1.2. Der Algorithmus von Boyer und Moore

Der Algorithmus KMP betrachtet stets den gesamten Textstring.

Frage:
Gibt es einen Algorithmus, der häufig nicht den gesamten Textstring betrachtet?

Idee:

Gibt es Für das Stringmatchingproblem
problem
↑ problem
↑ problem
↑

allgemein:

$$x = a_1 a_2 a_3 \dots \underset{\downarrow}{a_m} a_{m+1} a_{m+2} \dots a_n$$
$$y = b_1 b_2 b_3 \dots \underset{\uparrow}{b_m}$$

Platziere zunächst wie beim Algorithmus KMP den Musterstring $y = b_1 b_2 \dots b_m$ linksbündig zum Textstring $x = a_1 a_2 \dots a_n$.

Aber:
Vergleiche y mit dem korrespondierenden Teilstring $a_1 a_2 \dots a_m$ des Textstrings von rechts nach links.

Beobachtung:
Im Mittel wird häufig $a_m \neq b_m$ erfüllt sein. Dann kann wie folgt verfahren werden:

1. Fall: $a_m \notin y$

Platziere y unter $a_{m+1} a_{m+2} \cdots a_{2m}$.

2. Fall: $a_m \in y$

Platziere y derart, dass a_m von rechts gesehen über dem ersten Symbol b_r mit $b_r = a_m$ steht. D.h.,

i) $a_m = b_r$, wobei b_r unter a_m steht und
ii) $a_m \neq b_j \quad \forall j > r$.

Danach liegt folgende Situation vor:

$$x = a_1 a_2 \cdots a_{m-r+1} a_{m-r+2} \cdots a_m a_{m+1} \cdots a_{2m-r} \cdots a_n$$
$$\qquad\quad b_1 \qquad b_2 \qquad \cdots \quad b_r \; b_{r+1} \cdots b_m$$

Wir vergleichen y mit dem korrespondierenden Teilstring $a_{m-r+1} a_{m-r+2} \cdots a_{2m-r}$ des Textstrings von rechts nach links.

y und der korrespondierende Teilstring von x können in einem Suffix gleich sein und sich erst dann unterscheiden. D.h., allgemeiner liegt folgende Situation vor:

Annahme:
Wir vergleichen y mit dem Teilstring $a_{k+1} a_{k+2} \cdots a$ von x und folgendes ist erfüllt:

i) $a_{k+j} \neq b_j$ und
ii) $a_{k+j+1} a_{k+j+2} \cdots a_{k+m} = b_{j+1} b_{j+2} \cdots b_m$.

Idee:

- Shifte $y = b_1 b_2 \dots b_m$ unter Anwendung von "sinnvollen" Regeln nach rechts.
- Vergleiche den neuen korrespondierenden Teil-string des Textstrings x von rechts nach links mit y.

Frage:
Was sind sinnvolle Regeln bzw. Heuristiken für den Rechtsshift?

Derartige Regeln müssen die Eigenschaft besitzen, dass kein Shift über ein Vorkommen von y in x hinaus erfolgt. Boyer und Moore geben in ihrer Arbeit

R. S. Boyer, J. S. Moore, A fast string searching algorithm, CACM 20 (1977), 762 - 772.

zwei Heuristiken an, die jeweils einen Wert δ_1 bzw. δ_2 für einen sinnvollen Rechtsshift berechnen.

δ_1 basiert auf folgender Idee:

Der Textstringbuchstabe c, der einen Mismatch verursacht hat, muss einem Musterstringbuchstaben, der identisch zu ihm ist, zugewiesen werden. Dies bedeutet, dass wir den Musterstring y so weit nach rechts schieben, bis das in y am

weitesten rechts stehende c unter dem Textbuchstaben c, der den Mismatch verursacht hat, steht. Für $c \in \Sigma$ ist somit $\delta_1(c)$ definiert durch:

$$\delta_1(c) := \begin{cases} m & \text{falls } c \notin y \\ \min_{0 \le i \le m} \{ i \mid b_{m-i} = c \} & \text{sonst.} \end{cases}$$

Bemerkung:

- Im obigen Szenario, d.h., $a_{z+j} \neq b_j$ und $a_{z+j+1} \cdots a_{z+m} = b_{j+1} \cdots b_m$ definiert $\delta_1(a_{z+j})$ genau dann einen Rechtsshift von y, wenn $m - \delta_1(a_{z+j}) < j$. Ist dies der Fall, dann kann y um $\delta_1(a_{z+j}) - (m-j)$ Positionen nach rechts geshiftet werden.

- δ_1 kann durch eine Tabelle der Größe $|\Sigma|$ beschrieben werden.

- $\delta_1(c)$ gibt das in y links vom Stringende am weitesten rechts stehende c an. Natürlich kann man für jede Position j, $1 \le j \le m$ einen entsprechenden Wert im voraus berechnen. Dies würde allerdings die Vorbereitungszeit erheblich erhöhen und es wäre ein zusätzlicher Speicher der Größe $m \cdot |\Sigma|$ notwendig, wenn man dies für alle j tun würde.

δ_2 ist eine Funktion derjenigen Position in y, in der sich y vom gerade geprüften Teilstring von x

unterscheidet.

Annahme:

$a_{i+j} \neq b_j$ und $a_{i+j+1} \cdots a_{i+m} = b_{j+1} \cdots b_m$.

Idee:

Berechne eine Tabelle δ_2, die die notwendige Information über den kleinsten Rechtsshift > 0 enthält, so dass zwischen $a_{i+j+1} \cdots a_{i+m}$ und dem korrespondierenden Teilstring y' in Musterstring y kein Mismatch vorkommt und unmittelbar links von y' in y ein Symbol $\neq b_j$ steht.

Bemerkung:

Falls links von y' in y das Symbol b_j stehen würde, dann hätten wir nach dem Rechtsshift unter a_{i+j} wiederum das Symbol b_j stehen und somit wiederum ein Mismatch.

Interpretation:

Äquivalent zur obigen Idee könnten wir y unter y plazieren und für das untere y den geringstmöglichen Rechtsshift > 0 durchführen, so dass zwischen $b_{j+1} \cdots b_m$ und dem darunterstehenden Teilstring y' von y kein Mismatch vorkommt und unmittelbar links von y' ein Symbol $\neq b_j$ steht. D.h., wir benötigen das maximale $g < j$, so dass

i) $b_{g+1} b_{g+2} \cdots b_{g+(m-j)} = b_{j+1} b_{j+2} \cdots b_m$ und

ii) $b_g \neq b_j$.

Falls g nicht existiert, dann benötigen wir den maximalen Präfix von $b_1 b_2 \dots b_{m-j}$, der auch Suffix von $b_{j+1} b_{j+2} \dots b_m$ ist.

Wir berechnen zunächst eine Tabelle δ_2' und dann mit Hilfe dieser die Tabelle δ_2.

$\delta_2'(j)$ enthält den Index desjenigen Symbols, das unter b_m geschoben wird. Falls das untere y ganz über das obere y hinaus geschoben wird, dann erhält $\delta_2'(j)$ den Wert 0. Somit ergibt sich für $\delta_2'(j)$ folgender Wert:

$$\delta_2'(j) := \begin{cases} \bullet \max\{ g + (m-j) \mid 1 \le g < j,\ b_g \neq b_j \\ \qquad \text{und } b_{g+1} \dots b_{g+(m-j)} = b_{j+1} \dots b_m \} \\ \qquad \text{falls solches } g \text{ existiert.} \\ \bullet \max\{ e \mid b_1 \dots b_e \text{ ist Suffix von } b_{j+1} \dots b_m \} \\ \qquad \text{falls nicht } g \text{ aber solches } e \text{ existie} \\ \bullet\ 0 \qquad \text{sonst} \end{cases}$$

Falls der erste Fall eintritt, dann wird der Muster-string y derart nach rechts geschiftet, dass b_g unter a_{i+j} und somit auch $b_{g+(m-j)}$ unter a_{i+m} stehen.

Falls der zweite Fall eintritt, dann wird der Muster-string y derart nach rechts geschiftet, dass b_e unter a_{i+m} steht.

Im dritten Fall wird y ganz über a_{i+m} hinaus geschoben. D.h., b_1 steht dann unter a_{i+m+1}.

Bemerkung:
Im Gegensatz zur Tabelle δ_2' geben die Komponenten der Tabelle δ_1 die Länge eines möglichen Rechtsschiftes an. Besser wäre es, wenn $\delta_2'(j)$ dies auch tun würde. Dann könnte der Algorithmus den maximalen der beiden Rechtsschifte einfach bestimmen und diesen durchführen.

Ziel:
Äquivalente Umformulierung von $\delta_2'(j)$ zu $\delta_2(j)$, so dass $\delta_2(j)$ die Länge des durch $\delta_2'(j)$ spezifizierten Rechtsschiftes angibt.

Situation:

	b_1 b_2 ...	b_j ...	b_m
1. Fall	b_1 ... b_g ...	$b_{g+(m-j)}$ ...	b_m
2. Fall		$b_1 b_2 \ldots b_e$ ...	b_m
3. Fall		b_1 b_2 ...	b_m

Somit ergibt sich als Länge des korrespondierenden Rechtsschiftes:

1. Fall: $m - (g + (m-j)) = j - g$
2. Fall: $m - e$
3. Fall: m

Also erhalten wir

$$\delta_2(j) = m - \delta_2'(j).$$

δ_2 kann durch eine Tabelle der Größe m spezifiziert werden. Da δ_2 nur von y und nicht von x abhängt, kann diese Tabelle im voraus berechnet werden. Gegeben die Tabellen δ_1 und δ_2 sieht der Algorithmus von Boyer und Moore folgendermaßen aus:

Algorithmus BM

Eingabe: Textstring $x = a_1 a_2 \dots a_n$,
Musterstring $y = b_1 b_2 \dots b_m$ und
Tabellen δ_1 und δ_2.

