CSC-205: *Theory of Automata*
Project Report
Instructor: Sir Jawad Ahmed Bhutta
Date: 17 December 2025

Group Members

- 23FA-017-CS: **Javaria Owais**
- 23FA-021-CS: **Sahir Ul Hassan**

KMP-Based Mealy Machine Visualizer and String Matching Web Application

1. Introduction

String matching is a fundamental problem in computer science, with applications in text processing, search engines, compilers, and bioinformatics. Efficient pattern matching becomes crucial when large volumes of text must be analyzed or processed repeatedly. The Knuth–Morris–Pratt (KMP) algorithm is a classical approach to this problem, offering linear-time performance by preprocessing the pattern to avoid redundant comparisons.

A deeper insight into KMP reveals its connection to finite automata: the algorithm can be interpreted as a deterministic finite automaton (DFA) where each state corresponds to a matched prefix of the pattern. When outputs indicating successful matches are added to the transitions, this automaton becomes a Mealy machine. This project leverages that theoretical insight to create a web-based tool for both string matching and automaton visualization.

The primary aim of this project is to implement a KMP-based system that allows users to perform string operations efficiently while visualizing the underlying Mealy machine, bridging the gap between theory and practical software deployment.

2. Objectives

The project was developed with the following objectives in mind:

1. Implement the KMP algorithm for efficient string matching.
2. Construct a Mealy machine representation derived from the KMP failure function.
3. Provide dynamic visualization of automaton states and transitions.
4. Create a responsive web interface for user interaction.
5. Containerize the application for consistent deployment.
6. Deploy the application using cloud infrastructure to make it publicly accessible.

3. Theoretical Background

The string matching problem involves finding occurrences of a pattern `P` within a text `T`. A naive solution checks all possible alignments, resulting in **O(n·m)** time complexity for a text of length `n` and pattern of length `m`. The KMP algorithm improves this by preprocessing the pattern into a **Longest Prefix Suffix (LPS)** array. For each position in the pattern, the LPS array stores the length of the longest proper prefix that is also a suffix ending at that position. This allows the algorithm to skip previously matched characters upon a mismatch, achieving **O(n + m)** time complexity.

From an automata perspective, each prefix of the pattern corresponds to a state in a deterministic finite automaton. Transitions between states are determined by the input character, while fallback transitions are defined by the LPS array. Adding output values to transitions, indicating whether the accepting state is reached, yields a Mealy machine. This representation captures both the operational flow of the KMP algorithm and its theoretical structure.

4. System Architecture

The system follows a **client–server architecture**. The backend handles all computation and provides an API, while the frontend manages user interaction and visualization.

The backend, implemented in **Python** using **Flask**, exposes endpoints for string operations and automaton visualization. The frontend is built with **HTML, JavaScript, and Tailwind CSS** to provide a responsive interface. Users can submit a text and pattern, query containment or occurrence counts, and request Mealy machine diagrams dynamically. The server generates these diagrams using **Graphviz**, rendering the automaton with states, transitions, and outputs.

The architecture is containerized with **Docker**, ensuring consistent runtime behavior across environments. Deployment to **Google Cloud Run** allows scalable, serverless operation, with automatic HTTPS access and no server management required.

5. Implementation

The system is implemented as a client–server web application combining KMP-based string processing with Mealy machine visualization.

Backend.
The backend is written in **Python** using **Flask**, centralizing all algorithmic logic, including LPS computation, string operations, and Mealy machine construction. It exposes the following API endpoints:

- **`POST /api/contains`** – returns whether a pattern occurs in the text
- **`POST /api/count`** – returns the number of non-overlapping occurrences of a pattern
- **`POST /api/visualize`** – generates and returns a Mealy machine visualization for a pattern

Input validation ensures the text and pattern use only allowed alphabets (`a, b` or `0, 1`).

Frontend.
Implemented with **HTML, JavaScript, and Tailwind CSS**, the frontend communicates

asynchronously with the backend. Users can perform string matching operations and view dynamic visualizations without page reloads.

Visualization and Deployment.
The Mealy machine diagrams are created using **Graphviz**, with states representing matched prefix lengths and transitions labeled by input/output pairs. The application is containerized using **Docker** and deployed to **Google Cloud Run**, enabling a fully managed, scalable, and publicly accessible web service.

6. Testing and Validation

The application was tested extensively with patterns of varying lengths and texts containing different character sequences. Test cases verified:

- Correct computation of the LPS array
- Accurate results for `contains`, `count`, `startsWith`, and `endsWith` operations
- Proper generation of Mealy machine diagrams reflecting KMP transitions
- Successful deployment and accessibility on Cloud Run

The system consistently produced correct results and visualizations under all tested scenarios.

7. Limitations

The current implementation has the following limitations:

- The alphabet is restricted to `{a, b}` or `{0, 1}`.
- Very long patterns may produce complex visualizations that are difficult to interpret.
- Graphviz must be installed in the runtime container for visualization generation.

8. Future Work

Future enhancements could include:

- Support for arbitrary alphabets
- Overlapping pattern occurrence counts
- Interactive step-by-step automaton simulation

9. Conclusion

This project demonstrates the practical application of the KMP algorithm as both a string matching tool and a Mealy machine visualizer. By combining algorithmic rigor with a web-based interface and cloud deployment, it provides both educational value and a functional software tool. The architecture and implementation reflect industry best practices, offering a foundation for further extensions and research in automata visualization and pattern matching.