IOP Conf. Series: Materials Science and Engineering **122** (2016) 012034 doi:10.1088/1757-899X/122/1/012034

Combined string searching algorithm based on knuth-morris-pratt and boyer-moore algorithms

R Yu Tsarev[1], A S Chernigovskiy[1], E A Tsareva[2], V V Brezitskaya[2], A Yu Nikiforov[1], N A Smirnov[2]
[1]Siberian Federal University, Krasnoyarsk, Russia
79, Svobodny Prospect, Krasnoyarsk, Russia
[2]Siberian State Aerospace University, Krasnoyarsk, Russia
31 "Krasnoyarskiy Rabochiy" prospect, Krasnoyarsk, 660037, Russia

E-mail: tsarev.sfu@mail.ru

Abstract. The string searching task can be classified as a classic information processing task. Users either encounter the solution of this task while working with text processors or browsers, employing standard built-in tools, or this task is solved unseen by the users, while they are working with various computer programmes. Nowadays there are many algorithms for solving the string searching problem. The main criterion of these algorithms' effectiveness is searching speed. The larger the shift of the pattern relative to the string in case of pattern and string characters' mismatch is, the higher is the algorithm running speed. This article offers a combined algorithm, which has been developed on the basis of well-known Knuth-Morris-Pratt and Boyer-Moore string searching algorithms. These algorithms are based on two different basic principles of pattern matching. Knuth-Morris-Pratt algorithm is based upon forward pattern matching and Boyer-Moore is based upon backward pattern matching. Having united these two algorithms, the combined algorithm allows acquiring the larger shift in case of pattern and string characters' mismatch. The article provides an example, which illustrates the results of Boyer-Moore and Knuth-Morris-Pratt algorithms and combined algorithm's work and shows advantage of the latter in solving string searching problem.

1. Introduction

The searching problem is one of fundamental tasks of theoretical programming [10]. String searching is one of simple, but, nonetheless, extremely important problems. The importance of this problem is explained by the wide area of its solution results' application: text editors, online string matching, speech analysis and recognition, information retrieval, network content inspection, data compression, etc. [4].

Nowadays there are many algorithms for solving this problem developed. During the last thirty years numerous algorithms, which allow solving the problems of searching with some special characteristics, were offered. M. Ahmed, M. Kaykobad and R.A. Chowdhury developed a new string matching algorithm that unlike other sub-linear string-matching algorithms never performs more than n text character comparisons while working on a text of length n [1]. T. Lecroq proposed string matching algorithms based on hashing q-grams [9]. K. Fredriksson and S. Grabowski developed new exact bit-parallel string matching algorithm, based on the shift-or algorithm [2, 5]. They have shown how to adapt their techniques for the shift-add algorithm, obtaining optimal time for searching under Hamming distance [5]. L. He, B. Fang and J. Sui have used the concept of a window whose size is equal to pattern length. They presented a novel string matching algorithm named wide window algorithm [6]. A.

Hudaib et al. in [7] presented two sliding windows algorithm. The algorithm makes use of two sliding windows. Both windows slide in parallel over the text until the first occurrence of the pattern is found or until both windows reach the middle of the text.

The inexhaustible interest towards this sphere of problems confirms the importance and topicality of the string searching problem. This article offers an algorithm based upon two well-known and acknowledged string searching algorithms. This kind of combination allows increasing the effectiveness of solving the given problem.

2. Theoretical bases of the combined algorithm

The combined string searching algorithm is based upon two algorithms. One of them was developed by Knuth, Morris and Pratt, and the other one by Boyer and Moore. These algorithms belong to two quite big sub-classes of string searching algorithms. Knuth-Morris-Pratt algorithm belongs to forward pattern matching sub-class, and Boyer-Moore belongs to backward pattern matching sub-class. Two stages can be identified in the work of these algorithms:

1. Preparing of table d used in case of shifting of the pattern in the string.
2. String searching itself.

Let us have a look at the work of these string searching algorithms in detail. The description will be followed by examples and explanations with the help of C language.

