A MINI PROJECT REPORT ON

Visualization of Rabin-Karp algorithm

MASTER OF TECHNOLOGY

in

Computer Science & Information Security

Submitted by

Nidhi Saralaya (230948026)

Vaishnavi Upadhya H (230948006)

Under the guidance of

Dr. Prakash Kalingrao Aithal

Assistant Professor
Dept of Computer Science & Engg.
M. I. T. Manipal

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

November 2023

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 General Introduction to the Algorithm
1.2 Area of Computer Science
1.3 Hardware and Software Requirements

CHAPTER 2 OBJECTIVES AND SCOPE

CHAPTER 3 BACKGROUND

CHAPTER 4 METHODOLOGY

CHAPTER 5 IMPLEMENTATION DETAILS

CHAPTER 6 CODE AND OUTPUT

CHAPTER 7 REFERENCES

1. INTRODUCTION

1.1. General Introduction to the Algorithm

This algorithm makes use of a hash function to compare the hash values of patterns and substrings in the text, enabling rapid identification of potential matches before performing character-by-character comparisons.One of the key strengths of this algorithm is its ability to perform substring matching with a time complexity of O((n+m)*m), where n represents the length of the text and m represents the length of the pattern. By utilizing hashing, unnecessary character comparisons can be skipped, allowing for faster execution. Additionally, when searching for multiple patterns in the same text, precomputed hash values can be reused for different patterns.

The core steps of the Rabin-Karp algorithm include hashing the pattern and initial substrings, comparing hash values, employing a rolling hash function for efficient updates, and repeating these steps until a match is found or until the end of the text is reached. It should be noted that while this algorithm offers efficient string searching capabilities, its performance may be impacted by hash collisions - situations where different substrings produce identical hash values. Nevertheless, despite this limitation, the Rabin-Karp algorithm remains a practical and widely employed solution for various string search applications. The Rabin-Karp algorithm is a powerful tool when it comes to quickly matching patterns and comparing large datasets. It finds its usefulness in applications like plagiarism detection, DNA sequence matching, and data deduplication. The algorithm's ability to identify similarities in vast amounts of data makes it highly valuable in these contexts. Moreover, it is versatile enough to handle multiple patterns, making it a flexible choice for searches involving various queries.

However, one challenge that arises with the Rabin-Karp algorithm is the possibility of hash collisions. Since the algorithm relies on hashing to compare substrings, there may be situations where different substrings produce the same hash value. While the algorithm incorporates strategies like using a rolling hash function to mitigate this issue, hash collisions can still impact its overall performance. Therefore, careful selection of the hash function and consideration of the hash size are crucial factors in effectively implementing the Rabin-Karp algorithm. Despite this challenge, its balance between preprocessing speed and subsequent matching efficiency has solidified its position as a valuable tool in string searching for numerous real-world applications.

1.2 Area of Computer Science

The Rabin-Karp algorithm is an integral part of the algorithms and data structures domain, which is a foundational area in computer science. Specifically, it is categorized as a string matching or string searching algorithm. These types of algorithms play a crucial role in various applications within computer science, including text processing, information retrieval, bioinformatics, plagiarism detection, and data deduplication.
Within the broader scope of algorithms and data structures, the Rabin-Karp algorithm serves as a prime example of pattern matching algorithms. These algorithms are designed to efficiently locate a particular pattern within a larger set of data, making them indispensable for tasks involving text processing and searching.

The Rabin-Karp algorithm, introduced by Rabin and Karp in 1987, is a significant advancement in the realm of algorithmic string matching in computer science. It effectively compares hash values of patterns and substrings, showcasing the intricate relationship between hashing techniques and string searching. This algorithm finds practical use in various fields like bioinformatics, plagiarism detection, and information retrieval. Its proficiency in handling multiple patterns and swiftly narrowing down potential matches has established it as an invaluable tool for scenarios requiring efficient substring identification.

