USING DOUBLE KEYS HASHING FOR EFFICIENT STRING SEARCHING PROBLEM

Felix Chebon Ng’etich
flexx87@gmail.com
+254 713 047 019

Introduction

String matching also referred to as string searching algorithm, is a class of string algorithms that try to find a place where one or several strings (patterns) are found within a larger string or text. Hashing is any well-defined procedure or mathematical function that converts a large, possibly variable-sized amount of data into a small sized-datum, usually a single integer that may serve as an index to an array. It’s also mathematical process that converts a message (document) into a unique "message digest" that represents the original message. A hash function will not produce the same message digest from two different inputs.

Security of data and of any networked environment is in the hands or intrusion detector systems, viruses and spam filters which are all based on string matching. Performance demands integrity authentication and data security becomes an issue when threats are posed to your network. In this paper, We introduce the use of Hashing and String matching functions to curb with the threats and to ensure authentication of data in a network. [1]

We show that when hash and string matching algorithms are in cooperated together, data integrity is maintained and ease of information searching throughout the network is increased. This paper gives an input independent average linear time algorithm for storage and retrieval on keys. The algorithm makes a random choice of hash function from a suitable class of hash functions. Given any sequence of inputs the expected time (averaging over all functions in the class) to store and retrieve elements is linear to the length of the sequence. The number of references to the database required by the algorithm for any input is extremely close to the theoretical minimum for any possible hash function with distributed inputs. The ability to analyze the cost of storage and retrieval without worrying about the distribution of the input allows improvements on the bounds of several algorithms making the hash and string matching algorithm much appropriate algorithm for searching strings both from the rear and the front which save time of searching single instead of double search words.
Problem Statement

We show that when hash and string matching algorithms are cooperated together, data integrity is maintained and ease of information search throughout the network. This paper gives an input independent average linear time algorithm for storage and retrieval on keys. The algorithm makes a random choice of hash function from a suitable class of hash functions. Given any sequence of inputs the expected time (averaging over all functions in the class) to store and retrieve elements is linear in the length of the sequence. The number of references to the database required by the algorithm for any input is extremely close to the theoretical minimum for any possible hash function with distributed inputs. The ability to analyze the cost of storage and retrieval without worrying about the distribution of the input allows as corollaries improvements on the bounds of several algorithms.

The problem we want to solve is: given two strings, a text $T \ [1 ... n]$ and a pattern $P \ [1 ... m]$; find the substring of the text that is the same as the pattern (Substring is a continuous sub array), for any shifts $s$, let $T_s$ denote the substring $T \ [s..s + m -1]$. So we want to find the smallest shifts $s$ such that $T_s = P$, or report that there is NO match. For example if in the text CLEARED and the pattern is “RED”.

String matching uses brute force matching. This matching algorithm which compares the immediate loop substrings of $T_s$ with $P$

and if the two strings do not match then the string will terminate and will report match or character found, using the example above

\begin{verbatim}
ALMOSTBRUTEFORCE [2]
(T[1..n],P[1..m]):
    For s \ to n - m+1
    Equal \ true
    i \ 1
    While equal and i \ m
        Equal\i+1
        Else
            i \ i+1
            if equal
        return s
    return "none"
\end{verbatim}
Related Work

Systolic array hardware employs Dynamic programming [3] for string matching with hashing which is proper for short pattern and short text since the circuit size is proportional to the length of the pattern (starting point of the hash function) and text. Bloom filter string matching algorithm (BFSMA) hardware uses bloom filter to accelerate the average case time complexity for an exact matching algorithm. However, BFSMA in corporate both hashing and string matching to build a single bit vector for all patterns making it an infeasible algorithm. Most related work to our study is the BFSM algorithm because it uses multiple hashing functions to reduce the probability of giving false positive values (which is a false match) but any AND functions reports a positive value. When BFSM algorithm chooses constant $K$, independent hashing function to hash N patterns into a given vector size $M$, where the probability of false positive value $P_{fp}$ is obtained as: [3.1]. The pre hashing method if incorporated can test quickly the multiple partial patterns of the current state and against the compared substring of the text to avoid some slow matching. The matching can be skipped if true negative is indicated in the pre hashing matching. True negative is a condition that is absent in pre hashing vector of the suffix of the current state. [4]

Illustration of the algorithm [4.1]

$$ P_{fp} = \left(1 - (1 - \frac{1}{M})^{NK}\right)^k $$

String len ← length of the string

$i ← \text{patlen}$

then return false

$j ← \text{patlen}$

Loop: if $j ← 0$ then return $j+1$

if string$(i) = \text{pat}(j)$

then

$j ← j - 1$

$i ← i - 1$

go to loop

close $i ← i + \max(\text{delta}_1, (\text{string}(i)), \text{delta}_2(j))$

goto top.
Start

Construct a table A (pattern) and a table B (character)

Read the text T and fill the table each word of same character in Table A and Table B

Sort the Table B character with corresponding pattern given in A

Enter a pattern P to calculate the Hash value H

Search for each corresponding value starting with Table A

Is the character same?

Search the Hash value H using starting position of the pattern

Is the Hash found?

Compare words using pattern (starting position)

Is complete match found?

Occurrence of text reported

Jump to next table

Is it in the pattern?

No

End
Proposed Solution and future works

We propose the use of modified word searching algorithm (MWSA) which reduce search time (avoids searching only one word but instead searches the entire word document from front and end using the specified pattern given by the algorithm). Using Double key string searching with the incorporation of hashing perform better and faster than the modified word searching algorithm (MWSA) which uses single keys string search algorithm. But further work can be done on incorporating both MWSA with the double keys searching algorithms which will be able to handle multiple patterns on increased sized text and character as double keys search only works best on short text sized characters. [6]

References


