JOURNAL OF COMPUTING TECHNOLOGIES

ISSN 2278 – 3814

Available online at www.jctjournals.com

Volume 1 Issue 2, June 2012

A Relative Study of Pattern Matching Algorithms

Shivani Jain[#1], Dr. A.L.Nersimha Rao[*2], Dr. Pankaj Agarwal[#3]

#[1 IT], #[2 IT], #[3 CSE], MTU, MTU, MTU

Vidya College of Engineering, Meerut, UP, India

Galgotia Engineering College, Greater Noida, UP India

IMS, Gaziabad, UP, India

[1]shivanij_1110@yahoo.com

[2]pankaj7877@gmail.com

[3]dr.rao@aol.com

Abstract— **The arrival of computers has made the everyday use of pattern-matching in various applications such as text editing, DNA sequence analysis, word processors, web search engine, computational molecular biology and natural language processing etc. Since this has also moved the development of many algorithms in the field of pattern matching in a string. As with most algorithms, the main considerations for string searching are speed and efficiency. There are number of string searching algorithms in existence world, in this paper we will focus on various already exist exact string matching and approximate string matching algorithms such as Knuth-Morris-Pratt, Boyer-Moore, Quick-search, Horspool, Shift Or, Wu-Manber Algorithms.**

Keywords— ***Pattern matching, Exact String Matching, Approximate String Matching.***

I. INTRODUCTION

From many years, pattern-matching has been usually used in various computer applications, for example, in editors, retrieval of in sequence from text, image, or sound, and searching protein sequence patterns in DNA protein sequence databases. In the present day, pattern-matching algorithms match the pattern exactly or approximately within the text. An exact pattern-matching is to find all the occurrences of a particular pattern (P) p1 p2... pm) of m-characters in a text (T) t1 t2 ... tn) of n-characters which are put up over a finite set of characters of an alphabet set.String matching or searching algorithms try to find places where one or several patterns are found within a larger text [1]. When the pattern is a single string the problem is known as string matching, locate all occurrences of a pattern P of length m in a text T of length n. Approximate string matching consists in finding all approximate occurrences of pattern P in text T [2]. Approximate String matching is one of the main problems in classical string algorithms, with applications to text searching, biological applications, pattern recognition etc [3].An algorithm that returns near-optimal solutions is called an approximation algorithm. The approximate string matching problem is to find all of those positions in a given text which are the left endpoints of substrings. The problem of approximate string matching is typically divided into two sub-problems: finding approximate substring matches inside a given string and finding dictionary string that match the pattern approximately. The string matching problem is to find out a pattern in a text (another string). In approximation string matching algorithm substring is matched approximately with the large string.

The problem can be formally stated as follows: given a large text of length n, short pattern of length m, and a maximal number of errors allowed k, finds all text positions that match the pattern with up to k errors. The allowed errors are characters from an alphabet.Approximate string matching is a challenging problem in Computer Science and requiring a

large amount of computational resources. It has different areas such as computational biology, text processing, pattern recognition and signal processing. For these reasons, fast practical algorithms for approximate string matching are in high demand.

1.1. DEFINITION OF STRING MATCHING PROBLEM

Given: Two strings T[1..n] and P[1..m] over alphabet Σ.

Want to find all occurrences of P[1..m] –the pattern‖ in T[1..n] –the text

Example: Σ = {a, b, c}

text T

a	b	c	a	b	A	a	b	c	a	a	b	a	c

pattern P

a	**b**	**a**	**a**

Terminology:

- P occurs with shift s.
- P occurs beginning at position s+1.
- s is a valid shift.

Goal: Find all valid shifts.

1.2. APPLICATIONS OF EXACT STRING MATCHING OR APPROXIMATE STRING MATCHING ALGORITHMS

- Text editors
- Parsers.
- Spam filters.
- Digital libraries.
- Screen scrapers.
- Word processors.
- Web search engines.
- Natural language processing.
- Computational molecular biology.
- Feature detection in digitized images etc.