Ausgabe: k minimal, so dass

$$a_{k+1} a_{k+2} \dots a_{k+m} = b_1 b_2 \dots b_m,$$

falls solches k existiert und "Fehlanzeige" sonst.

Methode:

```
Ausgabe := n;
msende := m; (* markiert Musterstringende in x *)
while msende ≤ n
   do
      k := msende; (* Zeiger in x *)
      j := m;      (* Zeiger in y *)
      while j > 0 and a_k = b_j
         do
            j := j-1; k := k-1
      od;
```

```
        if j = 0
          then
            msende := n+1 ;
            Ausgabe := t
          else
            msende := max { t + δ1(a_t) , msende + δ2(j) }
        fi
    od ;
if Ausgabe < n
  then
      return (Ausgabe)
  else
      return ("Fehlanzeige")
fi.
```

Bemerkung:
Aus der Definition der Tabellen δ_1 und δ_2 folgt direkt, dass keiner der durch diese Tabellen definierten Shifts den Musterstring y über ein Vorkommen von y im Textstring x hinausschiebt. Somit müssen wir uns für den Beweis der Korrektheit des Algorithmus BM davon überzeugen, dass BM diese Shifts korrekt durchführt.

Nach dem durch $\delta_1(a_t)$ definierten Shift muss die Position des Musterstrings y die folgende sein:

$$\begin{cases} b_1 \text{ steht unter } a_{k+1} & \text{falls } a_k \notin y \\ b_{m-q} \text{ steht unter } a_k & \text{sonst,} \end{cases}$$

wobei $q = \min_{0 \le i < m} \{ i \mid b_{m-i} = a_k \}$.

Falls der Algorithmus BM den zu $\delta_1(a_k)$ korrespondierenden Shift durchführt, dann gilt danach

$$\text{usende} = k + \delta_1(a_k).$$

D.h., b_m steht unter

$$\begin{cases} a_{k+m} & \text{falls } \delta_1(a_k) = m \\ a_{k+q} & \text{falls } \delta_1(a_k) = \min_{0 \le i < m} \{ i \mid b_{m-i} = a_k \} =: q \end{cases}$$

Also steht

$$\begin{cases} b_1 \text{ unter } a_{k+1} & \text{falls } a_k \notin y \\ b_{m-q} \text{ unter } a_k & \text{sonst} \end{cases}$$

Demzufolge führt der Algorithmus BM den Shift korrekt durch.

$\delta_2(j)$ gibt die Länge des Rechtsshifts, um die der Musterstring nach rechts geschoben wird, an. Falls BM den zu $\delta_2(j)$ korrespondierenden Shift durchführt, dann gilt danach

$$\text{usende} = \text{usende} + \delta_2(j).$$

Dies bedeutet, dass der Musterstring exakt um $\delta_2(j)$ nach rechts geschoben wird. Also führt der

Algorithmus BM den Shift korrekt durch.

Bemerkung:
Falls für $q := \delta_1(a_z)$ der Wert $m-q$ größer als j ist, dann wäre der durch $\delta_1(a_z)$ definierte Shift ein Linksshift. Wegen $\delta_2(j) > 0$ gilt dann

$$m_{ende} + \delta_2(j) > k + \delta_1(a_z),$$

So dass der durch $\delta_2(j)$ definierte Rechtsshift vom Algorithmus BM durchgeführt wird. Also führt der Algorithmus BM stets einen Rechtsshift durch.

Bevor wir uns mit der Laufzeitanalyse des Algorithmus BM beschäftigen, überlegen wir uns, wie wir die Tabellen δ_1 und δ_2 effizient berechnen.

Die Berechnung der Tabelle δ_1 ist einfach und wird von folgenden Algorithmus durchgeführt:

Algorithmus δ_1
Eingabe: $y = b_1 b_2 \dots b_m$
Ausgabe: Tabelle δ_1
Methode:
for alle $c \in \Sigma$
do
$\delta_1(c) := m$
od;

```
for j from m to 1
  do
    if δ1(bj) = m
      then
        δ1(bj) := m - j
    fi
  od.
```

Übung:
Beweisen Sie die Korrektheit des Algorithmus δ_1 und führen Sie eine Laufzeitanalyse durch.

Die Berechnung der Tabelle δ_2 gestaltet sich wesentlich schwieriger. Für jedes $1 \leq j \leq m$ ist genau ein Wert $\delta_2(j)$ zu berechnen. Unser Ziel ist es, die Gesamttabelle in $O(m)$ Zeit zu berechnen. Für $1 \leq j \leq m$ betrachten wir noch einmal $\delta_2(j)$. Drei Fälle sind zu unterscheiden:

1. Fall:

$\exists\, 1 \leq g < j$ mit $b_{g+1} \dots b_{g+(m-j)} = b_{j+1} \dots b_m$ und $b_g \neq b_j$

Dann ist g_{max}, das maximale solche g, zu berechnen. Es gilt dann

$$\delta_2(j) = j - g_{max}.$$

2. Fall:

$\neg$ Fall 1 und $\exists\, 1 \leq \ell \leq m-j$ mit $b_1 \dots b_\ell$ ist Suffix von $b_{j+1} b_{j+2} \dots b_m$.

Dann ist ℓ_{max}, das maximale solche ℓ, zu berechnen. Es gilt dann

$$\delta_2(j) = m - \ell_{max}.$$

3. Fall:

¬ Fall 1 und ¬ Fall 2.

Dann gilt

$$\delta_2(j) = m.$$

Demzufolge ist folgende Grobstruktur für den Algorithmus zur Berechnung der Tabelle δ_2 sinnvoll:

(1) Für alle $1 \le j \le m$ initialisiere

$$\delta_2(j) := m.$$

(2) Für alle $1 \le j \le m$ berechne g_{max}, d.h., das maximale $1 \le g < j$ mit

$$b_{g+1} \cdots b_{g+(m-j)} = b_{j+1} \cdots b_m \quad \text{und} \quad b_g \neq b_j$$

und setze

$$\delta_2(j) := j - g_{max},$$

falls solches g existiert.

(3) Für alle $1 \le j \le m$, für die in (2) kein g_{max} berechnet wurde, für die also

$$\delta_2(j) = m$$

gilt, berechne ℓ_{max}, d.h., das maximale $1 \leq \ell \leq m-j$ mit

$b_1 \dots b_\ell$ ist Suffix von $b_{j+1} \dots b_m$

und setze

$$\delta_2(j) := m - \ell_{max},$$

falls solches ℓ existiert.

Bemerkung:
Für die Durchführung von (3) benötigen wir lediglich $O(m)$ Zeit. Wir können hierzu den Algorithmus BERECHNUNG VON H verwenden.

Ziel:
Durchführung von (2) in $O(m)$ Zeit.

Idee:
Reduktion von (2) auf ein Problem, dessen Lösung in $O(m)$ Zeit die Durchführung von (2) in $O(m)$ Zeit impliziert

Hierzu ist es zunächst sinnvoll, die Aufgabenstellung (2) äquivalent umzuformen. Dafür betrachten wir nochmals die Situation, in der ein Wert $\delta_2(j)$ zu berechnen ist. Sei hierzu $g_0 := g_{max}$.

$$x = a_1 a_2 \dots a_k a_{k+1} \dots a_{k+j} a_{k+j+1} \dots a_{k+m} \dots a_n$$

$$\neq$$

$$b_1 \dots b_{g_0} \underbrace{b_{g_0+1} \dots b_j}_{j-g_0} \; b_{j+1} \dots b_s \underbrace{b_{s+1} \dots b_m}_{j-g_0}$$

wobei $s := g_0 + m - j \iff g_0 = s + j - m$

Es gilt:

$$m - s = m - (g_0 + m - j) = j - g_0$$

Somit erhalten wir aus (2) folgende äquivalente Aufgabenstellung:

(2') Für alle $1 \le j \le m$ berechne maximales $s < m$ mit

$$b_{g+1} b_{g+2} \dots b_s = b_{j+1} b_{j+2} \dots b_m \text{ und } b_g \neq b_j,$$

wobei $g := s + j - m$ und setze

$$\delta_2(j) := m - s,$$

falls solches s existiert.

Zur Lösung dieser Aufgabenstellung möchten wir einmal den Musterstring y lesen und dabei die benötigten Werte $\delta_2(j)$ berechnen. Da die zu bestimmenden Eigenschaften vom Suffix von y abhängen, macht lesen von links nach rechts keinen Sinn. Beim Lesen von rechts nach links haben wir die Schwierigkeit, dass zum Zeitpunkt, wenn b_j betrachtet wird, b_s bereits betrachtet ist.

Bereits zum Zeitpunkt, wenn b_s betrachtet wird, muss die jetzt benötigte Information zumindest vorbereitet werden und spätestens dann, wenn das zu b_j korrespondierende b_g betrachtet wird, berechnet sein. Hierzu ist es nützlich, die Aufgabenstellung (2') nochmals äquivalent umzuformen. Folgende Beobachtung hilft uns hierbei:

- Für jedes $1 \leq s < m$ gibt es höchstens ein g, so dass

$$b_{g+1} b_{g+2} \cdots b_s = b_{j+1} b_{j+2} \cdots b_m \quad \text{und} \quad b_g \neq b_j,$$

wobei $j := m + g - s$

Bew. d. Beob.:
Betrachten wir $s < m$ fest. Falls kein $g < s$, das obige Eigenschaften erfüllt, existiert, dann ist nichts mehr zu beweisen.

