous suffix heuristic. O(n/m) is the order of the most exceptional performance evaluated by this algorithm. Space and time complexities for the pre-processing segment are the same i.e. order of O(σ+m), whereas the searching time complexity is in order of O(n*m).

2.10. Quick Search Algorithm [10] [11] [12]

The Quick Search algorithm considered to be an interpretation of the Boyer_Moore algorithm. Shifting of malicious characters is the extra step involved in Quick Search in comparison to the base algorithm. Comparatively the implementation is easy. The Quick Search algorithm is highly effective for colossal alphabets and tiny patterns. For the pre-processing segment, space and time complexities are in order of O(σ) and O(σ+m) respectively. The algorithm's searching time complexity is quadratic but good for applied conduct i.e. O(n*m).

2.11. Horspool Algorithm [10] [14]

This algorithm is an informal clarification of the Boyer_Moore algorithm. The enactment of the Horspool algorithm is comparatively easy. The pattern in the Horspool algorithm is compared as it performs in Boyer_Moore i.e. from right to left in direction; in either instance of hit or miss the shifting of the pattern is accomplished by the help of a pre-computed function. Here in this algorithm, the malicious character heuristic is figured in a little different way as in comparison to the way its base algorithm. For the pre-processing segment, space and the time complexities are in order of O(σ) and O(σ+m) respectively. However, for the searching segment time complexity is in order of O(m*n).For the single character, and the average comparisons are between ( )-1 and 2(+1)-1.

2.12. Apostolico_Giancarlo Algorithm [15]

The Apostolico_Giancarlo algorithm is a result of an amendment to the Boyer_Moore algorithm. In this algorithm, every single pattern suffix is remembered and information of it is stored in a table, at the end of each go the shifts are computed. Space and time complexities for the pre-processing segment are the same i.e. in order of O(σ+m), where as the searching time complexity is in order of O(n).In the nastiest situation, the maximum of n(3/2) text character evaluations is accomplished while searching.

2.13. Boyer_Moore_Smith Algorithms [16]

In the algorithm, the shift had made while scanning. The algorithms get benefited from the extreme shift worth among the figured shift with the text. The maximum value of shift computed when the rightmost character is taken in comparison to the just next to the rightmost text character is being considered. Space and reprocessing time complexities are in order of O(σ) and O(σ+m) respectively. Order of O(n*m) is the search time complexity of the algorithm.

2.14. Turbo_BM Algorithm [17]

Zest of the Turbo_BM algorithm is to remember the subpart of string in the transcript; this algorithm is as well a modified form of the Boyer_Moore algorithm. The algorithm doesn't need further pre-processing state rather than of base algorithm, it just entails a constant additional space only. Sub-string's matching is performed with a suffix of the pattern until the evaluations with the last character. Because of a turbo shift capability, the name is Turbo_BM. Space and pre-processing time complexities are in order of O(σ) and O(σ+m) respectively, whereas, the order of O(n) is the search time complexity. In the nastiest situation, the maximum of 2n text character comparisons is accomplished while searching.

2.15. Berry_Ravindran Algorithm [11] [18]

This algorithm is the fusion of the Quick_Search and Zhu_Takaoka algorithms. Malicious Character is being obtained using quick loop work as a base to the procedure. The time complexity of the pre-processing and searching segment of the Berry_Ravindran algorithm is in order of O(n*m) and O(2+m) respectively.

2.16. Wu_Manber Algorithm [10] [19]

This algorithm is an enhancement of the Boyer_Moore algorithm. Shifting of the wrong character is being performed in blocks, instead of doing it one character at a time. Hash, Shift, and Prefix tables are being used in algorithms. Order of O(|∑|+m) and O(n) is the time complexity for the pre-processing and searching segment for the Wu_Manber algorithm.

2.17. Backward_Oracle Algorithm [20]

The Backward_Oracle algorithm will more efficient for long patterns. In each turn, m size window of text is being processed. For the next window position, the algorithm seeks the pattern in the presently existing window. The searching and pre-processing time complexity for the Backward_Oracle algorithm is in-order of O(n*m) and O(m) respectively.