2.1. Principles of Knuth-Morris-Pratt string searching algorithm

The algorithm was developed by D. Knuth and V. Pratt, and independently by J. H. Morris in 1974, but it was published by them collaboratively only in 1977 [8]. This algorithm is based upon the idea that after partial matching of the initial part of the pattern with the corresponding characters of the string, we get certain information on the basis of the pattern itself, and this information will allow us to move forward along the string not by one, as in case of naive string matching, but further. In order to do that, when shifting, Knuth-Morris-Pratt string searching algorithm (or KMP algorithm) uses table d, which is preprocessed before the beginning of string searching. Since table d preprocessed in compliance with KMP algorithm will also be used in combined algorithm, let us denote it as d_{KMP}.

Thus, table d_{KMP} for Knuth-Morris-Pratt string searching algorithm is preprocessed on basis of the pattern and contains values, which will be used further for calculation of the shift amount of the pattern. The size of the table is equal to the length of the pattern. Therefore, table d_{KMP} is in fact a one-dimensional array, consisting of the number of elements equal to the number of characters in the pattern.

The first element of the array d_{KMP} is always minus one.

Let us have a look at the formation of the table d_{KMP} in the example of the pattern “barbarian”. As it is shown in Figure 1, the first element of the table d, corresponding to the first character of the pattern “b” is equal to minus one.

b	a	r	b	a	r	i	a	n
–1								

Figure 1. Pattern and value of the first element of table d_{KMP}.

For the other characters, the values of the elements of table d_{KMP} are calculated in the following way. The value of $d_{KMP}[j]$ corresponding to character j of the pattern is equal to the

maximum number of characters, which directly precede the given character and match with the beginning of the pattern. If k characters precede the given character, then only k−1 preceding characters are taken into account.

Thus, Figure 2 shows that one character “b” directly precedes the fifth character “a”, and matches with character b at the beginning of the pattern. Both characters “b” are in bold type. Since the number of matching characters is equal to one, element of the table, corresponding to the fifth symbol of the pattern “a” receives value one (see Figure 2).

b	a	r	**b**	a	r	i	a	n
–1				1				

Figure 2. Pattern and value of the fifth element of table d_{KMP}.

Two characters “ba” directly precede the sixth character of the pattern r and match with the characters “ba” at the beginning of the pattern. Both pairs of characters “ba” are in bold type. Since the character “r” is preceded by two characters, matching with the first two characters of the pattern, the corresponding element of the table is given the value 2 (see Figure 3).

b	**a**	r	**b**	**a**	r	i	a	n
–1				1	2			

Figure 3. Pattern and value of the sixth element of table d_{KMP}.

No other characters of the pattern “barbarian” are preceded by characters, matching with the beginning of the pattern; therefore the corresponding elements of the table d_{KMP} are equal to zero (see Figure 4).

b	a	r	b	a	r	i	a	n
–1	0	0	0	1	2	0	0	0

Figure 4. Pattern and table d_{KMP}.

The second stage of KMP algorithm work is comparing characters of the string and characters of the pattern and calculating the shift amount in case of their mismatch. The characters are considered from the left to the right, i.e. from the beginning to the end of the pattern. In case of mismatch of characters of pattern and string, the pattern is shifted to the right along the string by following value:

$$j - d_{KMP}[j],$$

in which j is an index of the currently considered character of the pattern, $d_{KMP}[j]$ is the value of the table d_{KMP}, corresponding to this character.

Figure 5 shows an example illustrating the shift of the pattern during the work of KMP algorithm.

String	***B***	***a***	***r***	***b***	***e***	***r***	**…**		
Pattern	***B***	***a***	***r***	***b***	***i***				
j	0	1	2	3	4				
$d_{KMP}[j]$	–1	0	0	0	0				
					B	***a***	***r***	***b***	***i***

$j = 4$, $d_{KMP}[4] = 0$,
shift $= j - d_{KMP}[j] = 5$

Figure 5. Partial match and shift of the pattern.

Let us give an example of string searching of the pattern “barbarian”. Figure 6 shows the process of KMP algorithm work. The characters that are being compared are underlined.

```
b a r   i s   f u l l   o f   b a r b a r i a n s
b a r b a r i a n
    b a r b a r i a n
      b a r b a r i a n
        …
                                b a r b a r i a n
                                  b a r b a r i a n
```

Figure 6. Process of work of Knuth-Morris-Pratt string searching algorithm.

