provadis Hochschule

WAB

Provadis School of International Management and Technology

Comparing Suffix Automata Against Suffix Arrays For Longest Common Substring Queries

Rubin Chempananickal James

rubin.chempananickal-james@stud-provadis-hochschule.de

Matriculation Number: D876

Department: Information Technology

Module: Algorithmen und Datenstrukturen

Reviewer: Prof. Dr. Volker Scheidemann

February 23, 2026

Abstract

The objective of this paper was to compare the performance of two algorithms, the Suffix Automaton and the Enhanced Suffix Array, for solving the Longest Common Substring problem in a pure Python implementation.

The algorithms were tested on multiple scenarios, from best case to worst case, and on multiple string lengths, from 100 to 10,000 characters, with multiple runs for each scenario and string length to ensure statistical significance of the results. Both the build time and the query time were measured, as well as the memory used both for building the data structure and for querying it.

The results showed that **so long as one is programming in pure Python**, an Enhanced Suffix Array offers practically zero advantages over a Suffix Automaton. The Suffix Automaton was substantially faster to build and to query, and although the Enhanced Suffix Array had an overall smaller index size, this would do very little in real-life memory-constrained applications, as its peak memory usage was much higher than that of the Suffix Automaton across all scenarios and string lengths.

This is potentially due to the nature of Python, as it is a high level interpreted language where the memory management is abstracted away from the programmer, leading to even simple data types like integers and lists having a much higher memory overhead than in lower level languages like C. Due to Python's memory management, the Enhanced Suffix Array would also lose its cache locality, which is the main reason for its compactness in lower level languages like C.

Abstrakt

Das Ziel dieser Arbeit war es, die Leistung von zwei Algorithmen, dem Suffix Automaton und dem Enhanced Suffix Array, bei der Lösung des Longest Common Substring-Problems in einer reinen Python-Implementierung zu vergleichen.

Die Algorithmen wurden in verschiedenen Szenarien getestet, von besten bis zu schlechtesten Fällen, und auf verschiedenen Stringlängen, von 100 bis 10.000 Zeichen, mit mehreren Durchläufen für jedes Szenario und jede Stringlänge, um die statistische Signifikanz der Ergebnisse sicherzustellen. Sowohl die Aufbauzeit als auch die Abfragezeit wurden gemessen, ebenso wie der für den Aufbau der Datenstruktur und für die Abfrage verwendete Speicher.

Die Ergebnisse zeigten, dass ***solange man in reinem Python programmiert****, ein Enhanced Suffix Array praktisch keine Vorteile gegenüber einem Suffix Automaton bietet. Der Suffix Automaton war wesentlich schneller im Aufbau und in der Abfrage, und obwohl der Enhanced Suffix Array insgesamt eine kleinere Indexgröße hatte, würde dies in realen, speicherbeschränkten Anwendungen nur sehr wenig ausmachen, da sein Spitzen-Speicherverbrauch in allen Szenarien und Stringlängen viel höher war als der des Suffix Automatons.*

Dies liegt vermutlich an der Natur von Python, da es eine hochstufige interpretierte Sprache ist, bei der die Speicherverwaltung von der Programmierer/in abstrahiert wird, was dazu führt, dass selbst einfache Datentypen wie Ganzzahlen und Listen einen viel höheren Speicher-Overhead haben als in Sprachen wie C. Aufgrund der Speicherverwaltung von Python würde der Enhanced Suffix Array auch seine Cache-Lokalität verlieren, was der Hauptgrund für seine Kompaktheit in Sprachen wie C ist.

Contents

List of Figures

List of Tables

Glossary

DNA Deoxyribonucleic Acid, the molecule that carries genetic information in living organisms. 1

RNA Ribonucleic Acid, a molecule that is transcribed from DNA and which then gets translated into proteins. 1

1 Introduction

1.1 Background

The Longest Common Substring (LCS) problem is a fundamental problem in computer science, with applications in multiple domains, bioinformatics being one of the most prominent. This is due to the fact that DNA and RNA sequences are usually encoded as strings for bioinformatics workflows, in a format called FASTA, originally described by Pearson and Lipman in 1985[1].

The LCS problem is defined as follows:

> Given two strings S and T, each of length at most n, the longest common substring (LCS) problem is to find a longest substring common to S and T.[2]

For example, given the strings "AG**CTAGC**" and "T**CTAGC**TA", the longest common substring is "CTAGC", which has a length of 5. The LCS problem can be solved using various algorithms, such as dynamic programming, suffix trees, suffix arrays, and suffix automata, each with different time and space complexities.

Of particular interest are the Suffix Automaton (SAM) (also known as Directed Acyclic Word Graph (DAWG)) and the Suffix Array (SA) with Longest Common Prefix (LCP) Array (also known as an Enhanced Suffix Array (ESA)), which both yield the search result in linear time, but differ in their construction time and space requirements.

1.2 Research Questions

[1] Lipman and Pearson 1985.
[2] Amir et al. 2020.

2 Methods

2.1 Design

2.2 Data Collection

Bibliography

Amir, Amihood et al. (2020). „Dynamic and internal longest common substring“. In: *Algorithmica* 82.12, pp. 3707–3743.

