String Search II:

Boyer-Moore and Knuth Morris-Pratt

Aidan Cox, Emily Lavoie, Nicholas Mendes, and Bilguuntur Munkhzaya

College of Arts and Sciences

University of Rhode Island, Kingston, RI 02881

E-mail(s): aidan_cox@uri.edu, emily_lavoie@uri.edu, nicholas_mendes@uri.edu, bilguuntur20@uri.edu

Abstract—The world of computing is frequently utilized with the hopes of lightening the difficulty, or processing time of a task, in comparison to human performance. String matching is a great example among many other processes in which digital computation and algorithms can aid in finding a more efficient path towards results. Two particular algorithms—researched and implemented—in this specific study are Boyer-Moore and Knuth-Morris-Pratt. The first algorithm is often regarded as string matching algorithm standard, in all ways benchmarking, and analysis. The second, being a historically significant example in the string searching world.

I. INTRODUCTION:

String searching is an incredibly vital part of computer science in general as its uses can be found in nearly all fields pertaining to computation. The goal for string searching, frequently colloquialized as string matching, is to find instances of a pattern, P, found in a text, T, and how many of said instances are present. String searching algorithms are most commonly found in database schema, network systems, and bioinformatics [1]. The two specific algorithms in question are Boyer-Moore, and Knuth-Morris-Pratt.

Discovered in 1977, Boyer-Moore was developed by Robert S. Boyer, and J. Strother Moore. The algorithm is very efficient, and is often found to be used in string searching involving literature [2]. Regarded as a must-learn for computing as a whole, Boyer-Moore is a considerably fast algorithm that is widely discussed, taught, and implemented [3]. Alongside Boyer-Moore is the Knuth-Morris-Pratt algorithm, often seen as standard, more so in an educational sense than practical. In its conceptual early stages, this algorithm pre-dating Boyer-Moore, as its

first semblance can be traced back to 1970. This initial state of the algorithm was found by Donald Knuth, and Vaughan Pratt. The algorithm finally came to full realization in 1977 however, as James H. Morris furthered its development, separate from Donald Knuth, and Vaughan Pratt, thus gaining its full name of Knuth-Morris-Pratt [4]. This algorithm in specific is cited to be the first ever linear-time performing string searching algorithm[4]. Our group's objective is to detail the intricacies of both Boyer-Moore and Knuth-Morris-Pratt algorithms, in the interest of delivering a concise yet substantive, general yet clear understanding of said algorithms.

II. RESEARCH:

After figuring out what days would be best for each member of the group to meet, we as a unit began to move together with the first objective being to identify what we were exactly working with. After a short search on the internet, we were able to get the overall gist of both algorithms. Due to their apparent popularity in the coding community, sources that covered these particular algorithms were plentiful. Many sources allowed us to find comparison data among a healthy amount of other string matching algorithms, which helped us get a further grasp on the performance aspect of both Boyer-Moore and Knuth-Morris-Pratt. A considerable amount of time and effort was allocated to solely doing research for the algorithms purely, concerning history and how they work. An even larger period of time was spent on Graphical User Interface implementation and how it could be practiced with the C++ programming language specifically. The remainder of the research time was spent on benchmarking and performance as a whole.

III. UNDERSTANDING THE ALGORITHMS

The Boyer-Moore string searching algorithm makes use of two key rules, otherwise known as heuristics in broader algorithm studies. These two heuristics are labeled as the Bad Character Rule, and the Good suffix rule [3]. Through the use of these two joint overarching rules, Boyer-Moore is able to perform under exemplary times for most usual cases. Boyer-Moore is a right to left scanning algorithm, and largely focuses on finding repeating patterns present in both the given search pattern and the pool of alphabet provided in the text. The first heuristic is the Bad Character Rule, which simply states that upon comparison, when a character in the text mismatches with the character in the pattern, then said character is labeled as a bad character. Once this occurs, the algorithm shits the location of the pattern until it fulfills one of two different conditions. One, if the mismatched comparison is finally found to be a match. Two, if the pattern of the search shifts entirely past the initially mismatched character. The second principle is the Good suffix rule. For simplicity, t, will stand in for a substring of the text that matches a

substring of the pattern. In this case, t will be shifted along the text until a separate instance of t is found to match in the text, and/or a prefix of the pattern is found to match the beginning of t, and/or the pattern moves entirely past the t. With each iteration of these two rules, the algorithm moves along the text in a jumping fashion from one similar pattern to another, until all fully matching patterns are found.

