Ankara Yıldırım Beyazıt University

Department of Computer Engineering

CENG303 - DESIGN AND ANALYSIS OF ALGORITHMS

String Matching Algorithms - Analysis Report

Muzaffer Batmaz 2205011021

Mikail Kaçmaz 22050111080

Instructor:Fatih Nar

Date: 07/12/2025

Introduction

This report presents the analysis, design rationale, and decision strategies used within the String Matching Project. The primary objective of this project is to evaluate and compare several string matching algorithms—both classical and custom-designed—based on their implementation characteristics and practical performance.
The project includes implementations of Boyer–Moore, KMP, Naive Matching, Rabin–Karp, and a newly created heuristic algorithm named *GoCrazy*.
Additionally, an automated rule-based analysis module, StudentPreAnalysis, is used to select the most suitable algorithm for a given input before executing any actual matching.

The following sections describe each algorithm in detail exactly as originally written.

1. Boyer–Moore Implementation Approach

In this project, the Boyer–Moore algorithm is implemented to correctly handle edge cases such as an empty pattern or a pattern longer than the text. The core of the implementation relies on the bad character heuristic, which is one of the main mechanisms that allow Boyer–Moore to skip large portions of the text.

First, the pattern is scanned to build a table recording the last occurrence of each character within the pattern. Instead of assuming a fixed alphabet, the table size is determined dynamically based on the highest character value observed in the pattern. This ensures that memory usage adapts to the actual content of the pattern.

During the search phase, the pattern is compared against the text from right to left. When a full match is found, the match position is added to the result. If a mismatch occurs, the algorithm checks where the mismatching text character last appears in the pattern and shifts the pattern accordingly. If the character does not appear in the pattern, it shifts by the maximum amount, allowing a potentially large jump forward.

This design ensures:

- Proper handling of special cases,
- A dynamic bad-character table built specifically for the given pattern,
- Efficient skipping behavior via right-to-left comparison,
- Strong practical performance, particularly for large inputs.

2. GoCrazy Algorithm – Design and Rationale

The GoCrazy algorithm is a custom heuristic string matching method designed specifically for this project. Its purpose is to quickly eliminate impossible matching positions through lightweight checks, especially for medium-length patterns.

Like Boyer–Moore, GoCrazy first handles trivial cases such as an empty pattern or a pattern longer than the text. After that, for each candidate position, it uses a three-stage filtering process:

First-character check

If the first character of the pattern does not match the corresponding text character, the position is immediately skipped.

Last-character check

If the last character of the pattern does not match the last character in the examined text window, the position is skipped as well.

Block-wise middle comparison

Only if both the first and last characters match, the algorithm continues by comparing the middle section of the pattern. It performs this comparison in small blocks, stopping as soon as a mismatch is found.

This approach allows GoCrazy to:

- Eliminate many candidate positions through cheap comparisons,
- Stop early when mismatches occur,
- Perform efficiently on patterns with diverse characters and medium length.

Although the worst-case performance matches that of the naive algorithm, the typical performance is significantly better due to the aggressive filtering strategy. GoCrazy is especially suited for patterns of moderate length that do not exhibit repetitive structure.

3. Pre-Analysis Strategy (StudentPreAnalysis)

The StudentPreAnalysis component implements a rule-based decision strategy for selecting the most appropriate string matching algorithm based on properties of the text and pattern. The goal is to avoid running all algorithms on every input and instead pick a strong candidate through lightweight analysis.The decision strategy follows these principles:

Very short patterns or trivial cases → Naive

For pattern lengths of 3 or fewer, the naive algorithm is more efficient because the preprocessing cost of advanced algorithms is not justified.

Medium-length patterns without special structure → GoCrazy

When the pattern length is between 4 and 50 and does not show repetitive characteristics, GoCrazy's filtering mechanism performs well and becomes the preferred choice.

Highly repetitive patterns → KMP

If the pattern consists of repeated characters or contains repeated prefixes, KMP is the ideal choice due to its prefix function and reuse of partial match information.

Patterns with a rare last character → Boyer–Moore

If the final character of the pattern occurs infrequently within the pattern, Boyer–Moore can perform large skips, making it highly efficient.

Very large text with a sufficiently long pattern → Boyer–Moore

For long inputs, Boyer–Moore is a strong general-purpose choice because of its ability to skip large sections.

Both text and pattern large → Rabin–Karp

When both pattern and text lengths are large, hashing-based approaches like Rabin–Karp can provide excellent performance in practice.

Default choice → Boyer–Moore

If none of the specific conditions are triggered, Boyer–Moore is selected as a robust, general-purpose algorithm.

The overall intention of this pre-analysis approach is to:

- Match algorithm choice to the structural characteristics of the pattern and text,
- Minimize unnecessary computation,
- Exploit KMP's efficiency on repetitive patterns,
- Utilize Boyer–Moore's skipping power on long inputs,
- Apply GoCrazy for mid-range cases where its heuristics excel.

4. Results Analysis

The performance results show that the pre-analysis strategy generally selected the most appropriate algorithm for each test case. GoCrazy performed very well on medium-sized and moderately complex patterns, such as simple matches, overlapping patterns, and repeating patterns. In these cases, the pre-analysis often matched or improved upon GoCrazy's direct runtime, since the selection overhead was minimal.