2. STUDY ON EXISTING STRING MATCHING ALGORITHMS

Algorithms for pattern matching depend on the type of output. In this paper we focus on the various exact and approximate pattern matching algorithm.

2.1. STRING MATCHING ALGORITHM

The object of string searching is to find the location of a specific text pattern within a larger body of text (e.g., a sentence, a paragraph, a book, etc.). In string matching algorithms, it is required to find the occurrences of a pattern in a text. These algorithms find applications in text processing, text-editing, computer security, and DNA sequence analysis.

1) *Naive (Brute-Force) Algorithm:* The naïve approach simply test all the possible placement of Pattern $P[1 \ldots m]$ relative to text $T[1 \ldots n]$. Specifically, there is shift $s = 0, 1, \ldots, n - m$, successively and for each shift, s. Compare $T[s+1 \ldots s+m]$ to $P[1 \ldots m]$ [4]. The algorithm can be designed to stop on either the first occurrence of the pattern, or upon reaching the end of the text. The main features of this algorithm it is easy but slow, there is no preprocessing phase, it shift only by 1 position to the right, there is only constant extra space needed, and comparisons can be done in any order and mn expected text characters comparisons.

   In this algorithm pattern compares to each substring of text of length M. For example, M=5. The worst-case complexity of this algorithm is Θ(mn), where m denotes the length of pattern and n denotes the length of text. The total number of comparisons: M (N-M+1), hence worst case time complexity is O(MN).

2) *Rabin-Karp Algorithm:* The Rabin-Karp string searching algorithm calculates a hash value for the pattern, and for each M-character subsequence of text to be compared. If the hash values are unequal, the algorithm will calculate the hash value for next M-character sequence. If the hash values are equal, the algorithm will do a Brute Force comparison between the pattern and the M-character sequence. In this way, there is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

   If a satisfactorily large prime number is used for the hash function, the hashed values of two different patterns will usually be distinct. If this is the case, searching takes O(N) time, where N is the number of characters in the larger body of text. It is always possible to construct a scenario with a worst case complexity of O(MN). This, however, is likely to happen only if the prime number used for hashing is small.

This is a simple randomized algorithm that tends to run in linear time in most scenarios of practical interest. The main features are using hashing function, preprocessing phase, constant space and good for multiple patterns x being used. The worst case running time is as bad as that of the naive algorithm [5].

3) *Finite automata string matching algorithm:* A finite state machine also known as a deterministic finite automaton or DFA, is the set strings matching some pattern).The main features are Building the minimal deterministic finite automaton (DFA) accepting strings from the language L = ∑*x, L is the set of all strings of characters from ∑ ending with the pattern x, Time complexity O(n) of the search in a string y of size n if the DFA is stored in a direct access table and Most suitable for searching within many different strings y for same given pattern x.

4) *Knuth-Morris-Pratt Algorithm:* The algorithm was invented in 1977 by Knuth and Pratt and independently by Morris, but the three published it jointly. Searches for occurrences of a pattern x within a main text string y by employing the simple observation: after a mismatch, the word itself allows us to determine where to begin the next match to bypass re-examination of previously matched characters.

The Knuth-Morris-Pratt string searching algorithm (or KMP algorithm) searches for occurrences of a ―word‖ W within a main ―text string‖ S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. The two portions of the algorithm (Efficiency of search algorithm and efficiency of the table-building algorithm) have, respectively, complexities of O(k) and O(n), the complexity of the overall algorithm is O(n + k).

- Knuth-Morris-Pratt's algorithm compares the pattern to the text in left-to-right, but shifts the pattern more intelligently than the brute-force algorithm.
- Knuth-Morris-Pratt's algorithm preprocesses the pattern to find matches of prefixes of the pattern with the pattern itself

5) *Boyer-Moore Algorithm:* The Boyer-Moore-Horspool algorithm is an algorithm for finding substrings in strings. It was published by Nigel Horspool in 1980. It is a simplification of the Boyer-Moore string algorithm which is related to the Knuth-Morris-Pratt algorithm. The algorithm trades space for time in order to obtain an average-case complexity of O(n) on random text, although it has O(MN) in the worst case. The length of the pattern is M and the length of the search string is N. the best case is the same as for the Boyer-Moore string search algorithm in big O notation.

