iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 1 **Issue:** III **Month of publication:** October 2013

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Suffix to Prefix Rule and Substring Matching Rules of Sting Matching Algorithms: An Analytical study and Correlations

Jamuna Bhandari[1], Anil Kumar[2]

[1]*Student Member IEEE*
[1] *Research Scholar, Dept. of CSE, Manipal University Jaipur, INDIA*
jamuna.bhandari@muj.manipal.edu bunu17_bhandari@yahoo.com

[2]*Senior Member IEEE*
[2]*Professor, Deptr of CSE, Manipal University Jaipur, INDIA*
anil.kumar@jaipur.manipal.edu dahiyaanil@yahoo.com

Abstract – *String matching is a profound problem in various fields and becomes a great challenge for running a program for pattern matching quickly and effectively with less or no complexities. The application of string matching or pattern matching is widely increased as new technologies are used in medical science, network security, forensic science, library sciences and many others for information detection and retrieval. Algorithms for string matching were designed since 1974 and till date lots of algorithms have been proposed. This analytical study presents the mechanism also called rules follows for pattern matching process. This includes analysis and correlations of suffix to prefix rule and substring matching rules with different cases, supported by appropriate figures and examples to justify the relations among them.*

Keywords: *String Matching, suffix to prefix, substring matching, 2-substring, 1-suffix, char-matching.*

1. INTRODUCTION

A process of finding some given string or pattern within the given text is known as string matching or sometimes called pattern matching. Sting matching is done in two ways via exact string matching and via approximate string matching. Exact matching[1],[2],[14] is a process of finding occurrence of pattern *P* exactly matched within a given text *T* whereas an approximate matching[2],[15] is a process of finding occurrence of patterns *P* approximately with some differences of mismatch characters within the text *T*.

String matching rules are the processes which are used to compare the characters of pattern *P* with given text *T* in order to find where the pattern *P* occurs exactly within the text *T*. Searching should be done with minimum comparing steps. Every algorithm follows some mechanisms to reduce the required time complexities used in preprocessing phase and in searching phase[1].

This paper analyzed the suffix to prefix rules and substring matching rules of exact string matching algorithms[16] The paper is organized as follows, section 2 includes Rule 1 (suffix to prefix) and Rule 2 (substring matching) with its sub-rules discussed in details, followed by the correlations among them in section 3. Finally concludes in section 4.

2. Analysis of suffix to prefix rule and substring matching rules.

This section analyzed the working of suffix to prefix rule known as Rule-1 and substring matching rules as Rule-2.

2.1. *Suffix to prefix rule:* Rule-1 suffix to prefix rule work as, if suffix of text *T* is found to be matched with the prefix of pattern *P* then shifting of pattern is done to align matching portion correspondingly as shown in figure 1.

Length of suffix can be *2 to m''* [where *m'' (m'-i)* is the length of text window *T* and m is length of pattern *P (m=m'')*]and length of prefix can be *1 to m-1*

Algorithm 1: Suffix to Prefix

If (m- j) == (m-k)

Then shift *(m – k)* characters.

Else shift *m* characters.

Where, *i, j and k* are the pointer used to point at positions of pattern and text.

Example 1:

```
        1 2 3 4 5 6
T: GCATCGACAGACTATACAGTACG
P:      GACGGATCA
        1 2 3 4 5 6 7 8 9
```

Here *m = 9, k = 3*

Skip *(m - k) = 9 – 3 = 6.* (by 6 characters)

```
T: GCATCGACAGACTATACAGTACG. . . . . .
P:          GACGGATCA
            1 2 3 4 5 6 7 8 9
```

Again m = 9, k = 0 (Zero because no in suffix and prefix)

So, m – k = 9 – 0 = 9 (skip by 9).

```
T: GCATCGACAGACTATACAGTACG. . . . . .
P:                    GACGGATCA
                      1 2 3 4 5 6 7 8 9
```

2.2. Substring matching rule

Substring matching rule is further divided into four sub-rules

- Substring matching rule as rule-2
- Character matching rule as rule-2.1
- 1-suffix rule as rule-2.2
- 2-substring matching rule as rule-2.3

This section analyzed all the sub rules of substring matching rules.

