Journal of Advanced Computer Science & Technology, 4 (1) (2015) 175-179
www.sciencepubco.com/index.php/JACST

doi: 10.14419/jacst.v4i1.4283
Research Paper

Comparison criteria between matching algorithms texts application on (horspool's and brute force algorithms)

Amin Mubark Alamin Ibrahim [1]*, Mustafa Elgili Mustafa [2]

[1] *Department of mathematics Faculty of Education Shagra University Afif, Saudi Arabia*
[2] *Computer Science Department Community College Shaqra University Shaqra, Saudi Arabia*
**Corresponding author E-mail: aminibrahim2006@gmail.com*

Abstract

The subject of matching text or search the texts is important topics in the field of computer science and is used in many programs such as Microsoft word program in correct spelling mistakes and search &replace, and other uses. The aim of this study was to learn how to trade-off texts matching algorithms, which are very much where we have the application on Horspool's and Brute Force algorithms. According to the standard number of comparisons and time of execution. The study pointed on preference Horspool's algorithm.

Keywords: *Space and Time Tradeoffs; Put Enhancement in String Matching; Horspool's and Brute Force Algorithms.*

1. Introduction

String-matching is a very important subject in the wider domain of text processing. String-matching algorithms are basic components used in implementations of practical software's existing under most operating systems. Moreover, they emphasize programming methods that serve as paradigms in other fields of computer science (system or software design). Finally, they also play an important role in theoretical computer science by providing challenging problems. Although data are memorized in various ways, text remains the main form to exchange information. This is particularly evident in literature or linguistics where data are composed of huge corpus and dictionaries. This applies as well to computer science where a large amount of data is stored in linear files. And this is also the case, for instance, in molecular biology because biological molecules can often be approximated as sequences of nucleotides or amino acids. Furthermore, the quantity of available data in these fields tends to double every eighteen months. This is the reason why algorithms should be efficient even if the speed and capacity of storage of computers increase regularly. String-matching consists in finding one, or more generally, all the occurrences of a string (more generally called a pattern) in a text.[14]So, the definition of string-matching g is an object maintaining a pattern and a string. It provides a collection of different algorithms for computation of the exact string matching problem. Each function computes a list of all starting positions of occurrences of the pattern in the string. The algorithms can be called explicitly by their name or implicitly by making an algorithm the "current" algorithm (specified by the set Algorithm-method) and using functions get First Occurence(), get Next Occurence(), and get All Occurences().[15] The aim of this study was to learn how to trade-off texts matching algorithms, which are very much where we have the application on Horspool's and Brute Force algorithms According to the standard number of comparisons and time of execution.

2. Space and time tradeoffs

Storage space can save calculation time. Tables can have common functional evaluations to reduce recursive calls. Another technique is to store preprocess calculations in an array to be used during the final calculations; this is called input enhancement. We consider one famous example for string matching [11].

3. NPUT enhancement in string matching

Recall the string matching problem, a pattern P [0...m-1] is searched for in a text T[0...n-1]. The brute force algorithm in the worst case makes m (n-m+1) comparisons, so the cost is Θ (nm). But on average only a few comparisons are made before shifting the pattern, so the cost is Θ (n). We consider two algorithms that also achieve this cost. [11].

