# A Fast String Matching Algorithm and Experimental Results

T. Berry and S. Ravindran

Department of Computer Science
Liverpool John Moores University
Liverpool L3 3AF
United Kingdom

e-mail: {T.BERRY,S.RAVINDRAN}@livjm.ac.uk

**Abstract.** In this paper we present experimental results for string matching algorithms which have a competitive theoretical worst case run time complexity. Of these algorithms a few are already famous for their speed in practice, such as the Boyer-Moore and its derivatives. We chose to evaluate the algorithms by counting the number of comparisons made and by timing how long they took to complete a given search. Using the experimental results we were able to introduce a new string matching algorithm and compared it with the existing algorithms by experimentation. These experimental results clearly show that the new algorithm is more efficient than the existing algorithms for our chosen data sets. Using the chosen data sets over 1,500,000 separate tests were conducted to determine the most efficient algorithm.

**Key words:** string matching, pattern matching, algorithms on words.

## 1 Introduction

Many promising data structures and algorithms discovered by the theoretical community are never implemented or tested at all. Moreover, theoretical analysis (asymptotic worst-case running time) will show only how algorithms are likely to perform in practice, but they are not sufficiently accurate to predict actual performance. In this paper we show that by considerable experimentation and fine-tuning of the algorithms we can get the most out of a theoretical idea.

The string matching problem [CR94] has attracted a lot of interest throughout the history of computer science, and is crucial to the computing industry. String matching is finding an occurrence of a pattern string in a larger string of text. This problem arises in many computer packages in the form of spell checkers, search engines on the internet, find utilities on various machines, matching of DNA strands and so on.

Section 2 describes string matching algorithms which are known to be fast. Section 3 gives experimental results for these algorithms. From the findings of the experimental results discussed in Section 3, we identify two fast algorithms to produce a new algorithm. The new algorithm is described in Section 4. In Section 5 we compare the new algorithm with the existing algorithms.

## 2 The String Matching Algorithms

String matching algorithms work as follows. First the pattern of length $m$, $P[1..m]$, is aligned with the extreme left of the text of length $n$, $T[1..n]$. Then the pattern characters are compared with the text characters. The algorithms can vary in the order in which the comparisons are made. After a mismatch is found the pattern is shifted to the right and the distance the pattern can be shifted is determined by the algorithm that is being used. It is this shifting procedure and the speed at which a mismatch is found which is the main difference between the string matching algorithms.

In the Naive or Brute Force (BF) algorithm, the pattern is aligned with the extreme left of the text characters and corresponding pairs of characters are compared from left to right. This process continues until either the pattern is exhausted or a mismatch is found. Then the pattern is shifted one place to the right and the pattern characters are again compared with the corresponding text characters from left to right until either the text is exhausted or a full match is obtained. This algorithm can be very slow. Consider the worst case when both pattern and text are all $a$'s followed by a $b$. The total number of comparisons in the worst case is $O(nm)$. However, this worst case example is not one that occurs often in natural language text.

An improved version of the BF algorithm, the Not So Naive (NSN) algorithm [HA93], changes the order of the comparisons. Suppose the pattern is aligned with the text characters, first the second pattern character is compared with the corresponding text character followed by comparisons of the rest of the pattern with corresponding text characters, and then the last characters to be compared are the first character of the pattern and the text character it is aligned with. A shift of two is made if a mismatch is made with the second character of the pattern and the first two characters of the pattern are the same, or if the second character of the pattern matches the text but a mismatch occurs and the first two characters are not equal.

The number of comparisons can be reduced by moving the pattern to the right by more than one position when a mismatch is found. This is the idea behind the Knuth-Morris-Pratt (KMP) algorithm [KMP77]. The KMP algorithm starts and compares the characters from left to right the same as the BF algorithm. When a mismatch occurs the KMP algorithm moves the pattern to the right by maintaining the longest overlap of a prefix of the pattern with a suffix of the part of the text that has matched the pattern so far. After a shift, the pattern character compared against the mismatched text character has to be different from the character that mismatched. The KMP algorithm takes at most $2n$ character comparisons. The KMP algorithm does $O(m+n)$ operations in the worst case.

The Colussi (COL) [CO91] algorithm is an improvement of the KMP algorithm. The number of character comparisons is $1.5n$ in the worst case. The set of pattern positions is divided into two disjoint subsets due to the combinatorial properties of their positions. First the comparisons are performed from left to right for the characters at positions in the first set. If there is no mismatch, the characters at positions in the second set are compared from right to left. This strategy reduces the number of comparisons.

Galil and Giancarlo (GG) [GG92] improved the COL algorithm by reducing the number of character comparisons in the worst case to $\frac{4}{3}n$. In these algorithms the

preprocessing takes $O(m)$ time.

The Boyer-Moore (BM) algorithm [BM77] differs in one main feature from the algorithms already discussed. Instead of the characters being compared from left to right, in the BM algorithm the characters are compared from right to left starting with the rightmost character of the pattern. In a case of mismatch it uses two functions, last occurrence function and good suffix function and shifts the pattern by the maximum number of positions computed by these functions. The good suffix function returns the number of positions for moving the pattern to the right by the least amount, so as to align the already matched characters with any other substring in the pattern that are identical. The number of positions returned by the last occurence function determines the rightmost occurrence of the mismatched text character in the pattern. If the text character does not appear in the pattern then the last occurence function returns $m$. The worst case running time of the BM algorithm is $O(mn)$.

The Turbo Boyer-Moore (TBM) algorithm [CC94] and the Apostolico-Giancarlo (AG) algorithm [AG86] are ameliorations of the BM algorithm. When a partial match is made between the pattern and the text these algorithms remember the characters that matched and do not compare them again with the text. The TBM algorithm and the Apostolico-Giancarlo algorithm perform in the worst case at most $2n$ and $1.5n$ character comparisons respectively [CL97b].