Pay attention to the fact that in every case of characters’ mismatch, the pattern is shifted by the whole passed length, because smaller shifts cannot lead to full match.

As far as the effectiveness of Knuth-Morris-Pratt string searching algorithm is concerned, its developers show that about n + m character comparisons are needed, which is better that n * m comparisons in case of naive string matching (where n is length of the string, m is length of the pattern, $0 < m < n$) [8, 10]. In this case, the string scanning indicator i never goes back, while in case of naive string matching, after a mismatch, the considering starts again from the first character, and therefore the string characters that have been considered before might be processed again.

However, string searching with the use of Knuth-Morris-Pratt string searching algorithm is helpful only in case if the mismatch of string and pattern characters was preceded by some number of matches. If the comparison of the string with the pattern shows that first characters are different, the pattern shifts by only one character. The pattern shifts more than by one only in case if several characters of string and pattern match.

2.2. Principles of Boyer-Moore string searching algorithm

In 1977 R.S. Boyer and J.S. Moore offered an algorithm, which not only improves the processing of the worst case, but also gives and advantage in general case [3]. Boyer-Moore string searching algorithm (or BM algorithm) is based upon an unusual idea –characters’ comparing starts not from the beginning, but from the end of the pattern. The speed of Boyer-Moore algorithm’s running is achieved at the expense of omitting parts of the text, which certainly do not participate in successful comparison.

As well as in case of KMP algorithm, table d is preprocessed basing upon the pattern before the beginning of searching, and then used when shifting of the pattern along the string.

Since this table will also be used in the work of the combined algorithm, let us denote it as d_{BM}.

Initially all the elements of the table d_{BM} are given the values equal to the length of the pattern.

On the next stage, every element of the table d_{BM} is given a value equal to the remoteness of the corresponding character of the pattern from the end of the pattern. Figure 7 shows the pattern and remoteness of each of its characters from the end of the pattern.

b	a	r	b	a	r	i	a	n
8	7	6	5	4	3	2	1	

Figure 7. Pattern and remoteness of each of its characters from the end of the pattern.

In case if a pattern contains several identical characters, the element of table d_{BM} corresponding to this character is given the value equal to remoteness from the end of the pattern of the rightmost character [10]. For example, the pattern "barbarian" contains three characters "a", and remoteness of the rightmost from the end of the pattern is equal to one. In connection with this, the elements of table d_{BM} corresponding to the rest of characters "a" in the pattern are given the value equal to one (see Figure 8). Similarly, elements of table d_{BM} corresponding to all the characters "r" are given the value of three, and the elements corresponding to characters "b" will have the value of five.

b	**a**	r	b	**a**	r	i	**a**	n
5	**1**	3	5	**1**	3	2	**1**	

Figure 8. Pattern and table d_{BM}.

During the implementation of Boyer-Moore string searching algorithm, as well as during the implementation of the combined algorithm, it is recommended to use the table of character codes ASCII for creation of table d_{BM}.

Every printed character has its own ASCII code. For example, the code of the character "a" is 97, the code of "r" is 114, and the code of space is 32. It can also be mentioned that the codes of Cyrillic characters are situated between 192 and 255.

Table ASCII contains 256 characters, that is why table d_{BM} can be named a one-dimensional integer-valued array consisting of 256 elements: int d[256].

Let us have a look at the peculiarities of software implementation of table d_{BM} through the example of the pattern "barbarian". Since this pattern contains nine characters, its length is nine. Correspondingly, on the stage of initialization all the elements of the table d_{BM} are given the value of nine:

$$d[0] = d[1] = \ldots = d[254] = d[255] = 9.$$

Then elements of table d_{BM} are given the corresponding values equal to the remoteness of the given character from the end of the pattern. The index of the element of table d_{BM} that is given a new value is identified by the ASCII code of the considered character.

For example, ASCII code of character "a" is 97. The value of the elements of table d_{BM} corresponding to it, as it was already shown on Figure 9, is equal to one. Therefore:

$$d[97] = 1.$$

It can also be written in the C language:

$$d[‘a’] = 1.$$

It can also be mentioned that during the implementation the element of array d with the index of 97 corresponds to all the characters “a” in the pattern “barbarian”. Thus, all the elements of table d_{BM} corresponding to the characters “a” of the pattern in such case are given the value of 1 (see Figure 9).

