Implement the Naive String Matching Algorithm and Rabin-Karp Algorithm for string matching.

Design and Analysis of Algorithms Mini Project (LP-III)

Submitted by

Aranav Mahalpure Roll No.BA63

Ashish Chandhekar Roll No. BA67

Aayush Bhalavi Roll No.BA26

Vansh Kanakia Roll No.BA52

FINAL YEAR COMPUTER ENGINEERING



**Department of Computer Engineering**

**International Institute of Information Technology Hinjawadi, Pune – 411057**

**SEMESTER I (AY 2024-25)**

|  |  |  |
| --- | --- | --- |
|  | **TABLE OF CONTENTS** |  |
|  | **TITLE** | **PAGE NO.** |
| **1.** | ABSTRACT | **1** |
| **2.** | INTRODUCTION | **2** |
| 2.1 SCOPE | | **3** |
| 2.2 REQUIREMENT ANALYSIS | | **3** |
| 2.3 SOFTWARE AND HARDWARE DETAILS | | **5** |
| **3.** | Example Run | **6** |
| **4.** | OUTPUT | **8** |
| **5.** | CONCLUSION | **10** |

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

**1. Abstract**

String matching is an essential operation in computer science, with applications ranging from search engines and data mining to pattern recognition and bioinformatics. In this project, we explore two fundamental string matching algorithms—the Naive String Matching Algorithm and the Rabin-Karp Algorithm—both of which solve the problem of finding occurrences of a pattern string within a larger text. This project integrates these algorithms into a user-friendly web interface, built using HTML, CSS, Flask, and Python, allowing users to test and compare their performance on the same input. By running both algorithms simultaneously, we analyze their computational efficiency, accuracy, and scalability in real-world applications.

*Department of Computer Engineering 1*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

**2. Introduction**

String matching is a core function in numerous software applications where it is necessary to search for occurrences of a pattern within a larger text. Efficient string matching algorithms are crucial in areas such as:

**Search Engines:** To retrieve relevant documents based on user queries.

**Bioinformatics:** For identifying genetic sequences in DNA analysis.

**Data Mining:** Where large volumes of text data are processed to discover patterns. In this project, we implement and compare two string matching algorithms:

**Naive String Matching Algorithm:** This straightforward approach systematically checks every position in the text for a match with the pattern. Though simple to understand and implement, it can be inefficient for large inputs due to its brute-force nature, having a time complexity of O(mn), where m is the length of the pattern and n is the length of the text.

**Rabin-Karp Algorithm:** This algorithm improves efficiency by using hashing to quickly compare substrings in the text to the pattern. By converting the text and pattern into hash values, potential matches can be found more quickly, making this algorithm more suitable for larger texts or when multiple patterns are being searched.

Both algorithms are integrated into a web-based interface to allow users to compare their performance and behavior under identical input conditions.

*Department of Computer Engineering 2*

*1*

**2.1. SCOPE**

The Naive String Matching Algorithm and Rabin-Karp Algorithm are widely studied due to their practical significance in a variety of computational problems. This project aims to

Implement both algorithms using Python.

Compare their performance using identical input conditions (same text and pattern).

Provide an interactive user interface where users can test the algorithms in real-time.

Demonstrate the strengths and limitations of each algorithm, with a particular focus on:

Time Complexity: Understanding how both algorithms scale with increasing input sizes.

Efficiency: How the use of hashing in Rabin-Karp can improve performance over Naive string matching.

Practical Applications: Real-world scenarios where either algorithm may be preferable.

The system is designed to be flexible and scalable, allowing further modifications and integration with other advanced algorithms such as the Knuth-Morris-Pratt or Boyer-Moore algorithms.

**2.2. REQUIREMENT ANALYSIS:**

**Functional Requirements:**

**1.Textand Pattern Input:**

Users will input both a "text" string and a "pattern" string to search within the text. The text can be of any length, while the pattern will typically be smaller, as is the case in real-world applications like document retrieval or genetic sequence analysis.