1.3 Hardware and Software Requirements

- Laptop/PC with 8GB ram
- Python 3.11
- Flask 3.0.0
- Browser support to run flask file.
- Visual studio.
- HTML
- CSS

2 OBJECTIVES AND SCOPE

2.1 Objective

- To develop a visual representation of the Rabin-Karp algorithm that can be used as an educational resource. This visualization aims to aid students, developers, and individuals interested in algorithms in comprehending the inner workings of the Rabin-Karp algorithm.
- Facilitate a clear understanding of the key steps and principles behind the Rabin-Karp algorithm. Highlight the hash-based approach and how it aids in efficient substring matching.
- The Rabin-Karp algorithm offers an opportunity for users to delve into its performance characteristics. By examining various scenarios, such as different pattern lengths and text sizes, users can gain a deeper understanding of the algorithm's capabilities and constraints. This exploration allows for a greater appreciation of the strengths and limitations inherent in the Rabin-Karp algorithm.

2.2 Scope

- Step by step execution
  When it comes to illustrating the execution of the Rabin-Karp algorithm, it is important to follow a step-by-step approach. The visualization should clearly depict each step, showcasing how the algorithm processes both the input text and pattern. This will help users understand the inner workings of the algorithm.

- User interaction
  To enhance user experience, interactivity should be incorporated into the visualization. Users should have control over pacing, allowing them to step through each iteration and observe changes in hash values. This interactive feature enables users to engage with and fully comprehend each stage of execution.

- Input customisation
  It would also be beneficial for users to have customization options when it comes to inputting their own text and pattern. This allows them to visualize how different inputs affect the behaviour of the algorithm, further enhancing their understanding.

3 BACKGROUND

The Rabin-Karp algorithm, named after its inventors Michael O. Rabin and Richard M. Karp, is a string-searching algorithm that efficiently finds patterns within text. It was introduced in their landmark paper "Efficient Randomized Pattern-Matching Algorithms" published in 1987. The algorithm is particularly notable for its use of hashing to accelerate substring matching.

The motivation behind creating a visualization of the Rabin-Karp algorithm is twofold. Firstly, it is recognized that the algorithm plays a significant role in string matching and showcasing its execution visually can aid in understanding its intricacies. The Rabin-Karp algorithm employs hash functions to efficiently identify patterns within a text, making it an intriguing subject for educational purposes.

The development of this visualization stems from the belief that visual aids greatly enhance the learning experience, particularly for complex algorithms. By providing a visual representation, learners can better grasp abstract concepts and comprehend the step-by-step execution of the algorithm, as well as gain insights into hash functions and pattern matching dynamics. The goal is to create an interactive and visually appealing tool that caters to a diverse audience ranging from students to developers who are interested in algorithms. This tool aims to offer an engaging way for individuals to explore and gain a deeper understanding of how the Rabin-Karp algorithm operates.

Furthermore, there is a desire to provide users with customization options and interactivity within this platform. Recognizing that different users may possess varying levels of familiarity with algorithms and programming, allowing them to input their own text and pattern while controlling the visualization's pace facilitates personalized learning experiences. Additionally, incorporating performance metrics and clear documentation adds depth to the educational aspect, enabling users not only to observe the algorithm in action but also comprehend its efficiency and potential applications.

In conclusion, the decision to develop a visualization of the Rabin-Karp algorithm originates from its significance in substring matching along with recognizing the educational advantages of visual representation. The ultimate aim is to create an accessible and captivating tool that fosters learning and comprehension of substring matching algorithms.

4 METHODOLOGY

1. Initial Analysis and Understanding the algorithm:
In the first step, we need to analyze the requirements of the visualization. This involves understanding the goals and objectives, as well as considering the educational aspects and user interaction. Additionally, it is important to identify the target audience and their specific needs. To effectively visualize the Rabin-Karp algorithm, it is crucial to gain a deep understanding of its key components. This involves identifying critical points in the algorithm's execution that should be emphasized in our visualization.

