2.6 Boyer Moore

2.6.1 right-to-left-scan

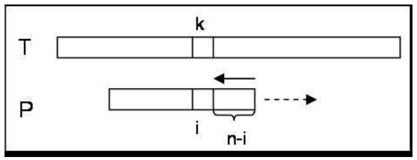
2.6.2 Bad Character- ule

Definition 11

für $a \in \mathcal{A}$ ist R(x) die am weitesten rechts auftretende Position von $x \in P$ oder 0, falls $x \notin P$

$$R(x) := \max \quad \{i | 1 \leq i \leq n \wedge P(i) = x\} \cup \emptyset$$

bad character shift rule für eine gegebene Verschiebung von P gegen T



- match für $P[i+1..n]$
- mismatch für $P(i)$

verschiebe P um $\max\{1\ ,\ i - R(T(k))\}$

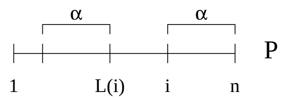
Erweiterung zur extended bad-character-shift-rule

2.6.3 (Strong) good-suffix-rule

2.6.4 Formalisierung und Vorverarbeitung

Definition 12 *(für good suffix rule, Fall Ia = mis match)*

sei P string für $1 \leq i \leq n = |P|$ definieren wir L(i) als die maximal (d.h. die am weitesten rechts stehende Position) $< n$, so dass



- *$P[i\,..\,n]$ ein Suffix von $P[1..L(i)]$ ist,*
  d.h. $\alpha = P[i\,..\,n] = P[L(i) - (n-i)\,..\,L(i)]$
- *oder $L(i) = 0$, falls es keine solche Position gibt*

formal: $L(i) = \max \quad \emptyset \cup \{j | j < n \wedge (P[i\,..\,n] = P[j - (n-i)\,..\,j)\}$

Definition 13
von $L'(i)$ analog, allerdings wird zusätzlich gefordert, dass das Zeichen vor den zwei Kopien von α – falls beide im String – ungleich sind:

$$\begin{aligned} L'(i) := \max \quad \emptyset \cup \{j | j < n \quad & \wedge (P[i\,..\,n] = P[j - (n-i)\,..\,j] \\ & \wedge (P(i-1) \neq P(j-(n-i)-1) \ \vee \ j - (n-i) - 1 = 0)\} \end{aligned}$$

Definition 14 *(für good suffix rule, Fall II = match oder Fall Ib)*

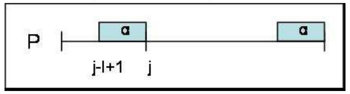
sei P string, $l'(i)$ ist die Länge des längsten Präfixes von P, das auch echtes Suffix von $P[i\,..\,n]$ ist

falls kein solches Präfix existiert ist l'(i)=0

formal: $l'(i) := \max \quad \emptyset \cup \left\{ j | 1 \leq j \leq \underbrace{n - i + 1}_{|P[i\,..\,n]|} \wedge P[1\,..\,j] = P[n - j + 1\,..\,n] \right\}$

Definition 15
sei P string, für $1 \le j < n$ ist

$N_j(P)$ *die Länge des längsten Suffixes von P[1 .. j], das auch Suffix von P ist*



$$N_j(P) := \max \emptyset \cup \left\{ l | 1 \le l \le j \wedge (\underbrace{P[j-l+1 \,..\, j]}_{\alpha} = \underbrace{P[n-l+1 \,..\, n]}_{\alpha}) \right\}$$

Berechung von $L(i) und L'(i)$

for i=1 to n do L'(i) := 0 endfor

for j=1 to n-1 do
if N_j(P)>0 then
$i := n - N_j(P) + 1$
$L'(i) := j$
endif
endfor

L(2):=L'(2)
for i=3 to n do
$L(i) := \max \{L(i-1), L'(i)\}$
endfor

strong good suffix shift rule

für eine gegebene Verschiebung von P gegen T trete

- ein Missmatch auf: $P(i-1) \neq T(k-1), i \le n$
  → falls $L'(i) > 0$, dann verschiebe um $n - L'(i)$-Positionen
  → falls $L'(i) = 0$, dann verschiebe um $n - l'(i)$-Positionen
- ein Missmatch auf: $P(n) \neq T(k)$
  → dann verschiebe um eine Position
- ein Vorkommen von P in T auf
  → dann verschiebe um $n - l'(2)$-Positionen

2.6.5 Kompletter Boyer Moore

//Vorverarbeitung
berechne

- Z_i für 1 ≤ i ≤ n
- N_i, L'(i), l'(i) für 2 ≤ i ≤ n
- $R(x), x \in \mathcal{A}$

```
//Vergleich
h:=n    //rechter Rand von P in T
while h ≤ m do
   i:=n //Vergleichsposition in P
   k:=h //Vergleichsposition in T
   while i > 0 ∧ P(i) = T(k) do
      i–;k–;
   endwhile
   if i==0 then
      printf'Vorkommen' //Vorkommmen von P in T
      h+ = n − l'(2)
   else
      h+ = max{Verschiebug aus b.c.s.r., Verschiebung aus g.s.s.r.}
   endif
endwhile
```