String Matching Algorithm Analysis Report

Name: Utkan DİNDAROĞLU
Number: 22050111042

December 6, 2025

1 Introduction

This report documents the implementation, design, and analysis of various string matching algorithms. The project involved implementing the Boyer-Moore algorithm, designing a custom "GoCrazy" algorithm, and creating a Pre-Analysis strategy to dynamically select the best algorithm based on input characteristics.

2 Boyer-Moore Implementation Approach

The Boyer-Moore implementation includes both the **Bad Character Heuristic** and the **Good Suffix Heuristic**.

Challenges and Optimizations

During the development process, we encountered a significant performance bottleneck and a crash related to Unicode characters.

- **The Issue:** Initially, the Bad Character table was implemented as an integer array of size 256. This caused an `ArrayIndexOutOfBoundsException` with Unicode inputs. Increasing the array size to 65,536 solved the crash but introduced massive initialization overhead (110,000 μs average execution time), making the algorithm slower than Naive.
- **The Solution:** We replaced the array with a `Java HashMap<Character, Integer>`. This allows us to map only the characters present in the pattern, reducing memory usage and initialization time drastically. The average time dropped to approximately 8,000 μs after this fix.

3 GoCrazy Algorithm Design (Tunneled Sunday-Raita)

The custom "GoCrazy" algorithm is a hybrid design combining the **Sunday** and **Raita** heuristics to maximize efficiency for standard text.

Design Rationale

1. **Sunday's Shift Rule:** unlike Boyer-Moore, which looks at the mismatch character, Sunday's rule looks at the character *immediately after* the current window to determine the jump distance. This often allows for larger shifts.
2. **Raita's Comparison Order:** Instead of scanning characters linearly (Left-to-Right or Right-to-Left), we compare in a probabilistic order to fail fast:

- Check the **Last** character (highest probability of mismatch).
- Check the **First** character.
- Check the **Middle** character.
- Check the rest.

3. **Internal Hybridization:** The algorithm includes an internal check. If the pattern length is tiny (≤ 2), it reverts to a simple loop to avoid the overhead of building the shift table.

4 Pre-Analysis Strategy

The goal of the `StudentPreAnalysis` class is to predict the fastest algorithm without scanning the full text (which would cost $O(N)$).

Selection Logic

1. **The "Tiny" Rule:** If the text length is < 500 or pattern length is ≤ 4, the overhead of initializing tables (for KMP/BM) usually outweighs the speed benefits. We select **Naive**.
2. **Small Alphabet Rule:** We scan the first 20 characters of the pattern. If fewer than 5 unique characters are found (suggesting DNA or Binary), we select **KMP** to avoid hash collisions or poor shifting behavior.
3. **Default:** For all other cases, we select **GoCrazy**.

5 Analysis of Results

Based on the final benchmark output:

Performance Overview

- **Naive Dominance:** Surprisingly, the Naive algorithm won the majority of test cases (Avg 6,027 μs). This is likely due to the small size of the test inputs and JVM optimizations for simple loops.
- **GoCrazy vs. The Rest:** GoCrazy (Avg 7,075 μs) successfully outperformed both KMP (7,390 μs) and Boyer-Moore (8,006 μs) on average.
- **The "Very Long Text" Victory:** The most significant result was the "Very Long Text" case:
  - **GoCrazy:** 16,960 μs (Winner)
  - **Naive:** 36,260 μs
  - **Boyer-Moore:** 41,240 μs
  - **KMP:** 48,540 μs

  This proves that for large-scale inputs, the overhead of the Sunday-Raita heuristic pays off significantly, achieving a speedup of over **2x** compared to Naive.

Critique of Pre-Analysis

The console output indicates that our Pre-Analysis selected "Naive" for almost every test case. While this resulted in a win for small inputs, it failed to switch to GoCrazy for the "Very Long Text" case, resulting in a performance loss (PreAnalysis was slower than GoCrazy by roughly 4.56 μs in that instance). Future improvements should lower the text-length threshold to favor GoCrazy earlier.

6 My Journey

This assignment was a significant learning experience, particularly in understanding the gap between theoretical complexity and real-world performance.

Key Learnings & Challenges:

- **The "Initialization" Trap:** The hardest challenge was understanding why my Boyer-Moore implementation was 20x slower than Naive. I learned that allocating a large array (size 65,536) for Unicode support has a massive hidden cost that destroys performance for short texts. Switching to a HashMap was a crucial lesson in memory management.
- **The Complexity of "Good Suffix":** Implementing the full Boyer-Moore Good Suffix rule was the most difficult coding task. I faced an "off-by-one" bug where the suffix table was misaligned, causing the search to fail at pattern boundaries. Tracing this required step-by-step debugging.
- **Creativity in Algorithms:** Designing "GoCrazy" was the most enjoyable part. I learned that combining different heuristics (Sunday's shift + Raita's comparison) can create a solution that beats standard library algorithms in specific contexts.

7 Documentation & Transparency

Research & Resource Usage: This project was developed using a combination of academic resources, online communities, and AI assistance.

- **Online Resources:** Concepts regarding string matching heuristics (Bad Character, Good Suffix, Sunday, Raita) were researched across various technical platforms including **GeeksForGeeks**, **Medium**, and **StackOverflow**. Discussions on **Reddit** and **HackerRank** provided insights into edge-case handling, while **YouTube** tutorials helped visualize the sliding window mechanisms.
- **AI Assistance:** AI tools were utilized as a "thought partner" and debugging assistant throughout the project. Specifically, AI assistance was used for:
  - **Debugging:** Helping trace the complex "off-by-one" indexing error in the Boyer-Moore Good Suffix table construction.
  - **Planning & Logic:** Brainstorming the "GoCrazy" hybrid approach by suggesting the combination of Sunday's shift logic with Raita's comparison order.
  - **Optimization:** Identifying the initialization bottleneck of the size-65536 array and suggesting the `HashMap` alternative to fix the performance regression.
  - **Formatting:** Generating the LaTeX structure for this report.