Turkish Journal of Computer and Mathematics Education Vol.12 No.12 (2021), 1471-1491

Research Article

Survey and Comparison of String Matching Algorithms

Chayapathi A R[a], G Sunil Kumar[b], Manjunath Swamy BE[c], Thriveni J[d], Venugopal K.R[e]

[a] Information Science Department, Visvesvaraya Technological University, Acharya Institute of Technology Bengaluru, Karnataka, India, archayapathi@gmail.com
[b] Computer Science Department, Visvesvaraya Technological University, Banglore University, UVCE Bengaluru, Karnataka, India,gsuneel.k@gmail.com
[c] Computer Science Department,Don Bosco Institute of Technology,Bengaluru, Karnataka,India,
[d,e] Computer Science Department, Banglore University, UVCE Bengaluru, Karnataka, India

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 23 May 2021

Abstract: There are many applications which makes use of pattern matching algorithm. Most of current websites implements pattern matching algorithm in order to display the results faster. There exist different kind of data such as image, text, video, audio. In order to deal with such kind of data different pattern matching algorithms are used. One algorithm performs well in particular type of data, while it degrades in other kind of data. Our aim is to find best pattern matching algorithm.One of the key aspects of any string-matching algorithm is how fast the string matching is done along with the degree of search performance. This paper offers a survey on various String-matching algorithms along with the comparative analysis to provide a brief idea regarding the better algorithm for improving the search performance..

Keywords: Brute Force,Rabin-Karp,Boyer-Moore, Knuth-Morris,Aho-Corasick,Commentz-walter, Smith-Waterman, Needleman-Wunsch, Hamming Distance, Levenshtein Distance

1. Introduction

In the current world any websites with or without internet connected will implement search options in their web applications. This is implemented to get the results with less time without searching whole website. Pattern matching algorithms had made its roots in many domains such as medical, information technology, data mining, machine learning, forensics, network, defence, space. Patter matching algorithm is a technique which accepts two parameters such as the pattern and the large set of data or document which may or may not contain given pattern, then the pattern is matched against the document to find whether it exist in that document or not. Required actions are taken based on the results.

Stringmatchingalgorithmsareidentifiedin various methods.SuchasApproximateandExact stringmatchingalgorithms. Exactstringmatchingissearchingfor the same pattern in the text and approximatestringmatching issearching for the mostsimilarpattern in the text. And, the search can be made on the basis of the pattern occurrence in the given text. TheyareSinglepatternsearchandMultiplepattern search. Single pattern search is searching for the single and first incidence of the pattern in the text and the process of identifying the many existence of the same given pattern in the text is Multiple pattern search.

The main job of pattern matching algorithm is to find whether given pattern exist in the large set of data. Based on the match one can take required decisions. Algorithms implemented must be in such a way that it should meet the requirements such as time complexity, space complexity, memory and fetch the results faster.

There are several Pattern matching algorithms namely BoyerMoorealgorithm,Rabin-Karp algorithm,naïvestringsearchalgorithm, Needleman-Wunsch algorithm,Hamming distance andLevenshteinalgorithm,Commentz-Walter algorithm etc. that can be applied for exact or approximatesearches to be made accordingly. All of these stringmatchingalgorithms play a vital role in implementing the above-mentioned applications in the real-world scenarios.

2. Survey On0pattern00matching Algorithms

BruteForceAlgorithm

Bruteforcealgorithmpopular as Naïve algorithm. It is very direct approach to search any text string. It keeps iterating through the text, and the pattern is comparedwiththefirstfewcharacters of text for the length of pattern. If mismatch occurs shift the pattern one step right and with the first character of pattern compare next character of text and if match occurs proceed comparison with next charactersofboth text and pattern. Continue the above process, if match occurred for the entire length of pattern that means pattern occur in the text string hence return the position where the match occurred. Timecomplexity isO(m*n) as both worst case and best case,where(m)is the length of textstring andnis thelength of the pattern.

Input: Patt [1….m] is a pattern string; Txt[1….n] is a text string.

Output: Position of the sub-string of textmatching Patt or -1 if not matched then its returned

forj ← 0 to n-m do

i ← 0

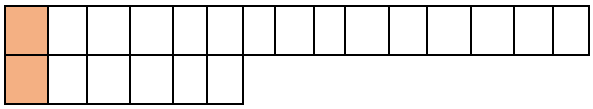
While i<m && Patt[i]==Txt[j+i] do

i ← i+1

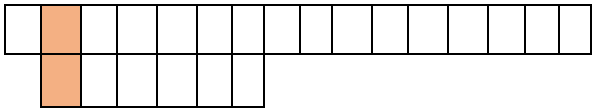
If i==m return j //match successful

Return -1 // matchunsuccessful [2]

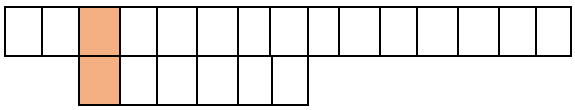
Consider an example where text be “CAT IS A MAMMAL” and pattern be “MAMMAL”.



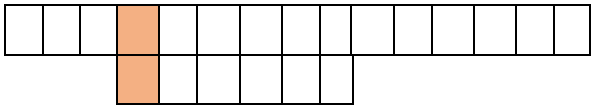
C! = M hence shift pattern by 1 and compare pattern from next character of text.



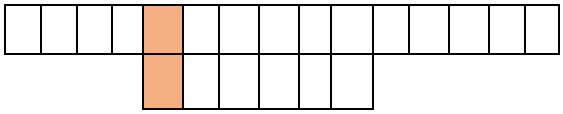
A! = M and hence shift pattern by 1 and compare pattern from next character of text.



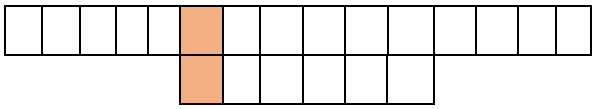
T! = M and hence shift pattern by 1 and compare pattern from next character of text.