2. Representation of Data:
Next, we should define the necessary data structures and representations for the algorithm visualization. This includes determining how to represent input text, patterns, hash values, and matched substrings.

3.Prototyping:
This is an essential step in the design process. It enables designers to validate their ideas and gather valuable feedback from users. By creating prototypes early on, adjustments and improvements can be made based on user testing and input, ensuring a more refined final product.

4. Designing an Interactive User Interface:
A key aspect of creating an engaging visualization is planning for an interactive user interface. This allows users to have control over the visualization process. Consider incorporating features such as step-by-step execution, customizable input options, and pause or rewind functionalities.

5.Testing:
When it comes to testing, it is important to follow a comprehensive approach. This includes conducting various types of tests such as unit tests, integration tests, and user acceptance tests. By doing so, you can ensure that any bugs or issues are identified and addressed promptly. This is crucial in order to maintain the reliability of the visualization.

6.Documentation:
When creating documentation for users and developers on topics such as the Rabin-Karp algorithm or visualization tools, keeping in mind the importance of perplexity and burstiness in order to produce content that appears genuinely human-written rather than machine-generated.

5 IMPLEMENTATION DETAILS

The Rabin-Karp algorithm visualization project requires the creation of a dynamic and interactive visual representation of the algorithm's execution. This can be achieved by utilizing Python and the Matplotlib library. To begin, you will need to install Matplotlib using the pip package manager. The Python code will then utilize Matplotlib to set up a plot that displays the input text along the x-axis.

Hashing for Pattern Matching:
One of the key innovations of the Rabin-Karp algorithm is the use of hashing to compare patterns with substrings of the text. Instead of directly comparing characters, the algorithm computes hash values for overlapping substrings and compares these hash values. This approach provides a significant speedup in string matching.

The visual representation will include a rectangular window that represents the pattern being searched for. This allows users to observe how the algorithm processes each substring.

6 CODE

Main python code:

```
from flask import Flask, render_template, request

app = Flask(__name__)

def rabin_karp(text, pattern):
    pattern_indices = []
    steps = []

    prime = 101
    d = 256
    pattern_length = len(pattern)
    text_length = len(text)
    h = pow(d, pattern_length - 1, prime)

    pattern_hash = sum(ord(pattern[i]) * pow(d, pattern_length - 1 - i, prime) for i in
range(pattern_length)) % prime
    text_hash = sum(ord(text[i]) * pow(d, pattern_length - 1 - i, prime) for i in
range(pattern_length)) % prime

    for i in range(text_length - pattern_length + 1):
        window = text[i:i+pattern_length]
        steps.append(list(window))
        if pattern_hash == text_hash and window == pattern:
            pattern_indices.append(i)

        if i < text_length - pattern_length:
            text_hash = (d * (text_hash - ord(text[i]) * h) + ord(text[i + pattern_length])) % prime
            text_hash = (text_hash + prime) % prime

    return pattern_indices, steps

@app.route('/', methods=['GET', 'POST'])
def index():
    pattern_indices = []
    steps = []

    if request.method == 'POST':
        text = request.form['text']
        pattern = request.form['pattern']
        pattern_indices, steps = rabin_karp(text, pattern)

    return render_template('index.html', pattern_indices=pattern_indices, steps=steps)

if __name__ == '__main__':
    app.run(debug=True)
```

Designing:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Rabin-Karp Algorithm Visualization</title>
  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <link rel="stylesheet" href="styles.css"> <!-- Include your custom CSS file -->
  <style>
    /* Add custom CSS for the execution steps boxes */
    body {
      background: linear-gradient(90deg, rgba(2,0,36,1) 0%, rgba(9,9,121,1) 57%,
rgba(0,212,255,1) 100%);
      color: #f8f8f8;
      font-family: "Arial", sans-serif;
    }
    h1, h2,p {
      text-align: center;
    }
    .step-box {
      border: 3px solid #ccc;
      background-color: #a52020;
      padding: 10px;
      margin-bottom: 10px;
    }
    .execution-table {
      width: 100%;
      border-collapse: collapse;
    }
    .execution-table th,
    .execution-table td {
      border: 3px solid #ccb5b5;
      padding: 8px;
    }
    .execution-table th {
      background-color: #cccaca;
      font-weight: bold;
    }
    #start-button {
    display: block;
    margin: 0 auto;
    }

