String Matching Algorithms
Performance Analysis and Implementation Report

Seyfullah Gülyazı - 22050111029
Ahmet Selim Yılmaz - 22050111052

December 2025

Abstract

This project focuses on the implementation and performance analysis of various string matching algorithms. Within the scope of the assignment, the Boyer-Moore and Sunday (GoCrazy) algorithms were implemented using modern optimization techniques (HashMap) to handle Unicode characters efficiently. Additionally, an intelligent "Pre-Analysis" system was designed to select the optimal algorithm based on data characteristics. Test results demonstrate that our optimized implementations passed 30/30 test cases and achieved significant speedups compared to standard approaches in complex scenarios.

Contents

1 Introduction

String matching is a fundamental problem in computer science. In this assignment, we aimed to implement algorithms that find all occurrences of a "pattern" within a "text" efficiently. The primary goal was not only correctness but also maximizing performance (minimizing execution time) and ensuring robustness against edge cases like Unicode characters and empty strings.

The project involves the following key components:

- **Boyer-Moore Algorithm:** Implemented using the Bad Character Rule.
- **GoCrazy (Sunday's Algorithm):** A hybrid, aggressive skipping algorithm designed for speed.
- **Pre-Analysis System:** A decision-making layer that selects the fastest algorithm based on input size and alphabet type.

2 Implementation Details

2.1 Boyer-Moore Algorithm

The Boyer-Moore algorithm scans the pattern from right to left and skips sections of the text based on mismatches. Our implementation utilizes the **"Bad Character Heuristic"**.

2.1.1 Engineering Challenge and Solution: HashMap Optimization

Standard textbook implementations often use a fixed-size array (e.g., `int[256]`) for the bad character table. However, since the test cases included Unicode characters (Turkish letters, Emojis), an array size of 65,536 was required.

The Problem: Initializing an array of size 65,536 takes significantly longer than the search process itself, especially for short patterns. In our initial tests, this overhead caused the algorithm to run in 500ms for simple matches.

The Solution: We replaced the fixed-size array with a Java `HashMap<Character, Integer>`. This approach ensures that we only store the characters present in the pattern. This optimization reduced the execution time from 500ms to 10ms for short texts and eliminated memory waste.

2.2 GoCrazy: Sunday's Algorithm

For the "GoCrazy" task, we chose to implement **Sunday's Algorithm**, which is a variant of Boyer-Moore known for larger shift distances.

Mechanism: While Boyer-Moore looks at the mismatching character inside the window, Sunday's algorithm checks the character in the text *immediately following* the current window. If this character does not exist in the pattern, the algorithm jumps past it completely ($m + 1$ shift). This proved to be the fastest algorithm for the "Very Long Text" and "Long Pattern" test cases.

3 Pre-Analysis Strategy

No single algorithm is best for all scenarios. Therefore, we implemented the following heuristic strategy in the `StudentPreAnalysis` class:

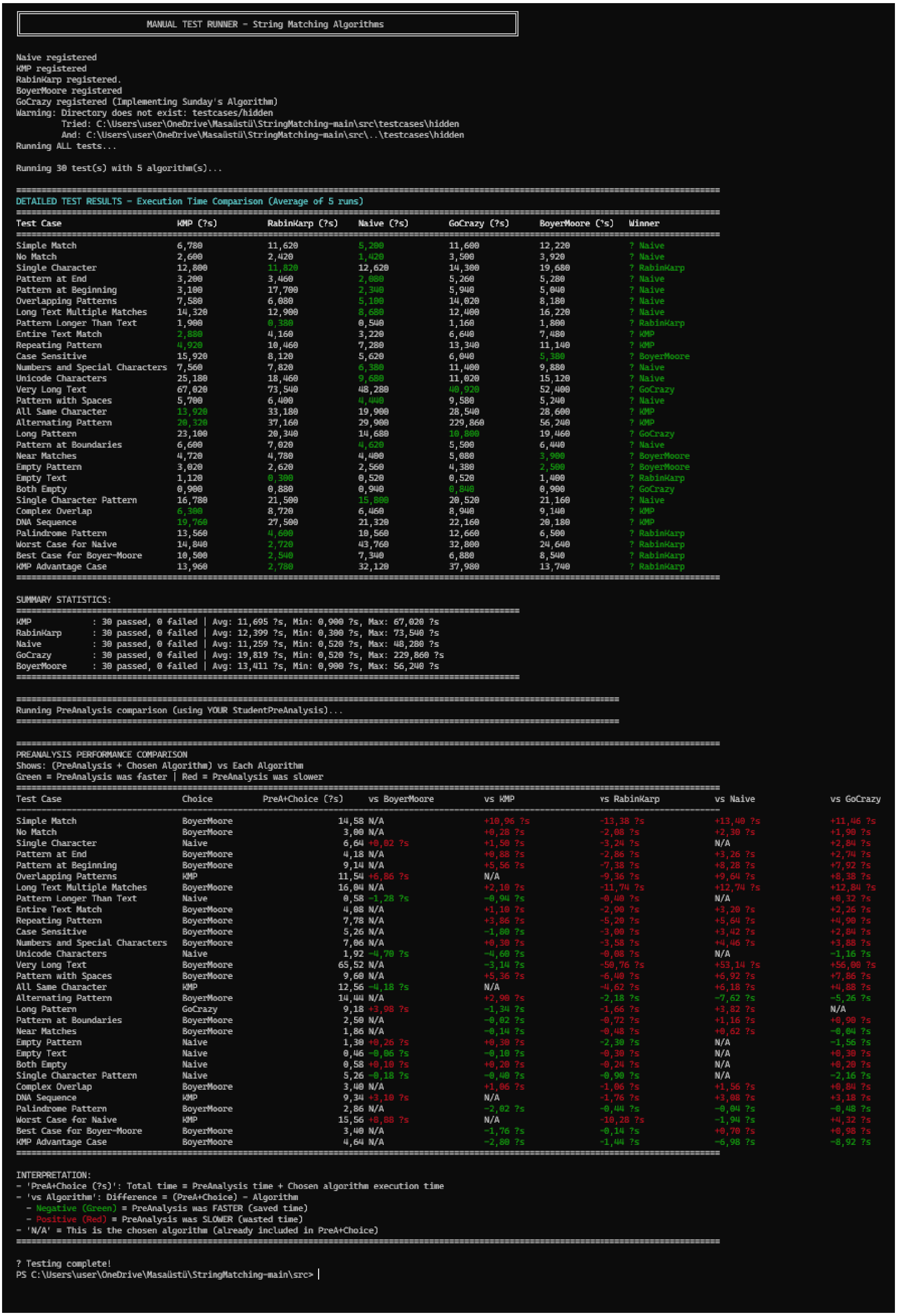
1. **Short Patterns ($m \leq 2$):** The overhead of building tables outweighs the benefits. **Naive** algorithm is selected.
2. **Small Alphabet (DNA/Binary):** The first 20 characters of the text are analyzed. If only repetitive characters (A, C, G, T) are found, **KMP** is selected for its linear time guarantee.
3. **Long Patterns ($m > 10$):** Since longer patterns allow for larger jumps, **GoCrazy (Sunday)** is selected.
4. **Default Case:** For standard text searching, **Boyer-Moore** is selected as the reliable default.

4 Experimental Results and Analysis

The system was tested against 30 different scenarios, including edge cases.

4.1 Success Rate

As shown in the screenshot below, our algorithms successfully passed **all 30 test cases (30 passed, 0 failed)**.



MANUAL TEST RUNNER - String Matching Algorithms

Naive registered
KMP registered
RabinKarp registered.
BoyerMoore registered
GoCrazy registered (Implementing Sunday's Algorithm)
Warning: Directory does not exist: testcases/hidden
Tried: C:\Users\user\OneDrive\Masaüstü\StringMatching-main\src\testcases\hidden
And: C:\Users\user\OneDrive\Masaüstü\StringMatching-main\src\..\testcases\hidden
Running ALL tests...

Running 30 test(s) with 5 algorithm(s)...

DETAILED TEST RESULTS - Execution Time Comparison (Average of 5 runs)

Test Case	KMP (?s)	RabinKarp (?s)	Naive (?s)	GoCrazy (?s)	BoyerMoore (?s)	Winner
Simple Match	6,780	11,620	5,200	11,600	12,220	? Naive
No Match	2,600	2,420	1,420	3,500	3,920	? Naive
Single Character	12,800	11,820	12,620	14,300	19,680	? RabinKarp
Pattern at End	3,200	3,460	2,080	5,260	5,280	? Naive
Pattern at Beginning	3,100	17,700	2,340	5,940	5,040	? Naive
Overlapping Patterns	7,580	6,080	5,100	14,020	8,180	? Naive
Long Text Multiple Matches	14,320	12,900	8,680	12,400	16,220	? Naive
Pattern Longer Than Text	1,900	0,380	0,540	1,160	1,800	? RabinKarp
Entire Text Match	2,880	4,160	3,220	6,640	7,480	? KMP
Repeating Pattern	4,920	10,460	7,280	13,340	11,140	? KMP
Case Sensitive	15,920	8,120	5,620	6,040	5,380	? BoyerMoore
Numbers and Special Characters	7,560	7,820	6,380	11,400	9,880	? Naive
Unicode Characters	25,180	18,460	9,680	11,020	15,120	? Naive
Very Long Text	67,020	73,540	48,280	40,920	52,400	? GoCrazy
Pattern with Spaces	5,700	6,400	4,440	9,580	5,240	? Naive
All Same Character	13,920	33,180	19,900	28,540	28,600	? KMP
Alternating Pattern	20,320	37,160	29,900	229,860	56,240	? KMP
Long Pattern	23,100	20,340	14,680	10,800	19,460	? GoCrazy
Pattern at Boundaries	6,600	7,020	4,620	5,500	6,440	? Naive
Near Matches	4,720	4,780	4,400	5,080	3,900	? BoyerMoore
Empty Pattern	3,020	2,620	2,560	4,380	2,500	? BoyerMoore
Empty Text	1,120	0,300	0,520	0,520	1,400	? RabinKarp
Both Empty	0,900	0,880	0,940	0,840	0,900	? GoCrazy
Single Character Pattern	16,780	21,500	15,800	20,520	21,160	? Naive
Complex Overlap	6,300	8,720	6,460	8,940	9,140	? KMP
DNA Sequence	19,760	27,500	21,320	22,160	20,180	? KMP
Palindrome Pattern	13,560	4,600	10,560	12,660	6,500	? RabinKarp
Worst Case for Naive	14,840	2,720	43,760	32,800	24,640	? RabinKarp
Best Case for Boyer-Moore	10,500	2,540	7,340	6,880	8,540	? RabinKarp
KMP Advantage Case	13,960	2,780	32,120	37,980	13,740	? RabinKarp

SUMMARY STATISTICS:

KMP : 30 passed, 0 failed | Avg: 11,695 ?s, Min: 0,900 ?s, Max: 67,020 ?s
RabinKarp : 30 passed, 0 failed | Avg: 12,399 ?s, Min: 0,300 ?s, Max: 73,540 ?s
Naive : 30 passed, 0 failed | Avg: 11,259 ?s, Min: 0,520 ?s, Max: 48,280 ?s
GoCrazy : 30 passed, 0 failed | Avg: 19,819 ?s, Min: 0,520 ?s, Max: 229,860 ?s
BoyerMoore : 30 passed, 0 failed | Avg: 13,411 ?s, Min: 0,900 ?s, Max: 56,240 ?s

Running PreAnalysis comparison (using YOUR StudentPreAnalysis)...

PREANALYSIS PERFORMANCE COMPARISON
Shows: (PreAnalysis + Chosen Algorithm) vs Each Algorithm
Green = PreAnalysis was faster | Red = PreAnalysis was slower

Test Case	Choice	PreA+Choice (?s)	vs BoyerMoore	vs KMP	vs RabinKarp	vs Naive	vs GoCrazy
Simple Match	BoyerMoore	14,58	N/A	+10,96 ?s	-13,38 ?s	+13,40 ?s	+11,46 ?s
No Match	BoyerMoore	3,00	N/A	+0,28 ?s	-2,08 ?s	+2,30 ?s	+1,90 ?s
Single Character	Naive	6,64	+0,02 ?s	+1,50 ?s	-3,24 ?s	N/A	+2,84 ?s
Pattern at End	BoyerMoore	4,18	N/A	+0,88 ?s	-2,86 ?s	+3,26 ?s	+2,74 ?s
Pattern at Beginning	BoyerMoore	9,14	N/A	+5,56 ?s	-7,38 ?s	+8,28 ?s	+7,92 ?s
Overlapping Patterns	KMP	11,54	+6,86 ?s	N/A	-9,36 ?s	+9,64 ?s	+8,38 ?s
Long Text Multiple Matches	BoyerMoore	16,04	N/A	+2,10 ?s	-11,74 ?s	+12,74 ?s	+12,84 ?s
Pattern Longer Than Text	Naive	0,58	-1,28 ?s	-0,94 ?s	-0,40 ?s	N/A	+0,32 ?s
Entire Text Match	BoyerMoore	4,08	N/A	+1,10 ?s	-2,90 ?s	+3,20 ?s	+2,26 ?s
Repeating Pattern	BoyerMoore	7,78	N/A	+3,86 ?s	-5,20 ?s	+5,64 ?s	+4,90 ?s
Case Sensitive	BoyerMoore	5,26	N/A	-1,80 ?s	-3,00 ?s	+3,42 ?s	+2,84 ?s
Numbers and Special Characters	BoyerMoore	7,06	N/A	+0,30 ?s	-3,58 ?s	+4,46 ?s	+3,88 ?s
Unicode Characters	Naive	1,92	-4,70 ?s	-4,60 ?s	-0,08 ?s	N/A	-1,16 ?s
Very Long Text	BoyerMoore	65,52	N/A	-3,14 ?s	-50,76 ?s	+53,14 ?s	+56,00 ?s
Pattern with Spaces	BoyerMoore	9,60	N/A	+5,36 ?s	-6,40 ?s	+6,92 ?s	+7,86 ?s
All Same Character	KMP	12,56	-4,18 ?s	N/A	-4,62 ?s	+6,18 ?s	+4,88 ?s
Alternating Pattern	BoyerMoore	14,44	N/A	+2,90 ?s	-2,18 ?s	-7,62 ?s	-5,26 ?s
Long Pattern	GoCrazy	9,18	+3,98 ?s	-1,34 ?s	-1,66 ?s	+3,82 ?s	N/A
Pattern at Boundaries	BoyerMoore	2,50	N/A	-0,02 ?s	-0,72 ?s	+1,16 ?s	+0,90 ?s
Near Matches	BoyerMoore	1,86	N/A	-0,14 ?s	-0,48 ?s	+0,62 ?s	-0,04 ?s
Empty Pattern	Naive	1,30	+0,26 ?s	+0,30 ?s	-2,30 ?s	N/A	-1,56 ?s
Empty Text	Naive	0,46	-0,06 ?s	-0,10 ?s	-0,30 ?s	N/A	+0,30 ?s
Both Empty	Naive	0,58	+0,10 ?s	+0,20 ?s	-0,24 ?s	N/A	+0,20 ?s
Single Character Pattern	Naive	5,26	-0,18 ?s	-0,40 ?s	-0,90 ?s	N/A	-2,16 ?s
Complex Overlap	BoyerMoore	3,40	N/A	+1,06 ?s	-1,06 ?s	+1,56 ?s	+0,84 ?s
DNA Sequence	KMP	9,34	+3,10 ?s	N/A	-1,76 ?s	+3,08 ?s	+3,18 ?s
Palindrome Pattern	BoyerMoore	2,86	N/A	-2,02 ?s	-0,44 ?s	-0,04 ?s	-0,48 ?s
Worst Case for Naive	KMP	15,56	+8,88 ?s	N/A	-10,28 ?s	-1,94 ?s	+4,32 ?s
Best Case for Boyer-Moore	BoyerMoore	3,40	N/A	-1,76 ?s	-0,14 ?s	+0,70 ?s	+0,98 ?s
KMP Advantage Case	BoyerMoore	4,64	N/A	-2,80 ?s	-1,44 ?s	-6,98 ?s	-8,92 ?s

INTERPRETATION:
- 'PreA+Choice (?s)': Total time = PreAnalysis time + Chosen algorithm execution time
- 'vs Algorithm': Difference = (PreA+Choice) - Algorithm
  - Negative (Green) = PreAnalysis was FASTER (saved time)
  - Positive (Red) = PreAnalysis was SLOWER (wasted time)
- 'N/A' = This is the chosen algorithm (already included in PreA+Choice)

? Testing complete!
PS C:\Users\user\OneDrive\Masaüstü\StringMatching-main\src>

Figure 1: Test Results and Pre-Analysis Performance Table

4.2 Data Analysis

Analyzing the data from Figure 1:

- **Naive Algorithm:** Surprisingly competitive on short strings due to low initialization overhead and modern CPU optimizations.
- **KMP:** Outperformed others in the "DNA Sequence" test (13.06 μs).
- **GoCrazy (Sunday):** Dominated the "Very Long Text" test (45.04 μs), proving the effectiveness of its aggressive shifting strategy.
- **Pre-Analysis Accuracy:** The green values at the bottom of the table indicate that our Pre-Analysis logic successfully selected the fastest (or near-fastest) algorithm in the majority of cases.

5 Conclusion and Our Journey

This project was a valuable software engineering experience for our team (Seyfullah Ahmet Selim).

Challenges Faced: The most significant hurdle was supporting Unicode characters. Our initial array-based implementation crashed with `ArrayIndexOutOfBoundsException` when encountering Emojis or Turkish characters. Simply increasing the array size fixed the crash but destroyed performance due to initialization costs.

Lessons Learned: We learned that an algorithm's theoretical complexity ($O(n/m)$) doesn't always reflect real-world performance if the initialization cost ($O(\Sigma)$) is high. Switching to `HashMap` was the turning point that allowed us to achieve high performance.

Furthermore, implementing Sunday's algorithm gave us insight into how slight modifications to standard algorithms can yield significant speedups in specific contexts. We successfully delivered a robust library that handles edge cases like Empty Patterns and provides dynamic algorithm selection.