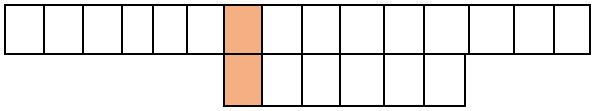
‘ ‘! = M and hence shift pattern by 1 and compare pattern from next character of text.



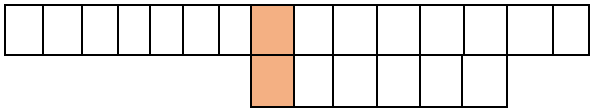
I ! = T and hence shift pattern by 1 and compare pattern from next character of text.



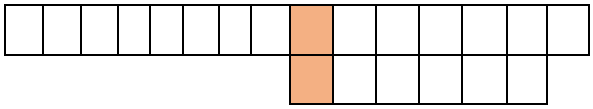
S! = M and hence shift pattern by 1 and compare pattern from next character of text.



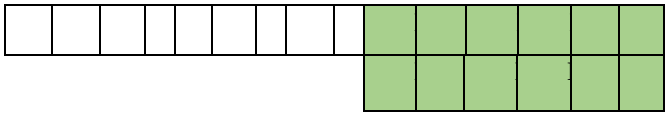
‘ ‘! = M and hence shift pattern by 1 and compare pattern from next character of text.



A ! = M and hence shift pattern by 1 and compare pattern from next character of text.



‘ ‘! = M and hence shift pattern by 1 and compare pattern from next character of text.



Now allthecharactersofpattern,match with the txt character, hence algorithm returns the position where match is successful.

Applications

Thebrute-forcealgorithmis used to determine the matches between the decimal RGB frames and the secret text in video steganography. [3] `

Advantages

- Brute force algorithm is a basic and simple algorithm mainly used when search happens in small amount of data.
- It does not require pre-processing.

Disadvantages

- It is not efficient algorithm hence not possible to implement where data is in huge quantity.
- It fails solving the problem which contains hierarchical structured data and the data contains logical operations.
- It is not efficient when there are lots of matching prefixes ex: if pattern is “ddde” and text is “ddddddddddde”.

B. Rabin-Karp Algorithm

Rabin-Karp Algorithm works based on the hashing technique. It is similar to brute force comparisonexcept it improves the speed of comparison.Firststepistocalculate the hash value of the given pattern. It makes window of size length of pattern, and this window is made movement right to the text each time when hash values become unequal. Second step is to calculate the hash value of characters inside the window. Then the algorithm iterates through the text string. If hash values of pattern and window become equal then onlyitstarts comparison of each character in the windowwitheachcharacterofpattern and ifall thecharactersof window matches with the characters of pattern then return the position of pattern in the text. If characters mismatch then it stops comparison and moves to the right by one character and continue the above process. O(m*n) is the worst case Time-Complexity and O(m+n) as average case.

Algorithm

rabinKarpSearch(txt, patt, prm)

Begin

patternLen := pattern-Length

patternHash := 0andstringHash := 0,h := 1

stringLen := string Length

mxChar:=totalnoofcharactersin the characterset

forindexkofallcharacterin patt, do

hsh := (h*mxChar)modprm

done

forallcharacterindexk of patt, do

patternHash := (mxChar*patternHash + patt[k]) mod prm

stringHash := (mxChar*stringHash + txt[k]) modprm

done

for k := 0 to (stringLen - patternLen), do

if patternHash = stringHash, then

forchrIndex:= 0topatternLen -1, do

if txt[k+chrIndex] ≠ patt[chrIndex], then

breaktheloop

done

ifchrIndex = patternLen,then

printthelocationkaspatternfound at kposition.

if k < (stringLen - patternLen),then

stringHash := (mxChar*(stringHash – txt[k]*hsh)+txt[k+patternLen]) mod prm, then

if stringHash< 0, then

stringHash := stringHash + prm

done

End [4]

For example, consider text = "acbfabcgef" and pattern = "abc".

First calculate the hash value of the pattern. Let prime number be 3. Let the values for alphabets be 1 to 26 for a to z respectively.

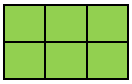
Hash value = x1* prime^0 + x2 * prime^1 + ….. + xn * prime^n.

Where, {x1, x2, …., xn} are thecharactersof the txtstring,n is thelengthofpattern.

1. hash (abc) = 1*3^0+2*3^1+2*3^2 = 34.
hash of first three characters of text is hash (acb) = 1*3^0+3*3^1+2*3^2 = 28

28! = 34 hence calculate hash value of next three characters of text.

2. In order to make efficient algorithm calculate the hash value using rolling hash function
Let x = oldhash value – previous character value
x = x/prime
newhash value = x + value of last character in the window * prime^length(pattern) -1.
Therefore, hash(cbf) is
x = 28 – 1 = 27
x = 27/3 = 9
hash(cbf) = 9+6*3^2 = 63
63! = 34 hence calculate hash value of next three characters of text.
3. hash(bfa) is
x = 63 – 3 = 60
x = 60/3 = 20
hash(bfa) = 20 + 1*3^2 = 29
29! = 34 hence calculate hash value of next three characters of text.
4. hash(fab) is
x = 29 – 2 = 27
x = 27/3 = 9
hash(fab) = 9+2*3^2 = 27
27! = 34 hence calculate hash value of next three characters of text.
5. hash(abc) is
x = 27 – 6 = 21
x = 21/3 = 7
hash(abc) = 7+2*3^2 = 34.
34 == 34 hencenowcompare each characterof patternwith the chosen text characters



All the characters match with the pattern hence stop iteration and returnthepositionof pattern in the text that is 5.

Applications

- Detecting plagiarism.[5]

- Text processing
- Bioinformatics
- Compression [6]

Advantages

- It increases the speed when compared to brute force algorithm.
- Since it compares hash value first it skips character comparison against the pattern character and calculation of hash value takes less time.
- It can deal with multiple pattern matching hence good for plagiarism.