The values of the elements of table d_{BM} corresponding to characters “i”, “r”, “b” change in the similar way:

$$d[‘i’] = 2 \text{ or } d[105] = 2,$$
$$d[‘r’] = 3 \text{ or } d[114] = 3,$$
$$d[‘b’] = 5 \text{ or } d[98] = 5.$$

Finally, the element of table d_{BM} corresponding to the character situated in the end of the pattern should be given the value of one:

$$d[‘n’] = 1 \text{ or } \quad d[110] = 1.$$

The necessity of assigning value of one and not zero (since its remoteness from the end of the pattern is equal to zero) is connected with the peculiarities of calculation of shift amount directly during the string searching.

The second stage of BM algorithm work is string searching itself. While comparing the pattern and the string, the pattern moves from the left to the right along the string. However, pattern and string characters are compared from the right to the left along the pattern. That is the peculiarly of backward pattern matching.

The comparison of the pattern and the string is carried out 1) until the whole pattern is considered, which indicates that there is a match between the pattern and some part of the string, 2) until the string ends, which means that there are no entries matching the pattern in the string, 3) until there is a mismatch of the pattern and string characters, which leads to shifting of pattern by several characters to the right and continuing the searching process.

In case of characters’ mismatch, shift of the pattern along the string is defined by the value of the element of table d_{BM}. However, the index of the given element is the ASCII code of the character of the string. It can be emphasized that although array d is formed on the basis of the pattern, the shift is defined by the mismatching character of the string. Figure 9 shows an example of BM algorithm work. The characters that are being compared are underlined.

```
b a r   i s   f u l l   o f   b a r b a r i a n s
b a r b a r i a n
                  b a r b a r i a n
                        b a r b a r i a n
                              b a r b a r i a n
```

Figure 9. Process of work of Boyer-Moore string searching algorithm.

In the first iteration there was a mismatch of the pattern character n and string character u. It should be pointed out that the values of all the elements of array d for all the possible characters on the stage of initialization are equal to the length of the pattern, i.e. to nine. Since character "u" is not found in the pattern, the value of the element d['u'] in further formation of the table is not changed and remains equal to nine. That is why the pattern is shifted by nine characters to the right. If these two characters matched, the last but one pattern character and the corresponding string character would be considered next, etc.

While comparing the pattern and the string there is a mismatch of characters "n" and "r". Once again, defining the shift with the help of the string character (d['r']), we get the value of three. The pattern shifts to the right by three characters. Similarly, the pattern gradually shifts along the string, until the pattern is found in the string or until the string ends.

The evaluation of BM algorithm effectiveness shows that almost in all cases the algorithm demands far less n comparisons. In the most favorable conditions, when the last pattern character falls into the mismatching string character, the number of comparisons is n / m [3, 10].

2.3. Combination of Knuth-Morris-Pratt and Boyer-Moore string searching algorithms

Based upon the mentioned principles of Knuth-Morris-Pratt и Boyer-Moore string searching algorithms concerning the creation of tables for defining the shift and string searching itself the following combined algorithm has been offered.

Step 1. Creating table d_{KMP} according to the principles of Knuth-Morris-Pratt string searching algorithm.

Step 2. Creating table d_{BM} according to the principles of Boyer-Moore string searching algorithm.

Step 3. Defining the initial value of index i, corresponding to the position of the pattern relative to the string.

Step 4. Defining the initial values of indices j_{KMP} and j_{BM}, which indicate the beginning and the end of the pattern respectively.

Step 5. Comparing the pattern character with index j_{KMP} and the corresponding string character, comparing the pattern character j_{BM} and the corresponding string character. If at least one comparison ends with a mismatch, go to Step 11.

Step 6. If j_{KMP} is less than j_{BM}, go to Step 9.

Step 7. Output of the message informing that the pattern matches with a part of the string.

Step 8. Go to Step 15.

Step 9. Increasing index j_{KMP} by one (i.e. going to the next character to the right), decreasing index j_{BM} by one (i.e. going to the next character to the left).

Step 10. Go to Step 5.