For trivial or very short patterns, such as single-character inputs or empty text/pattern cases, the strategy correctly chose the Naive algorithm. These scenarios do not benefit from preprocessing, and the results confirm that Naive consistently executed fastest.

When the pattern was highly repetitive, the pre-analysis selected KMP, which proved accurate. KMP handled uniform or prefix-repeating patterns much faster, validating the rule-based selection.

The strategy also successfully identified cases where Boyer–Moore is optimal, particularly on long texts or patterns with diverse character distributions. In tests like Very Long Text, Long Pattern, and Unicode Characters, Boyer–Moore significantly outperformed the other algorithms, and the pre-analysis captured these cases correctly.

Some test cases showed small slowdowns (marked in red), which occurred mainly when multiple algorithms performed similarly and the pre-analysis overhead outweighed the performance difference. These overheads were minor and did not affect the overall usefulness of the approach.

Overall, the results demonstrate that the pre-analysis module is effective and reliable, correctly leveraging the strengths of Naive, GoCrazy, KMP, and Boyer–Moore in most scenarios. It provides clear performance benefits across the test suite and justifies the design decisions made in the selection strategy.

5. Our Journey: Experience and Challenges

This project presented both intellectual challenge and significant learning opportunities. While implementing the core algorithms provided a deep understanding of string matching mechanics, the most demanding aspects were algorithm selection and optimization.

Key Challenges Faced:

- **Boyer-Moore Complexity (Good Suffix Rule):** Although the fundamental logic of Boyer-Moore was understood, implementing the **Good Suffix Rule** proved challenging and overly complex. Consequently, we focused solely on the **Bad Character Rule** for the final implementation, as it offered a more straightforward and achievable approach while still providing a strong level of efficiency.

- **Pre-Analysis Algorithm Conflict:** The greatest difficulty lay in designing the StudentPreAnalysis to accurately choose between the most efficient algorithms (KMP, Boyer-Moore, and the custom GoCrazy).
    - Initially, KMP and GoCrazy often claimed scenarios that were optimally solved by Boyer-Moore, leading to Boyer-Moore being rarely selected.
    - Similarly, GoCrazy often intercepted cases that KMP or Boyer-Moore should have handled, resulting in numerous incorrect (red-flagged) time results.
- **The Rabin-Karp Dilemma:** Despite fine-tuning the pre-analysis logic, we were consistently unable to find reliable, low-overhead heuristics that favored the selection of the Rabin-Karp algorithm effectively within the given test set.

Learning and Outcomes:

- **Algorithm Logic Deep Dive:** The implementation of Boyer-Moore successfully deepened our understanding of its right-to-left comparison and its ability to achieve large text skips, even using only the Bad Character heuristic.
- **Heuristic Design:** Creating the GoCrazy algorithm (which uses first/last character checks and block-wise comparison) was a rewarding exercise in designing heuristics to reduce loop iterations for specific pattern types.
- **Strategy Refinement:** Through iterative testing and the aid of **external tools (AI assistance)**, we developed specific, non-overlapping criteria based on pattern characteristics (e.g., repetitiveness for KMP, non-repetitiveness for GoCrazy, last-character rarity for Boyer-Moore). This refinement significantly improved the accuracy, moving initial "red-colored" (slow/incorrectly chosen) test results to accurate selections.

Conclusion:

The project was insightful and highly demanding. It successfully highlighted that the theoretical O(N) or O(N+M) complexity of advanced algorithms is often less important than the practical overhead of preprocessing and the ability to choose the *right* $O(N)$ algorithm for a specific input structure. The process of balancing algorithm strengths to create a robust pre-analysis system was both challenging and ultimately satisfying.

6. Research Process and References

The project relied on external resources and tools to ensure optimal performance, particularly in the most complex phases of implementation and analysis.

Research Methodology and Tool Use

- **Boyer-Moore Implementation:** The Bad Character Rule was implemented based on classical algorithm documentation. The more complex Good Suffix Rule was intentionally omitted, as research indicated its high implementation overhead.
- **Pre-Analysis Strategy (StudentPreAnalysis):** The greatest challenge was resolving conflicts where KMP, Boyer-Moore, and GoCrazy could all potentially be the

fastest algorithm. To achieve high accuracy, **AI assistance (Large Language Model - LLM)** was utilized.
    - The LLM helped to define specialized, non-overlapping heuristics, such as analyzing the **rarity of the pattern's last character** to trigger the efficient Boyer-Moore and focusing on **prefix repetition density** to confirm KMP selection.
    - This systematic approach corrected initial inaccuracies (red-marked results) and increased the overall number of fastest algorithm selections.
- **Rabin-Karp:** Due to the lack of low-overhead heuristics that reliably favored Rabin-Karp in the given test set, it remained the least-chosen algorithm in the final strategy.

References

1. **Classical Algorithm Documentation:** Used for theoretical validation of time complexities and core algorithm mechanics (KMP, Boyer-Moore).
2. **External Learning Resources:** Consulted for implementation details and edge case handling of the Bad Character Rule.
3. **Large Language Model (LLM) Assistance:** Utilized for developing specialized and differentiating heuristics within the `StudentPreAnalysis` logic.