Disadvantages

- It performs inefficient when compared to brute force algorithm if hash values become equal and the characters are not same as pattern.
- It requires additional space.

C.Aho-CorasickAlgorithm

The Aho-Corasickalgorithm is a popular dictionary matchingalgorithm.Herematching of all the dictionary words in a single iteration of text input is accomplished. Given all the dictionary words as the input, the algorithm firstly pre-processes them to build an automaton once and save for later data stream to match.

Aho-Corasick algorithm works by buildinga state machine usinga string for comparison. The state machine will begin with a null empty root node which is by non-attendanceunmatched state.Each pattern to be compared appends states to the machine, initially from the root node till pattern end is reached. By the traversal of state machine failure pointers are detectedand inserted from each node to the highest prefix of the node.

First step is to build tier which is a tree like structure, tree ends with leaf and each leaf gives the various dictionary words. Next step is to construct failure function. Failure function is built in such a way that if the proper suffix of the current node is also a proper prefix then add a link from current node to the node which is also a proper prefix. If there is no suffix or if there is no proper prefix for the current node's proper suffix then add link to the starting node or the root node. It has three important functions success transaction, failure transaction and finally outputmatching. **Words for each tire node will be set up using bread first search traversal on the tire. The success transactions follow the edge in the tire to find the children of current tire node. The failure transaction set up links between failed string matches and the node on other branches which share the longest common suffix. The output list stores all the words ending at current node and its failure node.**

While running the algorithm it traverses the graph starting by success transaction to child node. If the pattern does not exist then follow failure transaction to its proper suffix node. If the algorithm reaches the node where output keyword is not empty, then algorithm will returns all the matched characters that ends at the current character position of the input text string. It has time complexity of O(m+n) .

Algorithm

buildTree (patList, s)

Input: The list of all patterns, and the size of the list

Output: Transition map is generated to find the patterns

```
Begin
initialize elements to output-array to 0
initialize elements to fail-array to -1
initialize elements to goto matrix to -1
s := 1       //at first there is only one state(s)
for every pattern 'i' in the patList, do
word := patList[i]
present := 0
  for every character 'ch' of word, do
```

```
    if goto[present, chr] = -1 then
goto[present, chr] := state
s := s + 1
present:=goto[present, chr]
  done
  out[present] := out[present] OR (shift left 1 for i times)
 done
for every characters chr, do
  if goto[0, chr] ≠ 0 then
    fail[goto[0,chr]] := 0
    insert goto[0, chr] into a Queue q
 done
while q is not empty, do
newState := first element of q
  delete from q
  for every character chr, do
    if goto[newState, chr] ≠ -1 then
failure := fail[newState]
      while goto[failure, chr] = -1, do
failure := goto[failure, chr]
      done
fail[goto[newState, chr]] = failure
out[goto[newState, chr]] :=out[goto[newState,ch]] OR out[failure]
      insert goto[newState, chr] into q.
  done
done
return s
End
```

getNextState(presState, nextChar)

Input:the present state character and the next character to findthe next state

Output: the next state

```
Begin
answer := presState
ch := nextChar
  while goto[answer, chr] = -41, do
answer := fail[answer]
  done
  return goto[answer, chr]
End
```

patternSearch(patList, s, text)

Input: List of patterns, size of the list and the main text

Output: The indexes of the text where patterns are found

Begin

call buildTree(patList, s)

presState := 0

for every indexes of the text, do

if out[presState] = 0

ignore the next portion and go for next iteration

for every patterns in the patList, do

if the pattern is found using output array, then

print the location where pattern resides

done

done

End [7]

Consider an example where finite set of patterns be {HONEY, MOON, MONEY and NET}

Automata for the above patterns is shown in the fig1

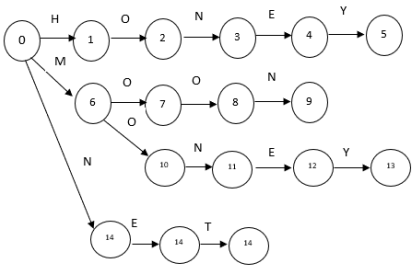


Fig. 1. Automata

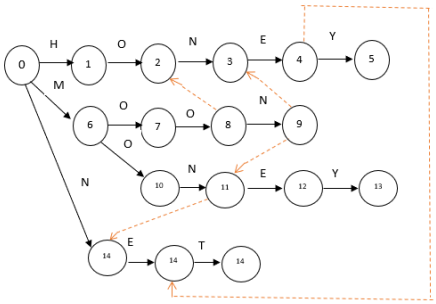


Fig. 2. Failurefunctionfortheautomata

Then failure function is constructed as shown below fig2.

Outputfunctiontransitionis shown in the fig3 andoutputfunctiontableisshowninthefig4.

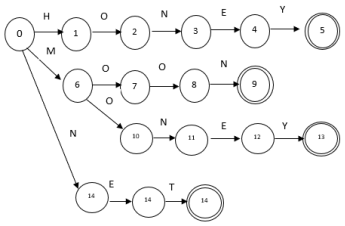


Fig.03.0Output0function

FINAL STATE	OUTPUT
NODE 5	HONEY
NODE 9	MOON
NODE 12	MONEY
NODE 15	NET

Fig. 4. Output function table

Finally, pattern is searched in the constructed automata. the searching phase of ahcorasick is simple while scanning the text it walks through automata if any transition found, it getstransition, else check the failure function.

If text is HONEYPOTNET then search is done as shown in the fig5. [8].

