The Scientific Journal of Cihan University – Sulaimanyia **PP: 97-107**
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

Comparative Study for String Matching Algorithms

Ammar Waysi AlTuhafi
Computer Science Department- Cihan university- Sulaimaniya, Iraq
dr.ammar.altuhafi@ieee.org
ammar.waysi@sulicihan.edu.krd

ABSTRACT

String matching became important application nowadays, the increasing of database such as websites, document, DNA, etc., leads to the urgent needs for string matching; string matching has many applications such as DNA, protein matching, internet search engine, all these types of application of string matching, beside the huge amount of database lead to increase the need to fast and efficient string matching algorithms. This study is about comparing among most well-known string matching algorithms; it focuses on four types of string matching algorithms, each one of them working in a different way. The four are tested with four types of data; ASCII (256 character), English alphabet (26 characters), DNA (4 character), and protein (20 character), with different pattern length (100, 50, 20, 10, 5) results shown based on number of comparisons and time.

Keywords: string matching, DNA, Protein, Boyer-Moore, Knuth-Morris-Pratt, Rabin-Karp.

الملخص

مطابقة السلاسل اصبحت من التطبيقات المهمة هذه الايام, زيادة قواعد البيانات مثل مواقع صفحات الانترنيت والوثائق والحمض النووي .. الخ, كل هذه العوامل ادت الى الحاجة الملحة الى تطبيقات تطابق السلاسل, هناك العديد من التطبيقات المستخدمة في هذا المجال مثل مطابقة الحمض النووي ومطابقة البروتين ومحركات البحث في الانترنيت، كل هذه التطبيقات تتعامل مع كمية هائلة من البيانات مما يؤدي الى الحاجة الى خوازيمات سريعة وكفوءة لمطابقة السلاسل. هذه الدراسة مبنية على المقارنة بين اربعة من اشهر الخوارزيمات في مطابقة السلاسل، كل سلسلة من هذه السلاسل تعمل وفق طريقة مختلفة، هذه الخوارزميات الاربعة تم فحص كل واحدة منهم على اربع انواع من البيانات وهي: احرف اللغة الانكليزية (٢٦ حرف) و احرف ال ASCII (٢٥٦ حرف) والحمض النووي (٤ حرف) واخيراً البروتين (٢٠ احرف)، مع استخدام اطوال مختلفة للسلسة (٥, ١٠, ٢٠, ٥٠, ١٠٠) النتائج كانت معتمدة على عدد المقارانات والوقت المستغرق في عملية المقارنة.

پوختە

هاوتاکردنی زنجیرەکان بووە بە یەکێك لە بەرنامە گرنگەکانی رۆژگار , زیادکردنی داتابەیس وەك ماڵپەری ئەلیکترۆنی و بەڵگەکان و ترشی ناوەکی ... هتد , ئەم هەموو فاکتەرانە وای خواستووە بۆ بەرنامەیەکی هاوتاکردنی زنجیرەکان , زۆر بەرنامە هەیە کە لەم بوارەدا بەکاردێت وەك هاوتاکردنی ترشی ناوەکی و هاوتاکردنی پرۆتین و بزوێنەری گەران لەسەر ئینتەرنێت , سەرجەمی ئەم بەرنامانە مامەڵە دەکات لەگەڵ بریکی زۆر لە زانیاری کە وا دەخوازێت ئەلگۆریتمی خێرا و بەتوانا هەبێت بۆ هاوتاکردنی زنجیرەکان. ئەم تویژینەوەیە بنیات نراوە لەسەر بەراوردکردنی چوار لە هەرە بەناوبانگترین ئەلگۆریتم لەبواری هاوتا کردنی زانیاری , هەر یەك لەو زنجیرانە کاردەکات بە رێگای جیاواز وە هەریەك لەم ئەلگۆریتمانە پشکنین بۆ کراوە لەسەر چوار جۆری داتای جیاواز , ئەوانیش پیتەکانی زمانی ئینگلیزی (٢٦ پیت) و زمانی ASCII

PP: 97-107 *The Scientific Journal of Cihan University – Sulaimanyia*
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

(٢٥٦ پیت) و پیتی ترشی ناوەکی (٤ پیت) وە پرۆتین (٢٠ پیت) , لەگەل بەکارهینانیز نجیرەی جیاواز لە دریژیدا (٥, ١٠, ٢٠, ٥٠, ١٠٠) ئە نجامەکان بنیات نراوە لەسەر کۆمەڵێك بەراوردکردن و کاتی خایەنراو بۆ ئەو پرۆسەیە.

