PROJECT REPORT
ON

String Matching GUI Application

Submitted By:	Submitted To:
Ritika Sharma	Divyanshi Sharma
UID: 24MCI10076	Assistant Professor
Class: MCA AI/ML	E17438
Section: 24MAM 2-A	

University Institute of Computing Chandigarh University, Gharuan, Mohali

Project Report
String Matching GUI Application

Student Name	**: Ritika Sharma**	**UID**	**: 24MCI10076**
Branch	**: MCA(AI & ML)**	**Section/Group**	**:24MAM2(A)**
Semester	**:1ST**	**Date of Performance**	**: 21/10/2024**
Subject Name	**: Design And Analysis of Algorithms LAB**	**Subject Code**	**: 24CAP-612**

1. Introduction

1.1. Project Overview

The String Matching GUI Application is designed to allow users to input a block of text and a pattern, and then search for the pattern's occurrences within the text. The search functionality is implemented using a naive string matching algorithm. The application uses Python’s Tkinter library to provide an intuitive graphical interface for inputting the text and pattern, performing the search, and displaying the results.

1.2. Objectives

The primary objective are:

Create a user-friendly GUI application for string pattern searching.
Implement the naive string matching algorithm to find occurrences of a given pattern within a text.
Display the results in a clear and concise manner to the user.

2. Technology Stack

2.1. Programming Language

Python: The application is written in Python, utilizing the Tkinter library for the GUI and implementing string matching using custom Python code.

2.2. Libraries and Tools

Tkinter: Python’s standard library for creating graphical user interfaces.
Naive String Matching Algorithm: A straightforward approach to finding substrings in a given text.

3. System Design

3.1. User Interface

The application interface is designed to be simple and intuitive. It consists of the following elements:

Text Input Field: An entry field where users can input the main block of text in which they want to search for a pattern.

Pattern Input Field: An entry field where users input the pattern they want to find within the text.

Search Button: A button that triggers the search function when clicked.

Result Display Label: A label where the result of the pattern search is displayed, showing either the positions where the pattern is found or indicating that no match was found.

3.2. Algorithm

The Naive String Matching Algorithm is a simple method to find all occurrences of a pattern in a text. The algorithm checks the text character by character to see if the pattern matches the text at each position.

Algorithm Steps:

Iterate over the text from the start until the point where the remaining substring is shorter than the pattern.
For each position, compare the pattern character by character with the substring of the text.
If the entire pattern matches, record the starting index of the match.
Return a list of positions where the pattern is found.