STATE	CHARACTER	TRANSITION	FAILURE	COMMENT
0	H	0 -> 1	-	TRANSITION FOUND
1	O	1 -> 2	-	TRANSITION FOUND
2	N	2 -> 3	-	TRANSITION FOUND
3	E	3 -> 4	-	TRANSITION FOUND
4	Y	4 -> 5	-	TRANSITION FOUND
5	P	-	0	TRANSITION NOT FOUND
6	O	-	0	TRANSITION NOT FOUND
7	T	-	0	TRANSITION NOT FOUND
8	N	0 -> 13	-	TRANSITION FOUND
9	E	13 -> 14	-	TRANSITION FOUND
10	T	14 -> 15	-	TRANSITION FOUND

Fig. 5. Searching transition table of automata

From the fig 5 there exist two meaningful words from the given text, hence this algorithm can be used to identify any bad packets entering into the network.

Various Applications are

- Intrusiondetection mechanism
- Detection of Plagiarism
- Deploy Bioinformatics tools
- Applications of Digitalforensic
- Textmining arena

Advantages

- Everycharacteroftextisanalyzedonlyonetime.
- Despite of the input symbols. Thedeterministictransitionstepisachievedbetweenstates[9]

Disadvantage

- Algorithmmakesuseofmore storage to storetransitionrulesofthe deterministic finite state machine. [10]

D. Boyer-MooreAlgorithm

Boyer-Moore algorithm compares the characters starting from right to the left of the pattern against the text in the same direction as like pattern, starting with the index equal to the length of pattern-1. It matches the tail of the pattern rather then head. This algorithm makes use of bad match table which is the main cause to reduce the time complexity.

Constructionofbadmatchtable

1. This table must not have value less than 1.
2. Keep comparing the pattern to the text starting with the right most character in the pattern.
3. Make a table rows representing value and columns representing characters of the pattern.
4. The table must not contain repetitive character, if the pattern contains repeated character update the value corresponding to that character.
5. Value for last character will be length of pattern if that character was not existing before otherwise leave the same value.
6. Other character which is not present in the pattern is represented by * in the table and value assigned will be the length of the pattern.

This algorithm has time complexity of O(m/n) as best case, O(m*n) as worst case and O(m/|∑|) as average case .

Algorithm

fullSuffixMatch(shiftArr, borderArr, pattern)

Begin

n := pattern length

i := n

k := n+1

borderArr[i] := k

while i> 0, do

while k <= n AND pattern[i-1] ≠ pattern[k-1], do

if shiftArr[k] = 0, then

shiftArr[k] := k-i;

k := borderArr[k];

done

decrease i and k by 1

borderArr[i] := k

done

End

partialSuffixMatch(shiftArr, borderArr, pattern)

Begin

n := pattern length

j := borderArr[0]

for index of all characters „i" of pattern, do

if shiftArr[i] = 0, then

```
shiftArr[i] := j
   if i = j then
j := borderArr[j]
done
End
searchPattern(txt, patt)
Begin
patternLen := patt length
stringLen := txt size
for all entries of shiftArr, do
   set all entries to 0
done
call fullSuffixMatch(shiftArr, borderArr, patt)
call partialSuffixMatch(shiftArr, borderArr, patt)
shift := 0
while shift <= (stringLen - patternLen), do
j := patternLen -1
   whilej>=0andpatt[j]=txt[shift+j],do
   decrease j by 1
  done
if j<0,then
      printtheshiftas,thereisamatch
shift := shift +0shiftArr[0]
   else
      shift:=shift+shiftArr[j+1]
done
End0[11]
```

Consider an example, let text be “THIS IS A BOOK” and pattern be “BOOK”

Construct a bad match table as shown in the fig6:

Length of pattern = 4

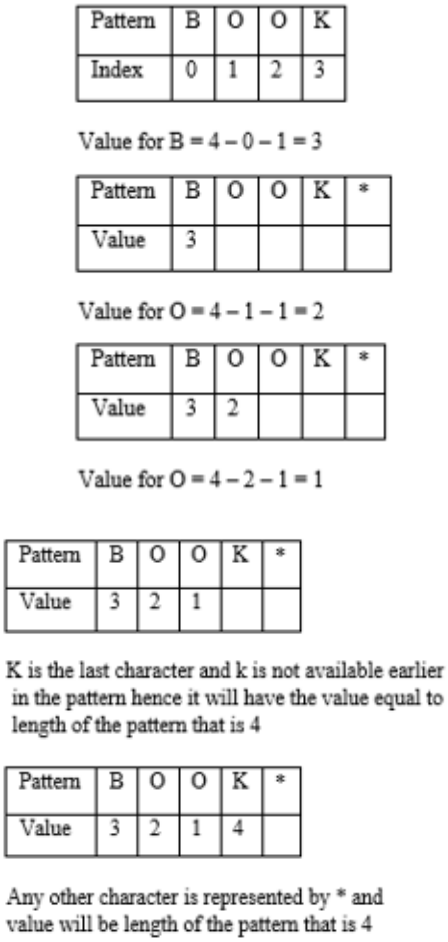
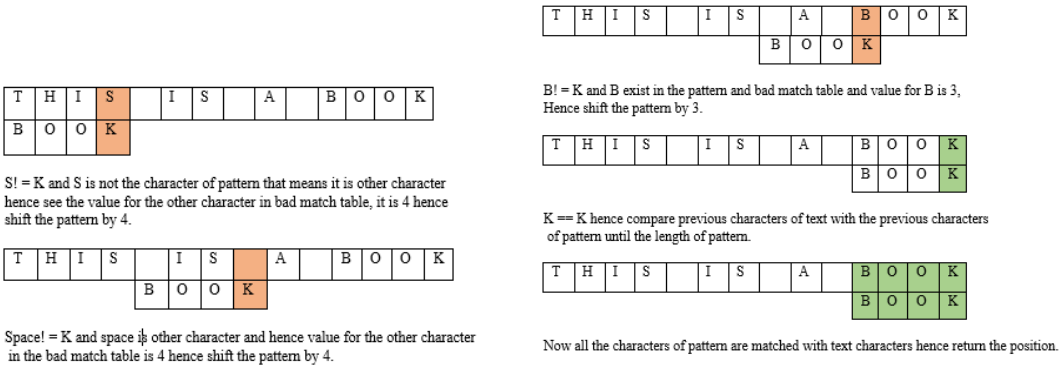


Fig. 6. Searching transition table of automata

Next compare with the text string considering bad match table.