**2.Algorithm Execution:**

Users can select either the Naive String Matching Algorithm or the Rabin-Karp Algorithm to execute. Additionally, both algorithms can be executed

*Department of Computer Engineering 3*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

simultaneously for comparative analysis. The system will output the matching positions and the execution time for each algorithm.

**3.Result Display:**

Results will be presented in an easy-to-read format, showing where the pattern occurs in the text (if at all), along with the execution time of each algorithm. This helps users understand the trade-offs between algorithmic complexity and performance.

**Non-Functional Requirements:**

**1.Performance:**

Both algorithms should execute within a reasonable time frame, even for moderately large inputs. The system should handle texts of up to several thousand characters without significant delay.

**2.Usability:**

The interface is designed to be simple and intuitive, enabling users to easily input text and pattern data, select algorithms, and view results.

**3.Scalability:**

The system is built to handle multiple concurrent users and can be scaled for larger datasets or more complex pattern matching tasks.

**4.Security:**

The system will handle user inputs securely, avoiding potential threats like input injection. All user data will be processed safely on the server, with no personal information retained.

*Department of Computer Engineering 4*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

**2.3. SOFTWARE AND HARDWARE REQUIREMENTS:**

**Software Requirements:**

**Operating System:**

Compatible with major operating systems such as Windows, macOS, and Linux (Ubuntu, CentOS).

**Programming Languages:**

Python (for algorithm implementation and Flask backend development).

**Framework:**

Flask (for the web application backend).

**Development Tools:**

IDE: Visual Studio Code or PyCharm.

Browsers: Latest versions of Chrome, Firefox, Safari, Edge (for cross-browser compatibility).

**Processor:**

Intel i5 or higher (recommended multi-core for faster performance).

**Memory:**

8 GB minimum (16 GB recommended for development purposes).

*Department of Computer Engineering 5*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

**3. Example Run:**

**Example 1: Naive String Matching Algorithm**

**Input:**

Text: "ABCABCDABABCD"

Pattern: "ABCD"

The algorithm checks each possible substring in the text:

Compare "ABCA" with "ABCD" → no match.

Compare "BCAB" with "ABCD" → no match.

Compare "CABC" with "ABCD" → no match.

Compare "ABCD" with "ABCD" → match found at position 4.

Continue searching, no further matches found.

**Result:**

Match found at index 4.

**Time Complexity: O(mn)**

Example 2: Rabin-Karp Algorithm

**Input:**

Text: "ABCABCDABABCD"

**Pattern: "ABCD"**

*Department of Computer Engineering 6*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

The algorithm computes the hash value of "ABCD" and compares it with the hash values of substrings in the text:

Hash of "ABCA" and "ABCD" → no match.

Hash of "BCAB" and "ABCD" → no match.

Hash of "CABC" and "ABCD" → no match.

Hash of "ABCD" and "ABCD" → match found at position 4.

Continue searching, no further matches found.

**Result:**

Match found at index 4.