2.2.1. Substring matching rule (Rule-2)

Consider any substring say either *α, u and z* of text *T* as shown in figure 2(a). Find the substring that to be matched in pattern *P*.

According to figure 2(a) matched portion *v* is considered as longest substring. Substrings *α, u and z* are the substrings within *v*. Substring *u* is found to be matched in pattern *P*. Shift the pattern in such a way that both the *u* aligns corresponding to each other as shown in figure 2(b); otherwise, skip pattern by pattern length *(j'-i).*

Length(*α, z*) may be *0 to m-1* and

Length*(u)* may be *1 to m*.

Algorithm 2: Substring matching

If [substring *T(j', k')* = = Substring *P(j, k)*]

Then shift pattern by *(j' – j).*

Else shift pattern *(j'-i).*

Where *j and k* are the reoccurrence position at pattern *P* and *j' and k'* are the occurrence position of text *T*.

Example 2:

```
              1 2 3 4 5 6 7 8 9
T:    GCATCGAGGAGAGTATACAGTACG
P:          GGAGCCGAG
```

Here, mismatch occurs between char A of text *T* and C of pattern *P*. Substring GAG at T(7, 9) position and reoccurs at pattern P(2, 4).

So now slide the window by (j'-j)=(7-2 = 5).

```
T: GCATCGAGGAGAGTATACAGTACG
         1 2 3 4 5
P:       GGAGCCGAG
```

2.2.2. Character matching rule (Rule-2.1)

Character matching rule, search for reoccurrence of mismatched character of text *T* also known as bad character in pattern *P*.

Suppose, char *x* of text *T* mismatch with corresponding character of pattern *P* at position *j* and reoccurs *x* in pattern *P* at position *k* as shown in figure 3(a).

Algorithm 3: Char matching rule.

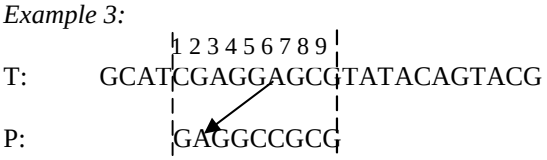
If bad char *x* occur in pattern P.

Then shift pattern by $(j-k)^{th}$ characters

Else Shift by *j*

Figure3(b) shows bad char *x* found in pattern *P*, so shift by position to align corresponding bad char together. Figure 3(c) shows, no bad character found in pattern *P*.

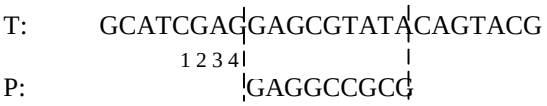
Example 3:



Here, A mismatch with C at position 6. A is considered as bad character. Now look whether char A occurs anywhere in pattern *P* or not.

If it found then shift the pattern by *(j-k)* position. According to example 3 A occurs at position 2 in pattern *P*.

So, (j-k) = (6 – 2 = 4) shift by 4.



2.2.3. 1- Suffix rule (Rule-2.2)

1-Suffix rule always consider the 1st character of suffix of text *T*. As shown in figure 4, Let *x* be the 1st char of suffix of text *T* at position *m'*. Now search reoccurrence of *x* in pattern *P*.

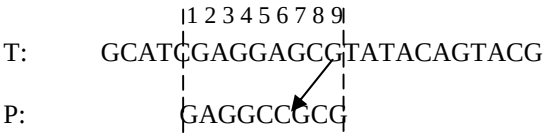
Algorithm 4: 1-suffix algorithm

If *T(char m')* found in pattern *P*

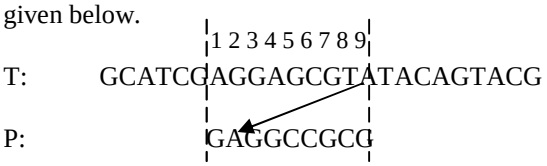
Then shift pattern by *(m-k)*

Else shift pattern by *m*

Example 4:



Shift by *(m-k) = (9 – 7) = 2* position, now shift will be as given below.



Shift by *(m-k) = (9 – 2) = 7* so on.

2.2.4. 2- Substring matching rule (Rule-2.3)

This rule works with two consecutive characters. One char should be bad char and another should be suffix of bad char as shown in figure 5(a).

