ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 6, Issue 10, October 2019

A Survey on Different Pattern Matching Algorithms of Various Search Engines

[1] SS.Swapna, [2] Yashdeep Jha, [3] Syed Zaheed, [4] Keertik Dewangan, [5]Sayyed Mujahid Pasha

[1] Assistant Professor, Department of CSE, Pallavi Engineering College

[2][3][4][5] B.Tech-CSE, Pallavi Engineering College,Nagol

***Abstract*— In Real-time world problems need fast algorithm with minimum error. Now a days many applications are use for searching results on web. There are many algorithms which are used for searching the results. Pattern matching method is one of them. In web application people deals with the different types of data, for example text searching, image searching, audio searching and Video searching. Every search engine uses different search algorithms for handling different types of data. This paper proposes an analysis and comparison of four algorithms for full search equivalent pattern matching like complexity, efficiency and techniques. The four algorithms are Naive string search algorithm, Rabin Karp String Search Algorithm, Knuth–Morris–Pratt algorithm, Boyer–Moore string search algorithm. This paper provides an analysis of above algorithms.**

***Keywords*— Pattern matching, Text searching, Image searching, Audio searching, Video Searching, Search Engines**

I. INTRODUCTION

In web search engine every searching operation is done online. Now a day's different search engine are in the market like Google, yahoo etc. The performance of any search engine depends on its searching capabilities. Searching a list for a particular item is a regular task. In real applications, the list items often are records and the list implemented as an array of objects. In search engine it deals with the different type of data (text, Image, Audio, Video). For handling such type of data there are two types of searching methods used. Linear and Binary searching method.

Linear search: finds an item in an unsorted sequence .For search algorithms, the main steps are the comparisons of list values with the target value. Counting these for data models representing the best case, the worst case, and the average case produces the following table.

TABLE I –LINEAR SEARCH COMPLEXITIES

Cases	*Complexity*
Best Case	O(1)
Worst Case	O(n)
Average Case	O(n)

Binary search algorithm: The binary search follows Divide and Conquer approach .It first Sorts the unsorted list, then it finds a middle value .It compares the key value which we are searching with the middle value if they are equal then we have successfully searched the values, if key value is greater then key value then we search the right sublist and if smaller we search the left sublist.This process go on till we have got the required value or we have reached our last value

TABLE II–BINARY SEARCH COMPLEXITIES

Cases	*Complexity*
Best Case	O(1)
Worst Case	O(log n)
Average Case	O(log n)

II . TYPES OF SEARCHES

A. Search by Text - In text searching we enter a text or a string about which we have to search in the search engine , then the search engine search all the related documents o the documents which have that string in them and display them to us.

B. Search by Image - It is a content-based image retrieval (CBIR) query technique that involves providing the CBIR system with a sample image that it will then base its search upon; in terms of information retrieval, the sample

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 6, Issue 10, October 2019

image is what formulates a search query .This effectively removes the need for a user to guess at keywords or terms that sometimes may not return a correct result. It also allows users to discover content that is related to a specific sample image.

C. Search by Video - There is no generic way currently to search by video. You can try a few tricks. Search by image for the thumbnail of the video or search for keywords related to the video. Search Engines like Google keep metadata, so searching for meta description or meta keywords may give results.

D. Search by Audio - In search by audio, the user must play the audio of a song either with a music player, by singing or by humming to the computer microphone. Subsequently, a sound pattern, A, is derived from the audio waveform, and a frequency representation is derived from its Fourier Transform. This pattern will be matched with a pattern, B, corresponding to the waveform and transform of sound files found in the database. The audio files in the database whose patterns are matching the pattern search will be displayed as search results.

III. NEED OF PATTERN MATCHING

Patterrn matching is the process of checking a perceived sequence of string for the presence of the constituents of some pattern.Alike pattern recognition, the match usually has to be same. The patterns have the form sequences of pattern matching include giving the locations of a pattern within a string sequence, to output some component of the matched pattern, and to substitute the matching pattern with any other string sequence (i.e., search and replace). Pattern matching concept has many applications Following figure shows the different applications.