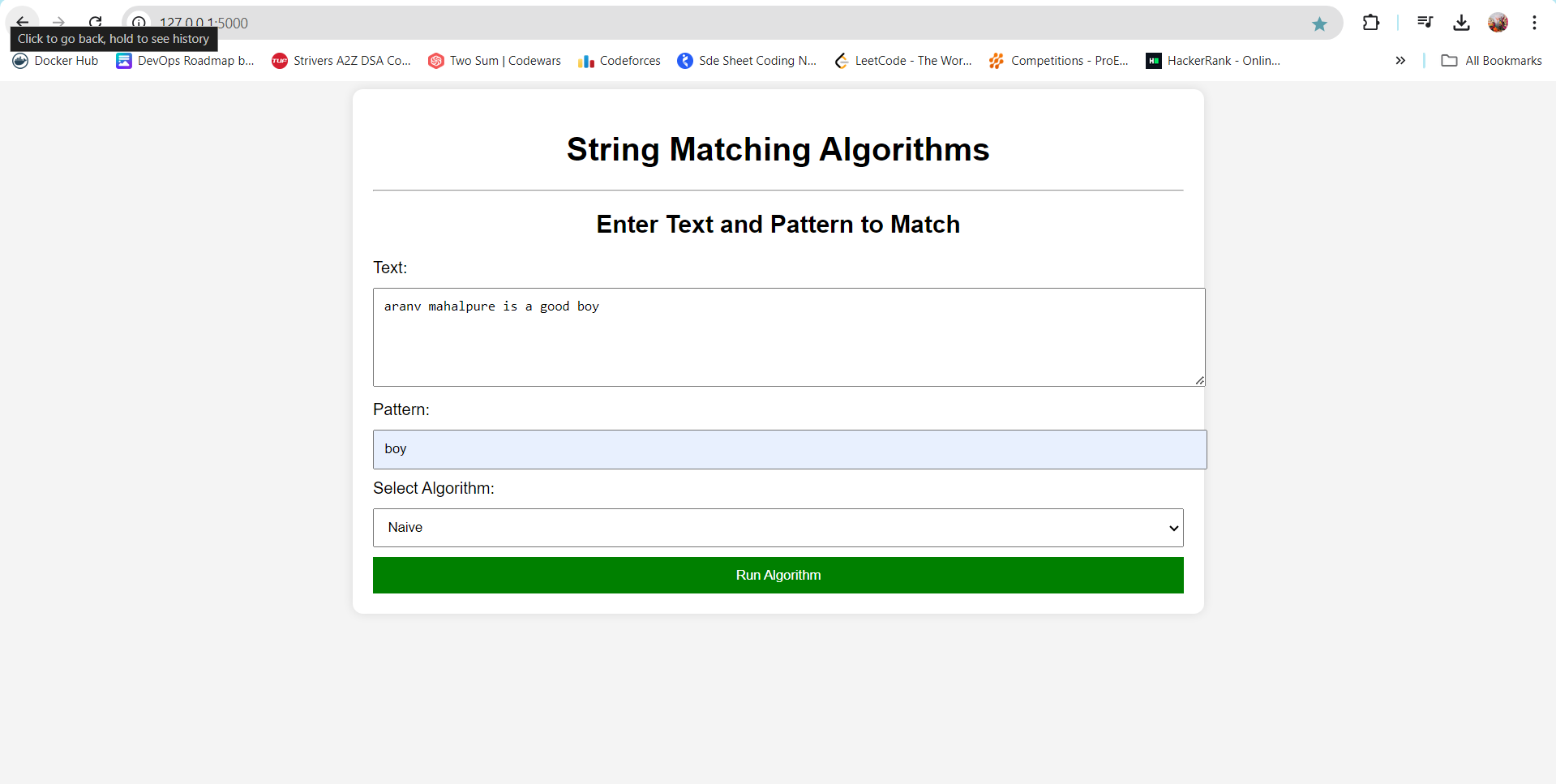
Time Complexity: O(n + m) on average due to hashing.