4. Horspool's algorithm(description and analysis)

In computer science, the Boyer–Moore–Horspool algorithm or Horspool's algorithm is an algorithm for finding substrings in strings. It was published by Nigel Horspool in 1980.[1]It is a simplification of the Boyer–Moore string search algorithm, which is related to the Knuth–Morris–Pratt algorithm .furthermore Horspool's algorithm shifts the pattern by looking up shift value in the character of the text aligned with the last character of the pattern in table made during the initialization of the algorithm. The pattern is check with the text from right to left and progresses left to right through the text [11]. The time efficiency for this algorithm is pretty fast in most cases where we are using the basic operation of comparison (where we compare m-1 characters.) The average time complexity is Theta no and for the wost case scenario the time complexity is O (nm) where n is the string in the algorithm and m is the substring/pattern we want to find in the string. And the average number of comparisons for one text character is between 1/п and 2/ (п+1), where п is the number of storing characters. Also preprocessing phase in O (m+ п) time and O (п) space complexity. [2], [3], [4], [5], [6], [7], [8], [9] and [10] Let c be the character in the text that aligns with the last character of the pattern. If the pattern does not match there are 4 cases to consider [3].The mismatch occurs at the last character of the pattern:
Case 1: c does not exist in the pattern (Not the mismatch occurred here) then shift pattern right the size of the pattern.
Case 2: The mismatch happens at the last character of the pattern, and c does exist in the pattern then the shift should be to the right most c in the m-1 remaining characters of the pattern.
The mismatch happens in the middle of the pattern:
Case 3: The mismatch happens in the middle (therefore, c is in pattern) and there are no other c in the pattern then the shift should be the pattern length.
Case 4: The mismatch happens in the middle of the pattern but there is other c in pattern then the shift should be the right most c in the m-1 remaining characters of the pattern.
The table of shift values, table(c), is a table of the entire alphabet of the text and should give
t (c) = m if c is not in the first m-1 characters of the pattern.
t (c) = distance of the right most c in the first m-1 characters of the pattern.
To make the shift table we initialize the table to m and then scan the pattern left to right writing m-1-j in for the j character in the pattern.
A. Algorithm ShiftTable (P[0...m-1])[11]
// output Table the size of the alphabet
// and gives the amount to shift the table.
Initialize all entries of Table to m
For j ← 0 to m-2 do Table [P[j]] ← m-1-j // character position from the end of pattern
Return Table
The line
For j ← 0 to m-2 do Table [P[j]] ← m-1-j
It finds the right most location in the pattern of the character.
Now we can use table in Horspol's algorithm.
B. Pseudo for horspool algorithm [13]:
//Implements Horspool’s algorithm for string matching
//Input: Pattern P [0.m − 1] and text T [0.n − 1]
//Output: The index of the left end of the first matching substring
// or −1 if there are no matches.
Shift Table (P [0.m – 1) //generate table of shifts.
i ← m − 1 //position of the pattern’s right end
While i ≤ n − 1 does
k ← 0 //number of matched characters
While k ≤ m − 1 and P [m − 1− k] = T [i − k] do
k← k + 1
if k = m
return i − m + 1
else i← i + Table [T [i]]
return −1

5. Brute force algorithm(description and analysis)

The brute force algorithm consists in checking, at all positions in the text between 0 and n-m, whether an occurrence of the pattern starts there or not. Then, after each attempt, it shifts the pattern by exactly one position to the right.The brute force algorithm requires no preprocessing phase, and a constant extra space in addition to the pattern and the text. During the searching phase the text character comparisons can be done in any order. The time complexity of this searching phase is O (mn) (when searching for am-1b in a for instance). The expected number of text character comparisons is 2n [12].

a. brute force algorithm Procedures:

In order to apply brute-force search to a specific class of problems, one must implement four procedures, first, next, valid, and output. These procedures should be taken as a parameter the data P for the particular instance of the problem, that is to be solved, and should do the following [12]:

First (P): generate a first candidate solution for P.

Next (P, c): generate the next candidate for P after the current one c.

Valid (P, c): check whether candidate c is a solution for P.

Output (P, c): use the solution c of P as appropriate to the application.

The next procedure must also, tell when there are no more candidates for the instance P, after the current one c. A convenient way to do that is to return a "null candidate", some conventional data value Λ that is distinct from any real candidate. Likewise, the first procedure should return Λ if there are no candidates at all for the instance P. The brute-force method is then expressed by the algorithm

B. Pseudo code for Brute Force algorithm[13]:

```
//Implements brute-force string matching.
//Input: An array T [0.n − 1] of n characters representing a text and.
// an array P [0.m − 1] of m characters representing a pattern.
//Output: The index of the first character in the text that starts a.
// matching substring or −1 if the search is unsuccessful.
for i ← 0 to n − m do
j ← 0
while j <mand P[j ]= T [i + j ] do
j ← j + 1
if j = m return i
return −1
```

6. Method of comparisons between bruteforce and horspool's algorithms:

The comparison between Bruteforce and Horspool's algorithms we used. Firstly, coefficient of number of comparisons for one text character by using spss program. Secondly, Account execution time for Bruteforce and Horspool's algorithms in the search for the sample in the text using the C ++ program. So, as to enhance the result of the first laboratories.

