Approximative Zeichenkettensuche

Prof. Dr. R. Parchmann

SS 2002

Gewidmet allen freizeitlosen Informatikern.

Korrekturexemplar !
Fehler bitte an sirtobi@unix-ag.uni-hannover.de mitteilen !

Download-URL: http://www.sirtobi.com/uni_script.shtml

Stand: 25. Juli 2004

Auch Pinguinen sitzt der Fehlerteufel im Nacken...

Dieses Dokument ist meine Mitschrift der Vorlesung "Approximative Zeichenkettensuche" von Prof. Dr. R. Parchmann im Sommersemester 2002.
Für eventuelle Fehler aller Art übernehme ich keine Verantwortung !
Tobias Müller

Inhaltsverzeichnis

(3.4.2002)

Diese Vorlesung wendet sich hauptsächlich an Studierende der Studienrichtung Informatik und des Nebenfachs Informatik nach dem Vordiplom und ist eine Ergänzung und Weiterführung der Vorlesung "Zeichenketten" aus dem Sommersemester 2001.

In dieser Vorlesung sollen Algorithmen für die angenäherte Suche von durch Muster beschriebene Zeichenketten in einer langen Zeichenkette vorgestellt und analysiert werden. Nach einer Wiederholung der wichtigsten Begriffe aus der Vorlesung "Zeichenketten" im Sommersemster 2001, sollen Verfahren zur angenäherten Suche behandelt werden, wobei etwa eine maximale Fehlerzahl gegeben ist oder aber es soll über eine Distanzmetrik eine möglichst gute übereinstimmung gesucht wird. Verfahren dieser Art sind besonders wichtig in der Textverarbeitung und in der Analyse von Gen-Sequenzen ("Computational Biology").

Problem:
Eine Zeichenkette p (Pattern) soll in einer (längeren) Zeichenkette t (Text) gesucht werden, wobei jedoch gewisse Fehler erlaubt sind.
Das Problem tritt auf bei

- Wiedererkennen von Signalen, die über gestörte Kanäle gesucht werden.
- Erkennen von Texten, die Schreibfehler enthalten.
- Erkennen von DNA-Sequenzen, die durch mögliche Mutationen verändert wurden.

Elementare Operationen sind mit Kosten versehen. Der Abstand zwischen zwei Zeichenketten x und y sind dann die minimalen Kosten, um x in y zu überführen.
Am einfachsten:
Einsetzungen, Löschungen und Substitutionen von Buchstaben (auch Transpositionen).
Zählt man nur die elementaren Operationen, so erhält man den Levenshtein Abstand.

Beispiel:
$x = INDUSTRY$
$y = INTEREST$
Levenshtein Abstand=6, etwa: $D \to T, U \to E, \varepsilon \to R, \varepsilon \to E, R \to \varepsilon, Y \to \varepsilon$:
$INDUSTRY$
$INTUSTRY$
$INTESTRY$
$INTERSTRY$
$INTERESTRY$
$INTERESTY$
$INTEREST$

Zählt man nur die Einsetz- und Löschoperationen, erhählt man die Editdistanz.

$x = INDUSTRY$
$y = INTEREST$
Edit-Abstand=8 ($D \to T$ ersetzen durch $D \to \varepsilon, \varepsilon \to T$, u.s.w.)

Zählt man nur Substitutionen, erhält man den Hemming-Abstand.
(Logisch: Geht nur mit gleich langen Ketten !)

$x = INDUSTRY$
$y = INTEREST$
Hemming-Abstand=6

Fehlerschranke k: $\frac{1}{k} \leq \frac{l}{|p|} \leq \frac{1}{2}$

---

1 Grundlegende Schreibweisen und Begriffe

Für das Folgende sei Σ ein Alphabet. Für $w \in \Sigma^*$ bezeichne $|w|$ die Länge des Wortes w.
Sei $w = a_1 \cdots a_n, n = |w|, a_i \in \Sigma, i \in [1:n]$. Dann bezeichne $w[i]$ den i-ten Buchstaben a_i von w. $w[i..j]$ bezeichne das Teilwort $a_i \cdots a_j$ von w. Ist $j < i$, dann bezeichnet $w[i..j]$ das leere Wort ε. Ein Teilwort $w[1..j]$ heißt Präfix von w, $w[j..n]$ heißt Suffix von w.
Sei $- \notin \Sigma$ ein neues Symbol (neutrales Zeichen) und $\Sigma \cup \{-\} = \tilde{\Sigma}$.
μ sei der durch $\mu(a) = \begin{cases} \varepsilon & \text{falls } a = - \\ a & \text{sonst} \end{cases}$ gegebene
Homomorphismus von $\tilde{\Sigma}^* \to \Sigma^*$ (Kompressionsfunktion)