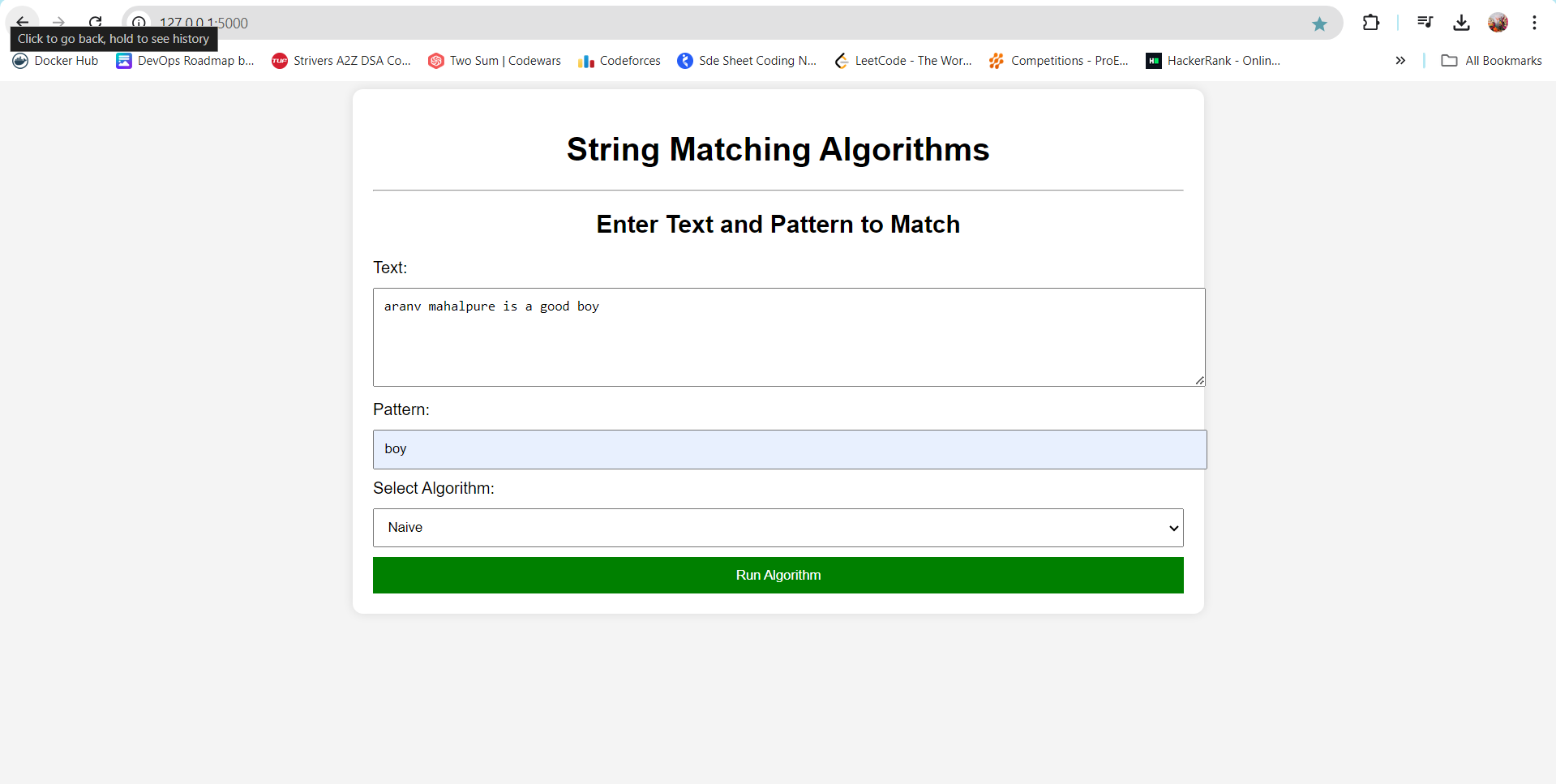
*Department of Computer Engineering 7*