The Horspool (HOR) algorithm [HO80] is a simplification of the BM algorithm. It does not use the good suffix function, but uses a modified version of the last occurrence function. The modified last occurrence function determines the right most occurrence of the $(k+m)$th text character, $T[k+m]$ in the pattern, if a mismatch occurs when a pattern is aligned with $T[k..k+m]$. This algorithm changes the order in which characters of the pattern are compared with the text. It compares the rightmost character in the pattern first then compares the leftmost character, then all the other characters in ascending order from the second position to the $m-1$th position.

The Raita (RAI) algorithm [RA92] again changes the order in which characters of the pattern are compared with the text. The process used to compare the rightmost character of the pattern, then the leftmost character, then the middle character and then the rest of the characters from the second to the $(m-1)$th position. If at any time during the procedure a mismatch occurs then it performs the shift as in the HOR algorithm.

The Quicksearch (QS) algorithm [SU90] is similar to the HOR algorithm and the RAI algorithm. It does not use the good suffix function to compute the shifts. It uses a modified version of the last occurrence function. Assume that a pattern is aligned with the text characters $T[k..k+m]$. After a mismatch the length of the shift is at least one. So, the character at the next position in the text after the alignment ($T[k+m+1]$) is necessarily involved in the next attempt. The last occurrence function determines the right most occurrence of $T[k+m+1]$ in the pattern. If $T[k+m+1]$ is not in the pattern the pattern can be shifted by $m+1$ positions. The comparisons between text and pattern characters during each attempt can be done in any order.

The Maximal Shift (MS) algorithm [SU90] is another variant of the QS algorithm. The algorithm is designed in such a way that the pattern characters are compared in the order which will give the maximum shift if a mismatch occurs.

The Smith (SMI) algorithm [SM91] uses HOR and Quick Search last occurrence functions. When a mismatch occurs, it takes the maximum values between these

functions.

The Zhu and Takaoka (ZT) algorithm [ZT87] is another variant of the BM algorithm. The comparisons are done in the same way as BM (i.e. from right to left) and it uses the good suffix function. If a mismatch occurs at $T[i]$, the last occurrence function determines the right most occurrence of $T[i-1..i]$ in the pattern. If the substring is in the pattern, the pattern and text are aligned at these two characters for the next attempt. The shift is $m$, if the two character substring is not in the pattern.

Searching can be done in $O(n)$ time using a minimal Deterministic Finite Automaton (DFA) [SI93]. This algorithm uses $O(\sigma m)$ space and $O(\sigma + m)$ pre-processing time, where $\sigma$ is the size of the alphabet. The Simon (SIM) algorithm [SI93] reduces the pre-processing time and the space to $O(m)$.

The pre-processing is needed for the algorithm to calculate the relevant shifts upon a mismatch/match except for the BF algorithm which has no pre-processing. The pre-processing cost of the algorithms does not effect the efficiency of the algorithms as they are relatively very small and all are approximately the same.

# 3 Experimental Results of the Existing Algorithms

Monitoring the number of comparisons performed by each algorithm was chosen as a way to compare the algorithms. All the algorithms were coded in C and their C code are taken from [CL97a] and animations of the algorithms can be found at [CL98]. This collection of string matching algorithms were easy to implement as functions into our main control program. The algorithms were coded as their authors had devised them in their papers. The main control program read in the text and pattern and had one of the algorithms to be tested inserted into it for the searching process. The main control program was the same for each algorithm and so did not affect the performance of the algorithms. Each algorithm had an integer counter inserted into it, to count the number of comparisons made between the pattern and the text. The counter was incremented by one each time a comparison was made.

A random text of 200,000 words from the UNIX English dictionary was used for the first set of experiments. The random text was constructed so as to simulate an actual English text. All the letters in the UNIX dictionary were made lower case to increase the probability of a match. In English text roughly only every 1 in 10 words begin with a capital letter. We decided to number each of the words in UNIX dictionary from 1 to 25,000. Then we used a pseudo random number generator to pick words from the UNIX dictionary and place them in the random text. Separating each word by a space character. Punctuation was also removed as we were concerned with finding words and the punctuation would not effect the results obtained. The reason for using a large text (200,000 words) was to ensure that as many of the 25,000 words in the UNIX English dictionary occurred somewhere in the random text generated. For each pattern in the dictionary, we searched the text (of 200,000 words) for the first occurrence of the pattern.

The text was searched for each word in the UNIX dictionary and the results are given in Table 1. The first column in Table 1 is the length of the pattern. The second

column is the number of words of that length in the UNIX English dictionary. For example, for a pattern length of 7, 4042 test cases were carried out and the average number of character comparisons made by the KMP algorithm was 197,000 (to the nearest 1000). The average was calculated by taking the total number of comparisons performed to find all 4042 cases and dividing this number by 4042. These columns are arranged in descending order of the average of the total number of comparisons of the algorithms. An interesting observation is that for (almost) each row the values are in descending order except for the last two columns.