Here, *u and x* are two consecutive characters with *x* as bad char of text *T*.

Algorithm 5: 2-substring matching rule

If *(k ==i && k+1 == i+1)*

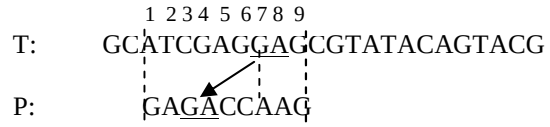
Then Shift by *(j-i)*

Else shift by *(j+1)*

(Where j is the mismatch position at pattern, k is the position of bad character, i is the position of bad character occurrence in pattern).

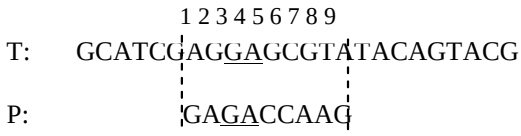
Figure 5(b) shows shifting after match found of 2-substing in pattern P

Example 5:



Here mismatch occurs between G and A of text and pattern. GA of text is now considered as 2-substring. Mismatch position *j* = 7 and reoccurrence position *i=3* in pattern *P (Also k=j && k+1=i+1).*

Now shift by *(j-i)=7-4=3* position



2. CO-RELATION AMONG SUFFIX TO PREFIX RULE AND SUBSTRING MATCHING RULES.

This section discussed the relations among rule-1 with rule-2 and its sub-types.

2.1. Rule-1 correlation with Rule-2

Rule-2, substring matching of pattern *P* is done by searching reoccurrence of substring of text *T* within the pattern *P* as shown in figure 6(a).

Suppose *v* is the matching portion of text *T* with pattern *P*.

α, u and z are the substrings of text *T*. (v= α| u| z).

Case 1: If *v* is found in prefix of pattern P then it becomes the special case of Rule-1 as shown in figure 6(b)

Case 2: If substring *u and z* are found in prefix of pattern *P* then it also becomes the special case of Rule-1 as shown in figure 6(c). Same is applicable for substring *z*.

2.2. Rule-1 correlation with Rule-2.1

Rule-2.1, char matching rule also called bad char rule looks for occurrence of only one char *(mismatched char of text T).* Rule-2.1, establishes the relation in following cases.

Case 1: If char mismatches at position *m* of pattern and bad char *a* reoccurs only at prefix of pattern *P* as shown in figure 7(a). Then it rule-2.1 becomes a special case of rule-1.

Case 2: If bad character of text *T* appears in prefix of pattern *P* followed by matched substring *u* as shown in figure 7(b) then this also becomes a case of Rule-1.

2.3. Rule-1 correlation with Rule-2.2

Rule-2.2, 1-suffix rule is look for 1st character (suffix) of text *T* within the pattern *P*. 1-suffix rules establishes the relation with rule-1 in following cases.

Case 1: If 1-suffix char reappears once only at prefix of pattern *P* as shown in figure 8(a).

Case 2: If 1-suffix character of text *T* found in prefix of pattern along with its prefix characters of Text *T* as shown in figure 8(b) then the shifting becomes the special case of Rule-1

2.4. Rule-1 correlation with Rule-2.3

Rule-2.3, 2-substring character is searching for substring within the pattern *P*. Rule-2.3 makes relation with Rule-1 in following cases.

Case 1: If the 2-substring character mismatches at position m-1 of pattern *P* and reappear only at prefix of pattern *P* then this will become a special case of Rule-1 as shown in figure 9(a).

Case 2: If 2-substring character of text *T* found in prefix of pattern along with its prefix characters of Text *T* as shown in figure 9(b) then the shifting becomes the special case of Rule-1

Following table shows the algorithms based on suffix to prefix rule and substring matching rules and its types.

3. CONCLUSIONS

This paper analyzed and correlates the suffix to prefix rule with substring matching rules and its sub-rules. The examples and figures mentioned in this paper compare the pattern and text from right to left. The comparisons may also be done from left to right and apply the same rules in same manner. Table 2 shows the different algorithms designed based on suffix to prefix rule and substring matching rules. KMP algorithm compares the pattern from left to right and BM compares from right to left but both of them also uses rule-1. So comparison done in any order applies any rules. It shows that they are interrelated to each others in terms of shifting the patterns not in scanning order.