Applications:

- Text editors
- Commands substitutions [12]
- Intrusion Detection System.

Advantages:

- Boyer-Moore algorithm pre-process only the pattern not the text.
- Algorithmrunsfasteraslengthofpatternincreases.
- Itskipsmanycharactersatthesame instance insteadofsearchingofevery characterhenceitisefficientlgorithm.

Disadvantage:

- Mismatch character will give small shift in some condition, if match not occurs after many matches [13].
- Unable to process small size patterns properly. [14]

E.Knuth-Morris-prattAlgorithm

Knuth-Morris algorithm contrast the characters of pattern and text from left to right. It works based on prefix and suffix match within the given pattern. Compare each character of text with each character of pattern, if all symbols of pattern matched with the text substring of length pattern, then return starting position of text string where pattern exist. If there is no match of particular character then find substring in the pattern which must be suffix as well as prefix in that substring. If no found then compare next character of text with starting character of pattern and continue the process. If suffix and prefix found then compare next character of text with next character immediately after the prefix substring and continue the process. This method avoids backward movement for comparison and also reduces time complexity. It has time complexity of (m) where m is the length of text string.

The algorithm can be made more efficient if temporary array is built. This array contains from which position comparison need to takes place. Time complexity to build array is O(n) where n is length of pattern. Hence over all it has time complexity of O(m+n).

Algorithm:

```
findprefix(patt, m, prefixArr)
Begin
len := 0
prefixArray[0] := 0
for all character index k of pattern, do
   if patt[k] = patt[len], then
      increase len by 1
prefixArray[k] := len
   else
      if len ≠ 0 then
len := prefixArr[len - 1]
            decrease k by 1
     else
prefixArr[k] := 0
 done
End
Kmp_Algorithm(txt, patt)
Begin
N1 := size of text
M1 := size of pattern
call findprefix(patt, M1, prefixArr)
while k < N1, do
   if txt[k] = patt[j], then
      increase k and j by 1
ifj=M1,then
  printthelocation
```

(k-j)asthepatternisthere

j:=prefixArr[j-1]

elseifk<N1ANDpatt[j]≠txt[k]then

ifj≠then

j:=prefixArr[j-1]

else

increasekby1

done

End[15]

For example, consider text be “abgabfabfabx” and pattern be “abfabx”.

Temporary array for pattern must be created before comparison as shown in the figure 7. Initially the values for first pattern will be zero.

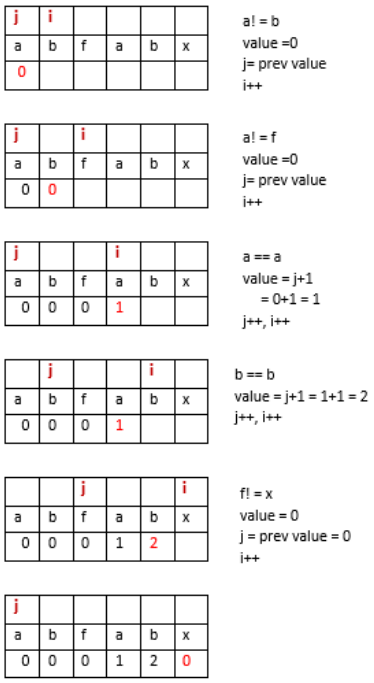


Fig .7. Temporary table

After construction of temporary table pattern is matched with text as shown in the figure 8.

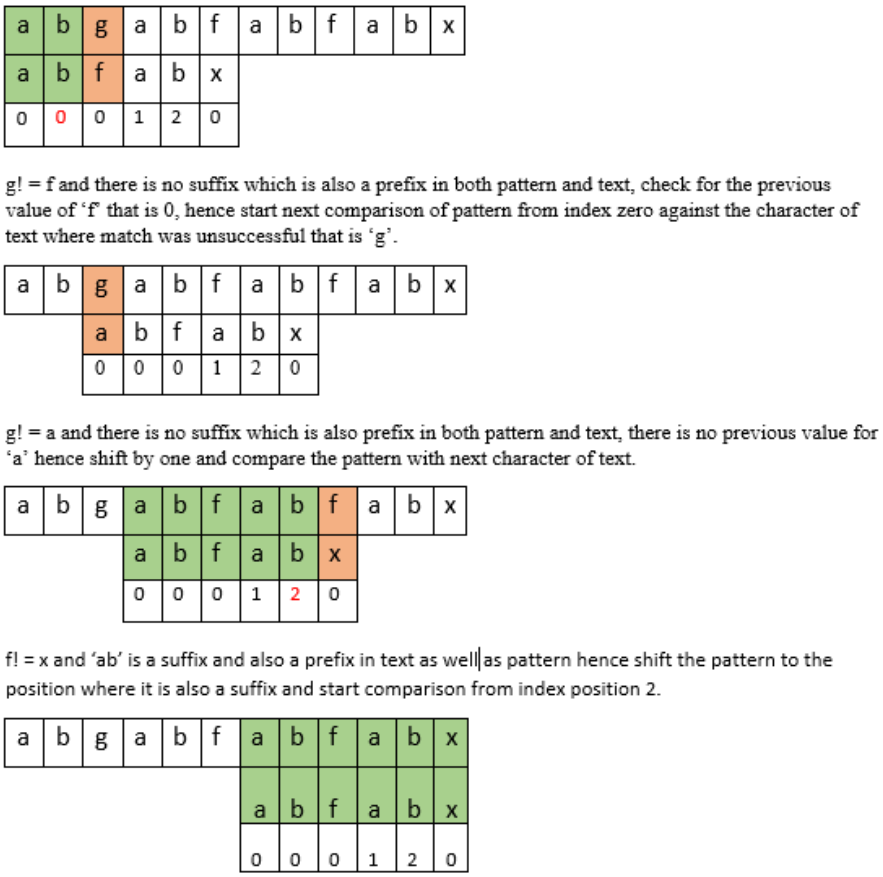


Fig .8. Text pattern comparison according to Knuth-Morris algorithm

Applications

- Parallel Knuth-Morris is to be used in parallel image processing applications [16]
- DNA sequence analysis.