Betrachten wir $g < s$ minimal, so dass

$$b_{g+1} b_{g+2} \cdots b_s = b_{j+1} b_{j+2} \cdots b_m \quad \text{und} \quad b_g \neq b_j,$$
wobei $j = m + g - s$

$\Rightarrow$
Für alle $g < n =: g+\ell \leq s$ gilt

$$b_n = b_{g+\ell} = b_{j+\ell} = b_{m+g-s+\ell} = b_{m+n-s}$$

$\Rightarrow$

Es existiert kein $h > g$, so dass

$$b_{h+1} b_{h+2} \cdots b_s = b_{j+1} b_{j+2} \cdots b_m \text{ und } b_h \neq b_j,$$
wobei $j = m + h - s$.

□

Aufgrund obiger Beobachtung erhalten wir aus (2') folgende äquivalente Aufgabenstellung:

(2") Für alle $1 \leq s < m$ berechne, falls es existiert, das eindeutige $g \in \{1, 2, \ldots, s\}$ mit

$$b_{g+1} b_{g+2} \cdots b_s = b_{j+1} b_{j+2} \cdots b_m \text{ und } b_g \neq b_j,$$
wobei $j := m + g - s$.

Falls g existiert und $\delta_2(j) > m - s$, dann führe

$$\delta_2(j) := m - s$$

aus.

Für $1 \leq s < m$ bezeichne N_s die Länge des längsten Suffixes des Teilstrings $b_1 b_2 \ldots b_s$, der auch Suffix von $b_1 b_2 \ldots b_m$ ist. Dann gilt für das zu s gemäß der Aufgabenstellung (2") korrespondierende g:

$$g = s - N_s$$

und daher

$$\begin{aligned} j &= m + g - s \\ &= m + s - N_s - s \\ &= m - N_s . \end{aligned}$$

Also können wir anstatt g auch N_s berechnen. Nach der Berechnung von N_s können wir, falls $\delta_2(m - N_s) > m - s$

$$\delta_2(m - N_s) := m - s$$

ausführen.

Wir arbeiten lieber mit aufsteigenden als mit absteigenden Indizes. Daher definieren wir den String y^R durch

$$y^R := b_m b_{m-1} \cdots b_1$$

Nach umindizieren erhalten wir den String

$$z := c_1 c_2 \cdots c_m .$$

D.h.,

$$c_q = b_{m-q+1} \quad \text{bzw.} \quad b_q = c_{m-q+1}$$

für $1 \leq q \leq m$.

Für $1 < i \leq m$ bezeichne Z_i die Länge des längsten Präfixes von $c_i c_{i+1} \cdots c_m$, der auch Präfix von $c_1 c_2 \cdots c_m$ ist.

Konstruktion $\Rightarrow$

$$N_s = Z_{m-s+1}$$

und

$$Z_r = N_{m-r+1}.$$

Demzufolge genügt zur Lösung der Aufgabenstellung (2") die Entwicklung eines Algorithmus für folgendes maximale Präfixproblem.

maximale Präfixproblem:
gegeben: String $z = c_1 c_2 \cdots c_m$
gesucht: Z_i für $1 < i \leq m$.

Unser Ziel ist nun die Entwicklung eines effizienten Algorithmus für das maximale Präfixproblem. Hierfür ist folgendes Konzept nützlich:

Betrachte die im folgenden Bild gezeichneten Boxen:

Jede Box startet in einer Position j mit $Z_j > 0$. Die Länge der Box, die in j startet, ist Z_j. Dies bedeutet, dass jede Box einen Teilstring maximaler Länge, der gleich einem Präfix von z ist und

nicht in Position 1 beginnt, repräsentiert.

D.h., wir definieren die Z-Box bzgl. j als das Intervall $[j, j + Z_j - 1]$.

Für jedes $i > 1$ bezeichnet r_i den am weitesten rechts liegenden Endpunkt aller Z-Boxen, die in i oder links von i beginnen. D.h.,

$$r_i := \max_{1 < j \leq i} \{ j + Z_j - 1 \mid Z_j > 0 \}.$$

Für jedes $i > 1$ bezeichnet l_i den am weitesten links liegenden Endpunkt aller Z-Boxen, deren rechter Endpunkt r_i ist. D.h.,

$$l_i := \min_{1 < j \leq i} \{ j \mid j + Z_j - 1 = r_i \}.$$

Idee:

Beginnend in $i = 2$ berechnen wir sukzessive die Werte Z_i, r_i und l_i. Dabei werden wir bereits erworbene Kenntnisse derart verwenden, dass insgesamt nur $O(m)$ Vergleiche benötigt werden.

Da wir in jeder i-ten Iteration lediglich die Werte r_{i-1} und l_{i-1} benötigen, verwenden wir zwei Variablen r bzw. l, in denen die zuletzt berechneten r- und l-Werte gespeichert werden.

In der nachfolgenden Beschreibung des Algorithmus enthalten die Klammern (* *) Kommentare.

Algorithmus MAXPREF

Eingabe: String $z = c_1 c_2 \dots c_m$

Ausgabe: Werte Z_i für $2 \leq i \leq m$.

Methode:

(1) $i := 2$;

(2) Vergleiche $c_2 c_3 \dots c_m$ und $c_1 c_2 \dots c_m$ von links nach rechts bis ein Mismatch gefunden ist.

(3) $Z_2 :=$ Länge des gemeinsamen Präfixes von $c_2 c_3 \dots c_m$ und z.

(4) if $Z_2 > 0$
then
$r := 2 + Z_2 - 1$; (* dann gilt $r = r_2$ *)
$\ell := 2$ (* dann gilt $\ell = \ell_2$ *)
else
$r := 0$; $\ell := 0$
fi;

(5) for $i := 3$ to m
do
if $i > r$
then
- Vergleiche $c_i c_{i+1} \dots c_m$ und z von links nach rechts bis ein Mismatch gefunden ist.
- $Z_i :=$ Länge des gemeinsamen Präfixes von $c_i c_{i+1} \dots c_m$ und z.
- if $Z_i > 0$
then

$r := i + Z_i - 1;$
$\ell := i$

fi

else (* $i \leq r$

D.h., die Position i ist in der Z-Box $[\ell, r]$ und c_i im Teilstring $\alpha := c_\ell c_{\ell+1} \cdots c_r$. Der Teilstring α ist auch Präfix von $z = c_1 \cdots c_m$.

$\Rightarrow$

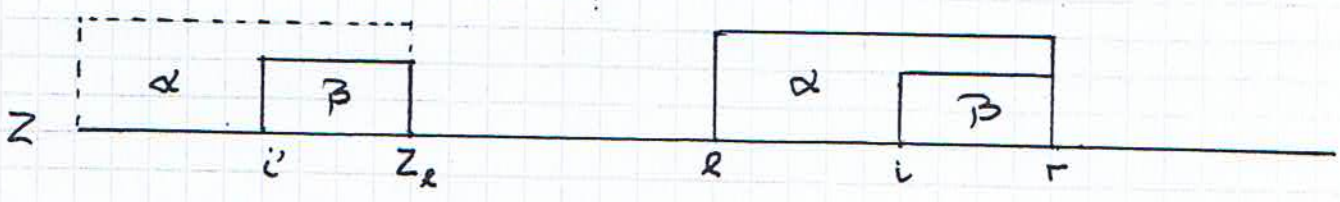
$c_{i-\ell+1} = c_i$

und auch

$$\underbrace{c_{i-\ell+1}}_{i'} c_{i-\ell+2} \cdots c_{Z_\ell} = \underbrace{c_i c_{i+1} \cdots c_r}_{\beta}$$

Da der Teilstring beginnend in Position i' ein Präfix der Länge $Z_{i'}$ von z hat, folgt somit:

Der Teilstring beginnend in Position i ist ein Präfix von z der Länge $\geq \min\{Z_{i'}, |\beta|\}$.



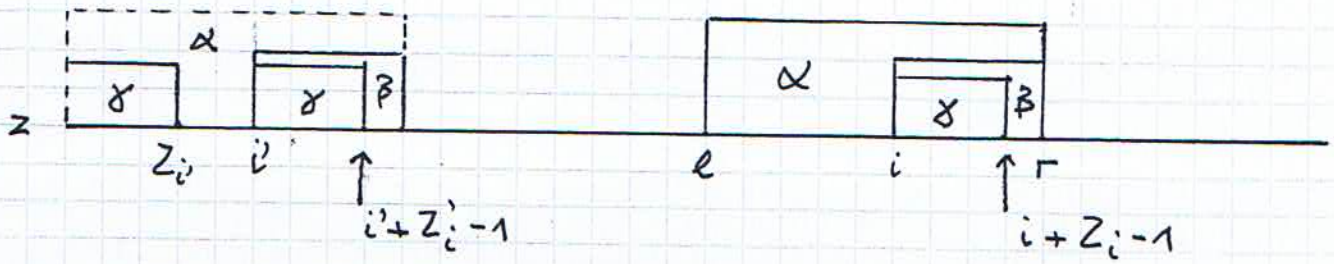
*)

if $Z_{i'} < |\beta|$

then

$Z_i := Z_{i'}$

(*



*)

else (* $Z_{i'} \geq |\beta|$

$\Rightarrow$

$c_i c_{i+1} \dots c_r$ ist Präfix von z und $Z_i \geq |\beta| = r-i+1$.

