String Matching Algorithms

Article *in* International Journal Of Engineering And Computer Science · March 2018

DOI: 10.18535/ijecs/v7i3.19

CITATIONS
8

READS
4,586

3 authors, including:

Preeti Narooka
Terna Engineering College
3 PUBLICATIONS **16** CITATIONS

SEE PROFILE

www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 7 Issue 3 March 2018, Page No. 23769-23772
Index Copernicus Value (2015): 58.10, 76.25 (2016) DOI: 10.18535/ijecs/v7i3.19

String Matching Algorithms

Mukku Bhagya Sri, Rachita Bhavsar, Preeti Narooka

Computer Department
Terna engineering college, Nerul
Computer Department
Terna Engineering college, nerul
Assistant professor
Computer Department
Terna Engineering college, Nerul

Abstract:
To analyze the content of the documents, the various pattern matching algorithms are used to find all the occurrences of a limited set of patterns within an input text or input document. In order to perform this task, this research work used four existing string matching algorithms; they are Brute Force algorithm, Knuth-Morris-Pratt algorithm (KMP), Boyer Moore algorithm and Rabin Karp algorithm. This work also proposes three new string matching algorithms. They are Enhanced Boyer Moore algorithm, Enhanced Rabin Karp algorithm and Enhanced Knuth-Morris-Pratt algorithm.
Findings: For experimentation, this work has used two types of documents, i.e. .txt and .docx. Performance measures used are search time, number of iterations and accuracy. From the experimental results, it is realized that the enhanced KMP algorithm gives better accuracy compared to other string matching algorithms. **Application/Improvements**: Normally, these algorithms are used in the field of text mining, document classification, content analysis and plagiarism detection. In future, these algorithms have to be enhanced to improve their performance and the various types of documents will be used for experimentation.

Keywords*: Brute Force, Boyer Moore, Information Retrieval, Knuth-Morris-Pratt, Pattern Matching, Rabin Karp*

I. Introduction

String searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text. Let Σ be an alphabet (finite set). Formally, both the pattern and searched text are vectors of elements of Σ. The Σ may be a usual human alphabet (for example, the letters A through Z in the Latin alphabet). Other applications may use binary alphabet (Σ = {0,1}) or DNA alphabet (Σ = A,C,G,T}) in bioinformatics.[11] We assume that the text is an array T[1..n] of length n and that the pattern is an array of length[1..m] of length m and that m<=n. The character arrays T and P are often called strings of characters. We say that pattern P occurs with shift s in text T (or equivalently that the pattern P occurs beginning at position s+1 in text T) if 0<=s<=n-m and T[s+1….s+m]=P[1..m]. If P occurs with shift s in T then we

calls a valid shift otherwise we call s an invalid shift. The string matching algorithm is the problem of finding all valid shift with which a pattern P occurs in given text.

Large number of algorithms is known to exist to solve string matching problem. Based on the number of patterns searched for the algorithms can be classified as single pattern and multiple pattern algorithms. Applications may require exact or approximate string matching.

Exact String Matching Problem

We are given a text string pattern string we want to find all occurrences of P in T. In Exact string matching problem the pattern is exactly found inside the text. Consider the following example:

T=AGCCTAAGCTCCTAAGTC

P=CCTA

There are two occurrences of P in T as shown below:

AGCCTAAGCTCCTAAGTC

A brute force method for exact string matching algorithm:

T=ACCACTAGA

P=ACTA

ACTA

ACTA

ACTA

If the brute force method is used, many characters which had been matched will be matched again because each time a mismatch occurs, the pattern is moved only one step. There are many exact string matching algorithms. Nearly all of them are concerned with how to slide the pattern. Few of them are listed below.

II. Metholodgy:

The main goal of this research work is to match the patterns of text by analyzing the contents of the documents using string matching algorithms. In order to perform this task, this research work uses four existing string matching algorithms; they are Brute Force algorithm, Knuth-Morris-Pratt algorithm (KMP), Boyer Moore algorithm and Rabin Karp algorithm. This work also proposes three new string matching algorithms. They are Enhanced Boyer Moore algorithm, Enhanced Rabin Karp algorithm and Enhanced Knuth-Morris-Pratt algorithm. The performance factors are used time taken for searching the pattern, number of iterations required and its accuracy for single word search, multiple words search and a file search. But in this research work we study in detail about , Knuth-Morris-Pratt algorithm (KMP) and Rabin Karp algorithm.

