String Matching

Gegeben: (endliches) Alphabet Σ

Text $T = t_1 t_2 \cdots t_n$ der Länge n

Pattern/Muster $P = p_1 p_2 \cdots p_m$ der Länge m

Gesucht: Erstes/alle Vorkommen von P in T.

Beispiel: $T = \texttt{abcdabfgabcaa}$, $P = \texttt{bc}$

```
 bc
       bc
```

Naiver Ansatz: Durchschieben des Patterns

for $i := 1$ **to** $n - m + 1$ **do**

$j := 1;$

while $t[i + j - 1] = p[j]$ **do**

if $j = m$ **then return** (i);

$j + +;$

return ('no match');

Worst Case Laufzeit: $\mathcal{O}((n - m + 1) \cdot m))$.

Dies wird auch angenommen: z.B. $T = a^{n-1}b, \; P = a^{m-1}b$

KMP-Algorithmus

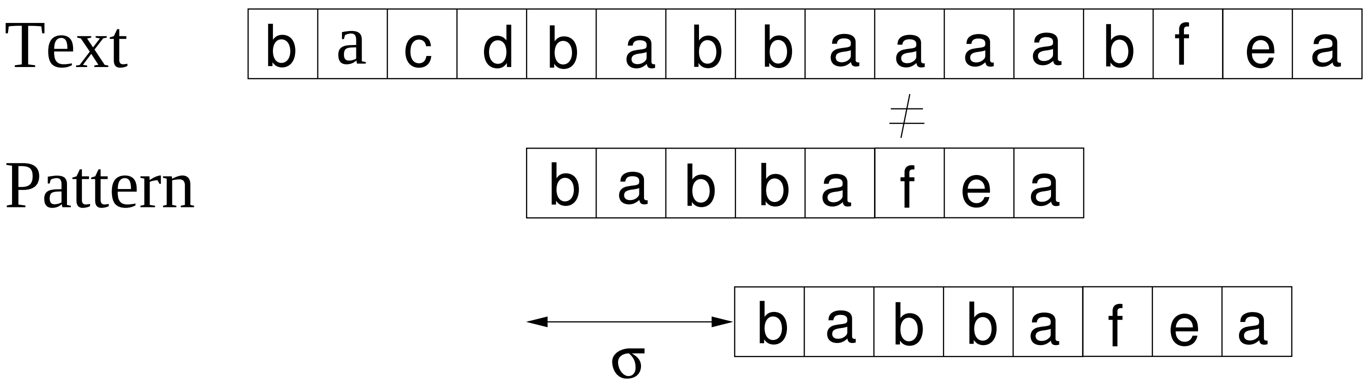
von Knuth, Morris, Pratt (1977)

Idee:

Tritt ein Mismatch an Position i des Patterns auf, so kann man

a) das Pattern unter Umständen um mehr als eine Position weiterschieben und

b) braucht man die Positionen *vor* dem Mismatch nicht nochmals zu vergleichen



KMP-Algorithmus

Berechne die Werte $\sigma[j], j = 1, \ldots, m$;

// $\sigma[j]$:= Anzahl Positionen um die Pattern weitergeschoben wird, wenn Mismatch an

// Position j des Pattern auftritt

$i := 1;\ j := 1;$

while $i \leq n - m + 1$ **do**

  while $t[i + j - 1] = p[j]$ **do**

    if $j = m$ **then return** (i);

    $j + +$;

  // Mismatch an Position j !

  $i := i + \sigma[j];\ j := \max\{1, j - \sigma[j]\}$;

return ('no match');

Boyer-Moore Algorithmus

Text:							a	b	c							
Muster:			a	b	c	b	c	b	c							
Zeichen-Heuristik:							a	b	c	b	c	b	c			

(schiebe bis zum rechtesten a im Muster)

Text:							b	b	c							
Muster:			a	b	c	b	c	b	c							
Suffix-Heuristik:							a	b	c	b	c	b	c			

(schiebe bis zum rechtesten ?bc im Muster, wobei ? ungleich c)