4. Figures and Table

This section includes all the figures and table used to justify the analysis and correlations.

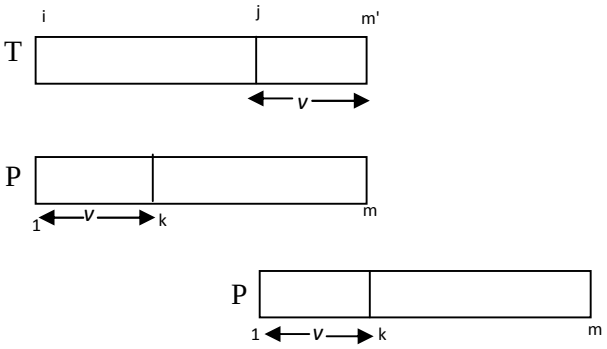


Fig. 1: Suffix to prefix rule

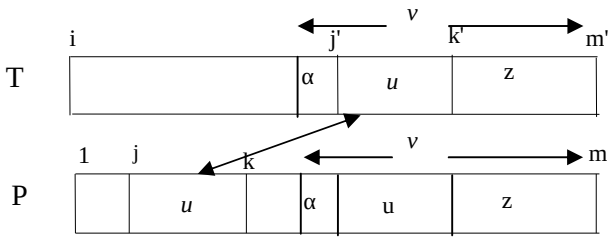


Figure 2(a): Substring *u* found in pattern

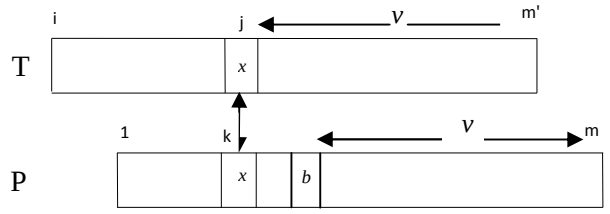


Figure 3(b): pattern shift after match is

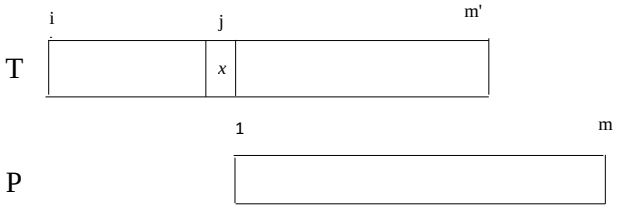


Figure 3(c): No bad char found in pattern

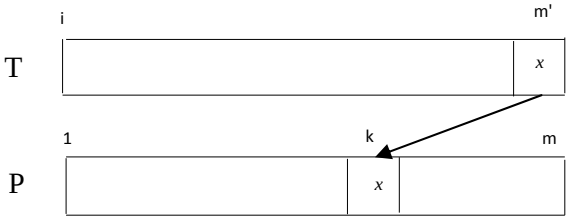


Figure 4: 1-suffix rule

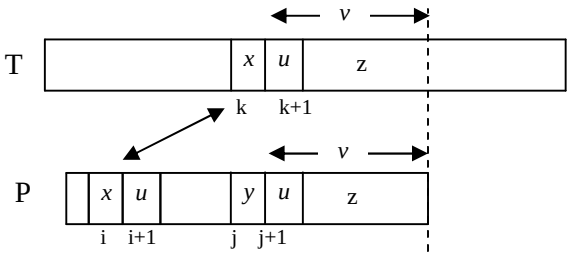


Figure 5(a): 2-sub string rule

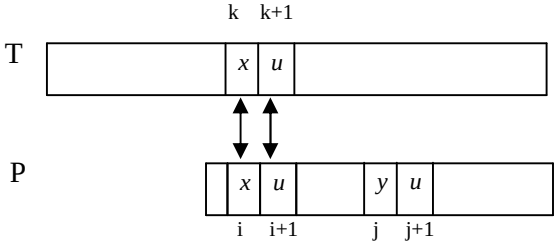


Figure 5(b): 2-substring matching shift

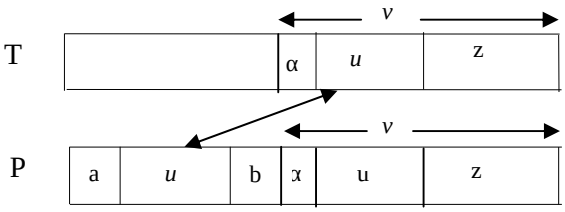


Figure 6(a): Substring matching rule

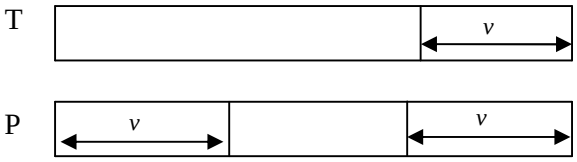


Figure 6(b): substring *v* shift as rule-1

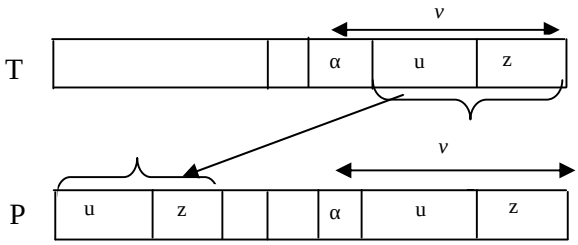


Figure 6(c): Substring u and z as rule-1

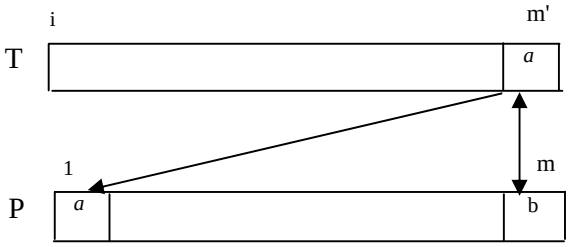


Figure 7(a): Bad Character *a* as rule-1

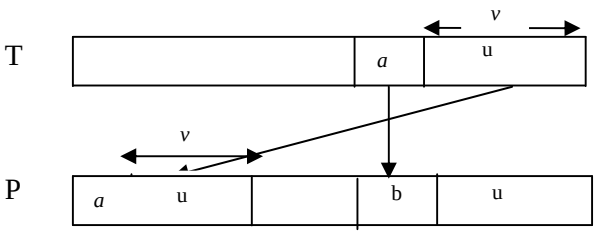


Figure 7(b): bad char *a* with *u* as rule-1 shift

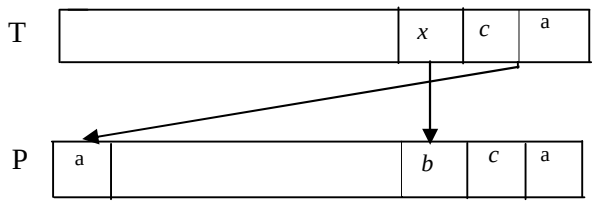


Figure 8(a): 1-suffix char at prefix as rule-1

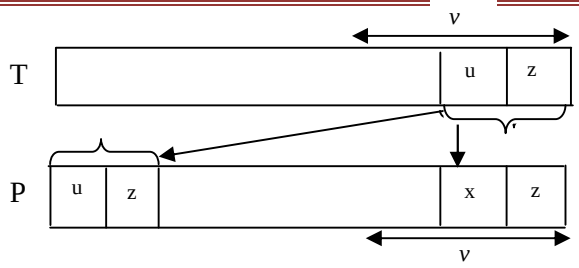


Figure 9(a): 2-substring as rule-1

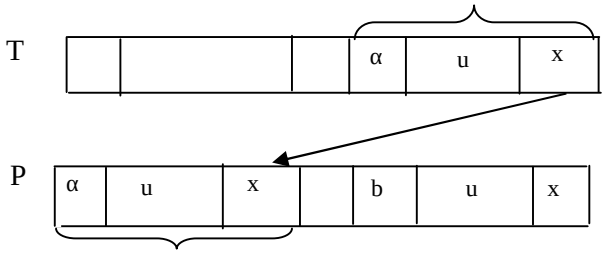


Figure 8(b): 1-suffix match with preceding char