1. INTRODUCTION

String matching is a method of giving two strings – a text and a pattern. String matching used for search if the pattern is in the text. It is also a part of any text processing editor. It is known as "the needle in a haystack problem." Generally, there are two types of String Matching: Exact String Matching and Inexact String Matching; this study focuses on Exact String Matching (ESM) algorithms. In string matching there are two strings; text and pattern, usually the pattern is shorter string than the text string, and the job of string matching algorithms is to find the pattern in the text with shortest time as much as possible [1] .

Within last years, there are many algorithms in String Matching that has been appeared for a different type of data; and they are; ASCII, English alphabet, DNA, Protein. ASCII code as known consist of 256 characters, English language has 26 characters, DNA consist of 20 characters, and Protein has four characters. The study is based on making performance comparison among string matching algorithms and determines their performance with different size and type of data [2].

There are different algorithms that are used in the string matching area such as: Naive, Knuth-Morris-Pratt (KMP), Boyer-Morro (BM) and Rabin-Karp (RK); these four algorithms considered the most well-known algorithms in string matching, each one of them work based on different techniques [3]. Therefore, evaluate the performance of these four algorithms can provide a significant estimation of the performance of other string matching algorithms by using similar techniques.

2. THE ALGORITHMS

The four, tested, algorithms in this research are; Naive, Knuth-Morris-Pratt (KMP), Boyer-Morro (BM) and Rabin-Karp (RK), the working mechanism of each one will be discussed as following:

2.1. Naïve Algorithm

The naive algorithm or sometimes called brute force algorithms, makes a comparison character by character of the text T[s...s+m-1] for all s E {0,...,n-m+1} and the pattern p[0...m-1].
The valid shifts found is returned. The problem of this method is its effectiveness.
The naive approach simply tests all the possible placement of Pattern P[1 .. m] relative to text T[1 . . n]. Specifically, we try shift s = 0, 1, . . n -m, successively and for each shift, s. Compare "T[s +1 . . s + m]" , "P[1 . . m]"

NAIVE_STRING_MATCHER (T, P)
n ← length [T]
m ← length [P]
For s ← 0 to n - m do
If P[1 . . m] = T[s +1 . . s + m]
Then return valid shift s
[4]

DOI: http://dx.doi.org/10.25098/2.2.33

The Scientific Journal of Cihan University – Sulaimanyia **PP: 97-107**
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

The following is the pseudo code for Naïve string matching algorithm:

```
n ← length [T]
m ← length [P]
for s ← 0 to n-m do
j ← 1
while j ≤ m and T[s + j] = P[j] do
j ← j +1
If j > m then
return valid shift s
return no valid shift exist
```

2.2. Knuth-Morris-Pratt Algorithm (KMP)

The KMP algorithm has complexity O(N+M), as a linear algorithm. KMP algorithm is contained from two phases, a searching phase, which is used to find the valid shifts in the text, in this phase the complexity is O(N), in this phase the comparison between pattern and text usually happened beside the shifting operation after each comparison process, before searching phase there is a preprocessing phase, which exists to preprocess the pattern, and also to avoid checking characters in T that we already know and match a prefix of P, based on a strategy of using information from partial matching of P. The preprocessing part does not only skip over portions of the text that cannot contain a match, it also avoids re-checking characters in T that are known [5, 6].
The following is the pseudo code for Naïve string matching algorithm:

```
KNUTH-MORRIS-PRATT FAILURE (P)
Input:    Pattern with m characters
Output: Failure function  f  for P[i . . j]
i ← 1
j ← 0
f(0) ← 0
while i < m do
   if P[j] = P[i]
      f(i) ← j +1
       i ← i +1
        j← j + 1
else if
       j ← f(j - 1)
else
      f(i) ← 0
      i ← i +1

KNUTH-MORRIS-PRATT (T, P)
Input:    Strings T[0 . . n] and P[0 . . m]
Output: Starting index of substring of T matching P
```

```
f ← compute failure function of Pattern P
i ← 0
j ← 0
while i < length[T] do
   if j ← m-1 then
      return i- m+1    // we have a match
i ← i +1
j ← j +1
else if j > 0
      j ← f(j -1)
   else
      i ← i +1
```