Step 11. Choosing the larger shift from $j_{KMP} - d_{KMP}[j_{KMP}]$ and $d_{BM}[i + \text{pattern length} - j_{BM}]$.

Step 12. Shifting the pattern to the right relative to the string, increasing the value of index i by the shift defined in Step 11.

Step 13. If the sum of index i and the length of the pattern is less than the length of the string, go to Step 4.

Step 14. Output of the message informing that the target pattern is not found.

Step 15. Stop.

The combined algorithm allows finding matches between the pattern and a part of the string with a smaller number of shifts, due to the fact that in Step 11 a larger shift is chosen out of two shifts, which can be received with Knuth-Morris-Pratt and Boyer-Moore string

searching algorithms. At the same time, these algorithms themselves guarantee that however large the shift is, no match of the pattern and the string will be overlooked.

3. Results and discussion

The combined string searching algorithm based upon Boyer-Moore and Knuth-Morris-Pratt string searching algorithms was implemented within computer programme InfoSearch. This programme allows carrying out the analysis of work of Boyer-Moore, Knuth-Morris-Pratt algorithms and the combined algorithm. The interface of the programme is shown in Figure 10.

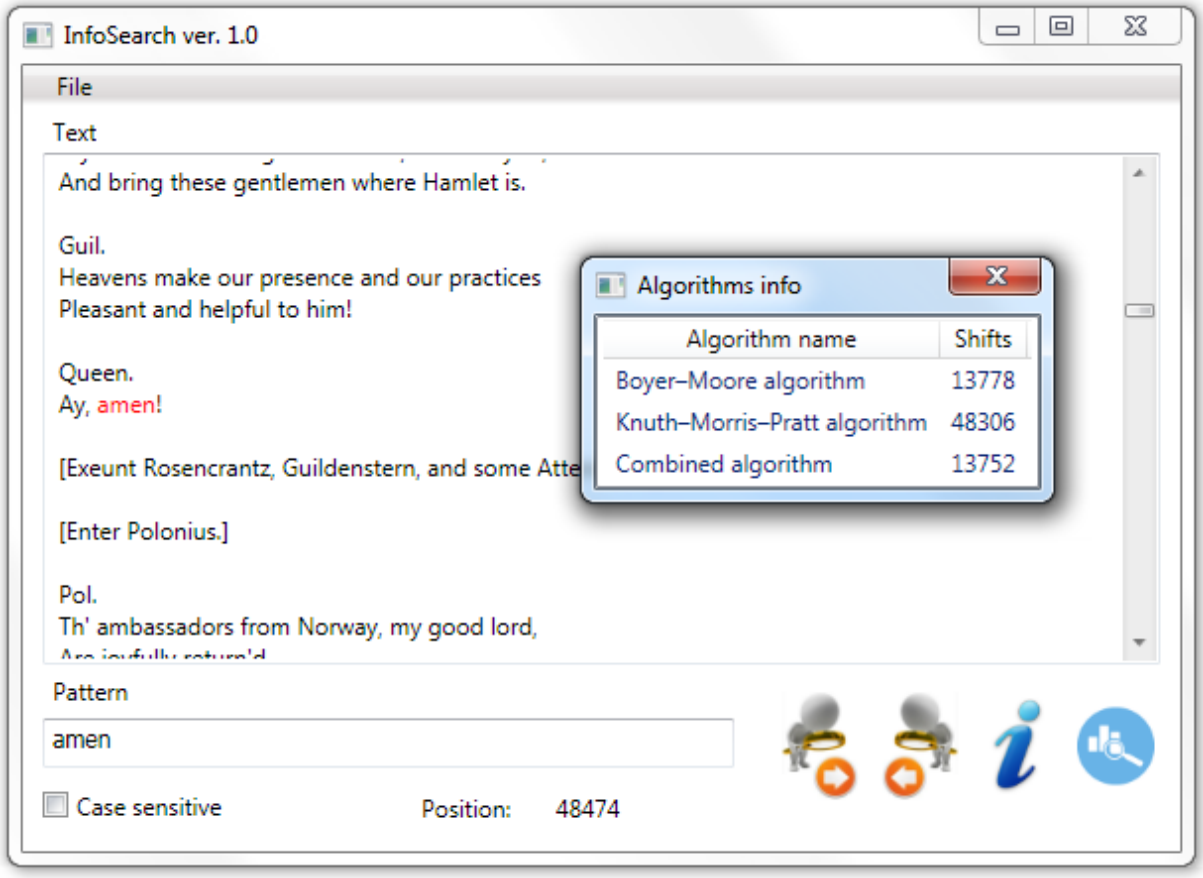

Figure 10. Interface of the programme implementing the combined string searching algorithm.