| p. len | num. | BF | KMP | DFA | SIM | NSN | COL | GG | BM | AG | HOR | RAI | TBM | MS | QS | ZT | SMI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 133 | 7 | 7 | 7 | 7 | 6 | 6 | 6 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 3 | 2 |
| 3 | 765 | 38 | 38 | 37 | 37 | 37 | 37 | 37 | 13 | 13 | 13 | 13 | 13 | 11 | 10 | 13 | 10 |
| 4 | 2178 | 82 | 82 | 80 | 80 | 80 | 79 | 79 | 23 | 23 | 23 | 23 | 22 | 19 | 19 | 22 | 18 |
| 5 | 3146 | 151 | 150 | 145 | 145 | 145 | 145 | 145 | 34 | 34 | 34 | 34 | 34 | 30 | 30 | 32 | 28 |
| 6 | 3852 | 186 | 185 | 179 | 179 | 179 | 178 | 178 | 36 | 36 | 36 | 36 | 36 | 33 | 32 | 33 | 30 |
| 7 | 4042 | 198 | 197 | 191 | 191 | 191 | 190 | 190 | 34 | 34 | 34 | 34 | 34 | 32 | 31 | 30 | 28 |
| 8 | 3607 | 205 | 204 | 197 | 197 | 197 | 196 | 196 | 32 | 32 | 31 | 32 | 31 | 30 | 29 | 27 | 26 |
| 9 | 3088 | 212 | 211 | 204 | 204 | 204 | 203 | 203 | 30 | 30 | 30 | 30 | 30 | 29 | 28 | 25 | 24 |
| 10 | 1971 | 220 | 219 | 212 | 212 | 212 | 210 | 210 | 29 | 29 | 29 | 29 | 29 | 28 | 27 | 24 | 23 |
| 11 | 1120 | 209 | 207 | 201 | 201 | 200 | 198 | 198 | 26 | 26 | 26 | 26 | 25 | 25 | 24 | 21 | 21 |
| 12 | 593 | 218 | 217 | 210 | 210 | 209 | 207 | 207 | 25 | 25 | 25 | 25 | 25 | 24 | 24 | 21 | 20 |
| 13 | 279 | 224 | 222 | 215 | 215 | 213 | 212 | 212 | 24 | 24 | 24 | 24 | 24 | 23 | 23 | 19 | 19 |
| 14 | 116 | 228 | 227 | 220 | 220 | 219 | 217 | 217 | 23 | 23 | 23 | 23 | 23 | 23 | 23 | 19 | 19 |
| 15 | 44 | 151 | 150 | 144 | 144 | 143 | 142 | 142 | 15 | 15 | 15 | 15 | 14 | 14 | 14 | 11 | 12 |
| 16 | 17 | 227 | 225 | 217 | 217 | 215 | 214 | 214 | 20 | 21 | 21 | 21 | 20 | 20 | 20 | 18 | 16 |
| 17 | 7 | 233 | 231 | 222 | 222 | 221 | 218 | 218 | 20 | 20 | 20 | 20 | 19 | 19 | 20 | 15 | 16 |
| 18 | 4 | 236 | 234 | 225 | 225 | 223 | 221 | 221 | 19 | 20 | 20 | 20 | 19 | 19 | 20 | 14 | 16 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1 | 132 | 131 | 122 | 122 | 121 | 119 | 119 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 7 | 8 |
| 21 | 2 | 311 | 309 | 295 | 295 | 290 | 288 | 288 | 24 | 24 | 25 | 25 | 23 | 23 | 24 | 15 | 18 |
| 22 | 1 | 491 | 486 | 455 | 455 | 451 | 445 | 445 | 33 | 33 | 33 | 33 | 33 | 31 | 34 | 22 | 27 |
| total | 24966 | 180 | 179 | 174 | 174 | 173 | 172 | 172 | 31 | 31 | 30 | 30 | 30 | 28 | 28 | 27 | 25 |

Table 1: Results of searching a text of 200,000 words for each word in the UNIX dictionary.

The algorithm with the largest number of comparisons is the BF algorithm. This is because the algorithm shifts the pattern by one place to the right when a mismatch occurs, no matter how much of a partial/full match has been made. This algorithm has a quadratic worst case time complexity. But the KMP algorithm which has a linear worst case time complexity, does roughly the same number of comparisons as the BF algorithm. The reason for this is that in a natural language a multiple occurrence of a substring in a word is not common. For the same reason, the KMP variants, COL and GG algorithms have only a small improvement over the KMP algorithm. Other linear time algorithms, DFA and SIM, also have roughly the same number of comparisons as the BF algorithm. We will see below that the other quadratic worst case time complexity algorithms perform much better than these linear worst case time algorithms. This is a good example showing that asymptotic worst-case running time analysis can be indicative of how algorithms are likely to perform in practice, but they are not sufficiently accurate to predict actual performance.

The BM algorithm uses the good suffix function to calculate the shift which depends on a reoccurrence of a substring in a word. But, it also uses the last occurrence function. It is this last occurrence function that reduces the number of comparisons significantly. In practice, on an English text, the BM algorithm is three or more times faster than the KMP algorithm [SG82]. ¿From Table 1 one can see that the KMP algorithm is takes six times more comparisons than the BM algorithm on average. The other algorithms, TBM, AG, HOR, RAI, QS, MS, SMI and ZT, are variants of the BM algorithm. The number of comparisons for these algorithms is roughly the same number as in the BM algorithm.

The SMI algorithm and the ZT algorithm do the least number of comparisons for pattern lengths less than or equal to twelve and greater than twelve respectively.

# 4 The New Algorithm - the BR algorithm

¿From the findings of the experimental results discussed in section 3, it is clear that the SMI and ZT algorithms have the lowest number of comparisons among the others. We combined the calculations of a valid shift in SMI and ZT algorithms to produce a more efficient algorithm. If a mismatch occurs when the pattern $P[1..m]$ is aligned with the text $T[k+1..k+m]$, the shift is calculated by the rightmost occurrence of the substring $T[k+m+1..k+m+2]$ in the pattern. If the substring is in the pattern then the pattern and text are aligned at this substring for the next attempt. This can be done shifting the pattern as shown in the table below. Let $*$ be a wildcard character that is any character in the ASCII set. Note that if $T[k+m+1..k+m+2]$ is not in the pattern, the pattern is shifted by $m+2$ positions. The total number of comparisons in the worst case is $O(nm)$.

| $T[k+m+1]$ | $T[k+m+2]$ | $Shift$ |
|---|---|---|
| $*$ | $P[1]$ | $m+1$ |
| $P[i]$ | $P[i+1]$ | $m-i+1$, $1 \leq i \leq m-1$ |
| $P[m]$ | $*$ | 1 |
| Otherwise | | $m+2$ |

For example, the following shifts would be associated with the pattern, onion.

| $T[k+m+1]$ | $T[k+m+2]$ | $Shift$ |
|---|---|---|
| $*$ | $o$ | 6 |
| $o$ | $n$ | 5 |
| $n$ | $i$ | 4 |
| $i$ | $o$ | 3 |
| $o$ | $n$ | 2 |
| $n$ | $*$ | 1 |
| Otherwise | | 7 |

After a mismatch the calculation of a shift in our BR algorithm takes $O(1)$ time. Note that for the substrings ni and n* have a value of 4 and 1 respectively. This ambiguity can be solved by the higher shift value being overwritten with the lower value. We will explain this later in this section. For a given pattern $P[1..m]$ the preprocessing is done as follows, and it takes $O(\sigma^2)$ time.

