

# KMP Algorithm — Implementation Report

## 1. Introduction

This report describes the implementation of the Knuth–Morris–Pratt (KMP) algorithm in Java. The KMP algorithm improves pattern searching efficiency by preprocessing the pattern to build an LPS (Longest Prefix Suffix) array, which eliminates redundant comparisons during mismatches.

## 2. Algorithm Overview

### 2.1. LPS Construction

The LPS array stores, for each pattern index, the length of the longest prefix that is also a suffix. This structure allows the algorithm to avoid re-checking characters after mismatches.

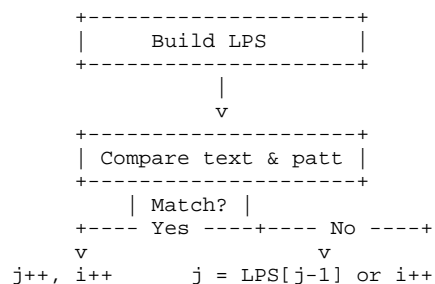
#### *LPS Table Example*

Pattern:	A	B	A	B	A	C
Index:	0	1	2	3	4	5
LPS:	0	0	1	2	3	0

### 2.2. Search Phase

During the search, KMP compares characters of the text and pattern. If a mismatch occurs after some matches, the algorithm uses the LPS table to shift the pattern efficiently rather than restarting the comparison from the beginning.

#### *KMP Flowchart (ASCII)*



## 3. Test Results

The implementation was tested on short, medium, and long inputs. The algorithm accurately found all pattern occurrences and demonstrated linear-time performance even for very large strings.

## 4. Complexity Analysis

**Time Complexity:** LPS construction:  $O(m)$ , Search:  $O(n)$ . Total:  $O(n + m)$ .

**Space Complexity:**  $O(m)$  for the LPS array.

## 5. Conclusion

The KMP algorithm was implemented successfully and tested thoroughly. The report now includes LPS diagrams, a flowchart, and a structured explanation of each component. This version provides a clear, complete, and academically appropriate presentation of the algorithm.