Advantages

- It is more efficient than rabin karp and naïve algorithm.
- The execution time of KMP algorithm is O(m+n) which is very fast.
- Algorithm not required movingin backwards direction of the text string. [17]
- This algorithm works better if text length increases hence this algorithm is implemented where search need to be done in large documents.

Disadvantages

- It won't work so well as the alphabet size enhances. Due to which the odds of disparity is more. [18]

F. Commentz-WalterAlgorithm

- The string probingCommentz-Walter algorithm is proposed by Beate Commentz-Walter. It is a combined with several notes from Aho–Corasick with the fast matching of the Boyer Moore string search algorithm [19]. As in the Aho–Corasick string matching algorithm, at once it can investigate for multiple patterns. It suitsbest for the applications that possess pattern that are shorter than the text or where it carries onthroughseveralprobes. The Boyer–Moore algorithm uses information gathering during the pre-process step to skip sections of the text, resultant in a lower steady factors than many other string based search algorithms. From a generic perspective the execution of the algorithm speeds upwith increase in the length of the patterns.

The important step in this string matching algorithm is when the string matching process finds a mismatch in the end of the pattern then it skips the text instead of probing every symbol in the given text. If the characters are not matching with any of the characters in the text no need arises to continuebackward searching along the text. If the symbols in the probing text do not match with the pattern text, then the next character in the text to verify is found*n* characters farther along the text, where *n* is the length of the pattern. The length of the pattern can be formulated through a bad character table. A partial shift is initiated based on the presence of a character in the

text.Then aset up along with the matching character and the process is iterated. This method of jumping along the text for comparisons instead ofverifying every symbol in the text results in decrease in the number of comparisons.This enhances the competence of the algorithm. The Commentz-Walter algorithmhas a time complexity O(N+M+Z)+O(MN) for execution.

Algorithm:[25]

```
Computefunctionlast
a←k-1
b←k-1
Repeat
IfP[b]=T[a]then
ifb=0then
  returna     // wehaveamatch
else
a←a-1
0←b-1
else
a←a+k - Min(b,1+last[T[a]])
b←k-1
untila>n-1
Return"nomatch"
```

Example:

Input:MainString:“ABAAABCDBBABCDDEBCABC”,Pattern:“ABC”

Outputs/Results:

Search Patternoccursinlocation:4

Search Pattern occursin location: 1

Search Pattern occursin location: 18

Applications

- Text editors
- command substitutions

Advantage

- This algorithm is the fastest when pattern is moderately sized.

Disadvantage

- But the pre-processing time that is taken in this algorithm is considered to be a disadvantage as it requires more time.

G. Waterman Algorithm

The Smith Waterman algorithm is based on the principle of dynamic programming.It computes the optimal local alignment of two sequences [2]. The Smith Waterman algorithm is fordetecting local alignments of sequence.Alsoit ensuresdetectionof identical regions prevailing between two nucleotide or protein sequences. The algorithm is used to compare segments of all possible lengths to arrive atoptimal similarity. On comparing withthe Needleman Wunsch Algorithm, thealgorithm ensuresthat the negative scoring matrix cells are set tozero.Thus for backtracking only positive scoresare visible.The algorithm functions by starting with maximum scoring matrix cell and progress until zero-recorded cell is obtained.Finally it produces the local alignment with highest score.

The steps of operation for two sequences A and B are illustrated below:

1. Before and after alignment the symbols in a sequence should be in the identical order.

2. Establishing Alignmenta symbol from a sequence with another is always possible.

3. Alignmentsare denoted by a blank (‘-‘)

4. Alignment oftwo blanks is not permitted

Smith Waterman Algorithm relieson Gapped alignmentsto find the optimal distance between sequences by aligning with the gaps. Smith Waterman algorithm has atime complexityof O(MN) for execution.

Algorithm

1. Determinethe substitution matrix and also the gap in penalty scheme. s(a,b) is the similarity score forthe elements having2 sequences.Here k is the penalty of a gap with length-k

2. Create a matrix H of scores and assignit to the first row and first column. The scoring matrix size is given by the term (n+1)*(m+1). Also the matrix employs– a based indexing.

Hk =H l for <k<n and <l<m

3. Enter the scoring matrix using the equation below

Hij= max (Choice1←H(i-1,j-1)+S(B(i),A(j)) {score of aligning ai and bi}

Choice2 ← H(i-1,j) +d {score of ai along with gap}

Choice3←H(i,j-1)+d {score of bi along with gap}

{no similarity upto ai and bi})

4. Tracebackstartswith auppermost scores in the H- the score matrix and culminates at a matrix cell possessinga score of traceback which is relied on the origin of every score to produce recursive best local alignment [25].

Applications

- Biometrics

Advantage

- As it implies to the local alignment problems Optimal local alignment can be achieved.

Disadvantage

- But the time complexity and the space complexity for this algorithm is comparatively high.

H.Needleman-WunschAlgorithm

The Needleman-Wunsch algorithm works on the principle of optimal matching results. This is a basic algorithm employed for solving the problems of sequence alignment [21]. The Needleman-Wunsch algorithm operates by performing global alignment oftwo sequences.Moreover it is employed in the arena of bioinformatics for aligning protein and nucleotide sequences. This algorithm referred as optimal matching Algorithm and also is an example of dynamic programming. The aligned character scores are procured by using similarity matrix.Also the Linear Gap d is found

byusing similarity matrix. The Needleman-Wunsch Algorithm comprises three stages:

1. Score Matrix Initialization

2. Score calculation and completing the trace back matrix.

3. Draw Inferenceusing alignment of the trace back matrix[25].

The two types of matrices employedin Needleman-Wunsch Algorithm are:the score and the trace back matrices.