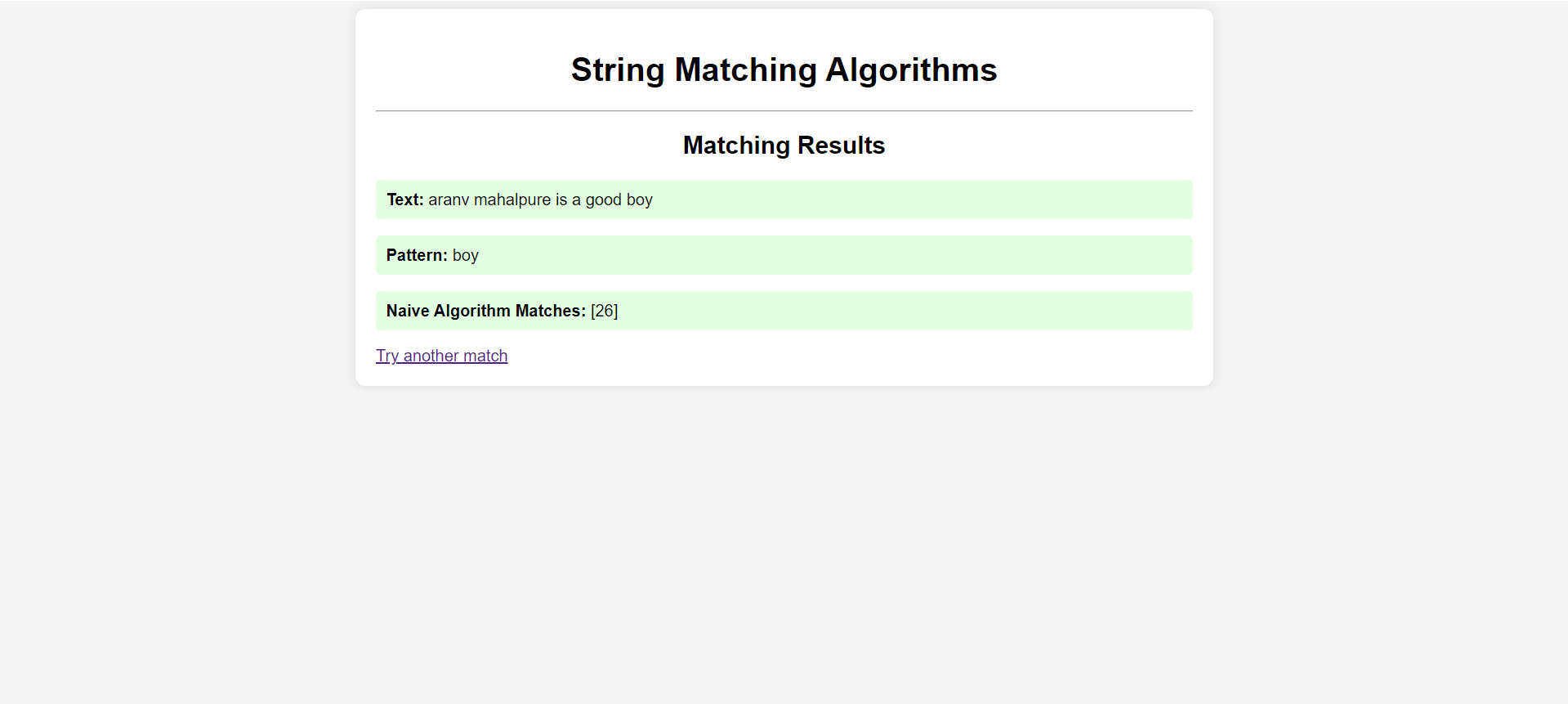
*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