2.18. Backward Non-deterministic DAWG Matching Algorithm [21]

The parallelism and encoding techniques are being used for simulation of Nondeterministic suffix automata. Shifting of m length window over the text is performed. For every window, the pattern is scanned the current window backward and update the configuration of automata. The backward non-deterministic DAWG matching algorithm's time complexity is in order of O(m) and O(n*m) for the pre-processing and searching segment, respectively.

2.19. Commentz_Walter Algorithm [10] [22] [23]

The Commentz_Walter algorithm comes in existence as the result of the analysis of the Boyer_Moore algorithm and the Aho_Corasick algorithm. A state machine is being erected from the patterns that are used to match the text in the pre-processing segment. The extract of the Boyer_Moore is being applied in the searching segment. The length of the matching window should be of the minimum pattern length. Scanning is achieved from the right to left for the pattern characters. In case of hit and miss, a pre-calculated shift table is being mapped to shift the positioning of the window towards the right side. The time complexity of the algorithm is in order of O(π+m) and O(n*m) for the pre-processing and the searching segments, respectively.

2.20. Aho_Corasick Algorithm [7] [22] [24]

This algorithm has automata in addition to the Knuth_Morris_Pratt algorithm. The tree-based state machine of the pattern is being prepared. Initially, the tree has an empty root. As patterns get matched, the states are created. In the search segment, the state machine is being passed through in anticipation of a perfect matching state. The time complexity is in order of O(nk+m) and O(n+m+z) for the pre-processing and searching segments respectively.

3. Comparison Chart

Table 1. Chart comparing the pre-processing and matching complexities of algorithms discussed in previous session.

Name of Algorithm	Pre-processing Complexity	Matching Complexity
Brute Force (Naïve)	----	O(n*m)
Finite Automata	O(m|∑|)	O(n)
Rabin_Karp	O(m)	O((n-m+1) m)
Shift OR	O(σ+m)	O(n)
Morris_Pratt	O(m)	O(n+m)
Knuth_Morris_Pratt	O(m)	O(n)
Colussi	O(m)	O(n)
Forward DAWG	O(n)	O(n)
Boyer_Moore	O (|∑|)	O((n-m+1)+|∑|)
Quick Search	O(σ+m)	O(n*m)
Horspool	O(σ+m)	O(n*m)
Apostolico_Giancarlo	O(σ+m)	O(n)
Boyer_Moore_Smith	O(σ+m)	O(n*m)
Raita	O(σ+m)	O(n*m)
Turbo_BM	O(σ+m)	O(n)
Berry_Ravindran	O(σ2+m)	O(n*m)
Wu_Manber	O(|∑|+m)	O(n)
Backward_Oracle_Matching	O(m)	O(n*m)
Backward Nondeterministic DAWG	O(m)	O(n*m)
Commentz_Walter	O(σ+m)	O(n*m)
Aho_Corasick	O(nk+m)	O(n+m+z)

4. Conclusion

The paper discussed some of the popular exact matching algorithms and found that few algorithms show linearity in pre-processing or searching time complexities. However, Shift OR, Morris_Pratt, Kunth_Morris_Pratt, Colussi, Forward DAWG, Boyer_Moore, Apostolico_Giancarlo, Turbo_BM, Wn_Manber are being shown linearity in pre-processing as well as in the search segment. Among the discussed algorithms, some are good for matching long patterns (i.e. Backward_Oracle), few shows high efficiency on working with short patterns (i.e. Quick Search). Many of the algorithms work on simple pattern matching while other use automata (i.e. Searching with Automation, Forward DAWG Matching, Backward Non-deterministic DAWG Matching, Aho_Corasick,) or hash function (i.e. Rabin Karp, Wu_Manber), Several needs storage while few obsolete needs the space requirement. The string matching /searching algorithms have a vast range, including an exact match, approximate match, parallel search, single pattern search, and multiple pattern search.