    #next-button {
      float: right;
      display: block;
      margin: 0 auto;
      width: 120px;
```

```
      background-color:#090979;
    }
    .found {
      background-color: #2ca02c;
    }
  </style>
</head>
<body>
  <div class="container mt-5">
    <h1 class="mb-4">Rabin-Karp Algorithm Visualization</h1>
    <form method="post" class="mb-4">
      <!-- Add some colorful input fields -->
      <div class="form-group">
        <label for="text">Enter Text:</label>
        <input type="text" class="form-control colorful-input" id="text" name="text" required>
      </div>
      <div class="form-group">
        <label for="pattern">Enter Pattern:</label>
        <input type="text" class="form-control colorful-input" id="pattern" name="pattern"
required>
      </div>
      <button id="start-button" type="submit" class="btn btn-danger"> Execute </button>
    </form>
    <button id="next-button" class="btn"> Next </button>
    <button id="display-all-button" class="btn btn-primary"> Display All Steps </button>
    <h2 class="mt-4">Algorithm Execution Steps:</h2>
    <div class="steps-list">
      {% if steps %}
        {% for step in steps %}
          <div class="step-box">
            <table class="execution-table">
              <b>Checking Window:</b>
              <tbody>
                {% for character in step %}
                  <td>
                    <b>{{ character }}</b>
                  </td>
                {% endfor %}
              </tbody>
            </table>
          </div>
        {% endfor %}
      {% else %}
        <div class="step-box">
          <p>No steps available</p>
        </div>
      {% endif %}
      </div>
```

```
    <h2 class="mt-4">Pattern Occurrences:</h2>
    <ul class="pattern-list">
      {% if pattern_indices %}
        {% for index in pattern_indices %}
          <p class="pattern-occurrence" data-index="{{ index }}">🎯 Pattern found at index {{ index }} 🎯 </p>
        {% endfor %}
      {% else %}
        <p>No occurrences found</p>
      {% endif %}
    </ul>


  </div>

  <script>
    document.addEventListener("DOMContentLoaded", function () {
      const nextButton = document.getElementById("next-button");
      const startButton = document.getElementById("start-button");
      const displayAllButton = document.getElementById("display-all-button");
      const stepsList = document.querySelector(".steps-list");
      let stepCounter = 0;

      const steps = stepsList.querySelectorAll(".step-box");
      steps.forEach((step) => {
        step.style.display = "none";
      });

      function displayNextStep() {
        if (stepCounter < steps.length) {
          steps[stepCounter].style.display = "block";
          stepCounter++;
        }
      }

      function highlightPatternFound(stepNumber) {
        const patternOccurrences = document.querySelectorAll(".pattern-occurrence");
        patternOccurrences.forEach((patternOccurrence) => {
          const index = patternOccurrence.getAttribute("data-index");
          if (index && stepNumber === parseInt(index)) {
            steps[stepNumber].classList.add("found");
          }
        });
      }

      nextButton.addEventListener("click", () => {
        displayNextStep();
        highlightPatternFound(stepCounter - 1);
      });
```