3.3. Pseudocode of the Algorithm

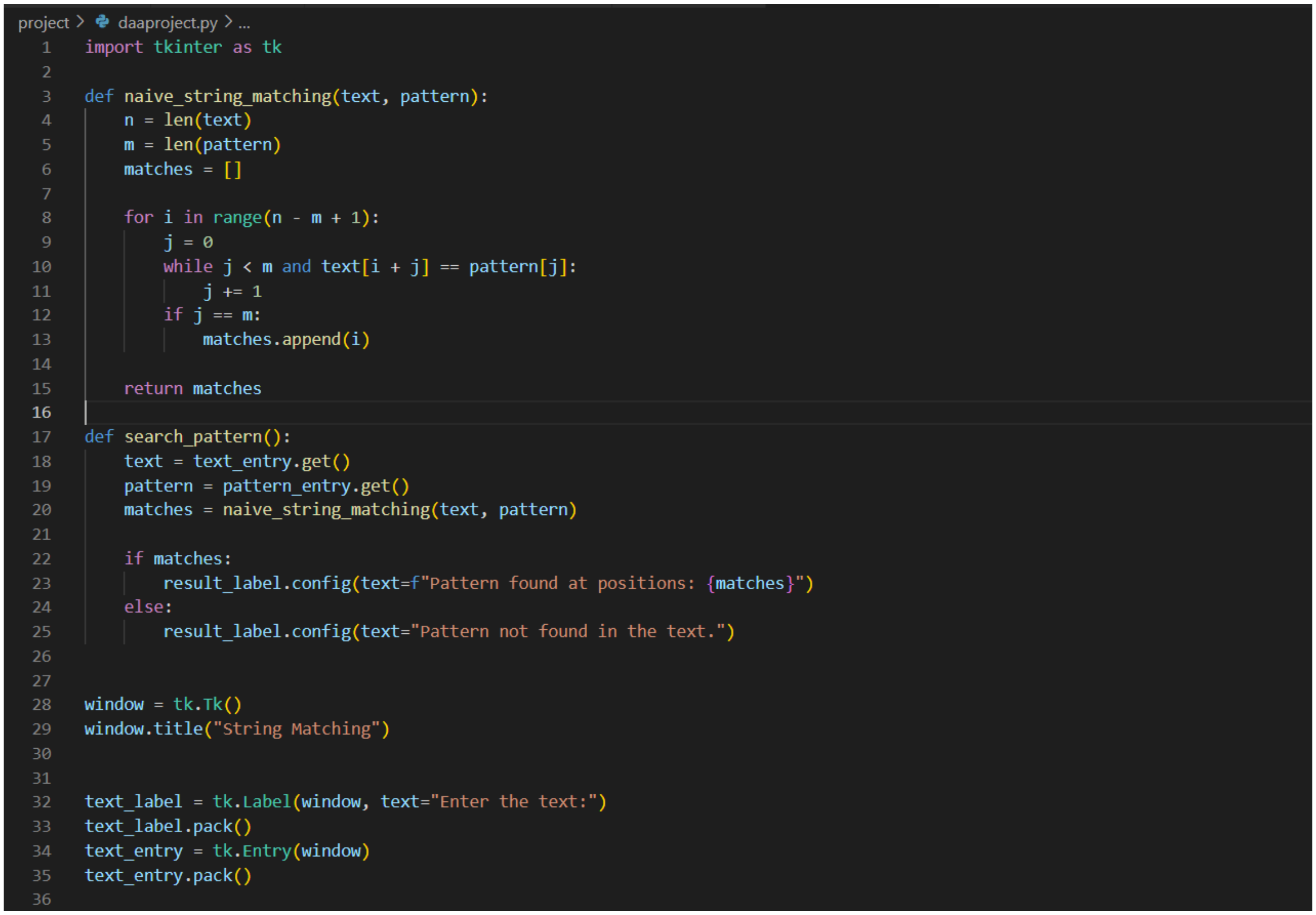
```
function naive_string_matching(text, pattern):
    n = length of text
    m = length of pattern
    matches = []

    for i in range(0, n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
        if j == m:
            append i to matches

    return matches
```

