

MY JOURNEY – String Matching Algorithms Homework

During this project, I had the opportunity to examine string matching algorithms from both theoretical and practical perspectives. At the start of the project, the Naive, KMP, and Rabin-Karp algorithms were provided as standard. My primary responsibility was to implement the Boyer-Moore algorithm and, in addition, develop a pre-analysis mechanism that would select the fastest algorithm based on the given text and pattern.

I used VS Code to develop the project and continuously compiled and ran the code for testing. Specifically, using the ManualTest class to run all algorithms on the same test cases and compare their processing times allowed me to clearly see the performance differences between the algorithms.

WHAT DID I DO?

In this project, I first implemented the Boyer-Moore algorithm. While writing the algorithm, I focused particularly on the "Bad Character Rule." I created a preprocessing stage for the pattern, preparing a table that keeps track of the last position of each character within the pattern. During matching, I performed comparisons from right to left of the pattern and, in case of a mismatch, used this table to apply an appropriate shift. This avoided unnecessary character comparisons.

After ensuring the Boyer-Moore algorithm produced correct results in all test cases, I implemented the StudentPreAnalysis class. The goal here was to predict which of the Naive, KMP, Rabin-Karp, and Boyer-Moore algorithms would run faster by examining the properties of the text and pattern. For this, I used criteria such as pattern length, the repeating nature of the pattern, alphabet size, and text length.

WHERE DID I FACE DIFFICULTIES?

The most challenging part of the project was correctly establishing the shift logic of the Boyer-Moore algorithm. Small errors, particularly in index calculations, could cause the algorithm to produce incorrect results or enter an infinite loop. The correct alignment of pattern and text indexes required careful attention to avoid "off-by-one" errors.

Another challenging aspect was the pre-analysis. In theory, it was easy to know which algorithm should be faster, but in practice, unexpected results in some tests made decision-making difficult. This demonstrated that algorithm selection is directly related not only to theoretical complexity but also to the structure of the input data.

HOW DID I SOLVE IT?

To solve these problems, I simulated the Boyer-Moore algorithm step-by-step on small examples. I corrected errors by individually checking each shift made on the pattern and text. I also revised the pre-analysis strategy several times. I redesigned the strategy so that Naive would be selected for short patterns, KMP for repeating patterns, and Boyer-Moore for long patterns and large alphabets. By examining the time tables in the ManualTest outputs, I analyzed which choices made sense.

RESEARCH AND RESOURCES USED

During this project, I utilized the following resources:

- Lecture notes on string matching algorithms provided in the course
- General explanations of Boyer–Moore, KMP, and Rabin–Karp algorithms (Wikipedia and similar open-source documentation)
- Online algorithm explanations and examples
- While using these resources, I focused on understanding the logic of the algorithms rather than directly copying code.

USE OF LLM (ChatGPT)

During this project, I used an LLM tool like ChatGPT. I used ChatGPT not as a solution-generating tool, but for learning and guidance purposes. Specifically:

- To better understand the logic of the Boyer-Moore algorithm's bad character rule
- To check for logical errors in my code
- To discuss which criteria might be meaningful in the pre-analysis section
- To learn how to run the project correctly in Windows and VS Code environments

I asked questions like "How does the Boyer-Moore bad character rule work?" and "Which algorithm is advantageous if the pattern repeats?" on ChatGPT and adapted the explanations I received to my own code. The process of writing, testing, and validating the final code is entirely my own.

GENERAL EVALUATION

Thanks to this assignment, I have gained a much better understanding of the practical behavior of string matching algorithms. In particular, I observed that the Naive algorithm can be faster than expected with short patterns, how stable the Key Matching Rule is with repeating patterns, and how powerful Boyer-Moore is under appropriate conditions. The pre-analysis section was quite instructive in understanding how much the choice of algorithm depends on the input data.

Github Url: <https://github.com/merttozkann/StringMatching>

Mert Özkan

22050111019