Definition 1.1
Seien x und $y \in \Sigma^*$. Dann heißt das Paar $(x', y') \in \tilde{\Sigma}^* \times \tilde{\Sigma}^*$ Ausrichtung (alignement) von x und y, falls gilt:

1. $\mu(x') = x$ und $\mu(y') = y$
2. $|x'| = |y'|$
3. Es ist $x'[i] \neq -$ oder $y'[i] \neq -$ für $i \in [1:|x'|]$

$|x'|$ heißt Länge der Ausrichtung (x', y').
$A(x, y)$ sei die Menge der Ausrichtungen von x und y.

Beispiel:

Mit $x' = abaa - cbc$ und $y' = -ba - bc - a$ ist (x', y') eine Ausrichtung für $x = abaacbc$ und $y = babca$.
Man schreibt eine derartige Ausrichtung häufig in der Form:

$$\begin{bmatrix} a & b & a & a & - & c & b & c \\ - & b & a & - & b & c & - & a \end{bmatrix}$$

Definition 1.2
Seien $x, y \in \Sigma^*$. Dann heißt $\tau(x, y) \subseteq [1 : |x|] \times [1 : |y|]$ Korrespondenz (trace) zwischen x und y, falls für alle $(i_1, j_1), (i_2, j_2) \in \tau(x, y)$ mit $(i_1, j_1) \neq (i_2, j_2)$ gilt: $i_1 \neq i_2, j_1 \neq j_2$ und $i_1 < i_2$ gdw. $j_1 < j_2$.
$T(x, y)$ sei die Menge aller Korrespondenzen von x und y.

Beispiel:
Sei $x = abaacbc$ und $y = babca$. Dann ist $\tau(x, y) = \{(2, 1), (3, 2), (5, 4), (7, 5)\}$ eine Korrespondenz zwischen x und y.
Man notiert eine derartige Korrespondenz auch in der Form

$$\begin{array}{ccccccccc} a & & b & & a & a & c & b & c \\ & & | & & | & & | & \diagup & \\ & & b & & a & b & c & a & \end{array}$$

(Kreuzungsfrei, nur maximal 1 Linie pro Buchstabe)

Bemerkung:
Jeder Korrespondenz zweier Zeichnketten x und y entspricht eine Ausrichtung von x und y. Dabei stehen die sich in Korrespondenz befindlichen Zeichen gegenüber. Die anderen Zeichen stehen dem neutralen Zeichen gegenüber.
(nicht eindeutig: $\begin{bmatrix} a & b & a & - & \mathbf{a} & c & b & c \\ - & b & a & \mathbf{b} & - & c & - & a \end{bmatrix}$)

Definition 1.3
Ein Paar $(r, s) \in \Sigma^* \times \Sigma^*$ heißt Edit-Operation (geschrieben als $r \to s$):

1. Gilt $|r| = |s| = 1$, dann heißt (r, s) Substitution
2. Gilt $|r| = 1, |s| = 0$, dann heißt (r, s) Löschung (deletion)
3. Gilt $|r| = 0, |s| = 1$, dann heißt (r, s) Einfügung (insertion)

Seien $x, y \in \Sigma^*$. x geht durch Anwendung der Edit-Operationen (r, s)[an der Stelle i] in y über (geschrieben $x \overset{(r,s)}{\underset{i}{\Rightarrow}} y$), falls $x = prq$, $|p| = i - 1$ und $y = psq$.
Entsprechend wird definiert, wie eine Folge von Edit-Operationen eine Zeichenkette x in eine Zeichenkette y überführt.
Sind außer der Operationen 1), 2) und 3) noch weitere Edit-Operationen erlaubt, so spricht man von erweiterten Edit-Operationen, etwa:

4) Gilt $r = ab, s = ba$ für $a, b \in \Sigma$, dann heißt (r, s) Transposition

Bemerkung:
Im Allgemeinen fordert man, daß nach einer Ersetzung von r durch s die Teilzeichenkette s nicht weiter verändert werden darf. Verzichtet man auf diese Einschränkung, so hat man ein allgemeines Term-Ersetzungs-System vor sich und der Abstand zweier Zeichenketten wäre allgemein nicht berechenbar!

