FHV

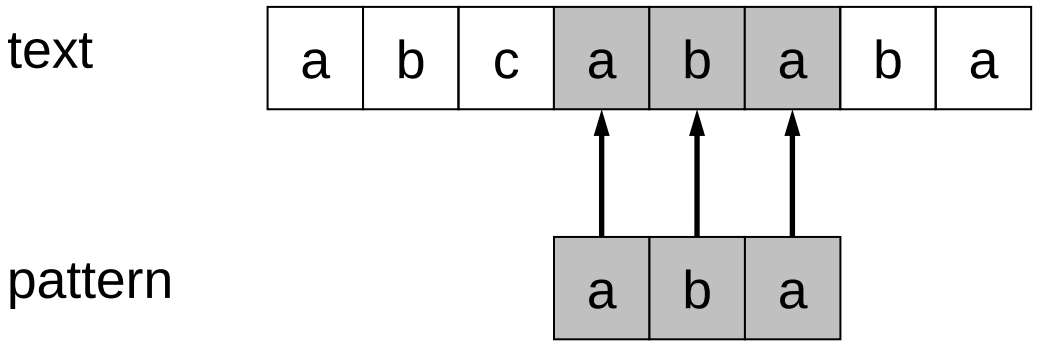
Algorithmen & Datenstrukturen

Pattern Matching

Wolfgang Auer

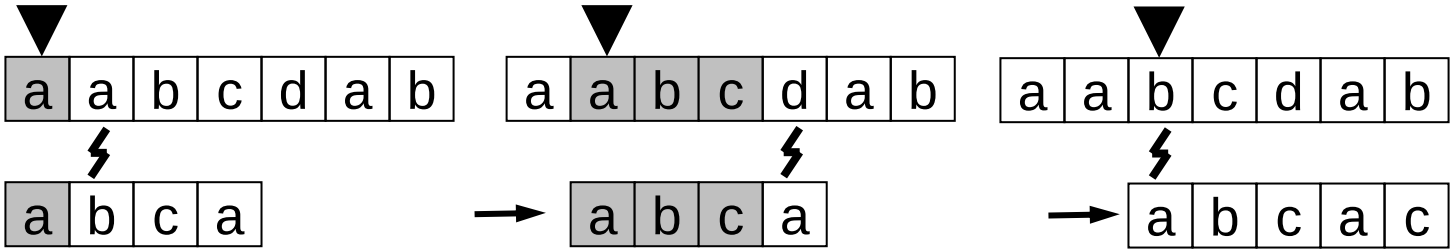
Problemstellung

- Geg: Zwei Zeichenketten
- Ges: Kommt die eine Zeichenketten (Muster) in der anderen (Text) vor?



Brute Force

Jeder Buchstabe das Texts wird von links nach rechts ungeachtet der Eigenschaften des Musters untersucht




```
i = 0
j = 0
while ( j < m ) and (i <= n - m) {
    j = 0
    while ( j < m ) and ( t[ i + j ] == p[ j ] ) {
        j++
    }
    i++
}
```

Worst case O ≈ m * n

m…Länge des Pattern

n…Länge des Texts

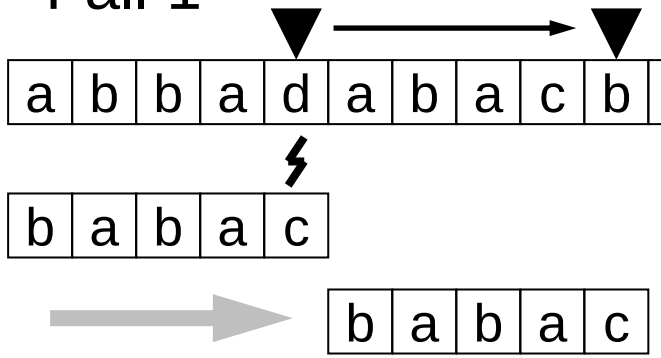
Boyer-Moore

- Ziel: Optimierung des Brute-Force Algorithmus

  $\Rightarrow$ *Anwenden der Kenntnis über das Muster*
- Strategie
  - Vergleiche das Muster von ***rechts nach links*** mit dem Text
  - Bei Ungleichheit (*Mismatch)* wird das Muster mit dem am weitesten rechts liegenden, identischen Zeichen unter die *Mismatch*-Stelle bewegt („*geshiftet*“)
  - Das Muster muss um mindestens eine Position nach rechts bewegt werden