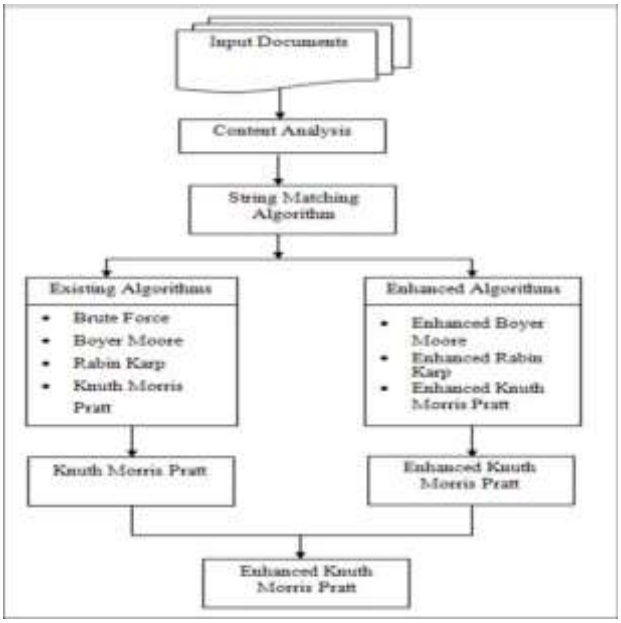

Fig: Methodology

Existing Algorithms:

1 Rabin- Karp Algorithm

Rabin-Karp Algorithm is the simplest string searching algorithm. This algorithm was developed by Michael O. Rabin and Richard M. Karp in 1987. This algorithm uses the hash function to discover the potential pattern in the input text. For the length of text n and pattern p of mutual length m, its average and best case running time is O (n+m) in space O (p), and also the worst-case time is O (nm) in space O (m). It is used to discover the hash value of the certain pattern substring and then it discovers the hash value of all possible m length substring of the input text. If the hash value of the pattern and text substring match than it returns the value otherwise next substring value is matched to calculate the string of length m.

Algorithm: Rabin-Karp

```
RABIN-KARP-MATCHER(T,P,d,q)
1   N=T.length
2   M=P.length
3   h=dm-1mod q
4   p = 0
5   t0=0
6   for i=1 to m
7   p =(dp+P[i])mod q
8   t0=(dt0+t[i])mod q
9   for s = 0 to n-m
10  if p == ts
11  if p[1..m] == T[s+1...s+m]
12  Print"Pattern occcurs with shift "s
13  If s<n-m
14  ts+1 =(d(ts – T[s+1]h)+T[s+m+1]) mod q
```

The procedure works as follows. All characters are interpreted as radix-d digits. The subscript on t are

provided only for clarity; the program works correctly if all the subsripts are dropped. Line 3 intializes h to the value of the high-order dogit podition of an m-digit window. Line 4-8compute p as the value of the of P[1...m] mod q and t0 as the value of T[1...m]modq. The for loop of lines 9-14 iterates thrugh all possible shifts s, maintaining the following invariant.

Knuth-Morris-Pratt Algorithm

The Knuth–Morris–Pratt were developed a linear time string searching algorithm by analysis of the brute force algorithm or naïve algorithm. The algorithm was developed in 1974 by Donald Knuth and Vaughan Pratt, and independently by James H. Morris and they published it jointly in 1977.The Knuth-Morris-Pratt algorithm moderates the total number of comparisons of the pattern against the input string. A matching time of O(n) is accomplished by evading associations with essentials of 'S' that have earlier been

1. The prefix function, Π The prefix function, Π for a pattern summarizes the knowledge regarding however the pattern matches in contradiction of shifts of itself. This information may be accustomed avoid unusable shifts of the pattern "p". In other words, this succeeds avoiding backtracking on the string "S".

2. The KMP Matcher With string "S", pattern "p" and prefix function "Π" as inputs, the prevalence of "p" in "S" is found and the algorithm yields the variety of shifts of "p" after which the existence is found.

3. Running - time analysis: The period of time for computing the prefix function is Θ (m) and period of time of matching function is Θ (n).

Algorithm:Knuth-Morris-Pratt

```
1  n = T.length
2  m=P.length
3  3.14 = Computer-Prefix-function(p)
4  q = 0
5  for i = 1to n
6  while q>0 and P[q+1]=/ T[i]
7  q = 3.14[q]
8  if P[q+1]== T[i]
9  q = q+1
10 if q == m
11 print"Pattern occurs with shift" i-m
12 q=3.14[q]
```

Enhanced Algorithms:

Enhanced Rabin Karp Algorithm

This searching algorithm that uses the hashing function to find any one of a set of pattern in input text. Hashing offers a simple method to avoid a total number of character comparisons. For length of text N and the pattern P of combined length M, its best case running time is O (N+M). And the worst case time is O (NM). First the algorithm used to find the hash value of the pattern. Then it checks the input text along with its hash value. If mismatch occurs, shift the window to the next character then calculate the hash value and the same process will continue. Otherwise it returns the index position of the particular character.