7. Result and discussion

We divided them into follows:

a) Testing the differences between bruteforce and horspool's algorithms according to the number of comparisons for one text character

We took 30 samples randomly in spss program to calculate the number of comparisons for one text character.Then the results of the T-test between Bruteforce and Horspool's algorithms with number of comparisons for one text character were found to be highly significant (Table 1). It is very clear from Table 1 that, there were a significant different (p>0.000).This proves that there was a high difference between Bruteforce and Horspool's algorithms,and which proves that the Horspool's the best choice (Figure (1) analysis of Bruteforce and Horspool's algorithms with number of comparisons for one text character)

Table 1: T-Test

One – Sample Statistics				
	N	Mean	Std. Deviation	Std. Error Mean
bruteforce	30	13.30	2.366	0.432
horspool	30	7.37	1.129	0.206

One – Sample Test						
	Test Value = 0					
	t	df	Sig. (2-tailed)	Mean Difference	95% Confidence Interval of the Difference	
					Lower	Upper
Bruteforce	30.793	29	0.000	13.300	12.42	14.18
horspool	35.738	29	0.000	7.367	6.95	7.79

Testing the Differences between Bruteforce and Horspool's Algorithms according to the Number of Comparisons for One Text Character.

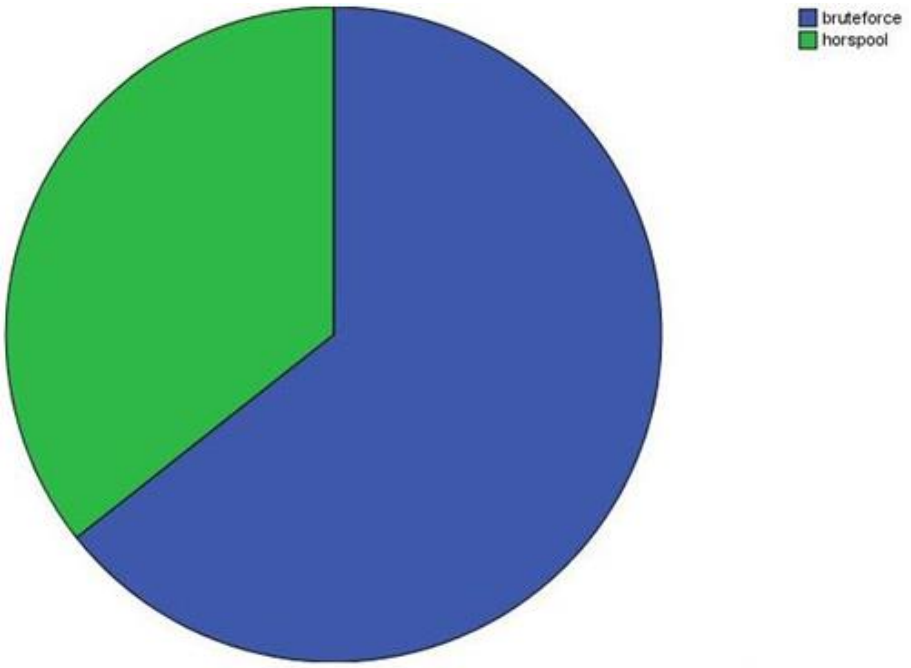

Fig. 1: Analysis of Bruteforce and Horspool's Algorithms with Number of Comparisons for One Text Character

B.Testing the differences between Bruteforce and Horspool's algorithms according to execution time:
We took 30 samples randomly in c++ program to calculate the execution time after that we took the result of the execution time to spss program to know the deference's .Then the results of the T-test between Bruteforce and Horspool's algorithms with execution time were found to be highly significant (Table 2). It is clearly from Table 2 that, there were a significant different (p>0.000).This proves, that there was a high difference between Bruteforce and Horspool's algorithms,and which proves that the Horspool's the best choice (Figure (2) analysis of Bruteforce and Horspool's algorithms execution time).

Table 2: T-Test

One – Sample Statistics				
	N	Mean	Std. Deviation	Std. Error Mean
bruteforce	30	0.5337	0.13954	0.2548
horspool	30	0.1310	0.13446	0.2455

One – Sample Test						
	Test Value = 0					
	t	df	Sig. (2-tailed)	Mean Difference	95% Confidence Interval of the Difference	
					Lower	Upper
Bruteforce	-528.569	29	0.000	-13.46633	-13.5184	-13.4142
horspool	-564.972	29	0.000	-13.86900	-13.9192	-13.8188

Testing the differences between Bruteforce and Horspool's algorithms according to execution time

