

# Knuth-Morris-Pratt (KMP) Algorithmus

## 1 Grundidee

Der **Knuth-Morris-Pratt (KMP)** Algorithmus sucht ein **Pattern P** in einem **Text T** effizient, ohne bei einem Mismatch im Text zurückzuspringen.

Kernidee:

- Wenn ein Mismatch passiert, nutzt KMP **Vorwissen über das Pattern**
- Dadurch werden unnötige Vergleiche vermieden
- KMP verwendet dazu ein Hilfsarray: **LPS** (Longest Prefix Suffix)

→ Laufzeit: **O(n + m)** (n = Länge Text, m = Länge Pattern)

---

## 2 Voraussetzungen

- Text T und Pattern P sind Strings
  - Fähigkeit, ein LPS-Array für P zu berechnen
- 

## 3 Laufzeiten & Eigenschaften

Eigenschaft	Wert
Laufzeit	$O(n + m)$
Speicherbedarf	$O(m)$
Backtracking im Text	nein
Anwendungsfall	String-Suche

**Hinweis:** Der naive Algorithmus kann im Worst Case  $O(n \cdot m)$  benötigen.

---

## 4 LPS-Array (Longest Prefix Suffix)

### Bedeutung

Für jede Position i im Pattern:

- $LPS[i]$  = Länge des **längsten echten Prefix** von  $P[0..i]$ , der gleichzeitig ein **Suffix** von  $P[0..i]$  ist.

„Echt“ bedeutet: nicht das ganze Teilwort selbst.

## Beispiel

Pattern:

```
P = "ababaca"
```

LPS:

Index:	0	1	2	3	4	5	6
P:	a	b	a	b	a	c	a
LPS:	0	0	1	2	3	0	1

## 5 Schritt-für-Schritt-Idee (Suche)

Wenn bei einem Vergleich  $\text{Text}[i] \neq \text{Pattern}[j]$ :

- $j$  wird nicht auf 0 gesetzt (wie beim naiven Ansatz)
  - sondern auf **LPS[j-1]**
- So „verschiebt“ KMP das Pattern intelligent, ohne Textzeichen erneut zu prüfen.

## 6 Besonderheiten / Prüfungsrelevante Hinweise

- KMP ist ein Standardbeispiel für „Preprocessing“
- Typische Prüfungsfragen:
  - Wie wird das LPS-Array berechnet?
  - Warum ist die Laufzeit  $O(n + m)$ ?
- Wichtig: KMP springt nie im Text zurück, nur im Pattern
- Warum KMP statt Automat?
  - KMP ist oft einfacher zu implementieren
  - Kein expliziter Automat notwendig
  - KMP ist ein spezialisierter, effizient konstruierter Automat für genau **ein Pattern**

## 7 Vor- und Nachteile

### Vorteile

- garantiert lineare Laufzeit  $O(n + m)$
- sehr effizient bei vielen Wiederholungen im Pattern

- keine erneuten Textvergleiche

## Nachteile

- Konzept LPS ist anfänglich schwer
  - Preprocessing erforderlich
- 



## Merksatz für die Prüfung

*KMP sucht Pattern in  $O(n + m)$ , indem es Mismatches mit dem LPS-Array behandelt und nie im Text zurückspringt.*

---

## 8 Python-Implementierung

```
In [1]: def build_lps(pattern):
    lps = [0] * len(pattern)
    length = 0 # Länge des aktuellen Prefix-Suffix
    i = 1

    while i < len(pattern):
        if pattern[i] == pattern[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1

    return lps

def kmp_search(text, pattern):
    if not pattern:
        return []

    lps = build_lps(pattern)
    result = []

    i = 0 # Index für text
    j = 0 # Index für pattern

    while i < len(text):
        if text[i] == pattern[j]:
            i += 1
            j += 1
        else:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1
```

```
    if j == len(pattern):
        result.append(i - j)      # Match gefunden
        j = lps[j - 1]           # weiter suchen
    elif i < len(text) and text[i] != pattern[j]:
        if j != 0:
            j = lps[j - 1]
        else:
            i += 1

return result

# Beispiel
text = "ababcabcabababd"
pattern = "ababd"
print(kmp_search(text, pattern)) # [10]
```

[10]