Algorithm: Enhanced Rabin Karp Algorithm

```
1  Functrion relation(S,P,n,m,k,q)
2  Begin
3  h – Km-1 mod q;
4  p – 0;
5  t0 – 0;
6  for i=1 to m do
7  P –(K, p+ p[i])modq;
8  T0 –(K, t0+s[i])modq;
9  End for
10 For j=0 to n-m do
     a) If  p-tj then
        i) If  p=s[j+1,j+m] then
                Out j+1;
        ii)     End if
     b) End if
11  If j<n-m then
12 Tj-1 =(K(tj-s[j+1]).h)+s[j+m+1])mod q;
13 End for
14 End.
```

Enhanced Knuth-Morris-Pratt Algorithm

Knuth-Morris-Pratt algorithm is one of the efficient string matching algorithms. This algorithm examines for existences of a pattern p within a main text t by using the reflection that while matching, the mismatch occurs, the word itself represents satisfactory information to regulate where the next match can begin, thus avoiding the re examination of formerly matched characters. The KMP algorithm uses a bit table to discover the mismatch of the pattern in an input text. This algorithm performs the comparison from left to right. It uses the bit table for the comparison, if match it returns the index of the text. Otherwise it checks the next bit.

Algorithm: Enhanced Knuth-Morris-Pratt Algorithm

```
1  KMP_search(E(p),E(T))
```

2 Begin
3 Preprocess E(p)to obtain the next_bit table
4 While (not emd of input)do
   a) Get next bit b;
   b) If (j>=0)&(b!=E(p)[j])do
   c) End if
   d) If (j=|E(p)|)
      i) Return a match
      ii) J--1
   e) End if
   f) J-j+1
   g) End while
5 End.

Variants:
Robin-Karp Algorithm

A. Long patterns and Σ For long patterns and Σ, Boyer-Moore algorithm gives much better efficiency compared to other string matching algorithms. The program involves two heuristics that allows the program to skip many text characters altogether. The algorithm makes successive comparisons from right to left. When a mismatch occurs, both heuristics proposes a value (maximum of which is chosen) by which shift is increased without skipping any valid shift.

B. Repetition Factors An efficient algorithm for string matching based on repetition factors was developed by Galil and Seiferas. The algorithm has linear running time complexity and requires only O(1) storage beyond P and T.

C. Approximate String Matching The Bitap algorithm performs approximate string matching based on Levenshtein distance between strings. The algorithm requires much lesser preprocessing and can uses mostly bitwise operations, making the algorithm extremely fast.

D. Dictionary Matching Aho-Corasick algorithm can perform multiple (but finite) pattern matching in a text in parallel achieving linear running time.

E. Polymorphic String Matching Combination of more than one string matching algorithm (example KMP and Boyer-Moore fusion) can be used to provide a better functional algorithm with decreased space and

Knuth-Morris-Pratt
A real-time version of KMP can be implemented using a separate failure function table for each character in the alphabet. If a mismatch occurs on character in the text, the failure function table for character is consulted for the index in the pattern at which the mismatch took place. This will return the length of the longest substring ending at matching a prefix of the pattern, with the added condition that the character after the prefix is With this restriction, character in the text need not be checked again in the next phase, and so only a constant number of operations are executed between the processing of each index of the text. This satisfies the real-time computing restriction.

III. Conclusion:

This research work analyzes the performance measures of existing and enhanced string matching algorithms. The performance factors are time, number of iteration and its accuracy for single line, multiple lines and a file. From the analysis, in existing the KMP algorithm gives the better accuracy for all the inputs. In enhanced algorithms, the enhanced KMP algorithm gives the better accuracy. Form the existing and enhanced KMP algorithms; the enhanced KMP algorithm gives the better accuracy.

IV. Acknowledgement

We feel privileged to express our deepest sense of gratitude. To our guide **Profs.Preeti mam.** Her prompt and kind help led to completion of work.

References

[1] Verma A, Kaur I, Singh I. Comparative analysis of data mining tools and techniques for information retrieval. Indian Journal of Science and Technology.

[2] Al-Mazroi A, Rashid NA. A Fast Hybrid Algorithm for the Exact String Matching Problem. American Journal of Engineering and Applied Sciences. 2011.

[3] Algorithm book by Cormen.