2.3. Boyer-Moore Algorithm (BM)

The main idea of this algorithm is that the match is performed from right to left. This technique allows the algorithm to skip more characters than the other algorithms and find the matching faster, for example if the first matched character of a text is not contained in the pattern P[0..m-1] we can skip m characters immediately. In O (m) time, scan characters of P, marking which characters of alphabet appear in P. Boyer-Moore "bad character" rule: if a character of T aligned with last character of P isn't in P, then none of the m characters of T (starting with this one) can align with the last character of P in a substring match [7]. The following is the pseudo code for Naïve string matching algorithm:

```
Input:    Text with n characters and Pattern with m characters
Output: Index of the first substring of T matching P
Compute function last
i ← m-1
j ← m-1
Repeat
   If P[j] = T[i] then
      if j=0 then
         return i        // we have a match
      else
         i ← i -1
         j ← j -1
   else
      i ← i + m - Min(j, 1 + last[T[i]])
      j ← m -1
 until i > n -1
Return "no match"
```

2.4. Rabin Karp (RK)

The Rabin-Karp algorithm uses a totally different technique based on mathematical equations to find pattern in the text. This method is based on hashing techniques. The Rabin Karp algorithm works by

DOI: http://dx.doi.org/10.25098/2.2.33

The Scientific Journal of Cihan University – Sulaimanyia **PP: 97-107**
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

"fingerprinting" the search string (of length K) and every substring of length K within the haystack, and compares fingerprints. If it finds a match, it reports success (with high probability). Fingerprinting is a process in which a single integer is made to represent an entire string of length K.
The probability of error depends on the choice of the fingerprinting function. After computing hash function h(x) for the pattern P[0...m-1]and the search operation done by using the same hash function for every substring of length m-1 of the text. The RK as KMP and BM algorithms also uses preprocessing technique before the search operation. Its preprocessing operation is the hashing of the pattern, which is O (M) complexity. So, the running time of the algorithm is O (M x (N-M+1)). However, the algorithms will run with a complexity O(N) [8].
The following is the pseudo code for Rabin-Karp string matching algorithm:

Compute ht (for the first substring of t with m length)
For i = 1 to n – m
If hp = ht
Match t[i . . . .i + m] with p, if matched return 1
Else ht = (d ht – t[i + 1] .dm–1 + t[m + i + 1]) mod q
End

3. TESTING ENVIRONMENT

The testing environment involves five types: DNA, Protein, ASCII, Binary and English alphabet, with different pattern length. All algorithms are performing their search with similar files (as their text) and similar patterns; the file size is 100 Mega Bytes for each data type. Applying each algorithm to each data type to find which data type is more useful for which algorithm. The results based on the speed and the number of comparisons. Figure 1 shows the testing environment.

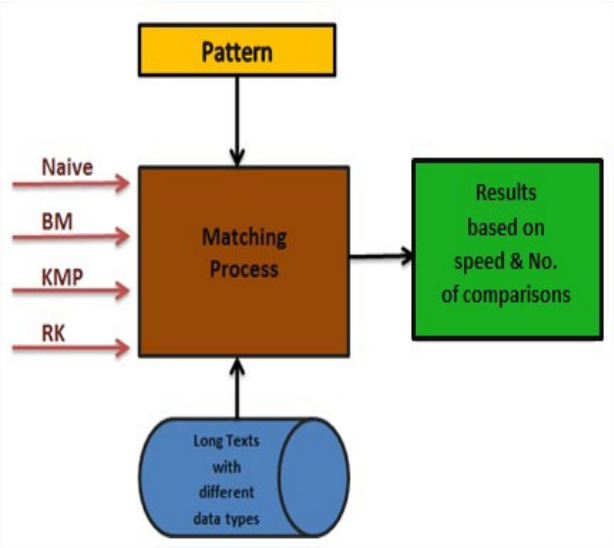

Figure 1: Testing environment

PP: 97-107 *The Scientific Journal of Cihan University – Sulaimanyia*
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

4. RESULTS

The result for each type of data with each algorithms using different pattern length are shown below.

