CS575: Final Project Report

Project Title: Implementation and comparison of String-Matching Algorithms

Team Member: Dua Jasmeet Kaur

I. Problem

String matching algorithms are a set of algorithms that finds a pattern within a larger string or text. Text can be anything like the human language alphabet, a binary alphabet or a DNA alphabet in bioinformatics. Purpose of the project is to discover a fitting string from the immense amount of data available by using the perception and understanding of numerous algorithms on String-Matching concepts.

II. Algorithms

String Matching has been an essential study area in this century of a growing technology era. It has been playing a significant role not only in computational area but other fields of science too. The basic concept is to discover the match of a pattern in an input string or a text.
To make this happen we will start with the basic Naïve algorithm and then we moved to the Rabin Karp algorithm and at last with Knuth-Morris-Pratt (KMP) algorithm to find the match.

A. Naïve Algorithm

B. Rabin Karp Algorithm

C. Knuth-Morris-Pratt (KMP) Algorithm

II.A. Naïve Algorithm [1]

Naïve Algorithm uses two loops to compare each value within a text to every other value in the pattern until it finds a match.

Pseudo Code:

```
Naïve_String_Matching(text, pattern)
  1. t← length [text]
  2. p← length [pattern]
  3. for s ← 0 to t -p
  4. do if pattern [1.....p] = text [s + 1....s + p]
  5. then print "pattern find on index", s
```

II. B. Rabin Karp Algorithm [2][3]

Rabin Karp Algorithm use hashing to match the pattern with the text. It generates the hash value of the pattern and then compare it with the hash value of the string, if the hash value matches then only it will compare all the characters of that string.

This algorithm also reduces the time complexity by comparing the two strings based on their hash values, and once the hash value matches it will check the individual character in it.

Pseudo Code:

```
Rabin_Karp (text, pattern)
  1. t← length [text]
  2. p← length [pattern]
  3. hp ← hash value of pattern
  4. ht ← hash value of text till index (t-p)
  5. for i← 1 to (t-p)
  6. if hp = ht
  7. then if pattern [1.....p] = T [i+1.....i + p]
  8. then print "pattern find on index", i
  9. else
  10. update the ht for the next window
  11.compare again with the new hash value
```

II. C. Knuth-Morris-Pratt (KMP) Algorithm [4][5]

Knuth Morris gives us the linear time complexity for string matching algorithms. Basically, for the given text Knuth Morris check the characters from left to right, it works on the lps (longest proper prefix which is also a suffix) to skip some comparisons by using the previous comparisons results.

Pseudo Code:

```
LPS(text)
    1. i ← 1
    2. j ← 0
    3. f(0) ← 0
    4. while i < p do
    5. if text[j] = text[i]
    6. f(i) ← j +1
    7. i ← i +1
    8. j← j + 1
    9. else if
```

```
10.  j ← f(j - 1)
11.  else
12.  f(i) ← 0
13.  i ← i +1
```

```
Knuth_Morris_Pratt(text,pattern)
    1. t ← length [text]
    2. p ← length [pattern]
    3. f ← compute LPS function of pattern
    4. i ← 0
    5. j ← 0
    6. while i < t do
    7. if j ← p-1 then
    8.  return i- p+1  then print "pattern find on index", s
    9. i ← i +1
    10.  j ← j +1
    11.  else if j > 0
    12.   j ← f(j -1)
    13.  else
    14.  i ← i +1
```

III. Software Design and Implementation

III.A. Software Design

All the algorithms are designed in such a way that they can be used to solve various string-matching problems. There are multiple real-time string-matching tools which are based on the same concept as above. The most common application is 'plagiarism check' which can be used by the universities to detect similar text, code symbols or can also detect the similar pattern/ structure of the document.

III.B. Implementation and Tools Used [7]

I used Java language for implementing the algorithms. All the algorithms are implemented and tested on IntelliJ tool, which contains all the supporting libraries of Java latest version to successfully get the required output based on the inputs.

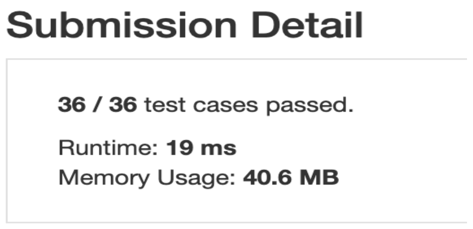
Below is the explanation of the problem I implemented to understand the functionality and analyze the time complexity of String-matching algorithms.

Given a string s and a non-empty string p, find all the start indices of p's anagrams in s.
Strings consists of lowercase English letters only and the length of both strings s and p will not be larger than 20,100.
The order of output does not matter.

- I implemented the solution of the problem in below ways as follows
  1. By using Rabin Karp algorithm: which basically use the hashing to solve the string matching.
  2. By using Knuth-Morris-Pratt: This algorithm uses the concept of the LPS and sliding window to find string-matching.

Below is the analysis of the above problem in Leetcode.

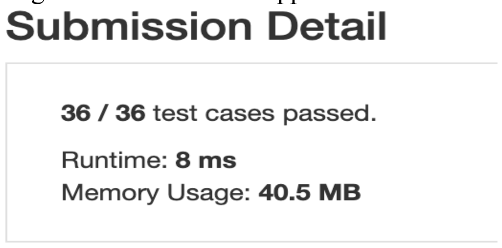
1. Using Rabin Karp approach:



Submission Detail

36 / 36 test cases passed.
Runtime: 19 ms
Memory Usage: 40.6 MB

2. Using Knuth-Morris-Pratt approach:



Submission Detail

36 / 36 test cases passed.
Runtime: 8 ms
Memory Usage: 40.5 MB

We can see how the run time is impacted while using the different approach on the same problem. Since KMP has better time complexity than Rabin Karp, it is shown clearly in the analysis above also that KMP take lass time to execute the problem with same input. Below is the time complexity comparison.

III.C. Performance Evaluation [6]

Algorithms	Complexity
Naive Technique	O(mn)
Rabin Karp Algorithm	Average O(n+m) worst case O(mn)
Knuth-Morris-Pratt Algorithm	O(n)

Attachments

https://github.com/Jasmeet10/String_Matching-Project

References

[1] . “Naive algorithm for Pattern Searching,” GeeksforGeeks, 27-Aug-2019. [Online]. Available: https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/. [Accessed: 23-Feb-2020]

[2] . “Rabin–Karp algorithm,” Wikipedia, 17-Dec-2019. [Online]. Available: https://en.wikipedia.org/wiki/Rabin–Karp_algorithm. [Accessed: 23-Apr-2020]

[3] . “Rabin-Karp Algorithm for Pattern Searching,” GeeksforGeeks, 23-Aug-2019. [Online]. Available: https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/. [Accessed: 29-Mar-2020]

[4] . “Knuth–Morris–Pratt algorithm,” Wikipedia, 23-Apr-2020. [Online]. Available: https://en.wikipedia.org/wiki/Knuth–Morris–Pratt_algorithm. [Accessed: 02-Apr-2020]

[5] . “KMP Algorithm for Pattern Searching,” GeeksforGeeks, 20-May-2019. [Online]. Available: https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/. [Accessed: 02-Apr-2020]

[6] . G. Gimel’farb “String Matching Algorithms” The University of Auckland - compsci369s1c - Lectures. [Online]. Available: https://www.cs.auckland.ac.nz/courses/compsci369s1c/lectures/GG-notes/CS369-StringAlgs.pdf. [Accessed: 19-Apr-2020]

[7] . Find all anagrams in a string. [Online]. Available: https://leetcode.com/problems/find-all-anagrams-in-a-string/. [Accessed: 21-Apr-2020]