AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय

Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

"DAA Miniproject"

Naive sting matching algorithm and Rabin Karp algorithm for string matching

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF ENGINEERING

In

COMPUTER ENGINEERING

Submitted By

Piyusha Rajendra Supe	23CO315
Iffa Irfan Shaikh	23CO314
Mangalam Sachin Sharma	22CO108

Under the Guidance of

Prof. A.U. Khandait

ALL INDIA SHRI SHIVAJI MEMORIAL SOCIETY'S COLLEGE OF ENGINEERING PUNE-411001

Academic Year: 2025-26 (Term-I)

Savitribai Phule Pune University

AISSMS
COLLEGE OF ENGINEERING
ज्ञानम् सकलजनहिताय

Approved by AICTE, New Delhi, Recognized by Government of Maharashtra
Affiliated to Savitribai Phule Pune University and recognized 2(f) and 12(B) by UGC
(Id.No. PU/PN/Engg./093 (1992)
Accredited by NAAC with "A+" Grade | NBA - 7 UG Programmes

Department of Computer Engineering

CERTIFICATE

This is to certify that **Piyusha Rajendra Supe** from **Fourth Year Computer Engineering** has successfully completed her work titled **"DAA Mini-project**" at AISSMS College of Engineering, Pune in the partial fulfillment of the Bachelor's Degree in Computer Engineering.

Prof. A. U. Khandait	**Dr. S. V. Athawale**	**Dr. D. S. Bormane**
(Faculty In-charge)	(Head of Department)	(Principal)
Computer Engineering	Computer Engineering	AISSMSCOE, Pune

ACKNOWLEDGEMENT

It is with profound gratitude and deep respect that I take this opportunity to acknowledge the invaluable support and guidance I have received throughout the course of my BCT project. This journey has been both intellectually enriching and personally fulfilling, significantly enhancing my understanding of tools, technologies, and their real-world application. First and foremost, I would like to express my heartfelt thanks to **Prof. A. U. Khandait.** for his expert guidance, unwavering support, and constructive feedback. His mentorship and encouragement were instrumental in shaping the direction, depth, and quality of this project. I also extend my sincere appreciation to the Head of the Department for their visionary leadership and for fostering an environment that encourages academic growth and innovation. The resources and opportunities provided under their guidance played a crucial role in the successful completion of this project. A special note of thanks goes to the supporting staff, whose timely assistance and cooperation ensured that the process was smooth and efficient. Their commitment and dedication were invaluable during every phase of this project. Finally, I would like to express my deepest gratitude to my parents, whose constant love, patience, and unwavering support have been my source of strength throughout this journey. Their belief in my abilities and their encouragement played a crucial role in keeping me motivated and focused. This project has not only strengthened my technical skills but has also taught me the importance of perseverance and continuous learning. I remain sincerely thankful to everyone who contributed to the successful completion of this project.

Academic Year: 2025-2026

Piyusha Rajendra Supe (23CO315)

TABLE OF CONTENTS

ABSTRACT

This project focuses on the implementation, visualization, and comparative analysis of two fundamental string matching algorithms: Naive String Matching and the Rabin-Karp Algorithm. String matching is a central problem in computer science and has wide-ranging applications in areas such as text processing, DNA sequence analysis, data retrieval, plagiarism detection, and pattern recognition. The main objective of this project is to provide an interactive, educational, and visually intuitive platform that allows users to understand the mechanics, efficiency, and differences between these algorithms. The application is developed using Python and Streamlit, creating a single-page, user-friendly interface where users can input a main text string and a pattern string. The system executes both algorithms step by step and displays each comparison in a structured and readable format. The Naive String Matching algorithm performs a straightforward comparison of the pattern with every possible substring of the main text, making it simple to understand but less efficient for large texts. In contrast, the Rabin-Karp algorithm leverages hashing and a rolling hash mechanism to quickly detect potential matches, significantly reducing unnecessary comparisons and improving performance for large datasets. A distinctive feature of the application is the step-by-step visualization of both algorithms. Each comparison, hash calculation, and pattern match is displayed in collapsible, styled boxes to enhance readability and learning. This approach allows users to track exactly how matches are identified or rejected and to observe the differences in algorithmic behavior in real time. Furthermore, the application includes a comprehensive comparison table summarizing key characteristics of the Naive and Rabin-Karp algorithms, including their approaches, best and worst-case time complexities, space requirements, suitability for different scenarios, and use of hashing techniques.