4.1. Protein

Table 1 shows the result of no. of comparisons and the time in milliseconds for the four algorithms for 100, 50, 20 and 5 characters' pattern length with DNA characters. Naïve algorithm has the highest value of number of comparisons and time because the shifting amount, which is done by one shifting every after every mismatch, Naïve algorithm, in the other hand, KMP algorithm has better performance in term of number of comparisons, because the good prefix technique in KMP algorithm, which can cause shifting more than one after mismatch, the BM algorithm has second better result in term of number of comparisons, BM algorithm has two techniques in shifting, bad character and good suffix, the combination between these two techniques cause better result comparing to KMP algorithm, finally RK has best performance in term number of comparisons because the depending on hash function calculation which has the ability to reduce the number of comparisons based in mathematical model.

In term of time, BM has better performance because the high shifting based on two techniques, instead one technique in KMP, the Naïve algorithm needs longer time comparing to BM and KMP algorithm because the simple shifting after each mismatching consume more time, while the worst algorithm in term of time is RK, because the long mathematical calculation and produce hash value for the text needs to long time comparing to simple matching method in the other three algorithms.

Table 1: The results for the four algorithms with DNA characters

No. of characters	Result Type	Naive	BM	KMP	RK
100	No. of comparisons	105266449	61747464	94996099	1041958
	Time in milliseconds	9531664	3375181	8437954	18438420
50	No. of comparisons	105266452	64999934	99999978	990845
	Time in milliseconds	8437894	3375157	8281662	18594631

DOI: http://dx.doi.org/10.25098/2.2.33

The Scientific Journal of Cihan University – Sulaimanyia **PP: 97-107**
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

20	No. of comparisons	105266452	61747464	94996099	1043745
	Time in milliseconds	9531710	4875184	8437945	18438365
10	No. of comparisons	105266452	5987732279	94996099	1042262
	Time in milliseconds	9687960	3875184	8437911	18594644
5	No. of comparisons	105265784	59875766	94996099	1042303
	Time in milliseconds	9531684	3812673	8437924	18594600

4.2. English Letters

Table 2 shows the result of no. of comparisons and the time in milliseconds for the four algorithms for 100, 50, 20 and 5 characters' pattern length with English characters. The result of English letters is close to protein letters because the close number of characters between them, in term of number of comparisons, still RK algorithm has the best result, then BM algorithm has the second best algorithm, followed by KMP algorithm, while Naïve algorithm has the worse result in term of number comparisons, although there are some slightly changes comparing to DNA results.

In term of time still BM algorithm has better performance, and KMP algorithm has second better performance, while naïve followed, RK algorithm has the worst results.

Table 2: The results for the four algorithms with English characters

No. of characters	**Result Type**	**Naive**	**BM**	**KMP**	**RK**
100	No. of comparisons	103997688	62501497	96156150	1030140
	Time in milliseconds	9531698	3812679	8281650	18594648
50	No. of comparisons	103997690	58801497	96156150	1028929
	Time in milliseconds	9219208	3687683	8125400	18438339
20	No. of comparisons	103997690	58701052	96156150	1029898

	Time in milliseconds	9062959	3625183	8281662	18438411
10	No. of comparisons	103997692	58900083	96156150	1030334
	Time in milliseconds	9219139	3687655	8125353	18438390
5	No. of comparisons	103997478	59874351	96156150	1030144
	Time in milliseconds	9219182	3687672	8281658	18594673

4.3. DNA

Table 3 shows the result of no. of comparisons and the time in milliseconds for the four algorithms for 100, 50, 20 and 5 characters' pattern length with Protein characters. In term number of comparisons, there are big enhancements, the small number of protein characters (four) produce high probability to have long good prefix for KMP algorithm and good suffix for BM, which help these two algorithms to do long skip of character matching after each mismatching, for this reason the number of comparison for these two algorithm have big enhancements, comparing to protein and English characters. Although the big enhancements of the number of comparison, the RK algorithm has second best performance, while KMP has the third best performance, these two algorithms come after BM algorithm which has the better performance, finally naïve algorithm has the worst results in term number of comparisons.

In term of time still BM algorithm has the better performance, because the two techniques that used in BM algorithm, KMP algorithm has second better performance in term of time, naïve algorithm follow BM and KMP algorithms, while RK still worse performance because the mathematical operation that hash function needs to.