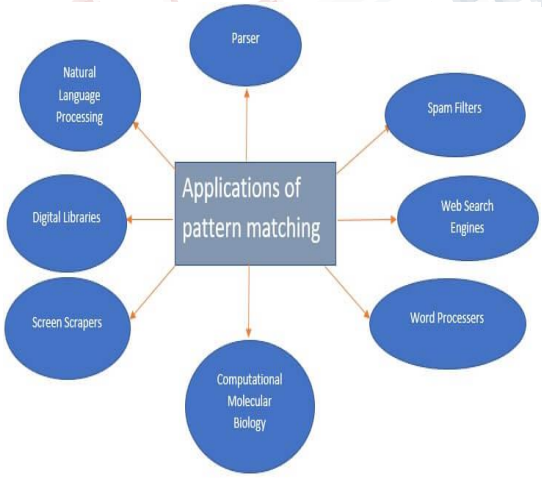

FIG I – APPLICATIONS OF PATTERN MATCHING

In pattern matching I focused on the web search engine amongst others application. Now a day's almost everybody use the web application to get the required results. But peoples are not only searching for text every time. They may search different type of data like audio, image and video. To handle such kind of data we need more efficient method for searching. Pattern matching will help us to find right and appropriate result. There are a lot of algorithms used for pattern matching.

IV . ALGORITHMS USED FOR PATTERN MATCHING

A . Naive string search algorithm - Naïve pattern searching is the simplest method among other pattern searching algorithms. It checks for all the characters of the main string to the pattern. This algorithm is helpful for smaller texts. It does not need any pre-processing phases. We are able tofind substring by checking once for the string. It also does not occupy extra space to perform the operation. In worst cases the time complexity of Naïve Pattern Search method can be O(m*n), where n is the size of string and n is the size of the pattern.

Algorithm:

```
naivePatternSearch(pattern, text)
Begin
  patLen := pattern Size
  strLen := string size

  for i := 0 to (strLen - patLen), do
    for j := 0 to patLen, do
      if text[i+j] ≠ pattern[j], then
        break the loop
    done
    if j == patLen, then
      display the position i, as there pattern found
  done
End
```

B. Rabin Karp String Search Algorithm - Rabin-Karp is another pattern searching algorithm to find the pattern in a more efficient way. It also checks the pattern by moving window one by one, but without checking all characters for all cases, it finds the hash value. When the hash value is matched, then only it tries to check each character. This procedure makes the algorithm more efficient.

Algorithm:

```
rabinKarpSearch(text, pattern, prime)
Begin
  patLen := pattern Length
  strLen := string Length
  patHash := 0 and strHash := 0, h := 1
  maxChar := total number of characters in character set
```

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 6, Issue 10, October 2019

```
  for index i of all character in pattern, do
    h := (h*maxChar) mod prime
  done

  for all character index i of pattern, do
    patHash := (maxChar*patHash + pattern[i]) mod prime
    strHash := (maxChar*strHash + text[i]) mod prime
  done

  for i := 0 to (strLen - patLen), do
    if patHash = strHash, then
      for charIndex := 0 to patLen -1, do
        if text[i+charIndex] ≠ pattern[charIndex], then
          break the loop
      done

      if charIndex = patLen, then
        print the location i as pattern found at i position.
    if i < (strLen - patLen), then
      strHash := (maxChar*(strHash – text[i]*h)+text[i+patLen]) mod prime, then
    if strHash < 0, then
      strHash := strHash + prime
  done
End
```

C. Knuth–Morris–Pratt algorithm - Knuth Morris Pratt (KMP) is an algorithm, which checks the characters from left to right. When a pattern has a sub-pattern appears more than one in the sub-pattern, it uses that property to improve the time complexity, also for in the worst case.

Algorithm :

```
findPrefix(pattern, m, prefArray)
Begin
  length := 0
  prefArray[0] := 0

  for all character index 'i' of pattern, do
    if pattern[i] = pattern[length], then
      increase length by 1
      prefArray[i] := length
    else
      if length ≠ 0 then
        length := prefArray[length - 1]
        decrease i by 1
      else
        prefArray[i] := 0
  done
End
kmpAlgorithm(text, pattern)
Begin
  n := size of text
  m := size of pattern
  call findPrefix(pattern, m, prefArray)

  while i < n, do
    if text[i] = pattern[j], then
      increase i and j by 1
    if j = m, then
      print the location (i-j) as the pattern is there
      j := prefArray[j-1]
    else if i < n AND pattern[j] ≠ text[i] then
      if j ≠ 0 then
        j := prefArray[j - 1]
      else
        increase i by 1
  done
End
```

D. Boyer–Moore string search algorithm -

The algorithm scans the characters of the pattern from right to the left beginning with the rightmost one. In case of a mismatch or a complete match of the whole pattern, it uses two pre-computed functions to shift the window to the right.The two shifts functions are as follows-

- good suffix shift or matching shift : It aligns only matching pattern characters against target characters already successfully matched.
- bad character shift or occurrence shift :It avoids repeating unsuccessful comparisons against a target character.