The project emphasizes educational value by bridging the gap between theoretical concepts and practical execution. By combining interactive input, detailed algorithmic steps, and clear visual comparison, the application serves as a powerful tool for students, researchers, and anyone interested in understanding string matching algorithms. The clean, white-themed design, organized layout, and attractive formatting make the platform visually appealing and easy to navigate, enhancing user engagement and comprehension. Overall, this project demonstrates not only the functionality of string matching algorithms but also the importance of algorithmic efficiency, practical implementation, and user-centric visualization. It provides a scalable, interactive, and informative platform that can be used in academic settings, workshops, or personal learning projects to understand, analyze, and compare the behavior and performance of different string matching approaches in real-time.

INTRODUCTION

String matching is a fundamental problem in computer science and plays a critical role in numerous applications, such as text processing, data retrieval, DNA sequence analysis, plagiarism detection, and cybersecurity. The goal of string matching is to find occurrences of a specific pattern string within a larger text string efficiently and accurately. Understanding how different algorithms solve this problem is essential for optimizing performance in real-world scenarios where large volumes of data need to be processed quickly.

Two commonly used algorithms for string matching are the Naive String Matching Algorithm and the Rabin-Karp Algorithm. The Naive String Matching Algorithm is straightforward and easy to understand, as it compares the pattern with all possible substrings of the main text one by one. Although it guarantees finding all occurrences, its efficiency decreases significantly with longer text strings due to repeated comparisons.

The Rabin-Karp Algorithm improves upon this by employing hashing techniques, particularly the rolling hash method, to compare substrings efficiently. Instead of checking each character individually for every possible substring, it calculates a numeric hash value for the pattern and for each substring of the text. When the hash values match, it performs a character-by-character comparison to confirm the match. This approach reduces the number of unnecessary comparisons and improves performance, especially for large texts or when searching for multiple patterns.

The purpose of this project is to develop an interactive, single-page Streamlit application that not only implements both Naive and Rabin-Karp algorithms but also provides a step-by-step visualization of their execution. The application allows users to input a text and a pattern, view how each algorithm processes the data, and understand their differences in terms of approach, efficiency, and practical applicability. This hands-on approach provides a clear, educational insight into string matching algorithms and demonstrates their relevance in real-world computing scenarios.

REQUIREMENTS

Category	Requirement
Hardware	Modern desktop or laptop with at least 4 GB RAM, 2 GHz processor or higher
Operating System	Windows, macOS, or Linux
Software	Python 3.9 or higher, Streamlit
Libraries	pandas, streamlit
Input	Main text string (T) and pattern string (P)
Output	Step-by-step execution of Naive and Rabin-Karp algorithms, pattern match indices
User Interface	Single-page web interface with collapsible steps and comparison table
Purpose	Educational demonstration of string matching algorithms, comparison of efficiency

METHODOLOGY

The methodology of this project focuses on implementing and analyzing two string matching algorithms: Naive String Matching and Rabin-Karp Algorithm. The project is developed as a single-page interactive Streamlit application, allowing users to input a main text string and a pattern string, observe step-by-step execution, and compare the algorithms' behavior.

1. Naive String Matching Algorithm

The Naive String Matching Algorithm is a simple and straightforward method to find occurrences of a pattern within a text. The algorithm works by sliding the pattern over the text one character at a time and checking for a match at each position.

Steps of the Naive Algorithm:

1. Determine the lengths of the text (n) and the pattern (m).
2. For each possible position i in the text from 0 to n-m:
   - Compare the substring of the text starting at i with the pattern character by character.
   - If all characters match, record the index i as a valid match.
   - If a mismatch occurs, move to the next position in the text.

Advantages:

- Simple to understand and implement.
- Guaranteed to find all occurrences of the pattern.