4. Implementation

4.1. Naive String Matching Code

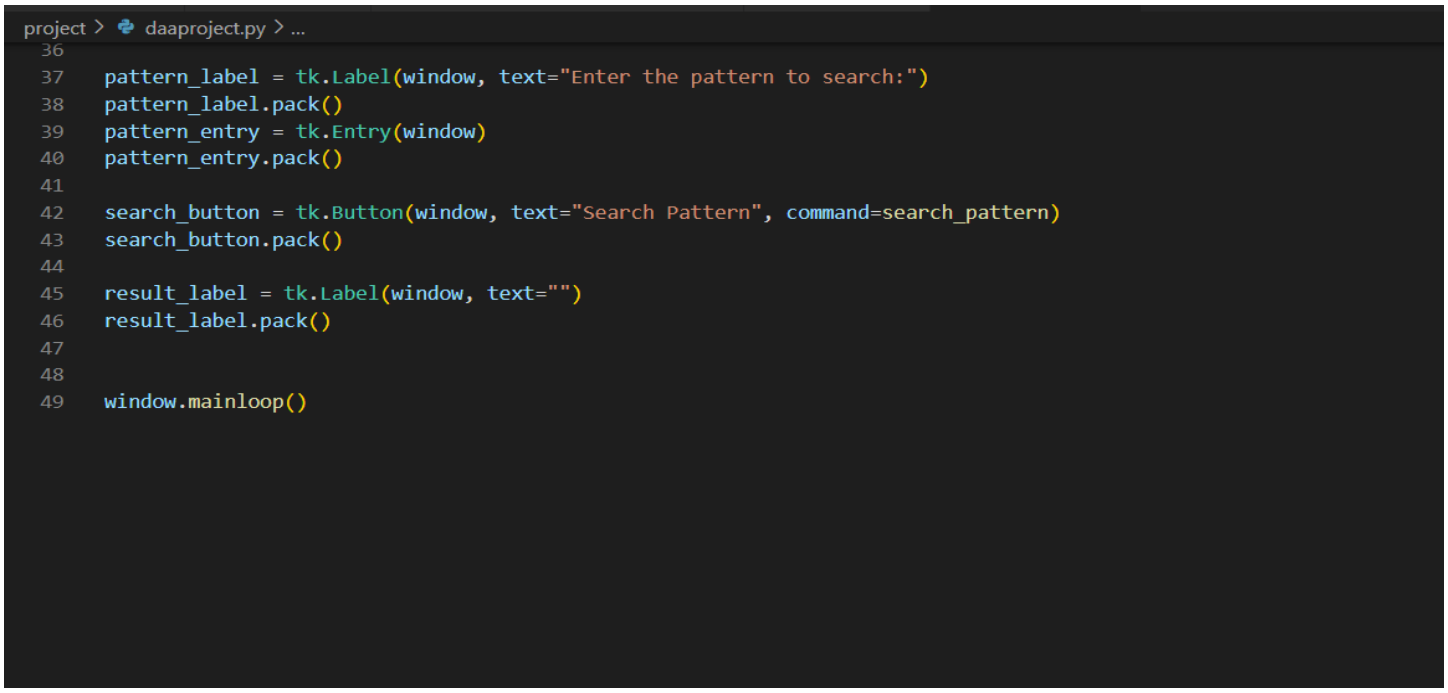
```python
def naive_string_matching(text, pattern):
    n = len(text)
    m = len(pattern)
    matches = []

    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
        if j == m:
            matches.append(i)

    return matches
```

4.2. GUI Code (Tkinter)



project > daaproject.py > ...

```python
import tkinter as tk

def naive_string_matching(text, pattern):
    n = len(text)
    m = len(pattern)
    matches = []

    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
        if j == m:
            matches.append(i)

    return matches

def search_pattern():
    text = text_entry.get()
    pattern = pattern_entry.get()
    matches = naive_string_matching(text, pattern)

    if matches:
        result_label.config(text=f"Pattern found at positions: {matches}")
    else:
        result_label.config(text="Pattern not found in the text.")


window = tk.Tk()
window.title("String Matching")


text_label = tk.Label(window, text="Enter the text:")
text_label.pack()
text_entry = tk.Entry(window)
text_entry.pack()

```



project > daaproject.py > ...

```python

pattern_label = tk.Label(window, text="Enter the pattern to search:")
pattern_label.pack()
pattern_entry = tk.Entry(window)
pattern_entry.pack()

search_button = tk.Button(window, text="Search Pattern", command=search_pattern)
search_button.pack()

result_label = tk.Label(window, text="")
result_label.pack()


window.mainloop()
```

Creating the main window

```
window = tk.Tk()
window.title("String Matching")
```

Adding Text Input Label and Entry Field

```
text_label = tk.Label(window, text="Enter the text:")
text_label.pack()
text_entry = tk.Entry(window)
text_entry.pack()
```

Adding Pattern Input Label and Entry Field

```
pattern_label = tk.Label(window, text="Enter the pattern to search:")
pattern_label.pack()
pattern_entry = tk.Entry(window)
pattern_entry.pack()
```

Adding Search Button

```
search_button = tk.Button(window, text="Search Pattern", command=search_pattern)
search_button.pack()
```

Adding Result Display Label

```
result_label = tk.Label(window, text="")
result_label.pack()
```

Running the application

```
window.mainloop()
```

4.3. Event Handling

The search_pattern function is connected to the "Search Pattern" button using the command parameter. When clicked, this function is executed, which:

Fetches the text and pattern from the entry fields.
Executes the string matching algorithm.
Displays the result in the result label.

