2 Exakter Mustervergleich *Exact String Matching*

Wir wissen, daß wichtige Biomoleküle Sequenzen über einem (kleinen) Alphabet sind, und ihre Funktion direkt oder indirekt durch diese Sequenz bestimmt ist (first fact der molekularen Biologie). Daher ist es klar/naheliegend, daß die Untersuchung auf gleiche (Teil)sequenzen oder Ähnlichkeit der Sequenzen für Fragestellungen der molekularen Bioinformatik von großer Wichtigkeit sind.

2.1 Fragestellung

gegeben sei ein *Text* T und ein *Muster* P, finde alle Vorkommen von P in T.

- Interessant ist das Durchsuchen von sehr großen Texten, daher sind effizient Methoden wichtig
- exakte Mustervergleiche in der molekularen Bioinformatik nicht der Regelfall, da oft Fehler/Abweichungen zu berücksichtigen sind

– aber es gibt doch einige Problemstellung
– darüberhinaus auch zum Kennen- und Verstehenlernen anderer Verfahren nützlich
– und als Teilschritte für approximative Verfahren

2.2 Anwendungen

2.3 Definitionen

Definition 1 *Alphabet $\mathcal{A}$ ist nicht leere Menge von Zeichen*

Definition 2 $\mathcal{A}^* = \bigcup_{n \geq 0} \mathcal{A}^n$ *ist die Menge der Zeichenketten (string) oder Wörter über* $\mathcal{A}$

Definition 3 $S(i)$ *bezeichnet das i-te Zeichen von S, Indizes beginnen bei 1*

Definition 4 $S[i..j]$ *ist das Teilwort (substring) von S, das an der Position i beginnt, und bei j endet (jeweils inklusiv)*

$S[i..j] = \lambda$, *falls* $i > j$

Definition 5 $S[1..i]$ *ist Präfix (prefix) von S*
$S[j..n] == S[j..]$ *ist Suffix von S, wobei n die Länge* $|S|$ *von S ist*

Definition 6 S^r *ist der zu* S *reverse String.*

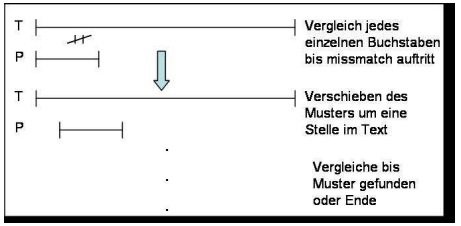
Definition 7 λ *ist der leere String (*$|\ \lambda\ | = 0$*)*

2.4 Der naive Algorithmus

m:= $|T|$ = Länge des Textes

n:= $|P|$ = Länge des Musters (Pattern)

$n \leq m$



Pseudocode:

```
for offset = 0 to m-n do
   found:=true;
   for j = 1 to n do
      if P(j) ≠ T(j+offset) then
        found:=false; break;
      fi
   endfor
   if found = true then 'gefunden';break; endif
endfor
```

worst-case Laufzeit: O($|T||P|$)

Es gibt (viel) bessere Verfahren, trotzdem:

- Verfahren ist sehr einfach (zu programmieren), Testen von komplizierte Algorithmen
- für große Alphabete und zufällige strings ist es im average-case nicht so schlecht (vgl)

Muster vorverarbeiten

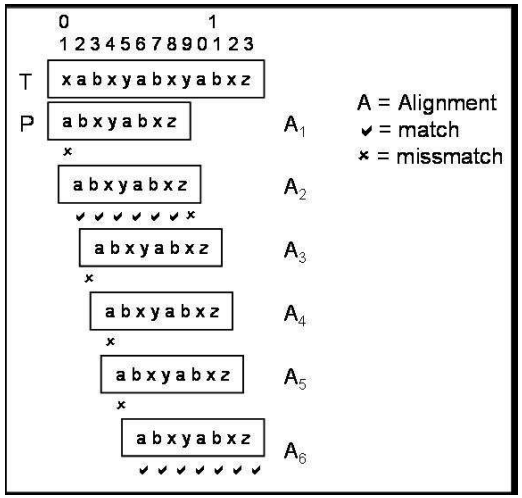
- möglichst mehr als eine Position shiften
- ev. nach shift Vergleiche im neuen “alignment” sparen
- Z-Algorithmus
- Boyer-Moore,

Text vorverarbeiten

- suffix-Bäume

2.5 Ein lineares Verfahren

Idee



2.5.1 Fundamental Preprocessing

für einen String S (das bei Anwendung für P eingesetzt wird) definieren wir:

Definition 8 *für einen String S,* $2 \leq i \leq |S|$ *ist*

$$Z_i(S) := max\{p|S[i \ldots i+p-1] = S[1 \ldots p]\}$$

die Länge des längsten Präfixes von Suffix $S[i \ldots]$*, das auch Präfix von S ist (beachte:* $S[i \ldots i-1] = \lambda$*)*

Bemerkung 1 *falls klar notieren wir* Z_i *statt* $Z_i(S)$

Definition 9 *für ein* $Z_i(S) > 0$ *heißt das Intervall* $[i, i+Z_i(S)-1]$ *die Z-Box an (der Position) i*

Definition 10 *für* $2 \leq i|S|$ *definieren wir* $[l_i, r_i]$ *als die Grenzen derjenigen Z-Box, die vor (inklusive) i beginnt und von diesen die maximale rechte Grenze hat:*

$V_i = \{[a_j, b_j] \mid [a_j, b_j]$*ist Z-Box an* $a_j \wedge a_j \leq i\}$

(die Menge der Grenzen aller Z-Boxen, die vor oder bei i beginnen)

$$[l_i, r_j] = \begin{cases} \underset{[a_j,b_j] \in V_i}{\operatorname{argmax}} \; b_j, & \text{falls } V_j \neq \emptyset \\ \\ [0,0] & \text{sonst} \end{cases}$$

Bemerkung 2 *für* $i < \tilde{i}$ *gilt stets* $V_i \subsetneqq V_{\tilde{i}}$ *(wegen der Definition),*

und daher $r_i \leq r_{\tilde{i}}$