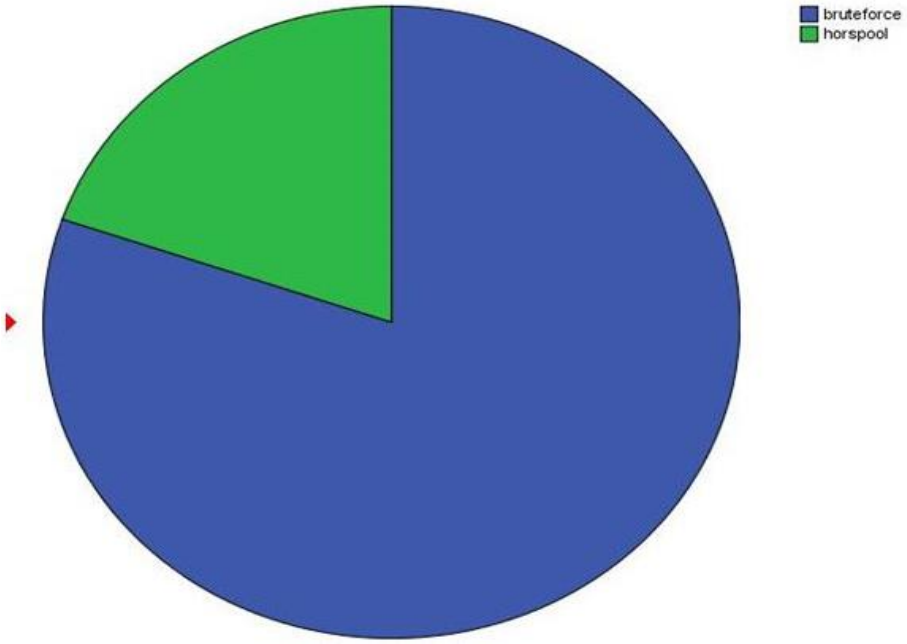

Fig. 2: Analysis of Bruteforce and Horspool's Algorithms with Execution Time

8. Conclusions

The study showed how important search algorithms matching texts and how to choose between these many algorithms. Where we have the application on the Horspool's algorithm and Bruteforce algorithm. The result proves that preference Horspool's algorithm more than Bruteforce. This study is subject to further discussion.

References

[1] R. N. Horspool (1980). "Practical fast searching in strings". Software - Practice & Experience 10 (6): 501–506. doi:10.1002/spe.4380100608. (Subscription required (help)). http://dx.doi.org/10.1002/spe.4380100608.
[2] AHO, A.V., 1990, Algorithms for finding patterns in strings. in Handbook of Theoretical Computer Science, Volume A, Algorithms and complexity, J. van Leeuwen ed., Chapter 5, pp 255-300, Elsevier, Amsterdam.
[3] BAEZA-YATES, R.A., RÉGNIER, M., 1992, Average running time of the Boyer-Moore-Horspool algorithm, Theoretical Computer Science 92(1):19-31. http://dx.doi.org/10.1016/0304-3975(92)90133-Z.
[4] BEAUQUIER, D., BERSTEL, J., CHRÉTIENNE, P., 1992, Éléments d'algorithmique, Chapter 10, pp 337-377, Masson, Paris.
[5] CROCHEMORE, M., HANCART, C., 1999, Pattern Matching in Strings, in Algorithms and Theory of Computation Handbook, M.J. Atallah ed., Chapter 11, pp 11-1--11-28, CRC Press Inc., Boca Raton, FL.
[6] HANCART, C., 1993. Analyse exacte ET en moyenne d'algorithmes de recherche d'un motif dans UN texte, Ph. D. Thesis, University Paris 7, France.
[7] HORSPOOL R.N., 1980, Practical fast searching in strings, Software - Practice & Experience, 10(6):501-506. http://dx.doi.org/10.1002/spe.4380100608.
[8] LECROQ, T., 1995, Experimental results on string matching algorithms, Software - Practice & Experience 25(7):727-765. http://dx.doi.org/10.1002/spe.4380250703.
[9] STEPHEN, G.A., 1994, String Searching Algorithms, World Scientific.
[10] http://www-igm.univ-mlv.fr/~lecroq/string/node18.html.
[11] http://polynomialtimes.com/algorithms/space-and-time-trade-offs/horspools-algorithm/.
[12] http://www-igm.univ-mlv.fr/~lecroq/string/node3.html#SECTION0030.
[13] Anany Levitin, the design and analyses of algorithms, third edition,9 October 2011 from the following web sites: ftp://doc.nit.ac.ir/cee/jazayeri/Algorithms/Books/Design%20&%20Analysis%20of%20Algorithm.pdf.
[14] http://www-igm.univ-mlv.fr/~lecroq/string/node2.html.
[15] http://www.algorithmic-solutions.info/leda_manual/string_matching.html.