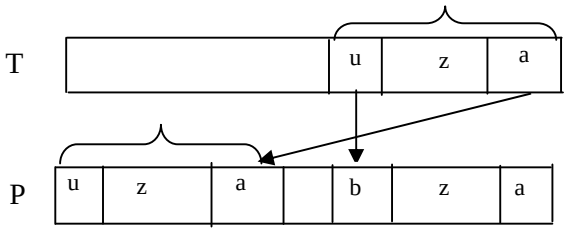


Figure 9(b): 2-substring with matched suffix

ALGORITHMS	RULE APPLIED
Fast pattern matching in string [2]	Rule-1: Suffix to prefix
on the exact complexity of string matching: upper bounds[11]	
String matching algorithms and automata[12]	
Fast String Algorithm [3]	Rule-1: Suffix to prefix Rule-2.1: Char Matching
Hybrid exact string matching algorithm for intrusion detection system[17]	
On improving the average case of BM algorithm [5]	Rule-2.3: 2-substring rule
A very fast search algorithm[6]	Rule-2.2: 1-suffix rule
Fast string searching[7]	
Practical fast searching in string[4]	
Tuning the BMH string searching algorithm[10]	
Experiments with a very fast substring search algorithm [8]	
Fastest pattern matching in string[9]	Rule-2.1: Char matching rule Rule-1: Suffix to prefix
A simple fast hybrid pattern- matching algorithm[18]	Rule-2.2: 1-suffix rule Rule-1: Suffix to prefix

Table 1: Suffix to prefix and substring based algorithm

Acknowledgement

This work is partially supported by HRD, Govt. Of Sikkim (India), vide notification no. 166/SCH/EDN 2003, Ref. No 82/SCH/EDN, issued on 20/7/2013

REFERENCES

[1] Knuth, D. E., Morris, JR, J. H., and Pratt, V. R. 1977. Fast pattern matching in strings. SIAM J. Comput. 6, 1, 323–350, 1977

[2] Patrick A. V. Hall. 1980. Approximate string matching, ACM

[3] Boyer, R. S. and Moore, J. S.1977, A fast string searching algorithm. Commun. ACM, 1977, 20, 762–772.