There are 128 characters in the ASCII set and $(128)2 = 16384$ distinct pairs. We define an array Shift Array (SA) of length 16384 and initialise it to $m+2$. Using a hash function we insert the values for each pair and the hash function we use is:

$T[m+k+1] \times 127 + T[m+k+2]$ where for $P[m+k+1]$ and $P[m+k+2]$ we use their ASCII values. This gives each pair of character a distinct value in SA and we insert into the SA the shift for the pair. If the same pair occurs more than once then the lower shift value overwrites the higher value. So for example for the pair $[i][o]$ we would insert the value 3 at the $[105 \times 127] + 111 = 13446th$ position in SA.

$[wildcard][o] = 6$ all array positions that satisfy $x[0] mod 127 = 111 mod 127 = 6$
$[o][n] = 5$ position $111 \times 127 + 110 = 14207$
$[n][i] = 4$ position $110 \times 127 + 105 = 14075$
$[i][o] = 3$ position $105 \times 127 + 111 = 13446$
$[o][n] = 2$ position $111 \times 127 + 110 = 14207$
$[n][wildcard] = 1$ position $110 \times 127 + 0..127 = 13970..14097$

The order of performing the steps is important in ensuring the correct values appear in the array. Note that the higher values have been over written by the lower

values.

In the RAI algorithm the first and last characters of the pattern are made variables. This cuts down the number of array look ups performed during a search. We adapted this idea to our algorithm and compared the least frequent pattern character with its corresponding text character. We then repeated the process for the second least frequent character and then the rest of the characters in order from right to left.

The UNIX dictionary used in the tests was used to see how many times each letter occurred in the dictionary. The frequency of each letter is given in the following chart.

| letter | frequency | ranking | letter | frequency | ranking | letter | frequency | ranking |
|---|---|---|---|---|---|---|---|---|
| a | 16395 | 25 | j | 432 | 3 | s | 10167 | 19 |
| b | 4110 | 10 | k | 1923 | 6 | t | 12789 | 22 |
| c | 8209 | 17 | l | 10013 | 18 | u | 6476 | 16 |
| d | 5763 | 14 | m | 5822 | 15 | v | 1890 | 5 |
| e | 20083 | 26 | n | 12062 | 20 | w | 1950 | 7 |
| f | 2660 | 8 | o | 12696 | 21 | x | 616 | 4 |
| g | 4125 | 11 | p | 5514 | 13 | y | 3618 | 9 |
| h | 5179 | 12 | q | 377 | 1 | z | 429 | 2 |
| i | 13963 | 24 | r | 13409 | 23 | | | |

Note that we choose the characters in the pattern that have the lowest ranking. If the character is not in the pattern then it has a ranking of 0 and is chosen as the least frequent character.

We now give an example of our BR algorithm in action to find the pattern onion. The SA array for the pattern onion were used to calculate the shift after a mismatch. $P[2]$ is the least frequent and $P[5]$ is the next least frequent character.

| w | e | | w | a | n | t | | t | o | | t | e | s | t | | w | i | t | h | | o | n | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ≠ | | | | | | | | | | | | | | | | | | | | | | | | |
| o | n | i | o | n | | | | | | | | | | | | | | | | | | | | | |

mismatch shift on SA($[n][t]$) = 110 × 127 + 116 = SA[14086] = 1

| w | e | | w | a | n | t | | t | o | | t | e | s | t | | w | i | t | h | | o | n | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ≠ | | | | | | | | | | | | | | | | | | | | | | | |
| | o | n | i | o | n | | | | | | | | | | | | | | | | | | | | |

mismatch shift on SA($[t][]$) = 116 × 127 + 32 = SA[14764] = 7.

| w | e | | w | a | n | t | | t | o | | t | e | s | t | | w | i | t | h | | o | n | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | ≠ | | | | | | | | | | | | | | | | |
| | | | | | | | | o | n | i | o | n | | | | | | | | | | | | | |

mismatch shift on SA($[s][t]$) = 115 × 127 + 116 = SA[14721] = 7

| w | e | | w | a | n | t | | t | o | | t | e | s | t | | w | i | t | h | | o | n | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | ≠ | | | | | | | | | |
| | | | | | | | | | | | | | | | o | n | i | o | n | | | | | | |

mismatch shift on SA($[][o]$) = 32 × 127 + 111 = SA[4175] = 6.

| w | e | | w | a | n | t | | t | o | | t | e | s | t | | w | i | t | h | | o | n | i | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | | | = | = | = | = | = |
| | | | | | | | | | | | | | | | | | | | | | 5 | 1 | 4 | 3 | 2 |
| | | | | | | | | | | | | | | | | | | | | | o | n | i | o | n |

So the word onion is found in 9 comparisons in a text of length 26. On the above full match the order in which the comparisons are conducted is shown on the third row.