Bad character heuristics BM (1)

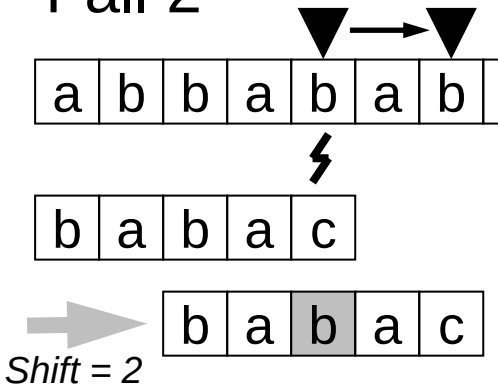
- Fall 1



Shift = Länge des Musters

„d“ kommt im Muster nicht vor ⇒ Verschiebe das Muster um dessen Länge nach rechts

- Fall 2

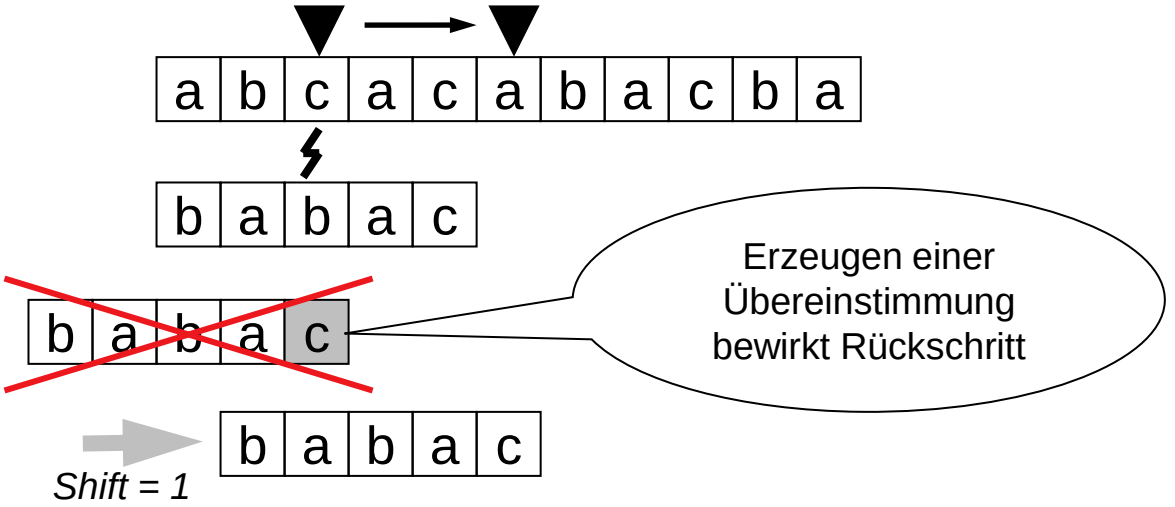


Shift = 2

„b“ kommt mehrfach im Muster vor ⇒ Verschiebe das Muster so, dass **letztes** „b“ auf das Textsymbol „b“ ausgerichtet ist

Bad character heuristics BM (2)

- Fall 3



Da Rückschritte nicht zielführend sind, wird das Muster um eine Position nach rechts verschoben.