[4] Horspool, R. N. 1980. Practical fast searching in strings. Softw. Pract. Exp. 10, 6, 501–506

[5] Zhu, R. F. and Takaoka, T. 1987. On improving the average case of the Boyer-Moore string matching algorithm. J. Inform. Process. 10, 3, 173–177.

[6] Sunday, D. M. 1990. A very fast substring search algorithm. Commun. ACM 33, 8, 132–142.

[7] Hume, A. and Sunday, D. M. 1991. Fast string searching. Softw. Pract. Exp. 21, 11, 1221–1248.

[8] Smith P.D.1991. Experiments with a very fast substring search algorithm. Software - Practice & Experience 21(10) pp. 1065-1074.

[9] Colussi, L. 1994. Fastest pattern matching in strings. J. Algorithms 16, 2, 163–189

[10] Raita, T. 1992. Tuning the Boyer-Moore-Horspool string searching algorithm. Softw. Pract. Exp. 22, 10, 879–884

[11] Galil, Z. GIancarlo, R. 1992, on the exact complexity of string matching: upper bounds, SIAM journal on computing, 21(3): 407 - 437.

[12] Simon, I. 1993. String matching algorithms and automata. In Proceedings of the 1st South American Work-shop on String Processing, R. Baeza-Yates and N. Ziviani, Eds. Universidade Federal de Minas Gerais, Brazil, 151–157

[13] Berry, T. and Ravindran, S. 1999. A fast string matching algorithm and experimental results. In Pro-ceedings of the Prague Stringology Club ’99, J. Holub and M. Sim ´anek, Eds. Czech Technical