# 5 Experimental Results and Comparisons with the BR Algorithm

We select the best nine algorithms from the results in Table 1 and the KMP algorithm, and compare with our BR algorithm. Experiments were carried out for different random texts as described in Section 3. The texts were constructed by randomly choosing words from the UNIX English dictionary. There were 2 different texts of 10,000 words, a text of 50,000 words and a text of 100,000 words. The results are described in Tables 3-6 (see appendix) respectively. Tables 3-6 (which can be found in the appendix at the back of this paper) show the average number of comparisons required for a search for the given pattern length. They are based on taking the total number of comparisons for the search for all the patterns of a length and dividing the number by the number of patterns of that size to give the average. So for example, in Table 3 the BM algorithm takes 12,000 comparisons (to the nearest thousand) on average if the pattern length is 7. From these tables one can observe that the relative order of their performance is the same as in Table 1. The main observation is that the BR algorithm performs better than the other algorithms for all pattern lengths and for all texts used in the experiments.

| p. len. | num. | KMP | AG | BM | HOR | RAI | TBM | MS | QS | ZT | SMI |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 133 | 199.98 | 93.96 | 93.96 | 94.00 | 93.96 | 93.89 | 35.94 | 32.92 | 93.96 | 31.48 |
| 3 | 765 | 366.02 | 64.09 | 64.18 | 64.20 | 64.19 | 63.70 | 28.78 | 28.21 | 60.03 | 24.93 |
| 4 | 2178 | 449.02 | 50.97 | 51.11 | 50.86 | 50.90 | 50.77 | 28.25 | 25.77 | 43.19 | 19.73 |
| 5 | 3146 | 540.11 | 44.91 | 45.02 | 44.58 | 44.46 | 44.72 | 28.33 | 26.47 | 33.91 | 18.13 |
| 6 | 3852 | 626.30 | 42.58 | 42.42 | 41.83 | 41.68 | 41.91 | 30.02 | 27.32 | 27.71 | 16.42 |
| 7 | 4042 | 719.01 | 42.07 | 41.38 | 40.92 | 41.00 | 40.72 | 31.49 | 28.83 | 24.94 | 16.08 |
| 8 | 3607 | 807.61 | 40.76 | 40.58 | 40.28 | 40.35 | 39.95 | 32.27 | 30.10 | 21.67 | 15.49 |
| 9 | 3088 | 896.18 | 41.85 | 41.52 | 40.92 | 40.84 | 40.69 | 34.75 | 32.19 | 19.29 | 15.45 |
| 10 | 1971 | 982.63 | 42.38 | 42.19 | 41.69 | 41.79 | 41.16 | 36.62 | 34.37 | 17.75 | 15.64 |
| 11 | 1120 | 1067.87 | 44.91 | 44.14 | 43.67 | 43.79 | 42.97 | 38.57 | 37.18 | 17.06 | 16.32 |
| 12 | 593 | 1164.14 | 45.36 | 45.28 | 44.58 | 44.68 | 44.20 | 40.06 | 39.28 | 16.14 | 17.34 |
| 13 | 279 | 1245.53 | 48.85 | 47.88 | 47.22 | 47.32 | 46.36 | 42.26 | 41.61 | 12.65 | 17.54 |
| 14 | 116 | 1322.70 | 46.46 | 46.74 | 46.46 | 46.60 | 45.16 | 42.62 | 42.26 | 11.32 | 17.03 |
| 15 | 44 | 1426.02 | 50.78 | 51.20 | 51.51 | 51.59 | 49.23 | 44.73 | 45.29 | 8.72 | 19.00 |
| 16 | 17 | 1527.28 | 48.99 | 49.34 | 50.44 | 50.60 | 47.37 | 46.60 | 49.06 | 24.80 | 20.02 |
| 17 | 7 | 1598.50 | 45.09 | 45.29 | 44.51 | 44.58 | 43.42 | 40.22 | 45.01 | 6.72 | 16.95 |
| 18 | 4 | 1700.81 | 50.34 | 50.58 | 53.96 | 54.06 | 48.54 | 50.12 | 53.59 | 6.09 | 22.21 |
| 19 | 0 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| 20 | 1 | 1948.74 | 58.37 | 58.37 | 58.12 | 58.07 | 58.37 | 52.25 | 63.51 | 3.01 | 29.43 |
| 21 | 2 | 1947.96 | 58.13 | 57.38 | 63.98 | 63.99 | 56.32 | 57.59 | 57.50 | 2.22 | 21.84 |
| 22 | 1 | 2129.14 | 50.97 | 50.97 | 49.87 | 49.89 | 50.97 | 45.07 | 55.43 | 1.04 | 25.09 |
| total | 24992 | 737.56 | 43.54 | 43.29 | 42.83 | 42.82 | 42.65 | 32.00 | 29.72 | 26.09 | 16.66 |

Table 2: The average difference between each of the existing algorithms and our BR algorithm as a percentage.

Table 2 summarises the results of Tables 3-6. The entries in Table 2 are in percentage form and describe how many fewer comparisons our BR algorithm uses, when compared with the existing algorithms. The figures are an average of the four different texts used. To calculate the difference as a percentage between our BR algorithm and the existing algorithms we used the following formula. The average number of comparisons was taken from the relevant cell in Tables 3-6 and divided by the value for that pattern length for our BR algorithm. This value was then deducted by 1 and multiplied by 100 to give the percentage difference between the two algorithms. An interesting observation of the existing algorithms when compared with the BR algorithm, is that for each individual text the percentages were within 1% for each specific algorithm. Each value in Table 2 is calculated by taking the difference as a percentage between each algorithm and our BR algorithm for each pattern length, adding them together and dividing by 4. For example, for a pattern length of 4 the BM algorithm takes on average 51.11% more comparisons than our BR algorithm.

The result of a full search for the dictionary over all four texts is given in the last

row of Table 2. From this we can see that the BM algorithm took on average 43.54% more comparisons than our BR algorithm (see 5th column, last row) for a complete search for all the words in the dictionary.