<u>5. Testing</u>

5.1. Test Scenarios

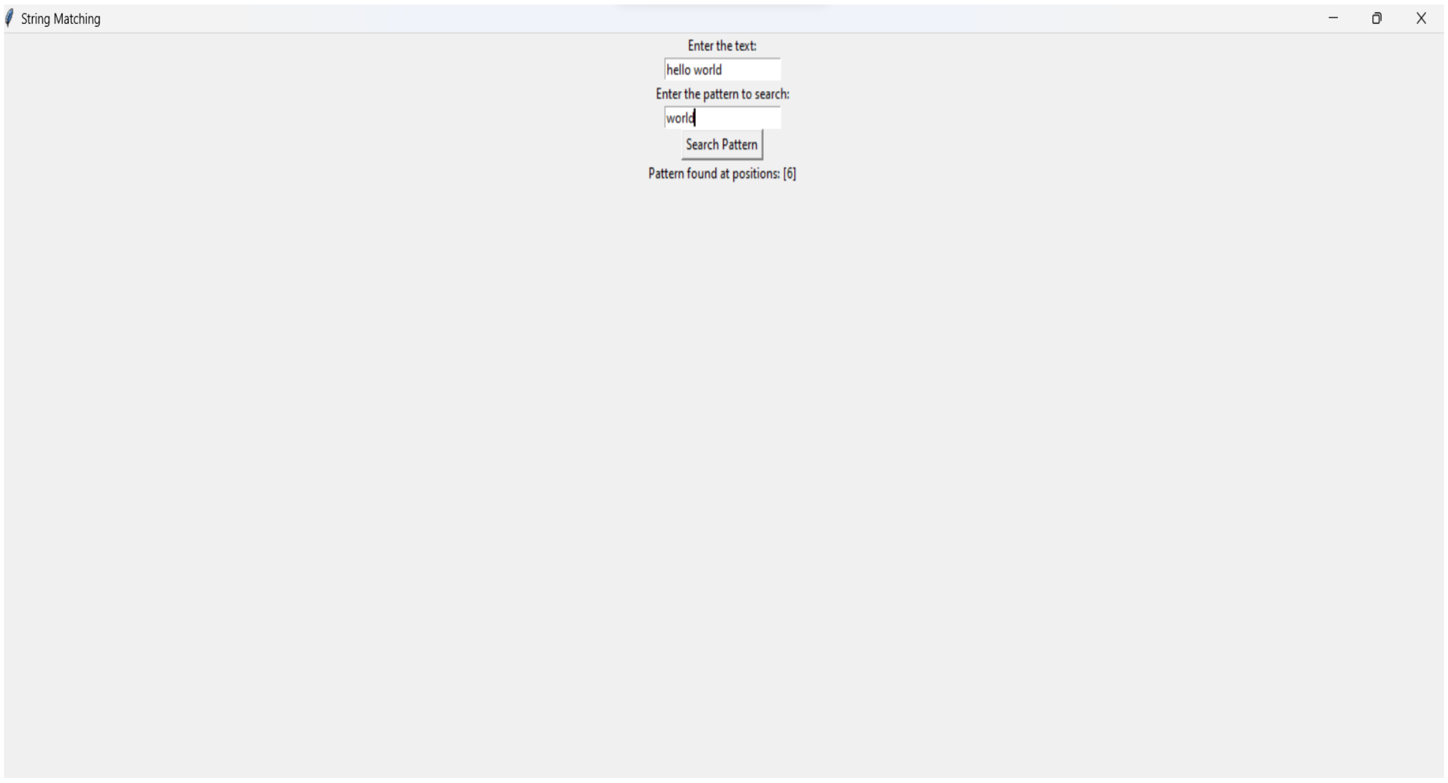
Test Case 1: Search for a pattern that exists within the text.
Input:
Text: hello world
Pattern: world
Expected Output: Pattern found at positions: [6]

Test Case 2: Search for a pattern that does not exist within the text.
Input:
Text: hello world
Pattern: python
Expected Output: Pattern not found in the text.