(10.4.2002)

Beispiel:
x=INDUSTRY
y=INTEREST

Um aus x y herzustellen, kann man folgende Edit-Sequenz verwenden:
$D \to T, U \to E, \varepsilon \to R, \varepsilon \to E, R \to \varepsilon, Y \to \varepsilon$

Definition 1.4
Sei $x \in \Sigma^*$. $z \in \Sigma^*$ heißt Subsequenz von x, falls es eine Folge $0 < i_1 < i_2 < \cdots < i_n \leq |x|$ und $|z| = m$ gibt mit $z[j] = x[i_j]$, $j \in [1:m]$.
$z \in \Sigma^*$ heißt gemeinsame Subsequenz (common subsequence) von x und y, falls z Subsequenz von x und von y ist.
$lcs(x, y)$ bezeichnet die Länge einer längsten gemeinsamen Subsequenz von x und y.
(lcs $\hat{=}$ longest common sequence)
z heißt Supersequenz von x, falls x Subsequenz von z ist.
z heißt gemeinsame Supersequenz von x und y (common supersequence), falls z Supersequenz von x und y ist. $scs(x, y)$ bezeichnet die Länge einer kürzesten gemeinsamen Supersequenz von x und y (shortest common Supersequenz).

Beispiel:
$lcs(a\underline{b}\overline{c}\underline{a}\overline{b}\overline{b}\underline{a}, \overline{c}\overline{b}\underline{a}\overline{b}\underline{a}\overline{c}) = 4$
denn es ist etwa *baba* oder *cbba* eine gemeinsame Subsequenz und es gibt keine längere.
$scs(abcabba, cbabac) = 9$
denn z.B. ist $\underline{ab}\overline{c}\underline{a}\overline{b}\underline{\overline{a}}\overline{b}\underline{a}\overline{c}$ eine gemeinsame Supersequenz und es gibt keine kürzere.

Es soll eine abstrakte Distanz $D(x, y)$ definiert werden, die den Abstand zwischen x und y als Element eines Halbrings angibt. Diese Distanz beruht auf einer vorgegebenen elementaren Distanzfunktion $w(a, b)$ für Zeichen aus $\tilde{\Sigma}$.

Definition 1.5
Sei T eine Menge, $\overline{0}, \overline{1} \in T$ und $+, \cdot : T \times T \to T$ binäre Operationen. $H = (T, +, \cdot, \overline{0}, \overline{1})$ heißt Halbring, falls

1. $(T, +, \overline{0})$ ist kommutative Halbgruppe mit Nullelement $\overline{0}$
   ($+$ ist kommutativ und assoziativ und $\overline{0} + a = a + \overline{0} = a \ \forall a \in T$)

2. $(T, \cdot, \overline{1})$ ist Halbgruppe mit Einselement $\overline{1}$
   ($\cdot$ ist assoziativ und $\overline{1} \cdot a = a \cdot \overline{1} = a \ \forall a \in T$)

3. Für alle $a, b, c \in T$ gilt
   $$a \cdot (b + c) = (a \cdot b) + (a \cdot c) \text{ und}$$
   $$(a + b) \cdot c = (a \cdot c) + (b \cdot c)$$

Eine Funktion $w : \tilde{\Sigma} \times \tilde{\Sigma} \to T$ heißt (abstrakte) elementare Distanzfunktion (Kostenfunktion).

Definition 1.6
Sei Σ ein Alphabet, $\tilde{\Sigma}$ das erweiterte Alphabet, $H = (T, +, \cdot, \overline{0}, \overline{1})$ ein Halbring, w eine elementare Distanzfunktion, $x, y \in \Sigma^*$ und $A(x, y)$ die Menge aller Ausrichtungen von x und y.
Dann ist die abstrakte Distanz $D : \Sigma^* \times \Sigma^* \to T$ (bezüglich Σ, H und w) definiert durch

$$D(x, y) = \begin{cases} \overline{1} & \text{falls } x = y = \varepsilon \\ \Sigma_{(x', y') \in A(x,y)} \prod_{i=1}^{|x'|} w(x'[i], y'[i]) & \text{sonst} \end{cases}$$

Satz 1.7
Sei D eine abstrakte Distanz bezüglich Σ, T, w.
Seien $x, y \in \Sigma^+, |x| = n, |y| = m$ und es sei $\overline{x} = x[1..n-1]$ und $\overline{y} = y[1..m-1]$.
Dann gilt:

1. $D(x, \varepsilon) = D(\overline{x}, \varepsilon) * w(x[n], -)$
2. $D(\varepsilon, y) = D(\varepsilon, \overline{y}) * w(-, y[m])$
3. $$\begin{aligned} D(x, y) = \quad & D(\overline{x}, \overline{y}) * w(x[n], y[m]) \\ & + D(x, \overline{y}) * w(-, y[m]) \\ & + D(\overline{x}, y) * w(x[n], -) \end{aligned}$$

Beweis:

1. Es ist $A(x, \varepsilon) = \{(x, -^n)\}$.
   Also gilt $D(x, \varepsilon) = \prod_{i=1}^{|x|} w(x[i], -) = \prod_{i=1}^{|x|-1} w(x[i], -) * w(x[n], -)$
   $= D(\overline{x}, \varepsilon) * w(x[n], -)$

2. wie 1)

3. Sei $(x', y') \in A(x, y)$ und sei $l = |x'|$. $A(x, y)$ läßt sich disjunkt zerlegen in
   $$A_1 = \{(x', y') \in A(x, y), x'[l], y'[l] \in \Sigma\}$$
   $$A_2 = \{(x', y') \in A(x, y), x'[l] = -\}$$
   $$A_3 = \{(x', y') \in A(x, y), y'[l] = -\}$$
   Dann ist $D(x, y) = \underbrace{\sum_{(x', y') \in A_1} \prod_{i=1}^{|x'|} w(x'[i], y'[i])}_{S_1}$

$$+ \underbrace{\sum_{(x',y')\in A_2} \prod_{i=1}^{|x'|} w(x'[i], y'[i])}_{S_2}$$

$$+ \underbrace{\sum_{(x',y')\in A_3} \prod_{i=1}^{|x'|} w(x'[i], y'[i])}_{S_3}$$

Es ist $S_1 = \sum_{A_1} \prod_{i=1}^{|x'|-1} w(x'[i], y'[i]) * \underbrace{w(x'[l], y'[l])}_{=w(x[n],y[m])}, l = |x'|$

$$= (\underbrace{\sum_{A_1} \prod_{i=1}^{l-1} w(x'[i], y'[i]))}_{|\overline{x}'|} * w(x[n], y[m]) \qquad \text{(Distributivität)}$$

$= (\sum_{(\overline{x}',\overline{y}')\in A(\overline{x},\overline{y})} \prod_{i=1}^{|\overline{x}'|} w(\overline{x}'[i], \overline{y}'[i])) * w(x[n], y[m])$
$= D(\overline{x}, \overline{y}) * w(x[n], y[m])$

Enstprechend zeigt man $S_2 = D(x, \overline{y}) * w(-, y[m])$ und $S_3 = D(\overline{x}, y) * w(x[n], -)$.

Beispiele:

Setzt man $w_1(a, b) = \begin{cases} 0 & \text{falls } a \neq b \\ 1 & \text{sonst} \end{cases}$ als elementare Distanzfunktion und setzt man $H_1 = (\mathbb{N}, max, +, 0, 0)$ (H_1 ist offensichtlich Halbring), so erhält man eine abstrakte Distanz D_1 bezüglich Σ, H, w_1. Es gilt

Satz 1.8
Es ist $D_1(x, y) = lcs(x, y)$ für $x, y \in \Sigma^*$.

Beweis:
Zu jeder Ausrichtung $(x', y') \in A(x, y)$ erhält man eine gemeinsame Subsequenz, in dem man alle $x'[i]$ aneinander fügt, für die $x'[i] = y'[i]$ gilt.
Also ist $lcs(x, y) = max_{(x',y')\in A(x,y)} \sum_{i=1}^{|x'|} w_1(x'[i], y'[i]) = D_1(x, y)$.

Beispiel:
$lcs(x, y)$ für $x = babc, y = abaacbc$

		b	a	b	c	a
	0	0	0	0	0	0
a	0	0	1	1	1	1
b	0	1	1	2	2	2
a	0	1	2	2	2	3
a	0	1	2	2	2	3
c	0	1	2	2	3	3
b	0	1	2	3	3	3
c	0	1	2	3	4	$\boxed{4}$

Nach Satz 1.7 (Seite 8) gilt:
$D(\varepsilon, \varepsilon) = 0$
$D(x, \varepsilon) = D(\overline{x}, \varepsilon) * w(x[n], -)$
$D(\varepsilon, y) = D(\varepsilon, \overline{y}) * w(-, y[m])$
$D(x, y) = \quad D(\overline{x}, \overline{y}) * w(x[n], y[m])$
$\qquad + D(x, \overline{y}) * w(-, y[m])$
$\qquad + D(\overline{x}, y) * w(x[n], -)$

Also $lcs(x, y) = 4$.