*)

- Vergleiche $c_{r+1} c_{r+2} \dots c_m$ mit $c_{|\beta|+1} c_{|\beta|+2} \dots c_m$ von links nach rechts bis ein Mismatch gefunden ist. Sei $q \geq r+1$ diejenige Position, die den Mismatch enthält.
- $Z_i := q-i;$
  if $q-1 > r$
  then
  $r := q-1 ;\ e := i$
  fi

od. fi fi

Übung:

a) Beweisen Sie die Korrektheit des Algorithmus MAXPREF.

b) Arbeiten Sie den Algorithmus MAXPREF aus.

Lemma 1.2

Der Algorithmus MAXPREF hat die Laufzeit $O(m)$, wobei m die Länge des Eingabestrings ist.

Beweis:

Jede Iteration benötigt konstante Zeit zuzüglich der Anzahl der Vergleiche, die während der Iteration durchgeführt werden.

$\Rightarrow$

Gesamtzeit $= O(m) + O(GAV)$,

wobei GAV für die Gesamtanzahl der durchgeführten Vergleiche steht.

Ziel: Abschätzung von GAV

Jeder Vergleich resultiert in einen Match oder in einen Mismatch. Bezeichne

A_1 die Gesamtanzahl der Matches und
A_2 die Gesamtanzahl der Mismatches.

$\Rightarrow$

$GAV = A_1 + A_2$.

Jede Iteration, die Vergleiche durchführt, endet, sobald ein Mismatch auftritt.

$\Rightarrow$

$$A_2 \leq m.$$

Für $i \geq 3$ gilt: $r_i \geq r_{i-1}$.

Sei k eine Iteration, in der q Vergleiche mit einem Match enden. Dann gilt:

$$r_k = r_{k-1} + q.$$

Ferner gilt für alle i: $r_i \leq m$.

Insgesamt folgt somit

$$A_1 \leq m.$$

Also gilt

$$GAV \leq 2m.$$

■

Wir haben bewiesen, dass die Vorbereitungszeit für den Algorithmus BM linear in der Länge des Musterstrings ist. Offen ist noch die Laufzeitanalyse für den Algorithmus BM. Wie immer können zwei Kostenmaße zugrunde gelegt werden.

a) erwartete Laufzeit
b) worst case Laufzeit

a) erwartete Laufzeit

Diese kann ermittelt werden durch

- Experimente

  Dabei stellt sich die Frage: Was ist ein geeignetes Experiment? D.h., wie sehen die Problemstellungen in der Praxis aus?

  Literatur hierzu: Siehe Buch von Gusfield, S. 16.

- theoretische Analyse

  Hierbei stellt sich die Frage: Was ist ein realistisches Modell?

b) Laufzeit im worst case

Es ist nahezu trivial zu zeigen, dass der KMP-Algorithmus höchstens 2n Vergleiche benötigt. Wesentlich komplizierter ist es, den BM-Algorithmus zu analysieren. Dies beruht auf folgender Eigenschaft des Algorithmus:

- Im Gegensatz zum KMP-Algorithmus vergisst der BM-Algorithmus bereits berechnete Information.

Beispiel:

```
        ↓       ↓
a c a b a a b c d a b c a a b c a
a a b c d a b c
      ↑
  a a b c d a b c
```

In ihrer Arbeit beweisen Knuth, Morris und Pratt mit einer komplizierten Analyse eine $7n$ obere Schranke für die Anzahl der Vergleiche, die der BM-Algorithmus durchführt. Guibas und Odlyzko haben diese obere Schranke auf $4n$ verbessert. Ihre Analyse ist auch kompliziert.

L. J. Guibas, A. M. Odlyzko: A new proof of the linearity of the Boyer-Moore string searching algorithm, SIAM J. Computing 9 (1980), 672-682.

Cole hat 1994 einen wesentlich einfacheren $4n$-Beweis und einen schwierigen $3n$-Beweis publiziert. In derselben Arbeit hat er Eingaben für den BM-Algorithmus konstruiert und gezeigt, dass diese $3(n-m)$ Vergleiche benötigen.

R. Cole: Tight bounds on the complexity of the Boyer-Moore string matching algorithm, SIAM J. Computing 23 (1994), 1075-1091.

Wir werden uns Cole's $4n$-Analyse erarbeiten. Wir betrachten hierzu den BM-Algorithmus und unterteilen diesen in sogenannte Vergleiche/Shift-Phasen, die wir mit 1 beginnend durchnummerieren.

Vergleiche/Shift-Phase:

$$\begin{array}{lccccccccc} x & a_1 a_2 \dots a_q & a_{q+1} & \dots & a_k & a_{q+j+1} & \dots & & a_{q+m} & \dots a_n \\ & & & & \neq & = & & & = & \\ y & & b_1 & \dots & b_j & b_{j+1} & \dots & & b_m & \end{array}$$

Das rechte Ende des Musterstrings y steht unter einer Position im Eingabestring x. Der Musterstring y wird von rechts nach links mit dem darüber stehenden Teilstring von x verglichen, bis entweder y in x gefunden ist oder ein Mismatch gesichtet. Falls ein Mismatch vorliegt, dann wird y gemäß den Regeln δ_1 und δ_2 nach rechts geschiftet.

Ziel:
Abschätzung der Anzahl der in einer Vergleiche/Shift-Phase vorgenommenen Vergleiche.

Wir ordnen jeden Vergleich demjenigen Textsymbol zu, das an dem Vergleich teilnimmt und unterscheiden zwei Typen von Vergleichen:

- unkritisch: falls das Textsymbol zum ersten Mal an einem Vergleich teilnimmt
- kritisch: sonst.

Da jedes Textsymbol höchstens einmal an einem unkritischen Vergleich teilnimmt, ist die Anzahl aller unkritischen Vergleiche durch die Anzahl n der Textsymbolen beschränkt.

Falls für die Anzahl aller kritischen Vergleiche der Beweis gelingt, dass diese höchstens das Dreifache der Anzahl aller unkritischen Vergleiche ist, dann haben wir eine 4n obere Schranke für die Gesamtanzahl aller Vergleiche bewiesen.

Idee:
Zu jeder Vergleiche/Shift-Phase korrespondieren Länge des durchgeführten Shifts viele unkritische Vergleiche. Diese möchten wir kritischen Vergleichen derart zuordnen, dass wir daraus eine obere Schranke für die Anzahl dieser kritischen Vergleiche herleiten können.

Frage:
Welche kritische Vergleiche sind sinnvolle Kandidaten für diese Zuordnung?

Zur Beantwortung dieser Frage betrachten wir die Situationen, unmittelbar vor und unmittelbar nach dem in der Vergleiche/Shift-Phase durchgeführten Shift.

Vergleiche/Shift-Phase:

$$x = a_1 a_2 \cdots a_q a_{q+1} \cdots a_k a_{q+j+1} \cdots a_{q+m} \cdots a_{q+m+p} \cdots a_n$$

(*)
$$\begin{array}{ccccccc} & & & \neq & = & & = \\ & & b_1 & \cdots & b_j & b_{j+1} & \cdots & b_m \\ & & & \overset{?}{} & = & \cdots & = \\ b_1 & \cdots & b_{g_0} & b_{g_0+1} & \cdots & b_{g_0+(m-j)} & \cdots & b_m \end{array}$$

naheliegende Antworten:

1) Für jedes (der) Textsymbole $a_{q+j+1}, a_{q+j+2}, \ldots, a_{q+m}$, die in der aktuellen Vergleiche/Shift-Phase den Match mit $b_{j+1} b_{j+2} \cdots b_m$ bilden, der nächste Vergleich, an dem dieses Textsymbol teilnimmt.

2) Die kritischen Vergleiche, die in der aktuellen Vergleiche/Shift-Phase vor dem Shift statt-gefunden haben.

Bemerkung:
Beide Zuordnungsarten sehen plausibel aus und sind es auch wert, weiter verfolgt zu werden. Die zweite Zuordnungsart wird zum Ziel führen. Wir werden in der Vorlesung nur die zweite Zuordnungsart analysieren.

Der Algorithmus BM führt den Shift mittels der Operation

$$msende := max\{k + \delta_1(a_k), msende + \delta_2(j)\}$$

durch. Jenachdem, ob dieser gemäß der δ_1- oder der δ_2-Heuristik durchgeführt wird, unterscheiden wir zwei Fälle.

1. Fall: Durchführung gemäß δ_2-Heuristik.

D.h.,

$$msende := msende + \delta_2(j)$$

Es gilt:

$$\delta_2(j) = m - \delta_2'(j),$$

wobei

$$\delta_2'(j) = \begin{cases} \max\{ g + (m-j) \mid 1 \le g < j,\ b_g \neq b_j \\ \qquad \text{und } b_{g+1} b_{g+2} \cdots b_{g+(m-j)} = b_{j+1} \cdots b_m \} \\ \qquad \text{falls solches } g \text{ existiert,} \\ \max\{ \ell \mid b_1 b_2 \cdots b_\ell \text{ ist Suffix von } b_{j+1} \cdots b_m \} \\ \qquad \text{falls nicht } g \text{ aber solches } \ell \\ \qquad \text{existiert} \\ 0 \qquad \text{sonst} \end{cases}$$

Somit können drei Unterfälle eintreten:

1.1

$$\delta_2'(j) = \max\{ g + (m-j) \mid 1 \le g < j,\ b_g \neq b_j \text{ und } b_{g+1} \cdots b_{g+(m-j)} = b_{j+1} \cdots b_m \}.$$

$\Rightarrow$

$$\delta_2(j) = j - g_0,$$

wobei $g_0 + (m-j) = \delta_2'(j)$.