Knuth-Morris-Pratt on the other hand, scans the text from left to right, in a very simple manner. Unlike the brute force method, KMP skips a word entirely if a sufficient portion of the pattern mismatches the current word within the text. This sufficient portion aspect is achieved through the use of a prefix function. The prefix function keeps track of previous comparisons in order to determine a substring of the pattern that is dedicated as the threshold for further comparisons. Through this simple and elegant method, Knuth-Morris-Pratt is able to bring down most performance time cases from the baseline of the brute force method. In specificity, the worst-case is brought down from quadratic time to linear time.

IV. IMPLEMENTATION:

String search algorithms are rather simple to implement, as in most cases, they do not require extensive libraries or data structures. In addition, pseudo-code and source-code for both Boyer-Moore and Knuth-Morris-Pratt can be found abundantly across many sources. The pseudo-code for Boyer-Moore is detailed as follows:

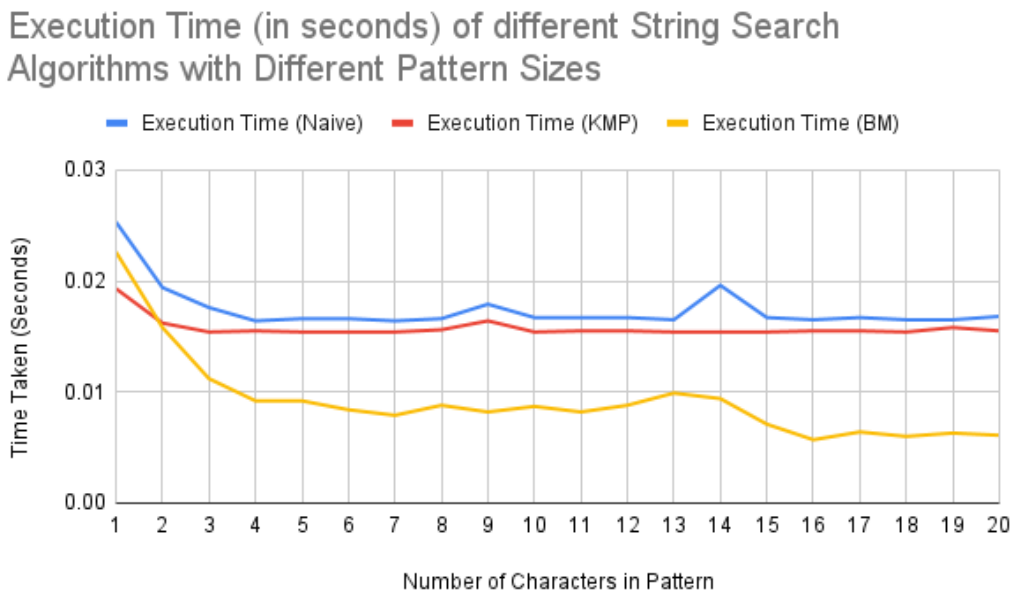
```
      While the end of the text has not been reached
           While the characters in the pattern and text match
                compare the characters in the pattern and text at a certain
alignment from right to left
          If a match has been found
               Save the index as being found
               Move to next alignment
          else if a mismatch has been found
      Check to see if the mismatched character in the text exists elsewhere in
the in the pattern
      Check to see if the characters that have already been matched exist
elsewhere in the pattern

      Calculate the number of alignments which would be skipped by either
                method and apply the method which skips over the most shifts
```

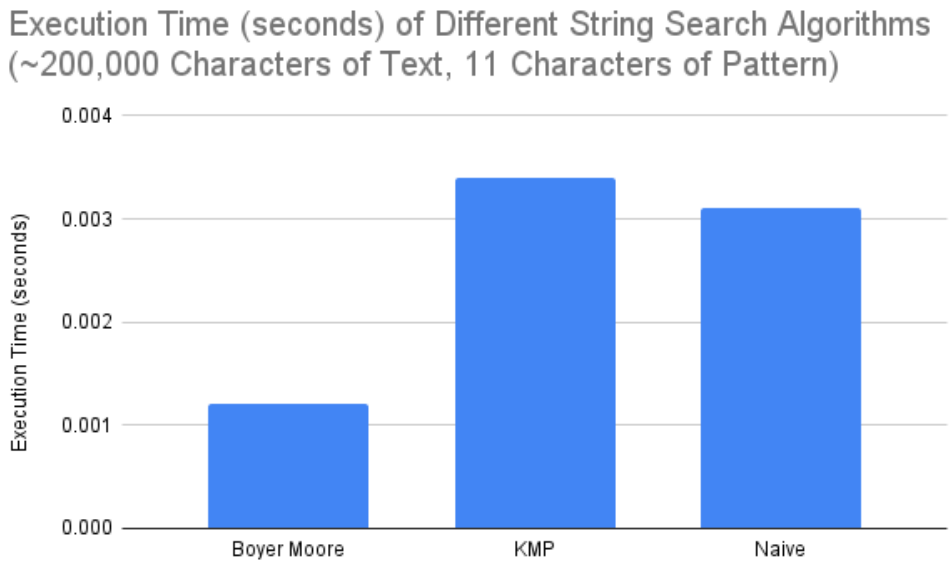
The pseudo code for Knuth-Morris-Pratt is as follows:

```
n = size of text
m = size of pattern
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 there is the pattern
      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
```