Table 3: The results for the four algorithms with protein characters

No. of characters	Result Type	Naive	BM	KMP	RK
100	No. of comparisons	133337152	141	74998712	1320212
	Time in milliseconds	14137355	4812710	12031777	18594669
50	No. of comparisons	133337165	143	74998712	1319680

DOI: http://dx.doi.org/10.25098/2.2.33

The Scientific Journal of Cihan University – Sulaimanyia **PP: 97-107**
Volume (2), Issue (2), December 2018
ISSN 2520-7377 (Online), ISSN 2520-5102 (Print)

	Time in milliseconds	14219439	4812728	12031820	18594665
20	No. of comparisons	133337174	263	74998712	1319092
	Time in milliseconds	14219477	4750234	11875587	18438399
10	No. of comparisons	133336601	131	74998712	1318187
	Time in milliseconds	14375637	4875225	12188064	18594669
5	No. of comparisons	132814224	48749162	74998712	1662286
	Time in milliseconds	14063172	4824363	12060908	18438394

4.4. ASCII

Table 4 shows the result of no. of comparisons and the time in milliseconds for the four algorithms for 100, 50, 20 and 5 characters' pattern length with ASCII characters. In term of number of comparisons RK algorithm has better performance, and BM algorithm has second better performance in term of number of comparisons, KMP algorithm has the third place, while naïve algorithm has worse result. The big difference between protein and ASCII results belong to big difference of number of characters, which cause low probability to produce good prefix in KMP algorithm, and good suffix in BM algorithm. Although the good suffix of BM algorithm has bad result with big number of characters such as ASCII (256), the bad character technique can cause good result of skipping.

In term of time, BM algorithm has best performance, followed by KMP and Naïve algorithms, the worse performance in term of time belongs to RK same other types of data.

Table 4: The results for the four algorithms with ASCII characters

No. of characters	Result Type	Naive	BM	KMP	RK
100	No. of comparisons	100392433	48889315	99609073	993313
	Time in milliseconds	8750427	8437907	7500357	22813608
50	No. of comparisons	100392433	48957425	99609073	994417

	Time in milliseconds	8594173	3500170	7500375	22501089
20	No. of comparisons	100392433	50171259	99609073	993321
	Time in milliseconds	8750388	3500155	7373209	18628709
10	No. of comparisons	100392433	52271078	99609073	993331
	Time in milliseconds	8594173	3437669	7344078	18594643
5	No. of comparisons	100392433	54752477	99609073	993632
	Time in milliseconds	1.0625535	4250214	7344040	18750884

5. CONCLUSION

The string matching applications is an integral part in computer science. The recent applications became much more than text editors, especially in bioinformatics applications, beside search engines. After testing the most well-known algorithms in string matching with different types of data and different length of patterns, the result shows the following: the best performance in terms of time is Boyer-Moore algorithm (with different data types with different pattern length) while Rabin-Karp algorithm shows best performance in terms of the number of comparisons except with protein BM has better permanence than RK algorithm.

6. RECOMMENDATIONS

According to the result of the research, the recommendations are the following: for application when time is the most important factor in string matching, using Boyer-Moore string matching algorithm is highly recommended. For application which tries to reduce the number of comparisons to the minimum, using Rabin-Karp algorithm for string matching is highly recommended. Whereas, for small applications such as text editors where users deal with small amount of text, it is recommended to use Naive algorithm for string matching, because the implantation is straightforward for this algorithm as well as the non-existence of pre-processing operation in order to increase the searching time when having large texts.

DOI: http://dx.doi.org/10.25098/2.2.33

REFERENCES

[1] D. Gussfield, "Algorithms on strings, trees, and sequences," *Computer Science and Computional Biology (Cambrigde, 1999),* 1997.

[2] G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR),* vol. 33, pp. 31-88, 2001.

[3] C. C.-T. Lecroq. (1997, 22/5). *EXACT STRING MATCHING ALGORITHMS*. Available: http://www-igm.univ-mlv.fr/~lecroq/string/

[4] Oracle. (2016). *Comparison: Exact String Match.* Available: http://www.oracle.com/webfolder/technetwork/data-quality/edqhelp/Content/processor_library/matching/comparisons/exact_string_match.htm

[5] D. E. Knuth*, et al.*, "Fast pattern matching in strings," *SIAM journal on computing,* vol. 6, pp. 323-350, 1977.

[6] K. A. Berman and J. L. Paul, *Algorithms: sequential, parallel, and distributed*: Course Technology Ptr, 2005.

[7] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM,* vol. 20, pp. 762-772, 1977.

[8] C. Charras and T. Lecroq, *Handbook of exact string matching algorithms*: King's College, 2004.