Somit liegt in der Tat die oben beschriebene Situation (*) vor, wobei $p := j - g_0$ die Länge des durchgeführten Shiftes ist.

Sei

$$\bar{y} := b_{j+1} b_{j+2} \cdots b_m = b_{g_0+1} b_{g_0+2} \cdots b_{g_0+(m-j)}.$$

Falls $j - g_0 \geq m - j$, dann können alle in der aktuellen Vergleiche / Shift-Phase durchgeführten kritische Vergleiche paarweise verschiedene durch den Shift der Länge $j - g_0$ bedingte unkritische Vergleiche zugeordnet werden. Somit ist der vorliegende Fall unproblematisch.

<u>Annahme</u>: $j - g_0 < m - j$.

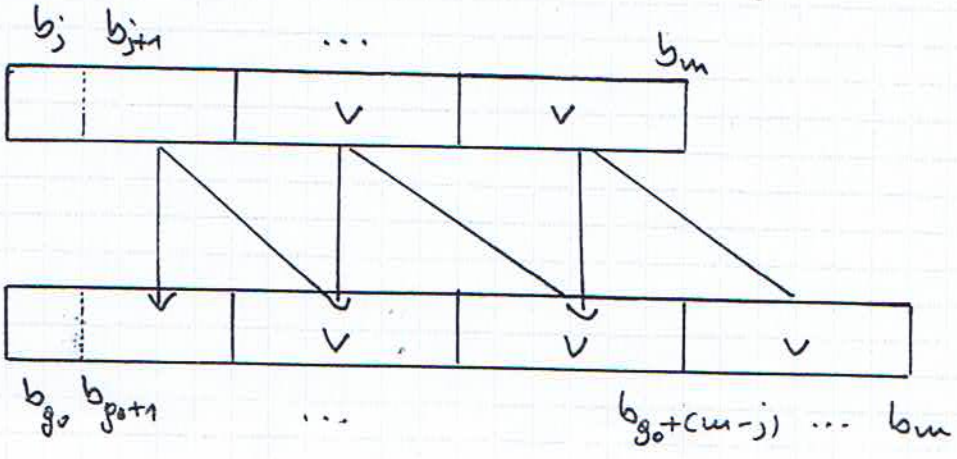
Dann liegt folgende Situation vor:

$b_j \; b_{j+1} \cdots b_m$

$b_{g_0} \; \underbrace{b_{g_0+1} \cdots b_{g_0+(m-j)}}_{\bar{y}} \; \overbrace{b_{g_0+(m-j)+1} \cdots b_m}^{\text{Länge } j-g_0} = v$

Wegen $j - g_0 < m - j$ ist $v := b_{g_0+(m-j)+1} \cdots b_m$ ein echter Suffix von $\bar{y} = b_{j+1} b_{j+2} \cdots b_m$.

Gemäß obiger Situation ist v auch Suffix von $b_{g_0+1} b_{g_0+2} \cdots b_{g_0+(m-j)}$. Diese Betrachtungsweise setzt sich fort, so dass sich folgende Situation ergibt: (insgesamt die)



Also können wir $\bar{y}$ wie folgt schreiben:

$$\bar{y} = u v^k,$$

wobei $k \geq 1$ und $u \in \Sigma^*$ ein Suffix von v ist. Ferner ist $b;u$ auch ein Suffix von v. Falls $u = \varepsilon$, dann gilt $k > 1$.

Wegen $|uv^k| > |v|$ können wir nicht auf triviale Art und Weise die zu uv^k korrespondierenden kritische Vergleiche paarweise unterschiedlichen zu v korrespondierenden kritischen Vergleiche zuordnen.

$\Rightarrow$

Eine genauere Analyse der obigen Situation ist notwendig.

Bezeichnungen:
Eine Periode eines Strings s ist die Länge $|v|$ eines Suffixes v von s, so dass $s = uv^k$ für ein $k \geq 1$ und einen möglicherweise leeren Suffix u von v. Periode(s) bezeichnet stets die kleinste Periode von s.

Lemma 1.3
Seien v und w zwei nichtleere Strings, so dass $vw = wv$. Dann existieren ein String u und natürliche Zahlen i und j, für die gilt:

$$v = u^i \quad \text{und} \quad w = u^j.$$

Beweis: (mittels Induktion über $|v| + |w|$)

$|v| + |w| = 2$:

Dann gilt: $vw = wv \Rightarrow v = w$.

Setze $u := v$, $i := 1$ und $j := 1$

$\Rightarrow$ $v = u^i$ und $w = u^j$.

Annahme:

Die Behauptung des Lemmas ist erfüllt für alle $v, w \in \Sigma^+$ mit $|v| + |w| \leq \ell$, wobei $\ell \geq 2$.

$\ell \rightsquigarrow \ell + 1$:

Betrachte $v, w \in \Sigma^+$ mit $|v| + |w| = \ell + 1$ beliebig, aber fest.

1. Fall: $|v| = |w|$

Dann gilt wiederum $vw = wv \Rightarrow v = w$

Setze $u := v$, $i := 1$ und $j := 1$

$\Rightarrow$ $v = u^i$ und $w = u^j$.

2. Fall: $|v| \neq |w|$

O.B.d.A. sei $|v| < |w|$. Dann gilt

$(vw = wv \wedge |v| < |w|) \Rightarrow$

v ist ein echter Präfix von w. D.h.,

$\exists\, z \in \Sigma^+$ mit $w = vz$.

Substitution von vz für w in der Gleichung $vw = wv$ ergibt dann

$$vvz = vzv.$$

Streichen der linken Kopie von v auf beiden Seiten der Gleichung ergibt

$$vz = zv.$$

Wegen

$$|v| + |z| = |w| < |v| + |w| = e+1$$

gilt

$$|v| + |z| \leq e.$$

Induktionsannahme $\Rightarrow$

$$\exists\, u \in \Sigma^+,\ i, j \in \mathbb{N} \text{ mit } v = u^i \text{ und } z = u^j$$

Also gilt

$$w = vz = u^i u^j = u^{(i+j)},$$

womit das Lemma bewiesen ist. ∎

Übung:
Seien $s = vw = wv$ für $v, w \in \Sigma^+$. Zeigen Sie, dass s Periodizität t mit $t \leq \min\{|v|, |w|\}$ besitzt.

Nun können wir mit der Analyse des Algorithmus BM fortfahren. Folgende Situation liegt vor:

$$
\begin{array}{l}
a_1 a_2 \dots a_q a_{q+1} \dots a_{q+p+1} \dots a_z a_{q+j+1} \dots a_{q+m} \dots a_{q+m+p} \dots a_n \\
\qquad\qquad\qquad\qquad\qquad\quad \neq \;=\; \dots \;=\; \\
\qquad b_1 \dots \qquad\qquad\quad b_j\, b_{j+1} \dots b_m \\
\qquad\qquad\qquad\qquad\quad \overset{?}{=} \;=\; \dots \;=\; \\
\qquad\qquad\quad b_1 \dots b_{g_0}\, b_{g_0+1} \dots b_{g_0+(m-j)} \underbrace{\dots b_m}_{v}
\end{array}
$$

wobei $p := j - g_0$ und $j - g_0 < m - j$.

Idee:
Zeige, dass in der aktuellen Vergleiche/Shift-Phase viele der durchgeführten Vergleiche unkritisch sind, so dass die Anzahl der kritischen Vergleiche die Länge $j - g_0$ des durchgeführten Shiftes nicht allzu sehr überschreiten kann.

Durchführung:

Sei z der kürzeste Teilstring von v, so dass $v = z^\ell$ für ein $\ell \in \mathbb{N}$. Wegen $v = v^1$ existiert der Teilstring z. z heißt Generator von v.

Oben haben wir uns überlegt, dass

$$\bar{y} = u v^k,$$

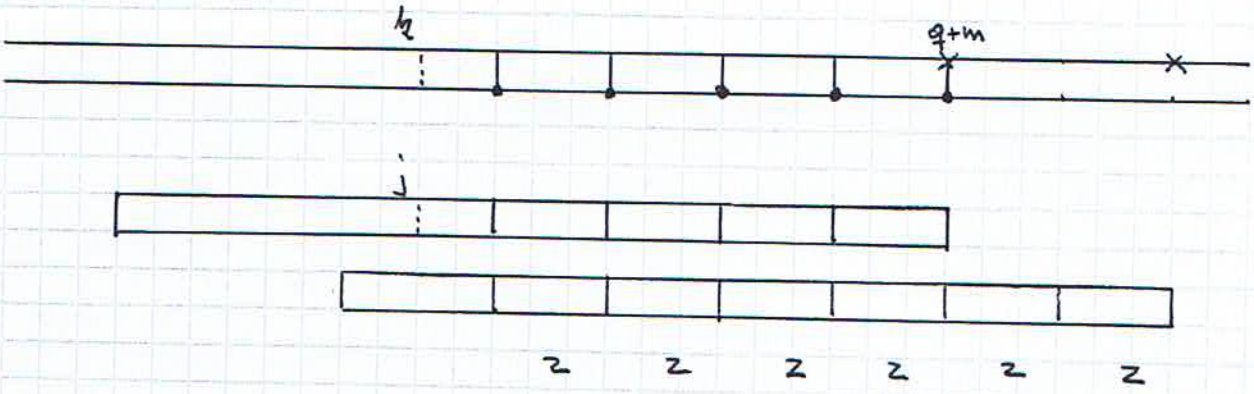
wobei $k \geq 1$ und $u \in \Sigma^*$ ein Suffix von v ist. Ferner ist auch $b_j u$ ein Suffix von v. Also gilt folgendes

Lemma:

Lemma 1.4

$|z|$ ist Periode von $\bar{y}$ und auch von $b_j \bar{y}$.