When it came to benchmarking, we used multiple different formats of text, and varying lengths of patterns to be searched within said text. The two main types we tested were DNA sequences, as to simulate bioinformatic research, and randomly generated text. The DNA sequence text came to 1,000,000 characters, while the randomly generated text was around 200,000,000 characters in length. Both tests reflected similar outcomes. Visualization of DNA sequence tests:



Randomly generated text tests:



All tests were compared against the Naive/Brute force method in order to create a sense of context. In all cases, both BM, and KMP performed faster than Naive. Best case for Boyer-Moore, for a text of length n and a fixed pattern of length m, is O(n/m) [3].While the worst case performs at O(n) time, regardless of whether the text contains a match or not [3]. The best performing instances of the Knuth-Morris-Pratt algorithm came to O(n) time. Following suite, the worst case performance time came to O(n) as well. When the range of unique characters used increases, the slower the algorithm performs as more chances of a mismatch occurring arise.Since the weaknesses of the KMP algorithm is a wide range of characters used, DNA sequence matching greatly benefits from KMP. Due to the compared characters being boiled down to A, C, T, and G, KMP's performance stays optimal.

After source code for both algorithms were done through their own respective .cpp files, bench-marking was an aspect which needed to be implemented. This part was easily handled as our CSC-212 lab ii showcased benchmarking in c++. The main concern in terms of implementation stemmed from the Graphical User Interface. Aidan Cox handled all the various GUI implementations of the project. The initial implementation was able to achieve features including, showing the program usage, parsing the command line args, validating the input, calling into our search algorithms, and ultimately showing the results to the user. The consecutive implementations followed with a native native windows GUI, which used an IDE from RAD Studio called C++ Builder. Through this IDE, we were able to include and call directly into our search algorithms to perform the search and show the results. Another

implementation was through a web based GUI. After research, it was found that Python and HTML/JS were great avenues to go through for GUI work in C++. Aidan's skills in both Python and HTML/JS helped to create a more modern-looking GUI running in a browser. Using a simple python-based web server, we could handle browser requests for the home page or perform a search. For search requests, Aidan's python script would get the data sent from the user and then call directly into the search.exe and wait for the results. The python script would then send the results to the browser.

V. CONTRIBUTIONS:

Team organization switched back and forth from dividing the total work for both algorithms into two units, and content based division on a per-person basis. At first we wanted to have Aidan and Nicholas tackle Boyer-Moore, while Emily and Bilguuntur handled Knuth-Morris-Pratt. This then evolved into each person handling a different aspect of the project, which became the ultimate decision to our workflow. For specifics, Aidan Cox implemented the main CLI, the Windows GUI application, Python/HTML/Js Web application, with the additional Github maintenance. Nicholas Mended handled both C++ implementations of Knuth-Morris-Pratt and Boyer-Moore, and their respective benchmarking implementations. Emily Lavoie was in charge of the presentation and its visualizations, with the use of GIFs examples. Bilguuntur Munkhzaya, in the initial steps, followed research into the KMP algorithm specifically, then led into the project's report, its organization, alongside the sources used in the report.

References

[1] “Applications of string matching algorithms,” GeeksforGeeks, 03-Sep-2020. [Online]. Available: https://www.geeksforgeeks.org/applications-of-string-matching-algorithms/.

[2] “Boyer Moore algorithm for Pattern Searching,” GeeksforGeeks, 08-Dec-2021. [Online]. Available: https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/.

[3] S. Oddiraju, “Boyer-Moore - Indiana state university,” BOYER-MOORE. [Online]. Available: http://cs.indstate.edu/~soddiraju/abstract.pdf.

[4] A. Chattopadhyay, M. Jalteh, and J. Khim, “Knuth-Morris-Pratt algorithm,” Brilliant Math & Science Wiki. [Online]. Available: https://brilliant.org/wiki/knuth-morris-pratt-algorithm/.