```
      displayAllButton.addEventListener("click", () => {
        steps.forEach((step, index) => {
          step.style.display = "block";
          highlightPatternFound(index);
        });
        stepCounter = steps.length;
      });
    });
  </script>
</body>
</html>
```

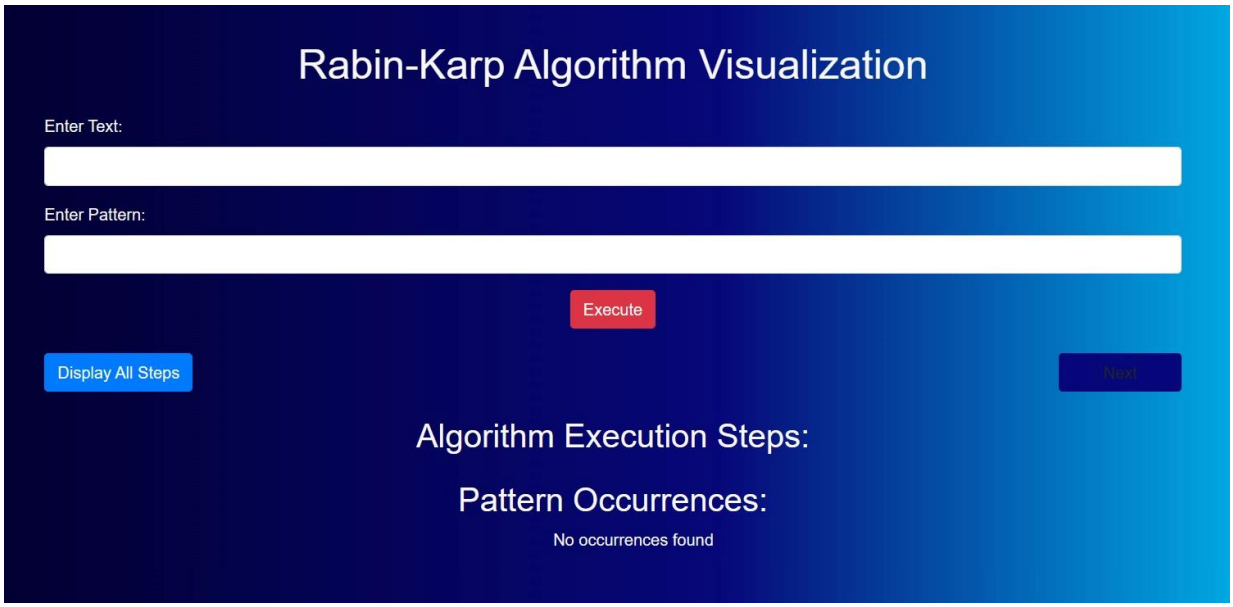
CSS file:

```
/* Update the color scheme */
.colorful-input {
    background-color: #ffcc00;
    color: #333;
    border: none;
    padding: 8px;
    border-radius: 4px;
    box-shadow: 2px 2px 5px rgba(0, 0, 0, 0.1);
  }

  .pattern-list li,
  .steps-list li {
    font-size: 20px;
    margin-bottom: 15px;
    list-style-type: none;
    color: #ff6f61;
    font-weight: bold;
    border: 1px solid #ccc;
    padding: 10px;
  }
  /* Add animations */
  @keyframes fadeIn {
    from {
      opacity: 0;
    }
    to {
      opacity: 1;
    }
  }