Lemma 1.4 in oben skizzierte Situation mit einbezogen ergibt folgendes Bild:



- Generatorendposition in x
- x Mstringendposition in x.

Zur Charakterisierung derjenigen Vergleiche, die kritisch sein können, müssen wir obige Situation sorgfältig analysieren und einige Eigenschaften herausarbeiten. Zunächst werden wir zeigen, dass in vorangegangenen Phasen die Mstringendposition niemals gleich einer aktuellen Generatorendposition war.

Lemma 1.5

In keiner vorangegangenen Vergleiche/Shift-Phase war die Endposition des Musterstrings gleich einer aktuellen Generatorendposition.

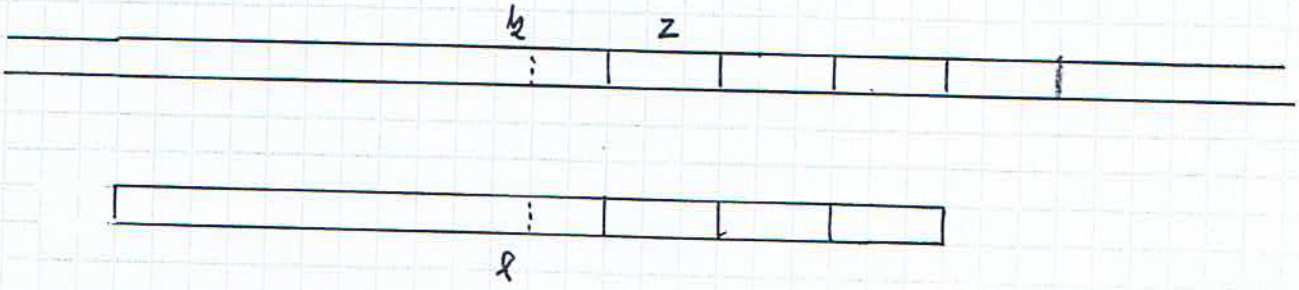
Beweis:

Solange der Musterstring nicht gefunden ist, schließt jede Vergleiche/Shift-Phase mit einem Rechtsshift ab. Da die Generatorendposition $q+m$ die aktuelle Endposition des Musterstrings ist, muss in jeder Vorgängerphase das Musterstringende echt links von $q+m$ liegen.

Annahme:

Die Endposition des Musterstrings liegt in einer Vorgängerphase in einer aktuellen Generatorendposition echt links von $q+m$.

Dann liegt folgende Situation vor:



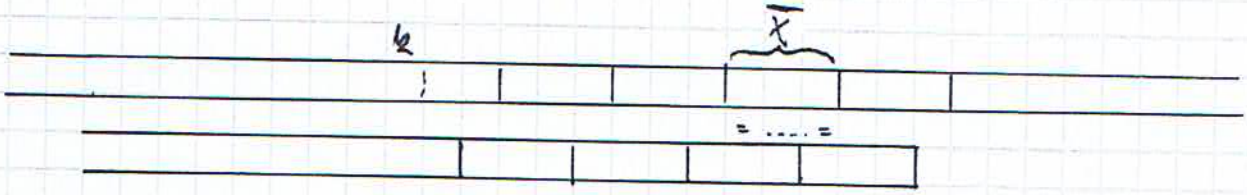
Lemma 1.4 $\Rightarrow$ $b_\ell = b_j$

Also muss die Vergleichsphase der betrachteten Vorgängerphase mit einem Mismatch zwischen b_ℓ und a_ℓ enden.

Wir betrachten nun den Shift, der gemäß der δ_2-Heuristik durchgeführt würde.

Da nach dem Shift gemäß der δ_2-Heuristik unter a_2 ein $b_4 \neq b_e$ steht, muss das Ende des Shifts echt innerhalb eines der Generatoren liegen.

Nach dem Shift liegt somit folgende Situation vor:



Gemäß unserer δ_2-Heuristik stimmen der unter $\bar{x}$ stehende Teilstring des Musterstrings und $\bar{x}$ überein.

$\Rightarrow$

$z = z_1 z_2 = z_2 z_1$ für zwei nichtleere Strings z_1 und z_2

Lemma 1.3 $\Rightarrow$

z ist nicht der kürzeste Teilstring von v, so dass $v = z^\ell$ für ein $\ell \in \mathbb{N}$.

Dies ist ein Widerspruch zur Wahl von z.

■

Bemerkung:
Für den Beweis des obigen Lemmas ist es unerheblich, ob der Algorithmus BM in der Tat den Shift gemäß der δ_2-Heuristik durchführt oder nicht.

Mit Hilfe von Lemma 1.5 lässt sich leicht das folgende Lemma beweisen.

Lemma 1.6

In jeder vorangegangenen Vergleich / Shift - Phase matcht der Musterstring y in $a_{q+j+1} a_{q+j+2} \cdots a_{q+m}$ höchstens $|z| - 1$ Symbole.

Beweis:

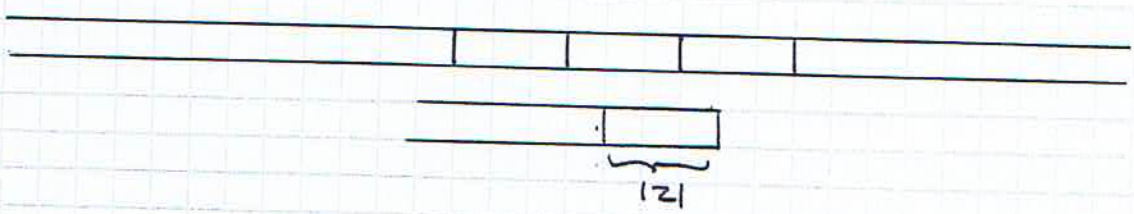
Annahme:

y matcht $a_{q+j+1} \cdots a_{q+m}$ in einem Teilstring der Länge $\geq |z|$.

Lemma 1.5 $\Rightarrow$

Die Endposition von y ist nicht gleich einer Generatorendposition

Also liegt folgende Situation vor:



Demzufolge wäre $z = z_1 z_2 = z_2 z_1$ für zwei nicht-leere Strings z_1 und z_2, was wegen der Wahl von z nicht sein kann.

∎

Folgender Lemma grenzt die möglichen Endpositionen des Musterstrings y während vorangegangenen Vergleiche/Shift-Phasen innerhalb von $a_{q+j+1} \cdots a_{q+m}$ ein.

Lemma 1.7

Falls in einer vorangegangenen Vergleiche/Shift-Phase die Endposition des Musterstrings y innerhalb von $a_{q+j+1}\, a_{q+j+2} \cdots a_{q+m}$ liegt, dann befindet sich diese innerhalb der ersten $|z|-1$ Positionen oder innerhalb der letzten $|z|$ Positionen in $a_{q+j+1} \cdots a_{q+m}$.

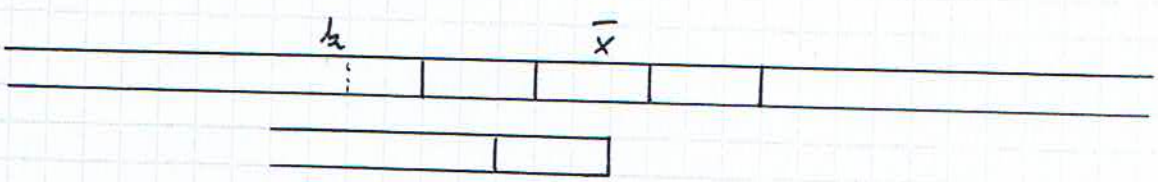
Beweis:

Annahme:

Die Endposition des Musterstrings befindet sich in einer vorangegangenen Phase rechts von den ersten $|z|-1$ und links von den letzten $|z|$ Positionen innerhalb von $a_{q+j+1}\, a_{q+j+2} \cdots a_{q+m}$.

Lemma 1.5 $\Rightarrow$

Die Endposition des Musterstrings y befindet sich echt innerhalb eines Generators $\bar{x}$. D.h., es liegt folgende Situation vor:



Lemma 1.6 $\Rightarrow$

Die Länge des Matches, mit dem die Vergleichsphase endet, ist $\leq |z|-1$.

$\Rightarrow$

Der Mismatch erfolgt innerhalb von $a_{q+j+1} \cdots a_{q+m}$.

Gemäß der δ_2-Heuristik wird der Musterstring nach rechts verschoben und zwar so wenig wie möglich, so dass

- der aktuelle Match nach dem Shift wieder gematcht wird und
- unter dem Symbol im Textstring, das den Mismatch bedingt hat, nach dem Shift ein anderes Musterstringsymbol steht.

Betrachten wir den Shift, der die Endposition des Musterstrings unter die Endposition von $\bar{x}$ platzieren würde.

Da $\bar{x} = z$ Generator von $\bar{y}$ ist, würde innerhalb von $a_{q+j+1} a_{q+j+2} \cdots a_{q+m}$ kein Mismatch erfolgen.
$\Rightarrow$

- der aktuelle Match wird nach dem Shift wieder gematcht und
- unter dem Symbol in x, das den Mismatch bedingt hat, steht nun ein anderes Musterstringsymbol.

Somit erfüllt der betrachtete Shift bis eventuell auf die Minimalität die δ_2-Heuristik.

Lemma 1.5 $\Rightarrow$

Dieser Shift ist nicht möglich.

