# REPORT
# Implementation of the Knuth–Morris–Pratt (KMP) Algorithm

Student: Madina Rakhmetulla
Group: SE-2430
Course: Design and Analysis of Algorithms
Date: _____

1. Introduction

The Knuth–Morris–Pratt (KMP) algorithm is a well-known method for searching a pattern inside a larger text. What makes it interesting is that it avoids rechecking characters that the algorithm has already matched. The goal of this report is to implement KMP from scratch in Java, run it on different input sizes, and analyze how efficiently it works. Compared to the naive approach, which may re-compare many characters, KMP takes advantage of previously gained information and therefore works much faster on long inputs.

2. Algorithm Description

The key idea of KMP is the LPS (Longest Prefix–Suffix) array. This array tells us how many characters of the pattern already match themselves, so when a mismatch happens, we do not start from zero. Instead, we shift the pattern to the right position without losing progress. The algorithm consists of two main phases: • Building the LPS array, which analyzes the pattern. • The actual search phase, where the pattern is compared with the text using the LPS values. In simple words, LPS helps the algorithm "remember" what it already matched. This is the reason why KMP manages to run in linear time. Optional pseudocode: 1. Preprocess the pattern and build the LPS array. 2. Scan the text and compare characters. 3. On mismatch, jump to the LPS value instead of starting over.

3. Implementation

The implementation was written in Java. The structure is straightforward: one method builds the LPS array (buildLps()), and another method performs the search (kmpSearch()). The code includes comments that explain each major step, making it easier to follow the algorithm’s workflow. The main class also contains three test examples of different sizes so that the performance can be observed.

4. Experiments and Testing

To understand how the algorithm behaves with different inputs, three tests were performed. Test cases: • Short string: "ababcabcababababd" with pattern "ababd". • Medium string: "the quick brown fox jumps over the lazy dog" with pattern "the". • Long string: artificially generated text of 100,000 characters. Each test demonstrates that KMP consistently finds matches correctly and efficiently.

| Test | Text Length | Pattern Length | Matches | Time (ms) |
|---|---|---|---|---|
| 1 | 15 | 5 | 1 | < 1 |
| 2 | 43 | 3 | 2 | < 1 |
| 3 | 100000 | 5 | 19999 | ≈ 6 |

## 6. Conclusion

In conclusion, the KMP algorithm provides a reliable and efficient method for substring search. Its ability to reuse information through the LPS array allows it to avoid unnecessary work and achieve linear complexity. This makes KMP a strong choice for tasks involving repetitive searches or large text data. In future work, it would be interesting to compare its behavior with other string-matching algorithms such as Rabin–Karp or Boyer–Moore.