Limitations:

- Inefficient for long texts or large patterns due to repeated comparisons.
- Time complexity is O((n-m+1)*m) in the worst case.

2. Rabin-Karp Algorithm

The **Rabin-Karp Algorithm** is an efficient string matching algorithm that uses hashing to compare the pattern with substrings of the text. Instead of comparing characters directly for every possible position, the algorithm computes a numeric hash value for the pattern and for each substring of the text of the same length.

Steps of the Rabin-Karp Algorithm:

1. Compute the initial hash value of the pattern and the first substring of the text of length m.
2. Slide the pattern over the text one character at a time:
   - Compare the hash values of the pattern and the current substring.
   - If the hash values match, perform a character-by-character check to confirm the match.
   - Use a **rolling hash** to efficiently compute the hash for the next substring without recomputing it from scratch.

3. Record the indices where matches occur.

Advantages:

- Efficient for large texts, especially when searching for multiple patterns.
- Reduces unnecessary character comparisons using hashing.

Limitations:

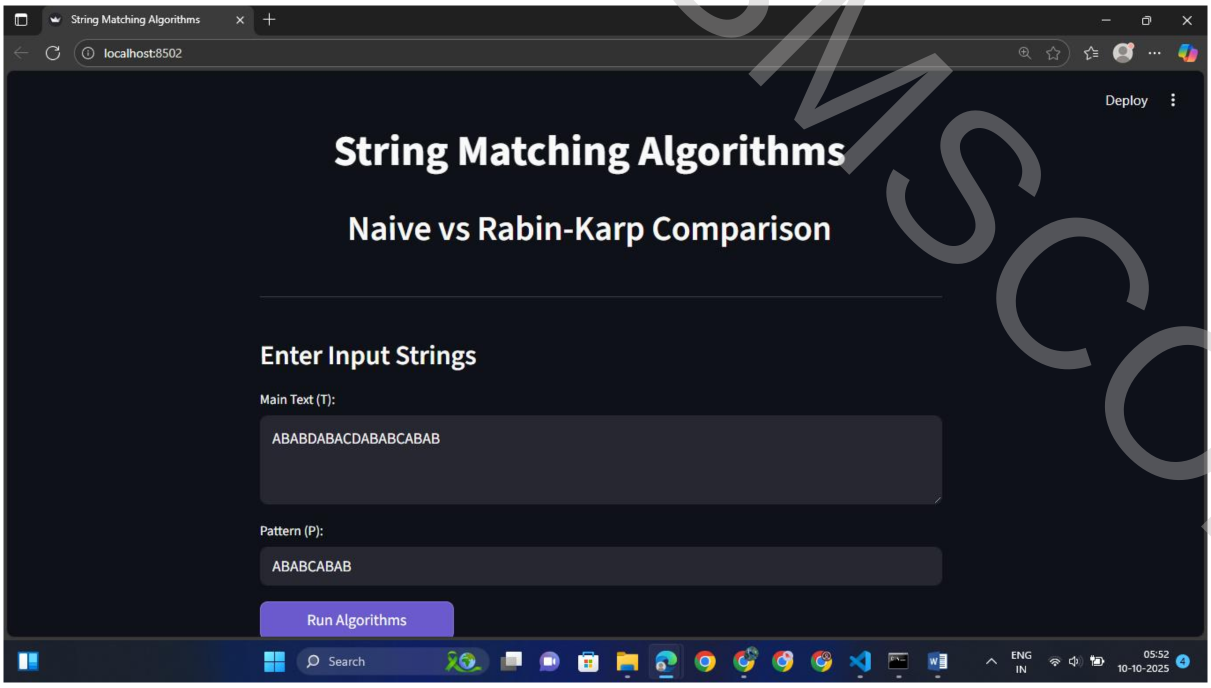
- Hash collisions can occur, requiring additional character comparisons.
- Slightly more complex to implement than the naive approach.

Workflow in the Project

1. **User Input:** Users enter the main text and the pattern in the Streamlit interface.
2. **Algorithm Execution:** Both algorithms are executed simultaneously, and every step is logged.
3. **Step Visualization:** Steps for each algorithm, including matches, mismatches, and hash updates (for Rabin-Karp), are displayed in collapsible, styled boxes for clarity.
4. **Pattern Indices:** The positions where the pattern occurs are displayed.
5. **Comparison Table:** A table summarizes differences in approach, complexity, hashing, and suitability.