$\Rightarrow$

Gemäß der δ_2-Heuristik muss ein Shift erfolgen, der das Musterstringende echt innerhalb $\bar{x}$ belässt.

Übung:
Zeigen Sie, dass auch gemäß der δ_1-Heuristik nur ein Shift erfolgen kann, der das Musterstringende echt innerhalb $\bar{x}$ belässt.

Da dies für jede Phase, die mit Musterstringende echt innerhalb $\bar{x}$ startet, gilt, kann das Musterstringende niemals über die Endposition von $\bar{x}$ hinausgeschoben werden.

Dies ist unmöglich, da in der aktuellen Vergleich/Shift-Phase das Musterstringende echt rechts von $\bar{x}$ steht.

Also ist unsere Annahme falsch und das Lemma somit bewiesen.

■

Nun können wir die Zuordnung der während der aktuellen Vergleiche/Shift-Phase erfolgten kritischen Vergleiche an die zum Shift korrespondierenden unkritischen Vergleiche durchführen.

Gemäß unserer Konstruktion ist die Länge $|v|$ des erfolgten Shiftes mindestens $|z|$, wobei z der Generator von v ist.

Lemma 1.7 $\Rightarrow$

Falls in einer vorangegangenen Vergleiche/Shift-Phase die Endposition des Musterstrings y innerhalb des aktuellen Matches $a_{q+j+1} a_{q+j+2} \cdots a_{q+m}$ liegt, dann befindet sich diese innerhalb der ersten $|z|-1$ oder innerhalb der letzten $|z|$ Positionen des Matches.

Lemma 1.6 $\Rightarrow$

In jeder vorangegangenen Vergleiche/Shift-Phase matcht der Musterstring y in $a_{q+j+1} \cdots a_{q+m}$ höchstens $|z|-1$ Symbole.

$\Rightarrow$

Ein Musterstring, dessen Endposition innerhalb der letzten $|z|$ Positionen sich befindet, kann nur Symbole innerhalb der letzten $2|z|-1$ Positionen matchen.

$\Rightarrow$

Höchstens $3|z|-2$ der $m-j$ Vergleiche können kritisch sein.

$\Rightarrow$

Wir können die kritischen Vergleiche der aktuellen Vergleiche/Shift-Phase oberat oder $\geq |z|$ zum Shift korrespondierenden unkritischen Vergleiche zuordnen, dass

1) jeder unkritische Vergleich höchsten drei kritische Vergleiche zugeordnet bekommt und
2) mindestens ein unkritisches Vergleich weniger als

drei kritische Vergleiche zugeordnet bekommt.

1.2

$\delta_2'(j) = \max\{\ell \mid b_1 b_2 \dots b_\ell \text{ ist Suffix von } b_{j+1} \dots b_m\}$.

$\Rightarrow$

$\delta_2(j) = m - \ell_0$, wobei $\ell_0 := \delta_2'(j)$.

Somit liegt folgende Situation vor:

$$\begin{array}{ccccccccccccc} a_1 \dots a_q & a_{q+1} \dots a_k & a_{q+j+1} & \dots & a_{q+m-\ell_0+1} & \dots & a_{q+m} & \dots & a_{q+2m-\ell_0} & \dots & a \\ & \neq \quad = & \dots & & = & \dots & = & & & & \\ b_1 \dots b_j & b_{j+1} & \dots & b_{m-\ell_0+1} & \dots & b_m & & & & & \\ & = & \dots & = & & & & & & & \\ & b_1 & \dots & b_{\ell_0} & \dots & b_m & & & & & \end{array}$$

Falls $m - \ell_0 \geq m - j$, dann können alle in der aktuellen Vergleiche (Shift-Phase durchgeführten kritischen Vergleiche paarweise verschiedene durch den Shift der Länge $m - \ell_0$ bedingte unkritische Vergleiche zugeordnet werden. Somit ist der vorliegende Fall unproblematisch.

Annahme: $m - \ell_0 < m - j$

Dann liegt folgende Situation vor:

$$\begin{array}{ccccc} b_1 b_2 \dots b_{m-\ell_0+1} & \dots & b_m & = & \text{Länge } m-\ell_0 \\ = & \dots & = & & \\ \underbrace{\qquad\qquad}_{\text{Länge } m-\ell_0} b_1 & \dots & b_{\ell_0} & \underbrace{b_{\ell_0+1} \dots b_m}_{v} & \end{array}$$

Analog zum Fall 1.1 überlegen wir uns, dass $u - l_0$ eine Periode des Musterstrings y ist. Wir betrachten nun den kürzesten Teilstring z von v mit $v = z^l$ für ein $l \in \mathbb{N}$.

Wir können nun zu den Lemmata 1.5 - 1.7 analoge Lemmata formulieren, wobei y die Rolle von $\bar{y}$ spielt. Diese können dann genauso wie die Lemmata 1.5 - 1.7 bewiesen werden.

..

Übung:
Führen Sie die Analyse des Falles 1.2 durch.

1.3

$$\delta_2'(j) = 0$$
$$\Rightarrow$$
$$\delta_2(j) = m.$$

D.h., der Musterstring wird um m Positionen nach rechts geschoben. Wegen $m > m - j$ ist der vorliegende Fall unproblematisch.

2. Fall Durchführung gemäß δ_1-Heuristik.

Die Analyse des 1. Falles hängt nicht davon ab, ob der Shift in der Tat gemäß der δ_2-Heuristik erfolgt. Falls der Shift nicht gemäß der δ_2-Heuristik durchgeführt wird, dann stehen mehr unkritische Vergleiche für die Zuordnung zur Verfügung, als dies nach Durchführung des Shiftes

gemäß der δ_2-Heuristik der Fall wäre. Somit kann auch bei der Durchführung des Shiftes gemäß der δ_1-Heuristik die Zuordnung durchgeführt werden.

In allen diskutierten Fällen können wir den zum Mismatch korrespondierenden Vergleich einem zum Shift korrespondierenden unkritischen Vergleich, dem maximal zwei kritische Vergleiche zugeordnet wurden, zuordnen. Insgesamt haben wir folgenden Satz bewiesen:

Satz 1.2

Der Algorithmus BM führt maximal $4n$ Vergleiche durch, wobei $n := |x|$ die Länge des Textstrings ist.

Ziel:

Berechnung aller Vorkommen des Musterstrings y in Textstring x in $O(n+m)$ Zeit.

Idee:

Nach jedem Vorkommen von y in x führe den minimalen Rechtsshift, so dass übereinander stehende Suffix gemäß der alten Position und Präfix gemäß der neuen Position von y übereinstimmen, durch. Setze dann den Algorithmus BM fort.

Wie folgendes Beispiel zeigt, könnte dies zu einer Laufzeit von $O(m \cdot n)$ führen:

Beispiel:

$$x = \underbrace{aa \dots a}_{n\text{-mal}} \quad , \quad y = \underbrace{aa \dots a}_{m\text{-mal}}$$

Der Algorithmus BM würde bei obiger Vorgehensweise für jedes der $n-m+1$ Vorkommen von y in x m Vergleiche durchführen. Die Gesamtanzahl der durchgeführten Vergleiche wäre dann $m \cdot n - m^2 + m$.

◆

Bemerkung:
Im obigen Beispiel hat der Musterstring y maximal mögliche Periodizität m. Eine kleine Periode des Musterstrings haben alle Beispiele, bei denen obige Vorgehensweise eine hohe Laufzeit nach sich zieht.

In

Zvi Galil, On Improving the Worst Case Running Time of the Boyer-Moore String Matching Algorithm, CACM 22 (1979), 505-508.

hatte Galil folgende Idee:

- Immer wenn der Musterstring y im Textstring x gefunden ist, vermeide mit Hilfe der Kenntnis von Periode (y) redundanten Testen.

Für die Durchführung obiger Idee, insbesondere für den Beweis der Korrektheit der Methode,

benötigen wir eine weitere Eigenschaft der Perioden eines Strings.

Erinnerung:
Für die Analyse des Algorithmus BM war folgende Definition der Periode eines Strings s sinnvoll:

Eine Periode eines Strings s ist die Länge $|v|$ eines Suffixes v von s, so dass $s = uv^k$ für ein $k \geq 1$ und einen möglicherweise leeren Suffix u von v.

Folgende äquivalente Definition ist für die Durchführung der obigen Idee geeigneter:

Eine Periode eines Strings s ist die Länge $|v|$ eines Präfixes v von s, so dass $s = v^k u$ für ein $k \geq 1$ und einen möglicherweise leeren Präfix u von v.

Übung:
Beweisen Sie, dass beide Definitionen einer Periode eines Strings s äquivalent sind.

Folgendes Lemma ist das in der Literatur wohlbekannte so genannte Periodizitätslemma.

Lemma 1.8
Seien p und q zwei Perioden eines Strings s. Falls $p + q \leq |s|$, dann ist $ggT(p,q)$ auch eine Periode des Strings s.

Beweis:

Falls $p = q$, dann ist wegen $ggT(p,q) = p$ die Behauptung trivialerweise erfüllt. Sei $p \neq q$. O.B.d.A. können wir $p > q$ annehmen.

Beh.(*):

Sei s ein String. Für alle Perioden a und b von s gilt: $(a+b \leq |s| \wedge a > b) \Rightarrow a-b$ ist Periode von s.

Zunächst werden wir uns davon überzeugen, dass die Beh. (*) das Lemma impliziert und danach die Beh. (*) beweisen.