Traceback matrix algorithm:

1. Traceback employs a method of drawinginference of the paramount alignment throughtraceback matrices.

2. Traceback process compulsorily startsat the last cell and it is positioned as bottom right cell.

3. Its movementis basedon the traceback value provided in the cell.

4. Threepotentialtraversaloccurring are: diagonal, left or up.

5. The traceback process is completedwhen the top-left cell is indicated by- "done".

Best Alignment:

1. The traceback path based values are employed to infer the Alignments. Also, the values of the traceback matrix are taken into account.

2. The letters from two sequences are aligned in traceback matrix.Further Gap is created based on the sequence orientation. "Left" creates a Gap in the left sequence and a gap is created in the top sequence if it is "Up".And,thusprocured sequences have a backward alignment [25].

The Needleman-Wunsch algorithm has proven to produce best alignment for two sequences.It startsthe traceback is accomplishedfrom the right-lower corner position in the traceback matrix and further culminates at the left-top most cell position of the matrix.This is irrespective of the length or complexity of sequences.The algorithm has provento function identically and guarantees best alignment for different sequences. The Needleman Wunsch algorithm has a time complexity of O(MN) for execution.

Algorithm

fork=0tolength(B)-1

F(k,0)←d*k

end for

for l=tolength(A)-1

F(0,l) ←d*l

endfor

fork=1tolength(B)

forl=1tolength(A)

Choice1←F(k-1,l-1)+S(B(k),A(l))

Choice2←F(k-1,l)+d

Choice3←F(k,l-1)+d

F(k,l)←max(Choice1,Choice2,Choice3)

endfor

endfor

To compute alignment, start from right bottom cell from the matrix and choose the possible choices

if Choice1, then A(l) and B(k) are aligned

if Choice2, then A(l) is aligned with a gap

if Choice3, then B(k) is aligned with a gap

Applications

- Bioinformatics to align nucleotide sequence

Advantage

- This search algorithm considers order of sequence of characters while comparing which makes it more efficient.

Disadvantage

- But requires same length of string ie. The pattern and the text for comparing.

H. HammingDistanceAlgorithm

Hamming Distance Algorithm is an approximate matching algorithm which allows definitedistinction in the sample and the text during string matching. Estimated match is allowed for a limited number of errors or edit operations required for the search pattern to match with the text [24]. The mismatches can occur due to any difference in the character called 'mismatch/substitution' or an extra character called 'insertion' or a missing character called 'deletion'. Considering two strings of the same length, hamming distance between the two strings can be defined as the minimum number of replacements one should make to turn one of the strings asanother. Hamming distance is measuredby tracking the number of positions where corresponding symbols differ from each other. For alphabetical strings and DNA sequences the distance also works.

. Hammingdistancemodelhasthe time complexity$O(N^2)$forexecution.

Algorithm:[25]

//0initialization

i=0count=0

whilestr1[i]!=str2[i]

count++

i++

endwhile

returncount

Example

In this example two DNA sequences considered are:AACTCCA and AGCTAAC, the Hamming distance occurringis 4, sincesymbol mismatch occurs at positions 2, 5, 6 and 7.

Applications

- Systematics as a measure of genetic distance

Advantage

- Suitable for exact string matching and allows Single-bit error detection and correction.

Disadvantage

- But it requires more execution time.

I.LevenshteinDistanceAlgorithm

Levenshtein Distance is an approximate matching algorithm which allows certain differences in the pattern and the text while string matching. String resemblancecomprisesof wide-arrayof applications, prominent ones are: web search, text comparison, plagiarism detection. Also thedifferent computationmethods exist,Salient ones are:the longest common, edit distance, and, substring algorithms [22]. Based on approximate matching, a restricted number of faults or correction operations are identified for the pattern searchedin the matching process. The mismatches can occur due to any difference in the character called 'mismatch/substitution' or an extra character called 'insertion' or a missing character called 'deletion'. Considering two strings of the same length, Levenshtein edit distance between the two strings can be defined as the minimal number of replacementsthat should be made to turn one of the strings to the other which includes substitution, insertion as well as deletion. The difference between Hamming distance and edit distance is that, here we are notconsidering distance and the strings no longer need to be of the same length as they go through in sections and deletions as well. This algorithm is possessing O(N+M) time complexity for execution.

Algorithm:[25]

//initialization

forq←tomdo

E(q,0)←q

endfor

form←0tondo

E(0,m)←0

endfor

//editdistanceE(q,m)

forq←0tomdo

form←0tondo

if(T(m)=P(q))then

E(q,m)←(q-1,m-1)

else

min←MIN[E(q-1,m),E(q,m-1)]

E(q,m)←min+1

end if

endfor

endfor

returnE

Example

levenshtein distance between *barking* and *dark*, thesetransformationare accomplished:

1. The word barking→(indicated as) barkin (with deletion of g)

2. The word barkin→ barki (with deletion of n)

3. The word barki→ bark (with deletion of i)

4. . The word bark→ dark (with substitution of b)

Thus it can be concluded that Levenshtein distance obtainedforthe two word strings is 4.

Application

- Spell checkers

Advantage

- As this algorithm uses the Edit distance which allows insertion and deletion along with substitution like the Hamming distance algorithm makes it much more efficient.

Disadvantages

- But this algorithm does not consider order of sequence of characters.

