22050111005 - Aybüke Karaçavuş
220501110053 - Rümeysa Kara

# String Matching Algorithms Homework Report

First, we reviewed the missing topics in string matching algorithms. We refreshed our understanding of how these algorithms work and scanned available resources as much as possible (YouTube videos, Medium blogs, and LLMs).
In particular, we focused on gaining a deeper understanding of the Boyer–Moore algorithm.

We learned in which scenarios each algorithm has advantages, and this provided us with foresight while designing the GoCrazy algorithm.

Before moving on to the implementation part of the assignment, the most time-consuming part for us was understanding the class structure. Once we grasped the logic between the classes, the assignment did not seem very difficult. On the contrary it turned out to be a useful task where we could concretely observe which algorithms are advantageous under which conditions.

**Boyer–Moore İmplementation:**
Boyer–Moore works by scanning the text from right to left. We improved the algorithm by using the bad character table and the good suffix heuristic. While scanning from right to left, when a mismatch occurs, we shift the pattern according to the table. It produces fast results when the alphabet size is large and the text length is long.

**GoCrazy Algorithm:**
For each test case, we selected the most suitable algorithm by considering the pattern length, text length, and alphabet size.
- Naive: if the pattern length is shorter than 4
- KMP: if the pattern contains repetitive substrings
- Rabin–Karp: if the text is long and the alphabet size is large
- Naive: at the end, if we are unsure

**Pre-Analysis Strategy:**
We followed an approach similar to GoCrazy. Before starting the assignment, we had already studied under which conditions each algorithm works more efficiently and

faster. We attempted to apply this knowledge during the pre-analysis phase as well. However, we are not fully confident that we made the correct choices.
- Short pattern and short text -> Naive
- Pattern with internal repetition -> KMP
Choosing between Naive and KMP was relatively easier because their use cases are more distinctive. However, we struggled the most when deciding between Rabin–Karp, Boyer–Moore, and GoCrazy.
- We know that Rabin–Karp should be used for long patterns and long texts, but we are not certain about what length threshold should be considered “long.”
- We attempted to use Boyer–Moore when the alphabet size was small. We tried to determine alphabet size by considering the number of distinct characters and relating it to the pattern length.
- The remaining cases were delegated to GoCrazy.
Despite all of these, we are still not confident that our selections were correct. :(

We were also unsure how to fully ınterpret the tables in the output of the code. The first table compares the execution times of the algorithms for each test case and identifies the fastest one, which is clear. However, we expected the algorithm selected by our pre-analysis code to match the best-case results in the first table, and we were not able to achieve this alignment consistently.

Additionally, the change in the README file close to the submission deadline was somewhat confusing for us.

**Resources:**

- https://medium.com/@AlexanderObregon/finding-patterns-with-boyer-moore-in-java-75c8a22d741a
- https://www.geeksforgeeks.org/dsa/pattern-searching/
- https://medium.com/tech-in-depth/string-matching-algorithms-271d50a2a265
- https://www.youtube.com/watch?v=ynv7bbcSLKE
- https://www.youtube.com/watch?v=qQ8vS2btsxI&t=50s
- https://www.youtube.com/watch?v=4Xyhb72LCX4
- https://chatgpt.com/share/693c5277-b384-8005-a853-9564b01c3974