Further to counting the number of comparisons we time the algorithms. The saving in the number of comparisons may be paid for by extra overhead due to accessing the precomputed shift table. We timed the search of the medium text of 50,000 words for all occurrences of the words in the UNIX dictionary. We used a 486-DX66 with 8 megabytes of RAM and a 100 megabyte hard drive running SUSE 5.2. In Table 7, the total number of comparisons for the search are given along with the time taken by each algorithm for the search, including any preprocessing performed by the algorithm. The number of comparisons are reduced by a factor of 1000. i.e. for BF 10911786 means 10911786000 comparisons.

| | medium1 | | | book1 | | | book2 | | papers | |
|---|---|---|---|---|---|---|---|---|---|---|
| | number | time | % dif BR | num. comp. | time sec. | % dif. BR | time | % dif. BR | time | % dif. BR |
| BF | 10911786 | 1315m 13s | 528.54 | | | | | | | |
| KMP | 10433340 | 1341m 25s | 541.06 | | | | | | | |
| DFA | 10433340 | 892m 59s | 326.75 | | | | | | | |
| SIM | 10433340 | 1688m 18 | 706.83 | | | | | | | |
| NSN | 10482487 | 777m 52s | 271.74 | | | | | | | |
| BM | 2002822 | 371m 51s | 77.71 | 3602739 | 674m | 79.73 | 663s | 69.57 | 264s | 58.08 |
| AG | 2005310 | 972m 10s | 364.60 | | | | | | | |
| HOR | 1985219 | 244m 41s | 16.93 | 3580863 | 442m | 17.87 | 446s | 14.07 | 249s | 49.10 |
| RAI | 1998657 | 238m 27s | 13.95 | 3601251 | 431m | 14.93 | 434s | 11.00 | 173s | 3.59 |
| MS | 1815486 | 318m 49s | 52.36 | | | | | | | |
| QS | 1785730 | 245m 58s | 17.55 | 3189368 | 444m | 18.40 | 452s | 15.60 | 180s | 7.78 |
| ZT | 1761716 | 420m 55s | 101.15 | | | | | | | |
| TBM | 1683516 | 1166m 4s | 457.26 | | | | | | | |
| SMI | 1621591 | 280m 41s | 34.14 | 2930285 | 513m | 36.80 | 514s | 31.46 | 207s | 23.95 |
| BR | 1489839 | 209m 15s | n/a | 2682916 | 375m | n/a | 391s | n/a | 167s | n/a |

Table 7: Timing for a complete search for the dictionary in the given texts.

¿From this table we can see that the algorithms that take a high number of comparisons are quite slow as well. The lower the number of comparisons the better the time. Although putting the algorithms in order of how many comparisons they take from highest to lowest starting at the BM we get the list: BM, RAI, AG, HOR, MS, QS, ZT, TBM, SMI and the BR. If we do the same for the timings we get ZT, BM, MS, SMI, QS, RAI and the BR. The reason for the difference in the lists is due to overheads in traversing the data structures which are present in the algorithms for the calculation of the correct shift value. Also the pre-processing of some of the algorithms are expensive. So we can not assume that because an algorithm takes a fewer number of comparisons that it will be more efficient than another.

We can also save time by performing the comparisons as in the RAI algorithm. This is done by making the least and second least likely characters variables instead having to look them up in the pattern array. Although counting the comparisons is a good estimate of which algorithm is the best to use we have to actual time the algorithms to find the best algorithm for the task of string matching.

We repeated the tests for the medium text for the book1 text for the 5 algorithms with the best times and our BR algorithm. From Table 7 we can see that our BR algorithm is still the quickest and the other algorithms are still over 14% more time than our algorithm. So our findings for a random text hold for this real English text. We then considered two other texts, book2 and the six papers concatenated together from the Calgary corpus [CAL]. We searched for 500 random words from the UNIX dictionary again for the best 5 algorithms and our BR algorithm. The results documented in Table 7 show that algorithm is the fastest algorithm for these tests. The main reason for the speed of our BR algorithm is the improved maximum shift

of $m + 2$ and the searching on the least likely to occur characters.

# Conclusions

The experimental results show that the BR algorithm is more efficient than the existing algorithms in practice for our chosen data sets. Over our 4 random texts and 3 real texts where the BR algorithm is compared to the existing algorithms, our algorithm is comfortably more efficient over each text. With the addition of punctuation and capital letters it does not affect the BR algorithm. If the pattern to be searched for began with a capital letter then this would make the capital letter the least frequent character and so it would be searched for first. We would expect the probability of a mismatch to rise and so the algorithm would speed up considerably. So in the real world we would expect our savings to remain and make our BR algorithm competitive with the existing algorithms. It is also possible to apply some of our finding to what makes a fast algorithm to the existing algorithms. This may make them faster but we were concerned with the original algorithms that were devised by their authors.

## Acknowledgments

We wish to thank Carl Bamford for comments and suggestions made to us during the writing of this paper.

# References

[AG86] Apostolico A., Giancarlo R., *"The Boyer-Moore-Galil string strategies revisited"*, SIAM Journal of Computing, 15(1), pages 98-105, 1986.

[BM77] Boyer R. S., Moore J. S., *"A fast string searching algorithm"*, Communications of the ACM, 23(5), pages 1075-1091, 1977.

[CAL] Calgary Corpus available at: ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/

[CL97a] Charras C., Lecroq T., Exact string matching available at: HTTP://www.dir.univ-rouen.fr/ lecroq/string.ps, 1997.

[CL98] Charras C., Lecroq T., Exact string matching animation in JAVA available at: HTTP://www. dir.univ-rouen.fr/ charras/string/, 1998.

[CO91] Colussi L., *"Correctness and efficiency of the pattern matching algorithms"*, Information Computing, 95(2), pages 225-251, 1991.

[CC94] Crochemore M., Czumaj A., Gąsieniec L., Jarominek T., Lecroq T., Plandowski W., Rytter W., *"Speeding up two string matching algorithms"*, Algorithmica, 12(4), pages 247-267, 1994.

[CL97b] Crochemore M., Lecroq T., *"Tight bounds on the complexity of the Apostolico-Giancarlo algorithm"*, Information Processing Letters, 63(4), pages 195-203, 1997.

[CR94] Crochemore M., Rytter W., *"Text algorithms"*, Oxford University Press, 1994.

[GG92] Galil Z., Giancarlo R., *"On the exact complexity of string matching: upper bounds"*, SIAM Journal of Computing, 21(3), pages 407-437, 1992.