BM Algorithmus

```
boyerMoore (↓text, ↓pattern, ↑pos) {
   /* text[0..n-1], pattern[0..m-1] */
   initShift(pattern, shift);

   i = m; /* pattern length */
   j = m; /* pattern length */
   while (j > 0) and (i <= n) {
      if (text[i - 1] == pattern[j - 1]) {
         i = i - 1; j = j - 1;
      } else {
         i = i + max(m – j + 1, shift[text[i – 1]]);
         j = m; /* pattern length */
      }
   }
   pos = i;
}
```

Worst case O ≈ m + n

Erstellung der Schiebetabelle für Boyer-Moore

- Bestimmen der maximalen Anzahl von Zeichen, um die sich der Lesecursor im Text bei einem Mismatch nach rechts bewegt werden darf
- Bsp: Shift für „TEXT“

A	B	..	E	..	T	..	X	Y	Z
4	4	4	2	4	0	4	1	4	4

Alle Buchstaben des verwendeten Alphabets

```
initShift (↓pattern, ↑shift) {
    for all possible characters c {
            shift[c] = m; /* pattern length */
    }

    for (i = 0; i < m; i++) {
            shift[pattern[i]] = m - i - 1;
    }
}
```

Knuth-Morris-Pratt

- Ziel: Optimierung des Brute-Force Algorithmus

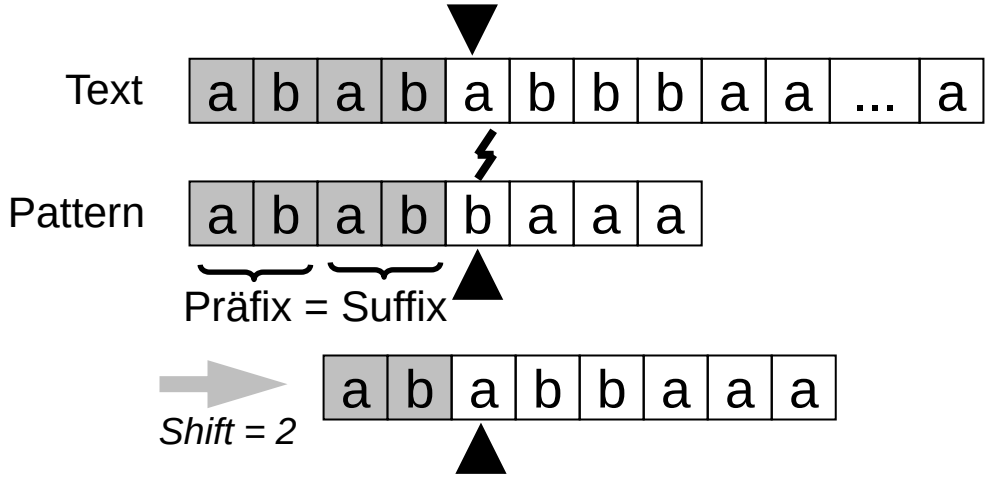
  $\Rightarrow$ *Anwenden der Kenntnis über das Muster*
- Strategie
  - Vergleiche das Muster von ***links nach rechts*** mit dem Text
  - Bei Mismatch wird anhand der Struktur des Musters die Anzahl der Zeichen bestimmt, die übersprungen (*„geshiftet“)* werden können.

Bestimmung des Shifts KMP (1)

- Definitionen
  - Sei $z = z_0..z_{k-1}, k \in N$, eine Zeichenkette der Länge k
  - Ein *Präfix* von z ist eine Teilkette s für die gilt:
    $s = z_0..z_{b-1}, b \leq k$
    Für $b < k$ sprechen wir von einem echten *Präfix*
  - Ein *Suffix* von z ist eine Teilkette s für die gilt: $p = s_{k-b}..s_{k-1}, b \leq k$
    Für $b < k$ sprechen wir von einem echten Suffix
  - Ein *Rand* r von z ist eine Zeichenkette, die gleichzeitig *echtes Präfix* und *echtes Suffix* ist.

    z.B. abaaxxxabaa

Bestimmung des Shifts KMP (2)



- Bei Mismatch wird der maximale Rand *b* des bereits analysierten Musters bestimmt
- Die letzten *b* Zeichen des analysierten Texts entsprechen den ersten *b* Zeichen des Musters
  $\Rightarrow$ Vergleich kann am (*b* + *1*)*-ten* Zeichen im Muster fortgesetzt werden.