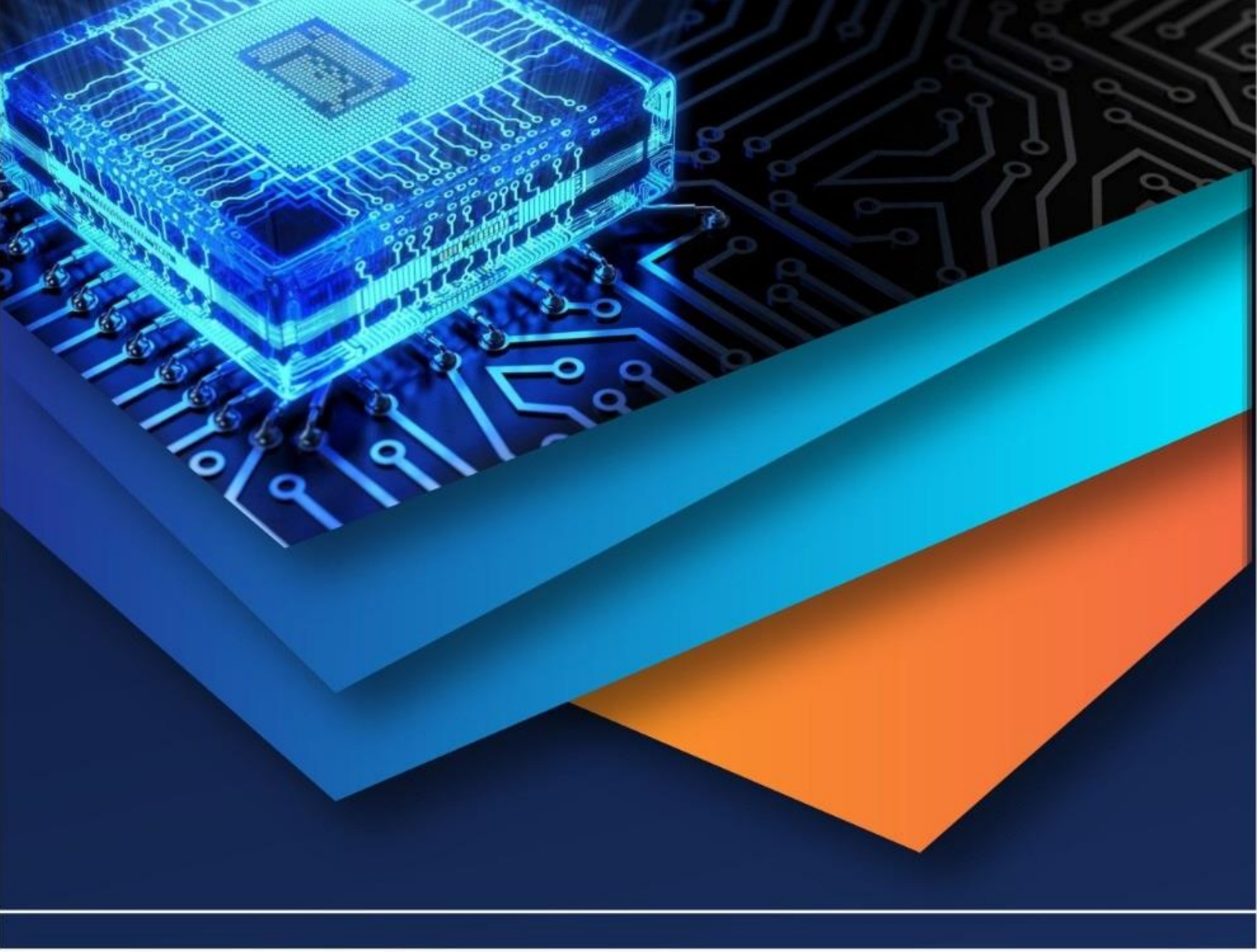
[14] C. Charras, T. Lecroq, "Handbook of Exact String Matching Algorithms", http://www.ezdoum. Com/upload/10/20020720023851/string.pdf, 2013.

[15] C. W. Lu, C. L. Lu, R.C.T. Lee, "A new filtration method and a hybrid strategy for approximate string matching", Theoretical Computer Science, Volume 481, 15 April 2013

[16]. W. Lu, C. L. Lu, R.C.T. Lee, Exact String matching rules for algorithms”, http://alg.csie.ncnu.edu.tw/lecture_notes_stringmatching.php.

[17] Awsan Abdulrahman Hasan and Nur’aini Abdul Rashid, Hybrid exact string matching algorithm for intrusion detection system, ICCIT, 2012.

[18] Frantisek Franek, Christopher G. Jennings, and William F. Smyth, A simple fast hybrid pattern- matching algorithm, springer, 2005.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY