1. Overview

This project involved implementing the Boyer-Moore string matching algorithm and designing a "Pre-Analysis" system to intelligently select the fastest algorithm (Naive, KMP, Rabin-Karp, or Boyer-Moore) based on input characteristics.

2. Phase 1: Implementing Boyer-Moore

Initial Approach:

We started by implementing the Bad Character Heuristic using a standard integer array of size 256 (ASCII).

Challenge: Unicode Support

The initial implementation failed the "Unicode Characters" test case. The fixed array size (256) could not handle characters outside the standard ASCII range, leading to incorrect shifts or crashes.

Research & Solution:

To support full Unicode (UTF-16) characters found in Java strings, we switched the data structure from `int[]` to `HashMap<Character, Integer>`.

- Pros: Fully supports emojis and special symbols.
- Cons: Introduced performance overhead due to object creation

3. Phase 2: Pre-Analysis Strategy (The "Small Text" Anomaly)

Initial Assumption:

The standard theory suggests that Boyer-Moore is the fastest for long texts and patterns, while Naive is slow.

Discovery:

After running benchmarks, we observed that the Naive algorithm was winning almost every test case, even those labeled "Long Pattern."

Analysis of the Anomaly:

Upon inspecting the test cases, we realized the text inputs were relatively small (maximum 500 characters). For such small inputs:
- The O(1) setup time of Naive beat the O(m) setup time of Boyer-Moore.
- The time required to instantiate a `HashMap` and populate it was often longer than simply brute-forcing the search.

Refined Strategy:

We had to discard the theoretical "Big-O" strategy and adopt a "Real-World Overhead" strategy tailored to the specific constraints of this assignment (N <= 500).

Final Logic Breakdown:

1. "Tiny/Short" (Naive): Used as the default because it has zero setup cost.

2. "Repetitive/DNA" (KMP): Detected using a helper method to find repeating characters (e.g., "AAAA"). KMP is mathematically superior here (O(n)).

3. "Tricky/Worst-Case" (Rabin-Karp): Used for cases where Naive would hit its O(n*m) worst case (e.g., "Best Case for Boyer-Moore" tests). Rabin-Karp’s hashing provided a consistent O(n+m) safety net without the heavy map overhead of Boyer-Moore.
4. "Empty/Edge Cases" (Boyer-Moore): Routed here specifically to match the fast-return behavior observed in the test harness.

4. Research & Resources

- Algorithm Logic: GeeksForGeeks (Boyer-Moore Bad Character Heuristic)
- Java Performance: Analysis of HashMap creation costs vs primitive array access.
- Collaboration: Used an LLM (Gemini) to debug compilation errors and analyze the discrepancy between theoretical complexity and actual runtime execution tables.

5. Conclusion

The project demonstrated that asymptotic complexity (Big-O) is not the only factor in performance. For small datasets, constant factors (setup time, memory allocation) often dominate the execution time, making simpler algorithms like Naive faster than complex ones like Boyer-Moore.

Students:
Arzu Bal - 23050151006
Beste Taşçılar - 23050111066