Bestimmung des Shifts KMP (3)

- Bestimme den breitesten Rand für alle Präfixe des Musters und speichere die Werte in einer Tabelle
- z.B.: Ränder für *ababbaaa*

	Zeichenkette	Breite des Randes
j = 1	a	0
j = 2	ab	0
j = 3	aba	1
j = 4	abab	2
j = 5	ababb	0
j = 6	ababba	1
j = 7	abababa	1

KMP Algorithmus

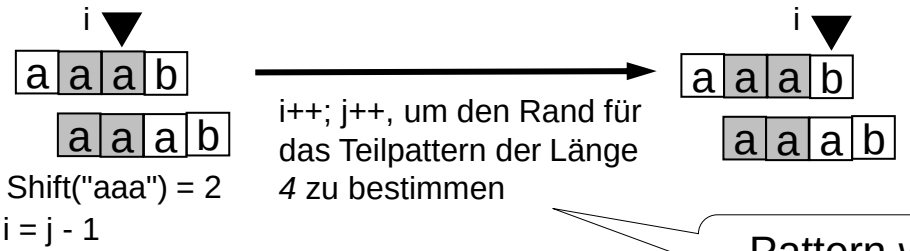
```
KMP (↓text, ↓pattern, ↑pos) {
   initShift(pattern, shift);

   i = 0;
   j = 0;

   while (j < m) and (i < n) {
      if (j == -1) or (text[i] == pattern[j]) {
         i = i + 1; j = j + 1;
      } else {
         j = shift(j);
      }
   }

   if (j == m) {
      pos = i - m;
   } else {
      pos = -1; //not found
   }
}
```

Weiterbewegung bei Mismatch an der ersten Stelle im Pattern (Implementierungstrick)

Worst case O ≈ m + n

Erstellen der Shift-Tabelle

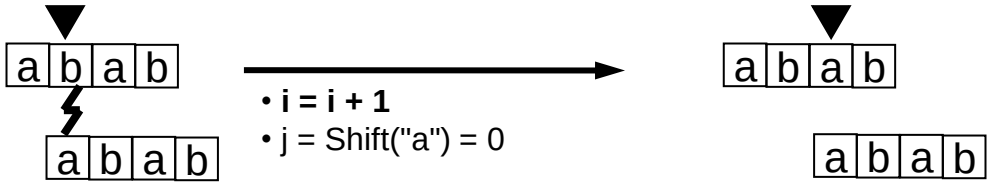
- $p_i = p_j$



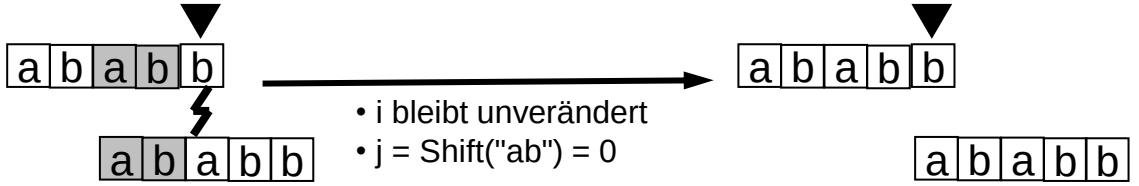
- $p_i \neq p_j$ (Mismatch)

Pattern wird mit sich selbst verglichen!

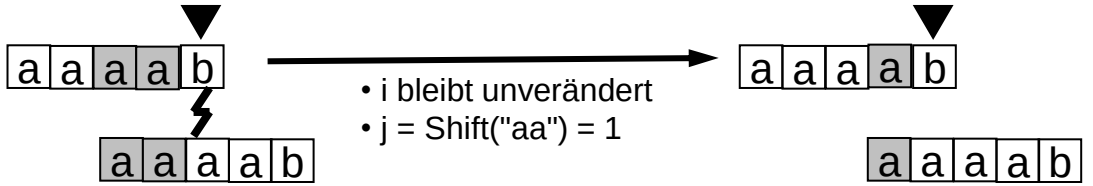
1. Fall



2. Fall



3. Fall



Erstellung der Schiebetabelle für KMP

```
initShift (↓pattern, ↑shift) {
   i = 0;
   j = -1;
   shift[0] = -1;

   while (i < m) {
      if (j == -1) or (p[i] == p[j]) {
         i = i + 1;
         j = j + 1;
         shift[i] = j;
      } else {
         j = shift[j]; //Strukturabhängigkeit
      }
   }
}
```