[HA93] Hancart C., *"Analyse exacte et en moyenne d'algorithmes de recherche d'un motif dans un texte"*. Thése de doctorat de l'Université de Paris 7, France, 1993.

[HO80] Horspool R. N., *"Practical fast searching in strings"*. Software Practice and Experience. 10(6), pages 501-506, 1980.

[KMP77] Knuth D. E., Morris Jr J. H., Pratt V. R., *"Fast pattern matching in strings"*, SIAM Journal of Computing, 6(1), pages 323-350, 1977.

[RA92] Raita T., *"Tuning the Boyer-Moore-Horspool string searching algorithm"*, Software Practice and Experience, 22(10), pages 879-884, 1992.

[SI93] Simon I., *"String matching algorithms and automata"*, First American Workshop on String Processing, ed. Baeza-Yates and Ziviani, pages 151-157. Universidade Federal de Minas Gerais, 1993.

[SM91] Smith P. D., *"Experiments with a very fast substring search algorithm"*, Software Practice and Experience, 21(10), pages 1065-1074, 1991.

[SG82] de Smit G. V., *"A Comparison of Three String Matching Algorithms"*, Software Practice and Experience, 12(1), pages 57-66, 1982.

[SU90] Sunday D. M., *"A very fast substring search algorithm"*, Communications of the ACM, 33(8), pages 132-142, 1990.

[ZT87] Zhu R. F., Takaoka T., *"On improving the average case of the Boyer-Moore string matching algorithm"*, Journal of Information Processing, 10(3), pages 173-177, 1987.

# Appendix

| p len | num | KMP | AG | BM | HOR | RAI | TBM | MS | QS | ZT | SMI | BR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 133 | 6 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 3 | 2 | 2 |
| 3 | 765 | 20 | 7 | 7 | 7 | 7 | 7 | 6 | 6 | 7 | 5 | 4 |
| 4 | 2178 | 41 | 11 | 11 | 11 | 11 | 11 | 10 | 10 | 11 | 9 | 7 |
| 5 | 3146 | 60 | 14 | 14 | 13 | 13 | 13 | 12 | 12 | 12 | 11 | 9 |
| 6 | 3852 | 67 | 13 | 13 | 13 | 13 | 13 | 12 | 12 | 12 | 11 | 9 |
| 7 | 4042 | 68 | 12 | 12 | 12 | 12 | 12 | 11 | 11 | 10 | 10 | 8 |
| 8 | 3607 | 69 | 11 | 11 | 11 | 11 | 11 | 10 | 10 | 9 | 9 | 7 |
| 9 | 3088 | 70 | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 8 | 8 | 7 |
| 10 | 1971 | 71 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 8 | 8 | 6 |
| 11 | 1120 | 70 | 9 | 9 | 9 | 9 | 9 | 8 | 8 | 7 | 7 | 6 |
| 12 | 593 | 70 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 6 | 7 | 5 |
| 13 | 279 | 72 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 6 | 6 | 5 |
| 14 | 116 | 69 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 6 | 5 |
| 15 | 44 | 72 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 6 | 5 |
| 16 | 17 | 70 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 5 | 5 | 4 |
| 17 | 7 | 75 | 7 | 7 | 7 | 7 | 6 | 6 | 6 | 5 | 5 | 4 |
| 18 | 4 | 87 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 6 | 5 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1 | 89 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 4 | 5 | 4 |
| 21 | 2 | 88 | 7 | 7 | 7 | 7 | 7 | 6 | 7 | 4 | 5 | 4 |
| 22 | 1 | 89 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 4 | 5 | 4 |
| total | 24966 | 64 | 11 | 11 | 11 | 11 | 11 | 10 | 10 | 10 | 9 | 7 |

Table 3: Averages for random TEXT A of 10,000 words

| p len | num | KMP | AG | BM | HOR | RAI | TBM | MS | QS | ZT | SMI | BR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 133 | 6 | 3 | 3 | 3 | 3 | 3 | 2 | 2 | 3 | 2 | 2 |
| 3 | 765 | 21 | 7 | 7 | 7 | 7 | 7 | 6 | 6 | 7 | 6 | 4 |
| 4 | 2178 | 42 | 12 | 12 | 12 | 12 | 12 | 10 | 10 | 11 | 9 | 8 |
| 5 | 3146 | 59 | 13 | 13 | 13 | 13 | 13 | 12 | 12 | 12 | 11 | 9 |
| 6 | 3852 | 66 | 13 | 13 | 13 | 13 | 13 | 12 | 12 | 11 | 11 | 9 |
| 7 | 4042 | 68 | 12 | 12 | 12 | 12 | 12 | 11 | 11 | 10 | 10 | 8 |
| 8 | 3607 | 69 | 11 | 11 | 11 | 11 | 11 | 10 | 10 | 9 | 9 | 8 |
| 9 | 3088 | 70 | 10 | 10 | 10 | 10 | 10 | 9 | 9 | 8 | 8 | 7 |
| 10 | 1971 | 71 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 8 | 8 | 7 |
| 11 | 1120 | 70 | 9 | 9 | 9 | 9 | 9 | 8 | 8 | 7 | 7 | 6 |
| 12 | 593 | 71 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 6 | 7 | 6 |
| 13 | 279 | 71 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 6 | 6 | 5 |
| 14 | 116 | 70 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 6 | 6 | 5 |
| 15 | 44 | 64 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 5 | 5 | 4 |
| 16 | 17 | 74 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 5 | 5 |
| 17 | 7 | 64 | 6 | 6 | 6 | 6 | 6 | 5 | 6 | 4 | 4 | 4 |
| 18 | 4 | 87 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 5 | 6 | 5 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1 | 41 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 |
| 21 | 2 | 72 | 5 | 5 | 6 | 6 | 5 | 5 | 5 | 4 | 4 | 3 |
| 22 | 1 | 89 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 4 | 5 | 4 |
| total | 24966 | 63 | 11 | 11 | 11 | 11 | 11 | 10 | 10 | 10 | 9 | 8 |