Algorithm :

```
fullSuffixMatch(shiftArray, borderArray, pattern)
Begin
  n := pattern length
  j := n
  j := n+1
  borderArray[i] := j

  while i > 0, do
    while j <= n AND pattern[i-1] ≠ pattern[j-1], do
      if shiftArray[j] = 0, then
        shiftArray[j] := j-i;
      j := borderArray[j];
    done

    decrease i and j by 1
    borderArray[i] := j
  done
End
 partialSuffixMatch(shiftArray, borderArray, pattern)
Begin
```

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 6, Issue 10, October 2019

```
  n := pattern length
  j := borderArray[0]

  for index of all characters 'i' of pattern, do
    if shiftArray[i] = 0, then
      shiftArray[i] := j
    if i = j then
      j := borderArray[j]
  done
End
searchPattern(text, pattern)
Begin
  patLen := pattern length
  strLen := text size

  for all entries of shiftArray, do
    set all entries to 0
  done

  call fullSuffixMatch(shiftArray, borderArray, pattern)
  call partialSuffixMatch(shiftArray, borderArray,
pattern)
  shift := 0

  while shift <= (strLen - patLen), do
    j := patLen -1
    while j >= 0 and pattern[j] = text[shift + j], do
      decrease j by 1
    done

    if j < 0, then
      print the shift as, there is a match
      shift := shift + shiftArray[0]
    else
      shift := shift + shiftArray[j+1]
  done
End
```

IV. TECHNIQUES USED BY ALGORITHMS

In this section we will see what techniques above algorithms are using -

ALGORITHMS	*TECHNIQUES*
Naive string search algorithm	Each character of the pattern is compared to a substring of the text which is the length of the pattern, until there is a match or a mismatch
Rabin Karp String Search Algorithm	Hashing
Knuth–Morris–Pratt algorithm	Two indices l and r into text t
Boyer–Moore string search algorithm	Uses good suffix shift and bad character shift

V. COMPEXITY ANALYSIS OF ALGORITHMS

In this section we will analyze the time complexity of preprocessing and matching as well as the space complexity of the string matching algorithms.

ALGORITHMS	TIME COMPLEXITY		SPACE
	PRE-PROCESSING	MATCHING	COMPLEXITY
Naive string search algorithm	0 (none)	O(nm)	O(1)
Rabin Karp String Search Algorithm	O(m)	avg O(n + m) worst O(n · m)	O(m)
Knuth–Morris–Pratt algorithm	O(m)	O(n)	O(m)
Boyer–Moore string search algorithm	O(m + |Σ|)	Ω(n/m), O(n)	O(m+|Σ|)

VI. CONCLUSION

Internet is a very impotant part of our lives .Today life without internet can't be imagined and we spend lots of our time in the intenet searching ,from searching videos on internet , searching your favourite series on Netflix to searching any required information on google . Searching is the first step we do on internet. So for efficient and fast searching we need good pattern matching algorithms.In our paper we have taken four algorithm and we have came to conclusion that according to preprocessing time complexity Boyre-Moore string search algorithm is the most efficient and according to matching time complexity Knuth-Morris-Pratt algorithm is the most efficient..

REFERENCES

[1] Rahul B. Diwate and . Satish J. Alaspurkar, "On Study of Different Algorithms for Pattern Matching", International Journal of Advanced Research in Computer Science and Software Engineering 3(3), March - 2013, pp. 615-620.

[2] Ananthi Sheshasayee and G. Thailambal, A comapartive analysis of single pattern matching algorithm in text mining , 2015 International Conference on Green Computing and Internet of Things (ICGCIoT),IEEE.

[3] Koloud Al-Khamaiseh and Shadi ALShagarin, "A Survey of String Matching Algorithms", Koloud Al-Khamaiseh Int. Journal of Engineering Research and Applications, ISSN : 2248-9622, Vol. 4, Issue 7(Version 2), July 2014, pp.144-156.

[4] A.A.PUNTAMBEKAR,Design Of Analysis Of Algorithm,R15 edition 2017,Techmical Publication.