References

[1] P. D. Michailidis and K. G. Margaritis, "On-Line String Matching Algorithms: Survey And Experimental Results," International Journal of Computer Mathematics, vol. 76, no. 4, pp. 411-434, 2001.
[2] C. Charras and T. Lecroq, Handbook of Exact String Matching Algorithms, 2004.
[3] C. Charras and T. Lecroq, "Exact string matching algorithms," Technical Report, 1997.
[4] C. Charras and T. Lecroq, "Exact string matching algorithms animation in java," Laboratoire d'Informatique de Rouen, 14 January 1997. [Online]. Available: http://www-igm.univ-

mlv.fr/~lecroq/string/index.html. [Accessed 20 February 2020].
[5] R. Baeza-Yates and G. H. Gonnet, "A new approach to text searching," Communications of The ACM, vol. 35, no. 10, pp. 74-82, 1992.
[6] M. Régnier, "Knuth-Morris-Pratt Algorithm: An Analysis," in MFCS '89 Proceedings on Mathematical Foundations of Computer Science 1989, 1989.
[7] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast Pattern Matching in Strings," SIAM Journal on Computing, vol. 6, no. 2, pp. 323-350, 1977.
[8] L. Colussi, "Correctness and efficiency of pattern matching algorithms," Information & Computation, vol. 95, no. 2, pp. 225-251, 1991.
[9] L. He and B. Fang, "Linear Nondeterministic Dawg String Matching Algorithm (Abstract)," in International Symposium on String Processing and Information Retrieval, 2004.
[10] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," Communications of The ACM, vol. 20, no. 10, pp. 762-772, 1977.
[11] D. M. Sunday, "A very fast substring search algorithm," Communications of The ACM, vol. 33, no. 8, pp. 132-142, 1990.
[12] V. SaiKrishna, A. Rasool and N. Khare, "String Matching and its Applications in Diversified Fields," International Journal of Computer Science Issues, vol. 9, no. 1, pp. 219-226, Jan 2012.
[13] J. Yan and A. E. Ahmad, "Breaking Visual CAPTCHAs with Naive Pattern Recognition Algorithms," in Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), 2007.
[14] R. N. Horspool, "Practical Fast Searching in Strings," Software - Practice and Experience, vol. 10, no. 6, pp. 501-506, 1980.
[15] A. Apostolico and R. Giancarlo, "The Boyer Moore Galil string searching strategies revisited," SIAM Journal on Computing, vol. 15, no. 1, pp. 98-105, 1986.
[16] P. D. Smith, "Experiments with a very fast substring search algorithm," Software - Practice and Experience, vol. 21, no. 10, pp. 1065-1074, 1991.
[17] M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski, and W. Rytter, "Speeding up two string-matching algorithms," Algorithmica, vol. 12, no. 4, pp. 247-267, 1994.
[18] T. Berry and S. Ravindran, "A Fast String Matching Algorithm and Experimental Results.," in Stringology, 1999.
[19] U. Manber and S. Wu, "Fast algorithm for multi-pattern searching," The University of Arizona, Tuscon, 1994.
[20] C. Allauzen, M. Crochemore, and M. Raffinot, "Factor Oracle: A New Structure for Pattern Matching," conference on current trends in theory and practice of informatics, vol. 1725, pp. 295-310, 1999.
[21] G. Navarro and M. Raffinot, "A Bit-Parallel Approach to Suffix Automata: Fast Extended String Matching," combinatorial pattern matching, pp. 14-33, 1998.
[22] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," Communications of The ACM, vol. 18, no. 6, pp. 333-340, 1975.
[23] B. Commentz-Walter, "A String Matching Algorithm Fast on the Average," in Proceedings of the 6th Colloquium, on Automata, Languages, and Programming, 1979.
[24] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," ACM Sigarch Computer Architecture News, vol. 33, no. 1, pp. 99-107, 2005.