This methodology ensures that users not only see the final output but also understand the **internal working of each algorithm**, making it a practical educational tool for learning string matching algorithms.

IMPLEMENTATION



String Matching Algorithms

Naive vs Rabin-Karp Comparison

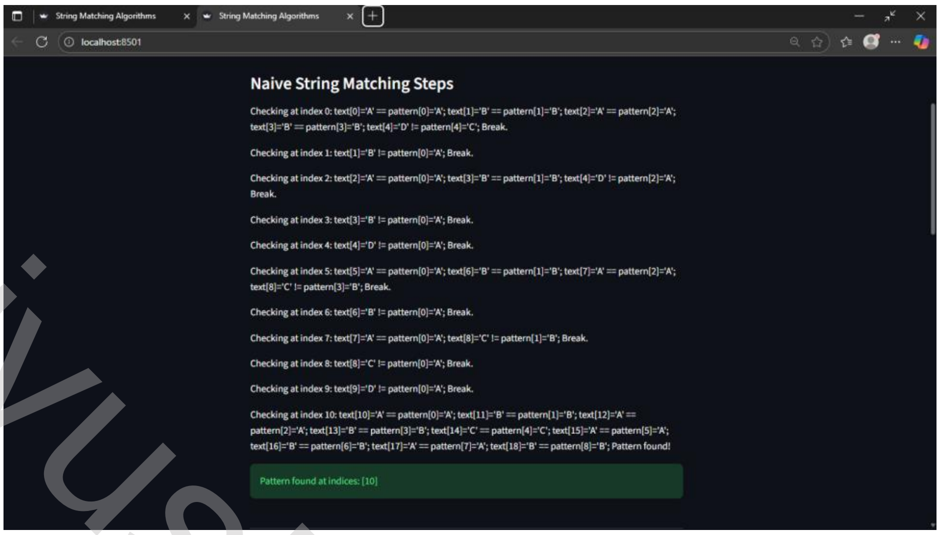
Enter Input Strings

Main Text (T):

ABABDABACDABABCABAB

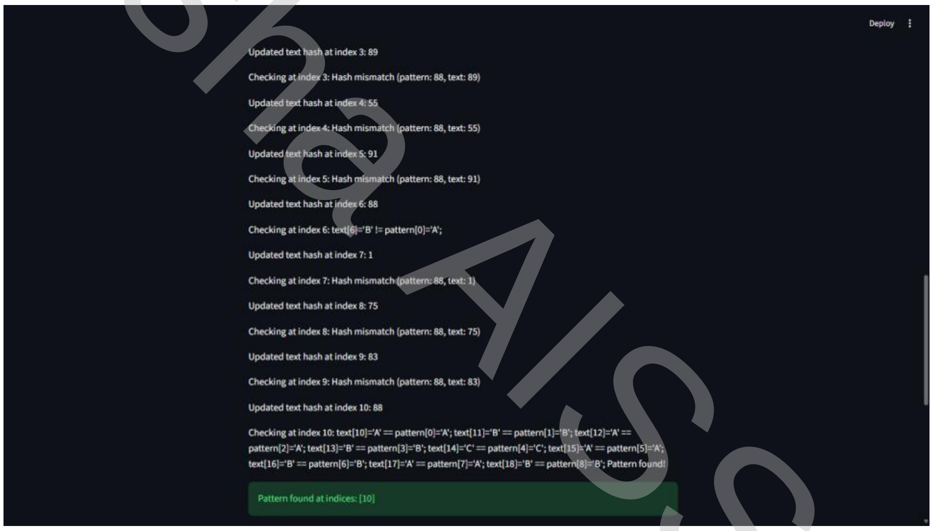
Pattern (P):