Für die Reduktion des Lemmas auf die Beh. (*) betrachten wir die Berechnung des $ggT(p,q)$ durch den Euklid'schen Algorithmus. Dieser sieht wie folgt aus:

$$
\begin{aligned}
p &= c_1 q + r_1 && 0 < r_1 < q \\
q &= c_2 r_1 + r_2 && 0 < r_2 < r_1 \\
r_1 &= c_3 r_2 + r_3 && 0 < r_3 < r_2 \\
&\vdots \\
r_{j-2} &= c_j r_{j-1} + r_j && 0 < r_j < r_{j-1} \\
r_{j-1} &= c_{j+1} r_j
\end{aligned}
$$

Es gilt dann: $ggT(p,q) = r_j$.

c_1 - maliges Anwenden von (*), wobei a zu Beginn den Wert p und danach den Wert der aktuellen Differenz und b stets den Wert q erhalten,

ergibt:

r_1 ist Periode von s.

c_2-maliges Anwenden von (*), wobei q die Rolle von p und r_1 die Rolle von q spielt, ergibt:

r_2 ist Periode von s

$\vdots$

r_{j-1} ist Periode von s

c_j-maliges Anwenden von (*) ergibt

r_j ist Periode von s

Wegen $ggT(p,q) = r_j$ folgt somit das Lemma.

Also fehlt nur noch der Beweis der Beh. (*).

Bew. d. Beh. (*):

Sei $s := a_1 a_2 \dots a_n$. Zu zeigen ist, dass a-b eine Periode von s ist. Hierzu genügt es zu zeigen, dass für $1 \leq i \leq n$ folgendes erfüllt ist.

$$a_i = \begin{cases} a_{i+(a-b)} & \text{falls } i+(a-b) \leq n \\ a_{i-(a-b)} & \text{falls } i-(a-b) \geq 1 \end{cases}$$

Betrachte $i \in \{1, 2, \dots, n\}$ beliebig, aber fest.

$a+b \leq n \Rightarrow$

i) $i + a \leq n \quad \vee \quad i - b \geq 1$ und

ii) $i - a \geq 1 \quad \vee \quad i + b \leq n$.

Wir können nun $i + (a-b)$ bzw. $i - (a-b)$ auf zwei Arten und Weisen umschreiben:

$$i + (a-b) = (i+a) - b \quad \text{oder}$$
$$i + (a-b) = (i-b) + a$$

bzw.

$$i - (a-b) = i - a + b \quad \text{oder}$$
$$i - (a-b) = i + b - a.$$

Wir diskutieren zunächst den Fall $i + (a-b) \leq n$.
Falls $i+a \leq n$, dann folgt, wegen a und b sind Perioden von s sind:

$$a_i = a_{i+a} = a_{(i+a)-b} = a_{i+(a-b)}.$$

Falls $i-b \geq 1$, dann folgt, wegen a und b sind Perioden von s sind:

$$a_i = a_{i-b} = a_{(i-b)+a} = a_{i+(a-b)}.$$

Den Fall $i - (a-b) \geq 1$ beweist man analog. ∎

Übung:
Führen Sie im Beweis der Beh. ⊛) den Fall $i - (a-b) \geq 1$ durch.

Folgendes Lemma besagt, dass eine große Anzahl von Vorkommen des Musterstrings y in Textstring x nur möglich ist, wenn Periode(y) klein ist.

Lemma 1.9

Seien $n := |x|$, $m := |y|$, $m < n$ und r die Anzahl der Vorkommen des Musterstrings y in Textstring x. Dann gilt: $r \geq \frac{2n}{m} \Rightarrow \text{Periode}(y) \leq \frac{m}{2}$.

Beweis:

Zunächst überlegen wir uns, dass es zwei Vorkommen von y in x, die sich mindestens in $\frac{m}{2}$ Positionen überlappen, gibt.

Annahme:
Keine Vorkommen von y in x überlappen sich in $\geq \frac{m}{2}$ Positionen.

$\Rightarrow$
Nach einem Vorkommen von y in x kann das nächste Vorkommen von y frühestens nach $\frac{m}{2}+1$ Positionen beginnen.

$\Rightarrow$

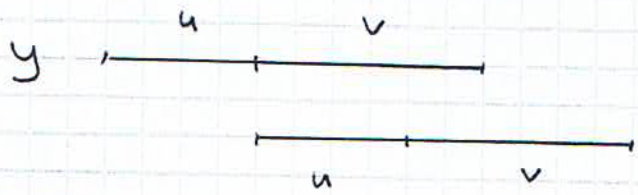
$$|x| \geq \left(\frac{2n}{m} - 1\right)\left(\frac{m}{2} + 1\right) + m$$

$$= n + \frac{2n}{m} - \frac{m}{2} - 1 + m$$

$$> n$$

Widerspruch!

Somit liegt folgende Situation vor:



mit $|u| \leq \frac{m}{2}$

Also ist $|u|$ eine Periode von y.

$\Rightarrow$

$\text{Periode}(y) \leq \frac{m}{2}$

∎

Übung

Beweisen Sie: $\text{Periode}(y) > \frac{m}{2} \Rightarrow$ Die Laufzeit des Algorithmus BM ist $O(n+m)$.

Für die folgenden Überlegungen nehmen wir stets $\text{Periode}(y) \leq \frac{m}{2}$ an. u bezeichnet stets denjenigen Präfix von y mit $|u| = \text{Periode}(y)$.

Übung:

Entwickeln Sie einen effizienten Algorithmus, der für einen gegebenen String s $\text{Periode}(s)$ berechnet.

Ziel:

Mittels einer einfachen Modifikation des Algorithmus BM dafür zu sorgen, dass auch bei Musterstrings y mit $\text{Periode}(y) \leq \frac{m}{2}$ stets der modifizierte Algorithmus lineare Laufzeit hat.

Betrachten wir hierzu die Vorkommen des Musterstrings y im Textstring x. Seien p und q zwei Positionen in x, in denen jeweils ein Vorkommen von y beginnt. Wir sagen, die Vorkommen p und q sind nahe beieinander, falls $|p-q| \leq \frac{m}{2}$.

Wir definieren die Nachbarschaftsrelation als reflexive, transitive Hülle von "nahe beieinander".

Ein Block ist eine maximale Klasse von Nachbarn. Folgendes Lemma ist leicht zu beweisen:

Lemma 1.10

Sei B ein Block und sei p die maximale Position in B. Dann gilt:

i) Ein neuer Block B' kann nicht vor Position $p + \frac{m}{2}$ beginnen.

ii) Die Anzahl der Blöcke ist $\leq \frac{2n}{m}$.

Beweis:

Übung

Folgendes Lemma besagt, dass aufeinanderfolgende Positionen innerhalb eines Blockes stets den Abstand Periode (y) haben.

Lemma 1.11

Sei B ein Block und seien $p_1, p_2, \ldots, p_r$, $r > 1$

die Anfangspositionen von y in TS in streng monoton wachsender Ordnung. Dann gilt für $1 < i \leq r$ $p_i - p_{i-1} = \text{Periode}(y)$.

Beweis:

Sei $k := \text{Periode}(y)$.

Betrachten wir $1 < i \leq r$ beliebig, aber fest.
Sei

$$k' := p_i - p_{i-1}.$$

Da p_i und p_{i-1} nahe beieinander liegen, gilt:

$$k' \leq \frac{m}{2}.$$

Genauso, wie im Beweis von Lemma 1.9 folgt nun

k' ist eine Periode von y.

Wegen $k' + k \leq m$ folgt aus Lemma 1.8

$ggT(k', k)$ ist eine Periode von y.

Wegen $\text{Periode}(y) = k$ gilt: $ggT(k', k) = k$

$\Rightarrow$

$$k' = \ell \cdot k \quad \text{für ein } \ell \in \mathbb{N}.$$

Falls $\ell = 1$, dann ist nichts mehr zu beweisen.

Annahme: $\ell > 1$

Aus

- Periode(y) $= k$ und
- in p_{i-1} und $p_{i-1} + \ell \cdot k$, $\ell > 1$ starten Vorkommen von y in x

folgt direkt:

In Position $p_{i-1} + k < p_i$ startet ein Vorkommen von y in x.

Dies ist ein Widerspruch zur Konstruktion des Blockes B. Somit war unsere Annahme falsch und das Lemma ist bewiesen. ∎

Idee:

Der Algorithmus BM findet jedes Vorkommen des Musterstrings y im Textstring x zu einem Zeitpunkt. Geschickter wäre es, stattdessen stets einen gesamten Block zu einem Zeitpunkt zu finden.

Den resultierenden modifizierten Algorithmus nennen wir BM'. Der Algorithmus BM' arbeitet wie folgt:

- Solange kein Musterstring y in x gefunden ist, arbeitet BM' genauso, wie der Algorithmus BM.

- Sobald ein Vorkommen von y in einer Position q gefunden wird, startet BM' genauso wie BM die Suche nach dem nächsten Vorkommen in Position q+k, wobei k = Periode(y). Im Gegensatz zu BM untersucht BM' jedoch nur die letzten k Symbole des Musterstrings y, ob diese mit den korrespondierenden Symbole im Textstring übereinstimmen.
- Ist dies der Fall, dann ist das nächste Vorkommen von y in x gefunden und BM' sucht nun genauso, wie oben beschrieben, in Position q+2k weiter.
- Ist dies nicht der Fall, dann arbeitet BM' genauso wie BM weiter, bis das nächste Vorkommen von y in x gefunden ist.

Übung:

a) Arbeiten Sie den Algorithmus BM' aus.
b) Beweisen Sie die Korrektheit des Algorithmus BM'.
c) Zeigen Sie, dass die Laufzeit des Algorithmus BM' linear ist.