Theoretically, the Boyer-Moore1 algorithm is one of the efficient algorithms compared to the other algorithms available in the literature. The algorithm preprocesses the pattern and creates two tables, which are known as Boyer-Moore bad character (bmBc) and Boyer-Moore good-suffix (bmGs) tables. For each character in the alphabet set, a bad-character table stores the shift value based on the occurrence of the character in the pattern. On the other hand, a good-suffix table stores the matching shift value for each character in the pattern. The maximum of the shift value between the bmBc (character in the text due to which a mismatch occurred) dependent expression and from the bmGs table for a matching suffix is considered after each attempt, during the searching phase. This algorithm forms the basis for several pattern-matching algorithms [6].

6) *Quick Search algorithm:* The bad-character shift used in the Boyer-Moore algorithm is not very efficient for small alphabets, but when the alphabet is large compared with the length of the pattern, as it is often the case with the ASCII table and ordinary searches made under a text editor, it becomes very useful. Using it alone produces a very efficient algorithm in practice.

Time complexity

- Preprocessing phase in $O(m+\sigma)$ time and $O(\sigma)$ space complexity, σ is the number of alphabets in pattern.
- Searching phase in $O(mn)$ time complexity.

7) *The Horspool Algorithm (HORSPOOL):* The Horspool algorithm is a derivative of Boyer-Moore1 and is easy to implement. When the alphabet size is large and the length of the pattern is small, it is not efficient to use Boyer-Moore's bad-character technique. Instead, it is always enough to find the bad-character shift of the right-most character of the window to compute the value of the shift. These shift values are computed in the preprocessing stage for all the characters in the alphabet set. Hence, the algorithm is more efficient in practical situations where the alphabet size is large and the length of the pattern is small [7].

Time complexity [10]

- Preprocessing phase in $O(m+\pi)$ time and $\boldsymbol{O}(\pi)$ space complexity.

- Searching phase in $O(mn)$ time complexity.
- The average number of comparisons for one text character is between $1/\pi$ and $2/(\pi+1)$.

(π is the number of storing characters)

2.2. APPROXIMATE STRING MATCHING ALGORITHM

Approximate string matching consists in finding all approximate occurrences of a pattern x of length m in a text y of length n. Approximate occurrences of x are segments of y that are close to x according to a specific distance. The distance must be not greater than a given integer k. We consider two distances, the **Hamming distance** and the **Levenshtein distance** [8].

1) *Levenshtein or edit distance [Levenshtein 1965]:* The Levenshtein distance is a string metric for measuring the amount of difference between two sequences. The term edit distance is often used to refer specially to Levenshtein distance. The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion, deletion, or substitution of a single character in the simplified definition, all the operation cost is 1. It is named after Vladimir Levenshtein, who considered this distance in 1965. Time complexity of this algorithm is O(mn). It may not be useful while comparing long query strings.

   Levenshtein distance is named after the Russian scientist Viadimir Levenshtein, who devised pronounce Levenshtein. The metric is also called edit distance. The edit distance $\delta(p, t)$ between two strings p (pattern) and t (text) ($m = |p|$, $n = |t|$ is the minimum number of insertions, deletions and replacements to make p equal to t. the term edit distance is sometimes used to refer to the distance in which insertions and deletions cost and replacement have twice the cost of an insertion.

2) *Hamming distance [Sankoff and Kruskal 1983]:* The Hamming distance is named after Richard Hamming, who introduced it in his fundamental paper on hamming codes Error detecting and error correcting codes in 1950. It is used in telecommunication to count the number of flipped bits in a fixed-length binary word as an estimate of error, and therefore is sometimes called the signal distance. The Hamming Distance allows only substitution, which cost is 1 in simplified definition. For comparing string of different lengths or strings where not just substitutions but also insertion or deletions have to be expected, a more sophisticated metric like the Levenshtein distance is more appropriate.