ABABCABAB

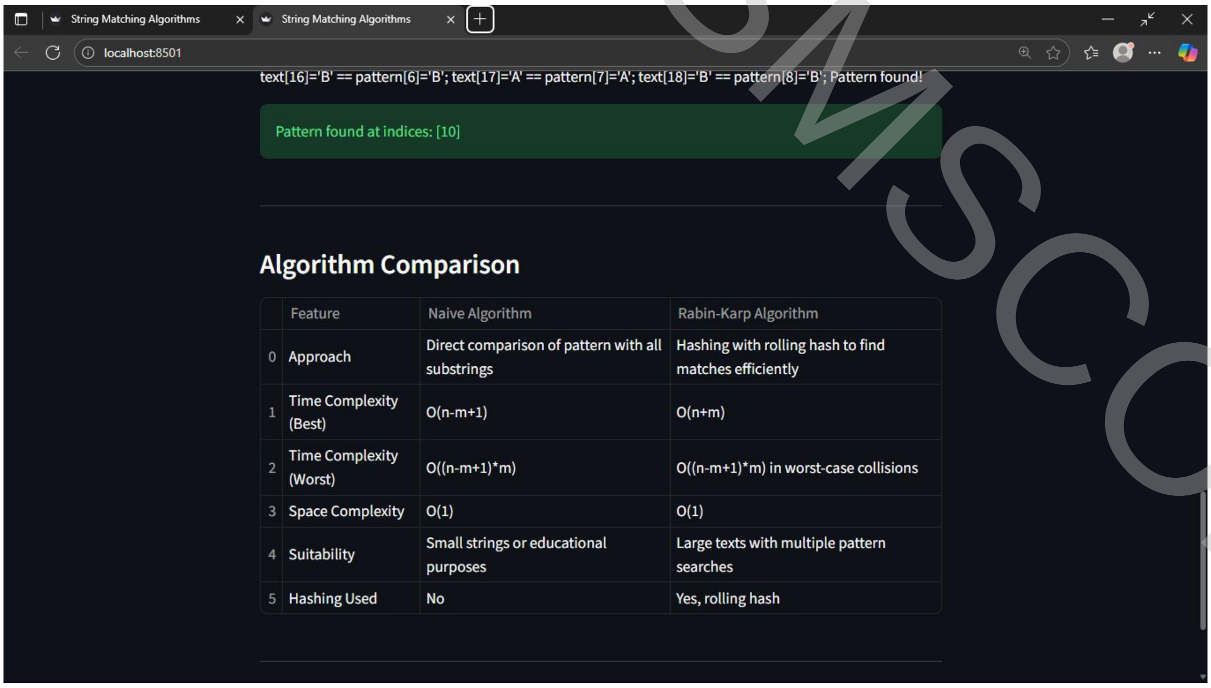


Naive String Matching Steps

```
Checking at index 0: text[0]='A' == pattern[0]='A'; text[1]='B' == pattern[1]='B'; text[2]='A' == pattern[2]='A';
text[3]='B' == pattern[3]='B'; text[4]='D' != pattern[4]='C'; Break.

Checking at index 1: text[1]='B' != pattern[0]='A'; Break.

Checking at index 2: text[2]='A' == pattern[0]='A'; text[3]='B' == pattern[1]='B'; text[4]='D' != pattern[2]='A';
Break.

Checking at index 3: text[3]='B' != pattern[0]='A'; Break.

Checking at index 4: text[4]='D' != pattern[0]='A'; Break.

Checking at index 5: text[5]='A' == pattern[0]='A'; text[6]='B' == pattern[1]='B'; text[7]='A' == pattern[2]='A';
text[8]='C' != pattern[3]='B'; Break.

Checking at index 6: text[6]='B' != pattern[0]='A'; Break.

Checking at index 7: text[7]='A' == pattern[0]='A'; text[8]='C' != pattern[1]='B'; Break.

Checking at index 8: text[8]='C' != pattern[0]='A'; Break.

Checking at index 9: text[9]='D' != pattern[0]='A'; Break.

Checking at index 10: text[10]='A' == pattern[0]='A'; text[11]='B' == pattern[1]='B'; text[12]='A' ==
pattern[2]='A'; text[13]='B' == pattern[3]='B'; text[14]='C' == pattern[4]='C'; text[15]='A' == pattern[5]='A';
text[16]='B' == pattern[6]='B'; text[17]='A' == pattern[7]='A'; text[18]='B' == pattern[8]='B'; Pattern found!

Pattern found at indices: [10]
```



```
Updated text hash at index 3: 89

Checking at index 3: Hash mismatch (pattern: 88, text: 89)

Updated text hash at index 4: 55

Checking at index 4: Hash mismatch (pattern: 88, text: 55)

Updated text hash at index 5: 91

Checking at index 5: Hash mismatch (pattern: 88, text: 91)

Updated text hash at index 6: 88

Checking at index 6: text[6]='B' != pattern[0]='A';

Updated text hash at index 7: 1

Checking at index 7: Hash mismatch (pattern: 88, text: 1)

Updated text hash at index 8: 75

Checking at index 8: Hash mismatch (pattern: 88, text: 75)

Updated text hash at index 9: 83

Checking at index 9: Hash mismatch (pattern: 88, text: 83)

Updated text hash at index 10: 88

Checking at index 10: text[10]='A' == pattern[0]='A'; text[11]='B' == pattern[1]='B'; text[12]='A' ==
pattern[2]='A'; text[13]='B' == pattern[3]='B'; text[14]='C' == pattern[4]='C'; text[15]='A' == pattern[5]='A';
text[16]='B' == pattern[6]='B'; text[17]='A' == pattern[7]='A'; text[18]='B' == pattern[8]='B'; Pattern found!

Pattern found at indices: [10]
```



```
text[16]='B' == pattern[6]='B'; text[17]='A' == pattern[7]='A'; text[18]='B' == pattern[8]='B'; Pattern found!

Pattern found at indices: [10]
```

Algorithm Comparison