Lipman, David J. and William R. Pearson (1985). „Rapid and Sensitive Protein Similarity Searches“. In: *Science* 227.4693, pp. 1435–1441. DOI: 10.1126/science.2983426. eprint: https://www.science.org/doi/pdf/10.1126/science.2983426. URL: https://www.science.org/doi/abs/10.1126/science.2983426.

AI Declaration

The usage of AI tools within this project is documented here. I solemnly declare that I have documented all interactions with AI tools, including the prompts used and the outputs received.

System	Prompt	Usage
GitHub Copilot 1	Asked to delete the existing project and provide a simple LaTeX template with a shared preamble, a Chapters subdirectory, TOC, glossary, abbreviations, and Roman numerals for non-content pages.	Template structure and LaTeX setup provided
GitHub Copilot 2	Requested additional title page lines (WAB header, reviewer, module) for the main document.	Title page metadata and layout updated
GitHub Copilot 3	Requested uppercase Roman numerals for front matter and lowercase Roman numerals for back matter page numbering.	Page numbering adjusted in main and exposee
GitHub Copilot 4	Requested exposee title page to include shared info fields (WAB header, department, module, reviewer).	Exposee title page updated to include shared metadata
GitHub Copilot 5	Requested adding the provadis-hochschule.pdf logo to the top right corner of both main and exposee title pages.	Logo added to top right of both title pages
GitHub Copilot 6	Reported that Glossary and Abbreviations sections were missing from compiled output.	Added example `\newacronym` entries to abbreviations.tex
GitHub Copilot 7	Reported that glossary section still not appearing in final PDF.	Updated settings.tex to include `automake` option in glossaries package to enable automatic glossary generation

System	Prompt	Usage
GitHub Copilot 8	Requested Python benchmark code for LCS comparison with multiple synthetic scenarios (random strings, implanted mutated substrings, etc.)	Created benchmark script and documentation in Code folder
GitHub Copilot 9	Asked to add a progress-bar-like output.	Added optional single-line in-place progress output with percentage, completed/total steps, elapsed time, ETA, and `-progress`/`-no-progress` flags
GitHub Copilot 10	Reported that ETA was inaccurate due to varying string lengths	Reworked ETA estimation to be algorithm- and length-aware using observed per-bucket runtimes
GitHub Copilot 11	Requested separate build and query timing (and plots).	Refactored benchmark to record build/query time separately and added total time summaries
GitHub Copilot 12	Requested plots using statistics beyond the mean (median, standard deviation, etc.).	Expanded summary stats and generated mean+std and median+IQR plots alongside memory plots
GitHub Copilot 13	Requested splitting benchmarking and plotting so plot generation can run separately, with a shared CLI entry point and default run-then-plot behavior.	Added CLI mode switch (run/plot/both) to reuse saved CSV results for plotting without rerunning benchmarks
GitHub Copilot 14	Requested an error if the two algorithms return different LCS result strings for the same input.	Added strict substring equality check (in addition to length) and raise an error on mismatch
GitHub Copilot 15	Requested that, in case of multiple valid LCS results, both algorithms return all results and compare the sets.	Updated both algorithms to return all max-length substrings and compare result sets for correctness
GitHub Copilot 16	Requested refactoring the benchmark script into helper modules under a helpers subdirectory, with separate algorithm imports and plotting helper, and clean top-level imports in the main file.	Split benchmark code into dedicated helper modules (SAM, ESA, plotting, generation, statistics, benchmarking) and simplified main script orchestration
GitHub Copilot 17	Requested mean, median, standard deviation, and IQR for memory usage plots as well.	Added memory quartile aggregation and generated memory mean+SD plus median+IQR plots
GitHub Copilot 18	Asked for a practical memory/space-complexity metric and whether build-time and query-time memory can be measured separately.	Added separate build/query phase memory metrics and a persistent index-size metric, with aggregation and plots for comparison

System	Prompt	Usage
GitHub Copilot 19	Asked whether adding total time (build + query) columns and graphs with full statistics would make sense.	Added full total-time statistics (mean, median, std, IQR, Q1, Q3) to summaries and created dedicated total-time plots
GitHub Copilot 20	Requested moving plot legends away from the top center because they obscured titles.	Repositioned all figure legends to the right side outside the plotting area to keep titles unobstructed
GitHub Copilot 21	Requested an additional output file containing both generated strings and their LCS value(s).	Added export of one CSV per generated case with ‘s‘, ‘t‘, LCS length, and all LCS substrings
GitHub Copilot 22	Asked to replace the lambda with a tuple-key-based approach.	Replaced lambda-based sorting with a tuple-key-based approach for suffix array construction, improving performance
GitHub Copilot 23	Asked to refactor the plotting code to reduce repetition and make it more modular.	Refactored plotting code to use a single generic plotting function

Declaration of Authorship

I hereby confirm that I have personally and independently prepared the present work and have not used any sources or aids other than those specified. All passages taken verbatim or in substance from other sources are identified as such. The drawings, illustrations and tables in this work are created by me or have been provided with an appropriate source reference. This work has not been submitted by me to any other university in the same or similar form for the acquisition of an academic degree.

Frankfurt, February 23, 2026 .....

Rubin Chempananickal James