The analysis of the considered string searching algorithms work results can be shown in the example of The Tragedy of Hamlet, Prince of Denmark written by William Shakespeare. Table 1 shows the results of the algorithms when searching for the words written in the first column.

Table 1. Results of string searching algorithms

Word	Position	Shifts		
		Knuth-Morris-Pratt algorithm	Boyer-Moore algorithm	Combined algorithm
amen	48474	48306	13778	13752
antique (1st occurrence)	68539	67756	11676	11667
antique (2nd occurrence)	179186	177135	30677	30646
cozenage	166058	165546	24932	24921
habit (1st occurrence)	24743	24519	5738	5729
habit (2nd occurrence)	29215	28949	6762	6751
habit (3rd occurrence)	113830	112669	26005	25973
habit (4th occurrence)	171586	169831	39245	39204
herb	136155	133534	37481	37423
marble	30208	30025	5985	5948
marvel	18318	18193	3624	3592
matron	111378	110925	22596	22588
theme (1st occurrence)	14063	13587	3171	3161
theme (2nd occurrence)	161372	156067	36456	36390
thieves	138812	135466	23527	23480
sea-fight	165436	164562	23227	23177
stone (1st occurrence)	129310	128472	31382	31341
stone (2nd occurrence)	140499	139599	34100	34057

The analysis that was carried out reveals that the combined algorithm needs by several times less shifts than Knuth-Morris-Pratt algorithm to find an entry of the pattern into the string.

The combined algorithm also shows better results in terms of required number of shifts in comparison with Boyer-Moore algorithm, although this time the advantage is not as vivid as the advantage of the combined algorithm over Knuth-Morris-Pratt algorithm.

4. Conclusion

This article presents a solution of a task, which is extremely important for computer analysis – namely, string searching. The offered solution develops theoretical basis of methods for language data computer analysis and serves the purposes of solving practical computer linguistics tasks.

The developed combined string searching algorithm has united the advantages of well-known Knuth-Morris-Pratt and Boyer-Moore string searching algorithms. The distinction of combined algorithm lies in higher effectiveness in comparison with the initial algorithms and larger shifts in case of mismatch of string and pattern characters, which increases the speed of pattern entry in the string. The combined string searching algorithm can be successfully implemented for searching in English and Russian texts.

5. Acknowledgements

The authors would like to thank Ahmad Hassanat from Mu'tah University, Jordan, for consultations while studying string searching algorithms.

References

[1] Ahmed M, Kaykobad M and Chowdhury R A 2003 A new string matching algorithm *Int. J.*

Comput. Math. **80** 825–834
[2] Baeza-Yates R A and Gonnet G H 1992 A new approach to text searching *Commun. ACM* **35** 74–82
[3] Boyer R S and Moore J S 1977 A fast string searching algorithm *Commun. ACM* **20** 762–772
[4] Faro S and Lecroq T 2013 The exact online string matching problem: A review of the most recent results *ACM Comput. Surv.* **45**
[5] Fredriksson K and Grabowski S 2005 Practical and optimal string matching *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **3772 LNCS** 376-387
[6] He L, Fang B and Sui J 2005 The wide window string matching algorithm *Theor. Comput. Sci.* **332** 391-404
[7] Hudaib A, Al-Khalid R, Suleiman D, Itriq M and Al-Anani A 2008 A fast pattern matching algorithm with two sliding windows (TSW) *J. Comput. Sci.* **4** 393-401
[8] Knuth D, Morris J H, Pratt V 1977 Fast pattern matching in strings *SIAM J. Comput.* **6** 323–350
[9] Lecroq T 2007 Fast exact string matching algorithms *IPL* **102** 229-235
[10] Wirth N 1985 *Algorithms and data structures* (NJ: Prentice Hall)