	Feature	Naive Algorithm	Rabin-Karp Algorithm
0	Approach	Direct comparison of pattern with all substrings	Hashing with rolling hash to find matches efficiently
1	Time Complexity (Best)	O(n-m+1)	O(n+m)
2	Time Complexity (Worst)	O((n-m+1)*m)	O((n-m+1)*m) in worst-case collisions
3	Space Complexity	O(1)	O(1)
4	Suitability	Small strings or educational purposes	Large texts with multiple pattern searches
5	Hashing Used	No	Yes, rolling hash

CONCLUSION

This project successfully implements, visualizes, and compares two fundamental string matching algorithms: the Naive String Matching Algorithm and the Rabin-Karp Algorithm. Using an interactive Streamlit application, users can input any text and pattern, observe the step-by-step execution, and understand how matches and mismatches are identified. The Naive Algorithm provides a simple and intuitive approach to pattern searching, suitable for small texts, while the Rabin-Karp Algorithm improves efficiency with hashing and rolling hash techniques, making it ideal for large texts or multiple pattern searches.

A key feature of this project is the step-by-step visualization, which allows users to follow each comparison, match, mismatch, and hash update, making abstract concepts tangible and easy to grasp. The inclusion of a comparison table highlights differences in approach, time complexity, space requirements, hashing usage, and suitability, providing a clear understanding of when to use each algorithm.

The clean white-themed interface, collapsible step boxes, and structured layout make the application visually appealing and user-friendly, enhancing the educational experience. Overall, this project serves as a comprehensive learning tool, bridging theory and practice, demonstrating algorithmic efficiency, and providing an engaging platform for students, educators, and algorithm enthusiasts to explore string matching techniques effectively.

REFERENCES

- https://www.geeksforgeeks.org/naive-algorithm-for-pattern-searching/
-
- https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/
- https://www.tutorialspoint.com/data_structures_algorithms/string_matching_algorithms.htm
- https://www.programiz.com/dsa/rabin-karp-algorithm
- https://www.javatpoint.com/string-matching-algorithms
- https://www.coursera.org/lecture/algorithms-on-strings/naive-string-matching-VQ5UX
- https://www.studytonight.com/data-structures/string-matching-algorithms
- https://www.softwaretestinghelp.com/string-matching-algorithms/
- https://www.educative.io/edpresso/what-is-the-rabin-karp-algorithm
- https://www.baeldung.com/cs/string-searching-algorithms