   The Hamming distance between two strings of equal length is the number of positions at which the corresponding symbols are different. Put another way, it measure the minimum number of substitutions required to change one string into the other, or the number of errors that transformed one string into the other.Hamming distance is defined only for string of the same length. For two string p and t, H (p, t) is the number of places in which the two strings differ. Running time is O(n).

3) *Shift-Or Algorithm:* The Shift-Or Algorithm is both very fast in practice and very easy to implement. It adapts to the two above problems. We initially describe the method for the exact string-matching problem and then we show how it can handle the cases of k mismatches and of k insertions, deletions, or substitutions. The main advantage of the method is that it can adapt to a wide range of problems.

   Main features [11]

   - Uses bitwise techniques.
   - Efficient if the pattern length is no longer than the memory-word size of the machine.
   - Preprocessing phase in $O(m + \sigma)$ time and space complexity.
   - Searching phase in $O(n)$ time complexity (independent from the alphabet size and the pattern length).
   - Adapts easily to approximate string matching.

4) *Wu-Manber Algorithm:* The WuManber algorithm is a suffix-search based multi-pattern search algorithm. The key idea of Wu and Manber is to use blocks of characters of length B to avoid the weakness of the Horspool algorithm [12]. Wu-Manber algorithm is a bitmap algorithm based on Levenshtein distances. The Wu-Manber algorithm assumes that the pattern length is no more than the memory-word size of the machine, which is often the case in applications. The preprocessing phase takes $O(\sigma m+km)$ memory space, and runs in time $O(\sigma m+k)$. The time complexity of the searching phase is O(kn).

3. CONCLUSION

This paper describes the concept of string matching algorithms. String matching or searching algorithms try to find places where one or several patterns are found within a larger text. We focus on various already exist exact string matching and approximate string matching algorithms such as Knuth-

Morris-Pratt, Boyer-Moore, Quick-search, Horspool, Shift Or, Wu-Manber Algorithms in this paper.

REFERENCES

1) Georgy Gimel'farb (with basic contributions from M. J. Dinneen, Wikipedia, and web materials by Ch. Charras and Thierry Lecroq, Russ Cox, David Eppstein, etc.)
2) M. Crochemore and T. Lecroq, Pattern Matching and Text Compression Algorithms, ACM Computing Surveys, 28,1 (1996) 39--41.
3) by G Navarro - Cited by 52 - Related articles department of computer science university of Chile blanco encalada 2120 citeseerx.ist.psu.edu/viewdoc/download.
4) http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/naiveStringMatch.htm
5) String Matching: Rabin-Karp Algorithm, Greg Plaxton, Theory in Programming Practice, Fall 2005, Department of Computer Science, University of Texas at Austin, http://www.cs.utexas.edu/~plaxton/c/337/05f/slides/StringMatching-1.pdf
6) A FAST Pattern Matching Algorithm, S. S. Sheik, Sumit K. Aggarwal, Anindya Poddar, N. Balakrishnan, and K. Sekar*, Bioinformatics Centre and Supercomputer Education and Research Centre, Indian Institute of Science, Bangalore 560 012, India, *J. Chem. Inf. Comput. Sci.* 2004, *44*, 1251-1256
7) D Horspool, R. N. Practical fast searching in strings. *Software – Practice Experience* 1980, *10(6)*, 501-506.
8) Pattern matching and text compression algorithms, Maxime Crochemore 1 Thierry Lecroq[2] , http://www-igm.univ-mlv.fr/~lecroq/articles/lir9511.pdf
9) A very fast substring search algorithm, SUNDAY D.M., Communications of the ACM, 33(8),1990, pp. 132-142.
10) HORSPOOL R.N., 1980, Practical fast searching in strings, *Software - Practice & Experience*, 10(6):501-506.
11) http://www-igm.univ-mlv.fr/~lecroq/string/node6.html
12) http://webpages.cs.luc.edu/~pld/courses/447/sum08/classA/groepl,klau,reinert.2002.fast_exact_string_matching.pdf