Table 4: Averages for random TEXT B of 10,000 words

| p len | num | KMP | AG | BM | HOR | RAI | TBM | MS | QS | ZT | SMI | BR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 133 | 9 | 6 | 6 | 6 | 6 | 6 | 4 | 4 | 6 | 4 | 3 |
| 3 | 765 | 37 | 13 | 13 | 13 | 13 | 13 | 10 | 10 | 13 | 10 | 8 |
| 4 | 2178 | 77 | 21 | 21 | 21 | 21 | 21 | 18 | 18 | 20 | 17 | 13 |
| 5 | 3146 | 133 | 30 | 30 | 30 | 30 | 30 | 27 | 26 | 28 | 25 | 20 |
| 6 | 3852 | 159 | 31 | 31 | 31 | 31 | 31 | 29 | 28 | 28 | 26 | 21 |
| 7 | 4042 | 170 | 29 | 29 | 29 | 29 | 29 | 27 | 27 | 26 | 24 | 20 |
| 8 | 3607 | 176 | 27 | 27 | 27 | 27 | 27 | 26 | 25 | 24 | 22 | 19 |
| 9 | 3088 | 181 | 26 | 26 | 26 | 26 | 26 | 25 | 24 | 22 | 21 | 18 |
| 10 | 1971 | 185 | 24 | 24 | 24 | 24 | 24 | 23 | 23 | 20 | 20 | 17 |
| 11 | 1120 | 184 | 23 | 23 | 23 | 23 | 23 | 22 | 22 | 18 | 18 | 15 |
| 12 | 593 | 186 | 21 | 21 | 21 | 21 | 21 | 21 | 20 | 17 | 17 | 14 |
| 13 | 279 | 183 | 20 | 20 | 20 | 20 | 20 | 19 | 19 | 15 | 16 | 13 |
| 14 | 116 | 194 | 20 | 20 | 20 | 20 | 20 | 19 | 19 | 15 | 16 | 13 |
| 15 | 44 | 164 | 16 | 16 | 16 | 16 | 16 | 16 | 16 | 12 | 13 | 10 |
| 16 | 17 | 217 | 20 | 20 | 20 | 20 | 20 | 20 | 20 | 17 | 16 | 13 |
| 17 | 7 | 172 | 15 | 15 | 15 | 15 | 14 | 14 | 15 | 11 | 12 | 10 |
| 18 | 4 | 147 | 12 | 12 | 13 | 13 | 12 | 12 | 13 | 9 | 10 | 8 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1 | 41 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 2 | 3 | 2 |
| 21 | 2 | 221 | 17 | 17 | 18 | 18 | 17 | 17 | 17 | 11 | 13 | 10 |
| 22 | 1 | 397 | 27 | 27 | 27 | 27 | 27 | 26 | 28 | 18 | 22 | 17 |
| total | 24966 | 155 | 27 | 27 | 26 | 26 | 26 | 24 | 24 | 23 | 22 | 18 |

Table 5: Averages for random text of 50,000 words

| p len | num | KMP | AG | BM | HOR | RAI | TBM | MS | QS | ZT | SMI | BR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 133 | 13 | 7 | 7 | 7 | 7 | 7 | 5 | 5 | 7 | 5 | 3 |
| 3 | 765 | 37 | 13 | 13 | 13 | 13 | 13 | 10 | 10 | 13 | 10 | 8 |
| 4 | 2178 | 80 | 22 | 22 | 22 | 22 | 22 | 19 | 18 | 21 | 17 | 15 |
| 5 | 3146 | 149 | 34 | 34 | 34 | 34 | 34 | 30 | 29 | 31 | 28 | 23 |
| 6 | 3852 | 182 | 36 | 36 | 36 | 36 | 36 | 33 | 32 | 33 | 29 | 25 |
| 7 | 4042 | 193 | 33 | 33 | 33 | 33 | 33 | 31 | 30 | 29 | 27 | 24 |
| 8 | 3607 | 201 | 31 | 31 | 31 | 31 | 31 | 29 | 29 | 27 | 26 | 22 |
| 9 | 3088 | 198 | 28 | 28 | 28 | 28 | 28 | 27 | 26 | 24 | 23 | 20 |
| 10 | 1971 | 198 | 26 | 26 | 26 | 26 | 26 | 25 | 25 | 22 | 21 | 18 |
| 11 | 1120 | 199 | 25 | 25 | 25 | 24 | 24 | 24 | 23 | 20 | 20 | 17 |
| 12 | 593 | 217 | 25 | 25 | 25 | 25 | 25 | 24 | 24 | 20 | 20 | 17 |
| 13 | 279 | 207 | 23 | 23 | 23 | 23 | 22 | 22 | 22 | 18 | 18 | 15 |
| 14 | 116 | 180 | 20 | 19 | 19 | 19 | 19 | 18 | 18 | 14 | 15 | 13 |
| 15 | 44 | 218 | 22 | 22 | 22 | 22 | 21 | 21 | 21 | 17 | 17 | 14 |
| 16 | 17 | 162 | 15 | 15 | 15 | 15 | 15 | 15 | 15 | 12 | 12 | 10 |
| 17 | 7 | 220 | 20 | 20 | 20 | 20 | 19 | 19 | 19 | 14 | 15 | 13 |
| 18 | 4 | 208 | 17 | 17 | 17 | 17 | 17 | 17 | 18 | 12 | 14 | 11 |
| 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | 1 | 157 | 12 | 12 | 12 | 12 | 12 | 12 | 13 | 8 | 10 | 8 |
| 21 | 2 | 89 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 11 | 5 | 4 |
| 22 | 1 | 315 | 21 | 21 | 21 | 21 | 21 | 20 | 22 | 14 | 18 | 14 |
| total | 24966 | 173 | 30 | 30 | 30 | 30 | 29 | 27 | 27 | 26 | 24 | 21 |

Table 6: Averages for random text of 100,000 words