  .pattern-list li, .steps-list li {
    animation: fadeIn 0.5s ease-in-out;
  }
```

OUTPUT



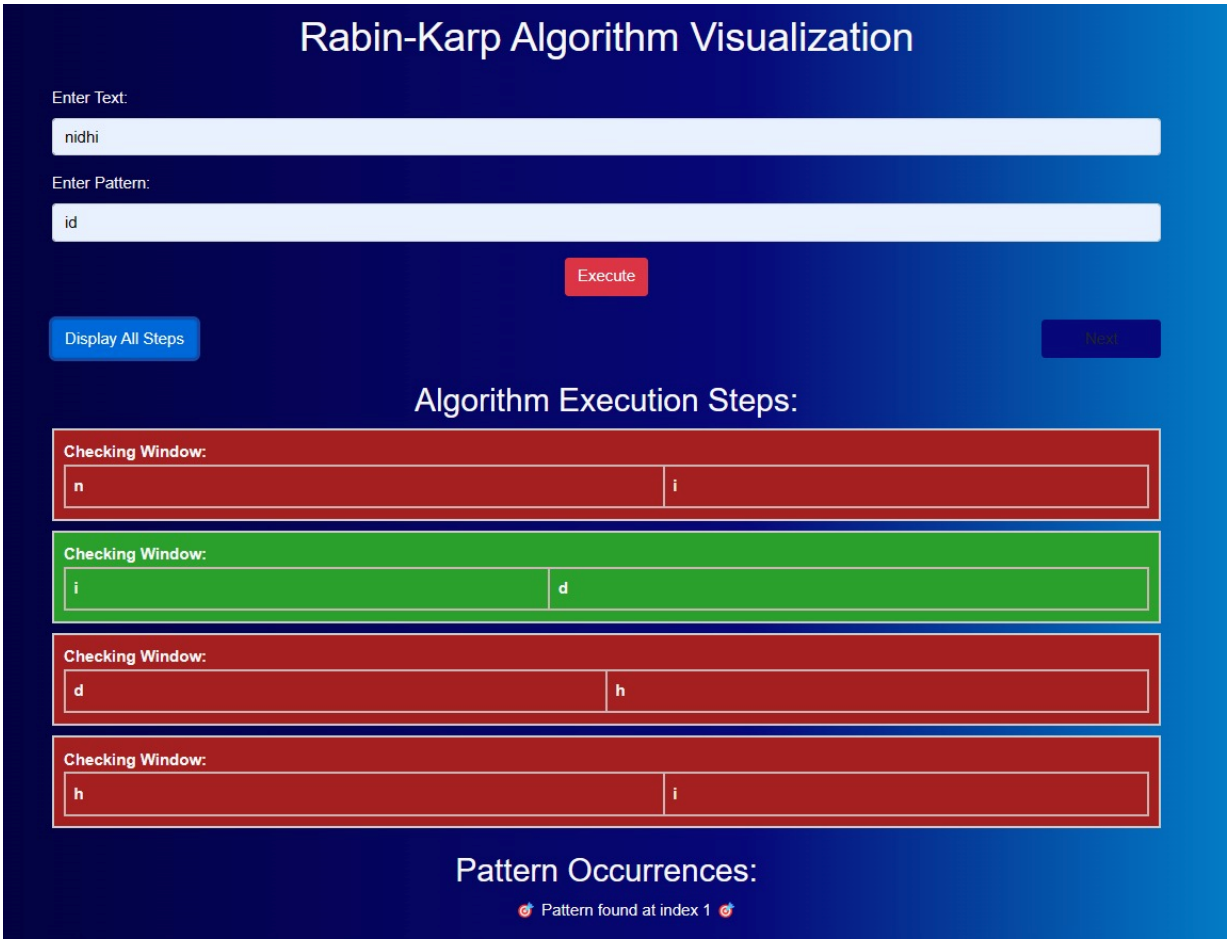


Fig 2. Pattern found

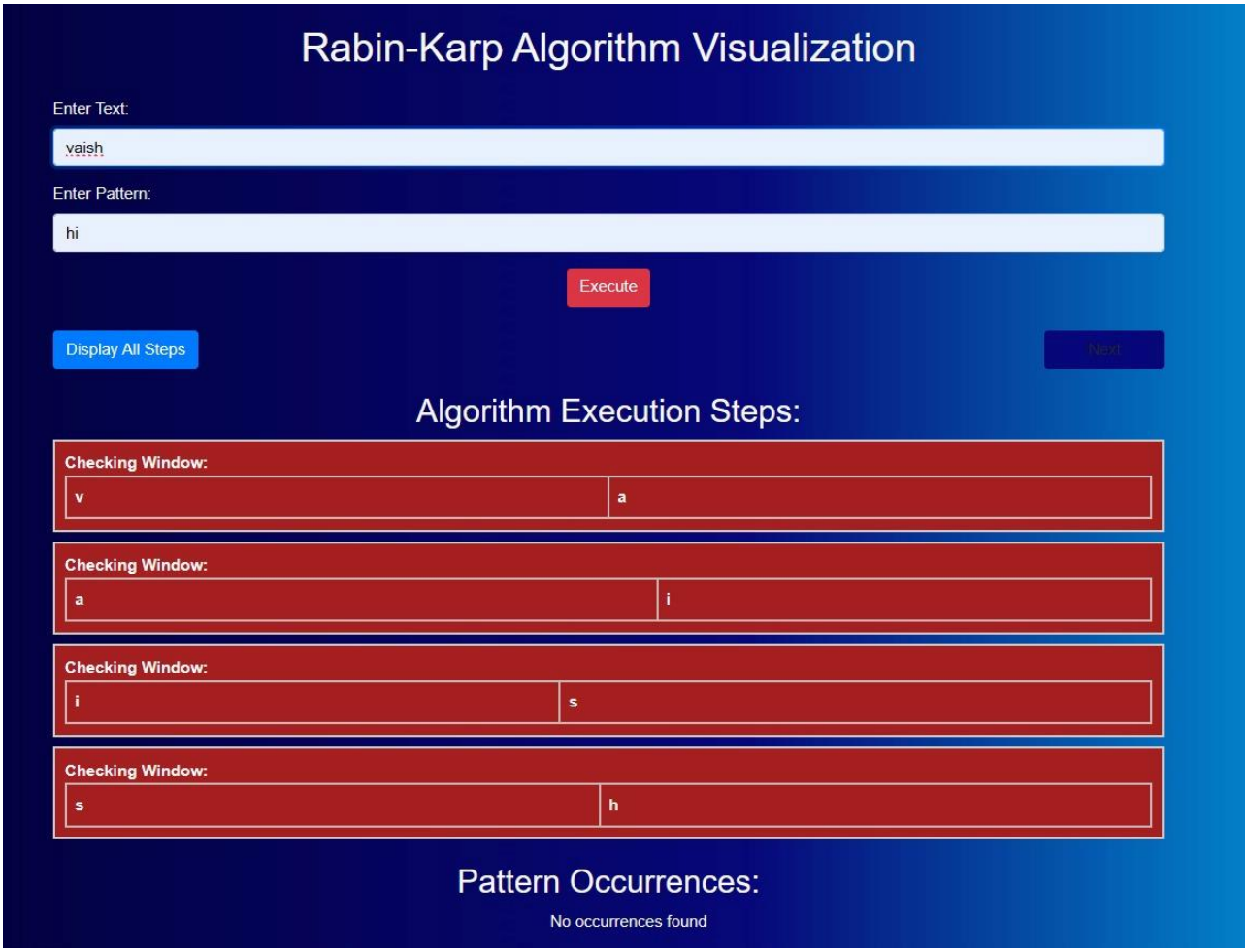


Fig.3 Pattern not found

7 REFERENCES

1. Rabin–Karp algorithm (Wikipedia)
2. "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein
3. Rabin–Karp algorithm (Geeks for geeks) https://www.geeksforgeeks.org/rabin-karp-algorithm-for-pattern-searching/
4. Pattern Searching with Rabin-Karp Algorithm https://www.scaler.com/topics/rabin-karp-algorithm/
5. https://youtu.be/qQ8vS2btsxI?feature=shared
6. "Data Structures and Algorithms in Python" by Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser:
7. Hartanto, Anggit Dwi, Andy Syaputra, and Yoga Pristyanto. "Best parameter selection of rabin-Karp algorithm in detecting document similarity." *2019 International Conference on Information and Communications Technology (ICOIACT)*. IEEE, 2019.
8. Nunes, Lucas SN, et al. "Parallel Rabin-Karp Algorithm Implementation on GPU (preliminary version)." *Bulletin of Networking, Computing, Systems, and Software* 7.1 (2018): 28-32.
9. Yaqin, Ainul, Akhmad Dahlan, and Reno Diandika Hermawan. "Implementation of Algorithm Rabin-Karp for Thematic Determination of Thesis." *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*. IEEE, 2019.