Test Case 3: Search for a pattern that occurs multiple times within the text.
Input:
Text: abcabcabc
Pattern: abc
Expected Output: Pattern found at positions: [0, 3, 6]

Test Case 4: Search with an empty pattern.
Input:
Text: abc
Pattern: ``
Expected Output: Pattern found at positions: [0, 1, 2, 3]

6.RESULT



Pattern found at positions: [6]

7. Learning Outcomes

Understanding the Naive String Matching Algorithm

- **Conceptual Clarity**: Gained an understanding of the basic naive string matching algorithm, including how it works by comparing each substring in the text with the pattern.
- **Complexity Awareness**: Recognized the time complexity of the naive approach, which is O((n−m+1)·m)O((n-m+1) \cdot m)O((n−m+1)·m), and understood its limitations in terms of efficiency, especially for large text or patterns.
- **Pattern Matching Techniques**: Gained practical insight into pattern matching techniques and the challenges of efficiently searching for substrings in larger texts.

2. Graphical User Interface (GUI) Design

- **Tkinter Basics**: Learned how to create a simple and intuitive graphical user interface using the Tkinter library in Python.
- **Event Handling in GUIs**: Acquired hands-on experience with event-driven programming, where user actions (such as clicking buttons) trigger specific functions to perform tasks (like searching for patterns).
- **User Input and Feedback**: Understood how to manage user input using entry fields, and display feedback or results using labels.

3. Integration of Algorithm with GUI

- **Algorithm-GUI Interaction**: Learned how to integrate backend logic (i.e., the string matching algorithm) with frontend GUI components to create a seamless user experience.
- **Data Flow Management**: Practiced managing data flow between GUI input fields and Python functions, ensuring that user inputs were properly passed to the algorithm and results were correctly displayed.

4. Python Programming Skills

- **Python Functions**: Strengthened knowledge of writing and using Python functions to encapsulate logic, such as the naive_string_matching function.
- **Control Structures**: Reinforced understanding of control structures like loops and conditionals when implementing the string matching algorithm.
- **Modular Programming**: Enhanced ability to break down a problem into modular components (e.g., separating the logic for the algorithm and the GUI).

5. Debugging and Testing

- **Handling User Input**: Learned how to validate and handle different types of user inputs, including empty patterns or incorrect text entries.
- **Testing Scenarios**: Understood the importance of testing different edge cases, such as patterns that don't exist in the text, patterns of varying lengths, or multiple pattern occurrences.

6. Project Design and Planning

- **Designing Simple Applications**: Gained experience in designing a complete Python application with a clear problem statement, objectives, and modular components.
- **Building Prototypes**: Learned how to quickly build and iterate over a functional prototype of the application, gradually improving both the algorithm and the user interface.
- **Documentation**: Understood the importance of documenting code, writing reports, and explaining the design, logic, and testing procedures for future reference.

7. Future Optimization Awareness

- **Efficiency Considerations**: Learned about the limitations of the naive algorithm and how more efficient string matching algorithms like KMP or Boyer-Moore could be implemented to optimize the search process.
- **Scalability**: Realized the need to consider algorithm scalability when working with larger data sets or more complex applications, and how the current solution can be expanded.

7. Conclusion

This project demonstrates how to create a simple yet effective string matching application using Python. The naive string matching algorithm is implemented for its simplicity, and the Tkinter library is utilized for building an easy-to-use graphical user interface. The program is functional, user-friendly, and meets the project objectives of enabling users to search for patterns within text.

7. Future Improvements
While the naive string matching algorithm works well for small text sizes, it becomes inefficient with large texts or complex patterns. Future improvements could include:

Implementing more efficient algorithms like the Knuth-Morris-Pratt (KMP) or Boyer-Moore algorithms.
Adding features like case-insensitive search and support for regular expressions.
Improving the user interface with additional features like file input for larger texts.