**4. Output**



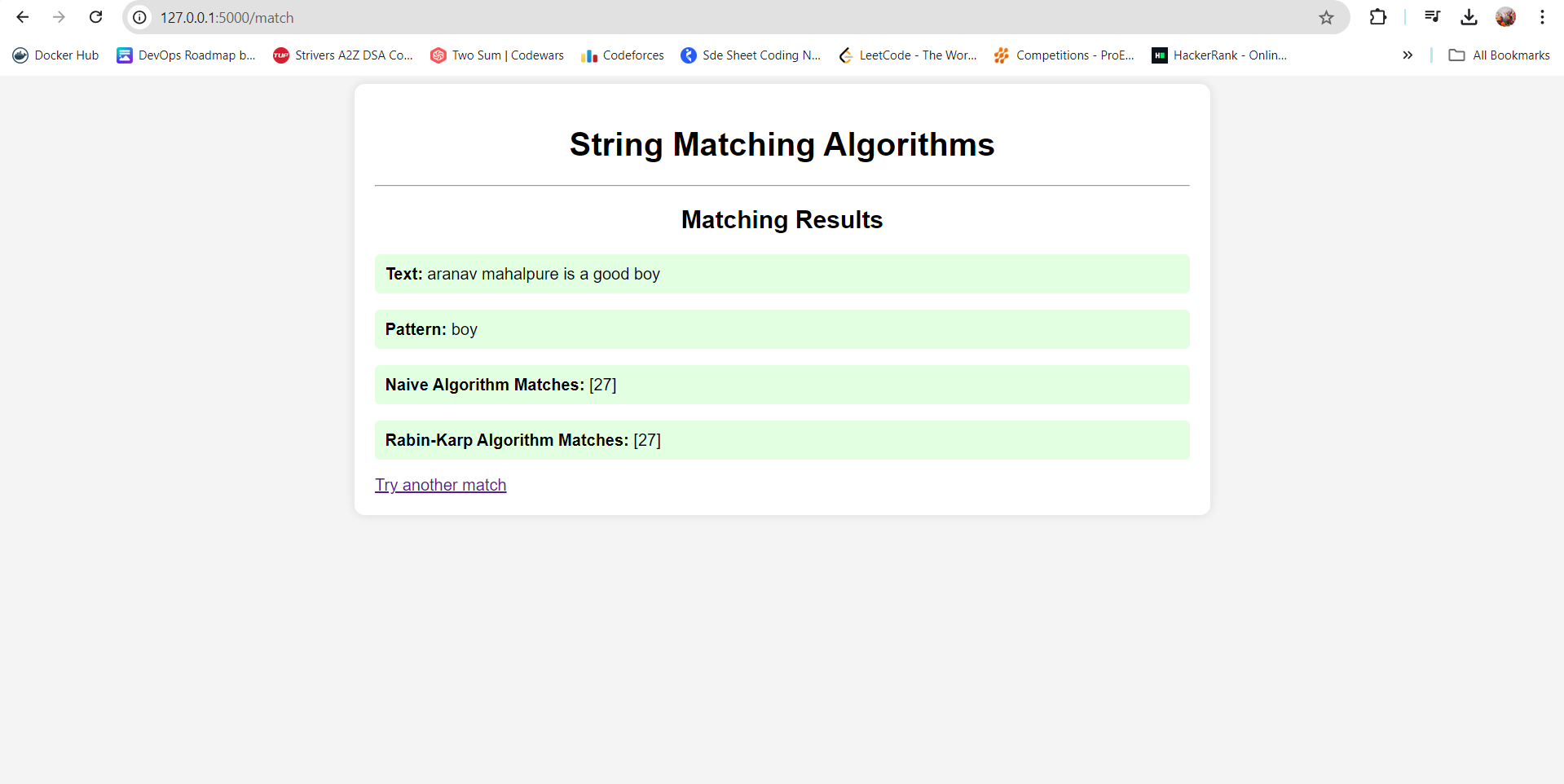




*Department of Computer Engineering 8*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**



*Department of Computer Engineering 9*

*1*

**Implement the Naive string matching algorithm and Rabin-Karp algorithm for string matching.**

**5. Conclusion**

This project successfully implements both the Naive string matching algorithm and the Rabin-Karp algorithm for string matching. While the Naive approach is simple, it can be inefficient for large texts. The Rabin-Karp algorithm, through the use of hashing, significantly improves performance for larger inputs or multiple pattern searches.

The project provides an interactive way to understand these algorithms through a user-friendly web interface, making it easier for users to experiment with different inputs and observe how the algorithms behave. In the future, this project could be extended to include more advanced algorithms such as Knuth-Morris-Pratt (KMP) or Boyer-Moore, allowing for even greater efficiency in string matching tasks.

*Department of Computer Engineering 10*

*1*