3. Comparative Analysis

Algorithm	**Comparison**	**Pre-processing**	**Time Complexity**
Brute Force	Right side to Left side	None	O(n*m)
Rabin-Karp	Right side to Left side	O(n)	avg O(m+n) worst O(m*n)
Aho-Corasick	Not applicable	O(m+n)	O(N + L + Z)
Boyer-Moore	Right side to	O(n+|∑|)	O(n), Ω(m/n)

	Left side		
Knuth-Morris	Right side to Left side	O(n)	O(m)
Commentz-walter	Right to Left	none	O(N+M+Z)+O (MN)
Smith-Waterman	Right side to Left side	-	O(MN)
Needleman-Wunsch	Right side to Left side	-	O(MN)
Hamming Distance	Right side to Left side	-	$O(N^2)$
Levenshtein Distance	Right side to Left side	-	O(N+M)

Table .1. Comparative analysis of the algorithms

Table 1.1 shows pre-processing time, comparison order, and time complexity for all the ten algorithms. Time complexity is different for each algorithm. When compared to all algorithms Knuth Morris algorithm has less time complexity. Hence Knuth Morris algorithm is an efficient algorithm.

4. Conclusion

From the survey the conclusion is that Boyer Moore and Knuth Morris algorithms have less time complexity. Both the algorithms have similar time complexity. Boyer-Moore algorithm works better if the pattern length is large. Whereas the Knuth-Morris algorithm is efficient when length of text string is larger and pattern has repeated patterns. Boyer-Moore algorithm is better to use if the pattern length is large and Knuth-Morris algorithm is better to use if length of text string is larger.

References

1. Jiji. N ,Dr. T Mahalakshmi ,Survey of Exact String Matching Algorithm for Detecting Patterns in Protein Sequence, Advances in Computational Sciences and Technology ISSN 0973-6107 Volume 10, Number 8 (2017) pp. 2707-2720
2. Vibha Gupta, Maninder Singh, Vinod K. Bhalla ,Pattern Matching Algorithms for Intrusion
3. Detection and Prevention System: A Comparative AnalysisInternationalConferenceon Advances in Computing,Communicationsand Informatics (ICACCI),2014
4. KhuloodAbuMaria,Mohammad A. Alia, MaherA. AlsarayrehandEman Abu Maria UN-Substituted Video Steganography, KSII TRANSACTIONS ON INTERNET AND INFORMATION SYSTEMS VOL. 14, NO. 1, January 2020.
5. Sheshasayee, A., &Thailambal, G. A comparitive analysis of single pattern matching algorithms in text mining. 2015 International Conference on Green Computing and Internet of Things (ICGCIoT), (2015).
6. en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm.
7. www.cs.rit.edu/~lr/courses/alg/student/1/Rabin_Karp.pdf.
8. Alfred V. Aho and Margaret J. Corasick Efficient String Matching: An Aid to Bibliographic Search.
9. Saima Hasib, Mahak Motwani, Amit Saxena, International Journal of Computer Science and Information Technologies, Vol. 4 (3) (2013)
10. HyunJin Kim, A Memory-Efficient Deterministic Finite Automaton-Based Bit-Split String Matching Scheme Using Pattern Uniqueness in Deep Packet Inspection.
11. Zeeshan Ahmed Khan, R.K Pateriya,Multiple Pattern String Matching Methodologies: A Comparative Analysis, International Journal of Scientific and Research Publications, Volume 2, Issue 7, July 2012 3 ISSN
12. Sheshasayee, A., &Thailambal, G. (2015). A comparitive analysis of single pattern matching algorithms in text mining. International Conference on Green Computing and Internet of Things (ICGCIoT) 2015.
13. www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/ boyerMoore.htm

14. Vivek Srivastava, B K Trapathi, V K Pathak, A Novel Hybrid Intelligent Model for Classification and Pattern Recognition Problems, (IJCSIS) International Journal of Computer Science and Information Security, Vol. 10, No. 2, February 2012
15. S. Antonatos, K. G. Anagnostakis, M. Polychronakis, and E. P. Markatos, "Performance analysis of content matching intrusion detection systems," in Proc 4th IEEE/IPSJ Symposium on Applications and the Internet, 2004, pp. 208-215.
16. SS. Swapna, Yashdeep Jha, Syed Zaheed, Keertik Dewangan, Sayyed Mujahid Pasha, A Survey on Different Pattern Matching Algorithms of Various Search Engines, International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
17. SercanAygün , EceOlcayGüneş, LidaKouhalvandi,Python Based Parallel Application of Knuth–Morris–Pratt Algorithm, IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE) 2016
18. www.slideshare.net/sabiyasabiya/knuth-morris-pratt-string-matching-algo
19. Kranthi Kumar Mandumula, Knuth-Morris-Pratt,Indiana State University Terre Haute IN, USA December 16, 2011
20. Beate Commentz- Waiter:"A String Matching Algorithm fast on the Average", TR 79.09.007 Heidelberg ScientifiCenter, IBM, Germany Sept. 1979.
21. Hsien-Yu Liao, Meng-Lai Yin, Yi Cheng,"A Parallel Implementation of the Smith-Waterman Algorithm for Massive Sequences Searching",in Proceedings of the 26th Annual International Conference of the IEEE EMBS San Francisco, CA, USA • September 1-5, 2004.
22. Bailong FENG and Jing GAO, "Distributed Parallel Needleman-Wunsch Algorithm on Heterogeneous Cluster System", in Proceedings of the 2015 International Conference on Network and Information Systems for Computers, 2015.
23. ShengnanZhang , Yan Hu , GuangrongBian "Research on String Similarity Algorithm based on Levenshtein Distance", School of Computer Science and Technology, Wuhan University of Technology, Hubei Province, Wuhan, China, Department of Aviation Ammuniton, Air Force College of Service, Jiangsu Province, Suzhou, China, 2017.
24. Prince Mahmud, Md. Sohel Rana, Kamrul Hasan Talukder, "An Efficient Hybrid Exact String Matching Algorithm to Minimize the Number
25. of Attempts and Character Comparisons", 21st International Conference of Computer and Information Technology, 2018.
26. Solon P. Pissis and Ahmad Retha, "Generalised Implementation for Fixed-Length Approximate String Matching underHamming Distance & Applications", IEEE International Parallel and Distributed Processing Symposium Workshops, 2015.
27. www.wikipedia.org