Verfahren:
Die Worte werden als Zeilen und Spalten verwendet, die erste Zeile und die erste Spalte wird leer gelassen.
Danach füllt man die Tabelle von oben links nach unten rechts nach folgendem Verfahren mit obigen Formeln:

		f
	b	c
e	a	x

$$x = max\{a, c, \left(b + \begin{cases} 0 & \text{falls } e \neq f \\ 1 & \text{falls } e = f \end{cases}\right)\} \qquad \textbf{(17.4.2002)}$$

Setzt man $w_2 : \tilde{\Sigma} \times \tilde{\Sigma} \to N \cup \{\infty\}$ mit $w_2(a,b) = \begin{cases} 1 & \text{falls } a = - \text{ oder } b = - \text{ oder } a = b \\ \infty & \text{sonst} \end{cases}$
und $H_2 = (N \cup \{\infty\}, min, +, \infty, 0)$, so erhält man eine abstrakte Distanz bezüglich Σ, H_2, w_2. Es gilt:

Satz 1.9
Für $x, y \in \Sigma^*$ ist $D_2(x,y) = scs(x,y)$.

Beweis:
Zu jeder Ausrichtung (x', y') mit $x'[i] = y'[i]$ für alle i mit $x'[i] \neq -$ und $y'[i] \neq -$ erhält man eine Supersequenz, in dem man alle $x'[i]$ und $y'[i]$, die ungleich " $-$ " sind, aneinander hängt. Es ist

$$scs(x,y) = min_{(x',y') \in A(x,y)} \sum_{i=1}^{|x'|} w_2(x'[i], y'[i])$$

Beispiel:
$x = abaacbc$, $y = babca$

$$\begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{a} & \mathbf{a} & \mathbf{c} & \mathbf{b} & \mathbf{c} & - \\ - & b & a & - & - & b & c & \mathbf{a} \end{bmatrix}$$

Die obige Σ wäre hier also gleich $1+1+1+1+1+1+1+1 = 8$.
Das **fett** gedruckte ist eine Supersequenz.

$$\begin{bmatrix} a & b & a & a & c & b & c \\ b & a & - & b & c & a & - \end{bmatrix}$$

Die obige Σ wäre hier also gleich $1+1+1+1+\infty+1+1 = \infty$.

Beispiel:
$x = abaacba$
$y = babca$

		$\rightarrow y$					
		b	a	b	c	a
	0	1	2	3	4	5
a	1	2	2	3	4	5
b	2	2	3	3	4	5
$\downarrow x$ a	3	3	3	4	5	5
a	4	4	4	5	6	6
c	5	5	5	6	6	7
b	6	6	6	$\boxed{6}$	7	8
a	7	7	7	7	8	$\underline{\underline{8}}$

$\overline{x} = x[1..n-1], |x| = n$
$\overline{y} = y[1..m-1], |y| = m$
Nach Satz 1.7 (Seite 8) gilt:
$D_2(x, \varepsilon) = D_2(\overline{x}, \varepsilon) + w_2(x[n], -)$
$D_2(\varepsilon, y) = D_2(\varepsilon, \overline{y}) + w_2(-, y[m])$

$$D_2(x,y) = min \left\{ \begin{array}{l} D_2(\overline{x},\overline{y}) + w_2(x[n], y[m]), \\ D_2(x,\overline{y}) + w_2(-, y[m]), \\ D_2(\overline{x}, y) + w_2(x[n], -) \end{array} \right\}$$

Aus dieser Tabelle kann man auch noch die Werte für alle Teilketten von a und b entnehmen, z.B.:
$\boxed{\ } = D_2(abaacb, bab) = 6$

$$\left[\begin{array}{cccccc} a & b & a & a & c & b \\ - & b & a & - & - & b \end{array} \right]$$

Verfahren:

	$\rightarrow y$	
		$y[j]$
$\downarrow x$		$\vdots$
$x[i]$	$\cdots$	$\bullet D_($x[1..i], y[1..j]$)$

a	b
c	x

$$x = min(c+1, b+1, a + \left\{ \begin{array}{ll} 1 & \text{falls } \cdots \\ \infty & \text{sonst} \end{array} \right.)$$

Es gilt:

Satz 1.10
Für $x, y \in \Sigma^*$ ist $lcs(x,y) = |x| + |y| - scs(x,y)$.

Beweis:
Es gibt eine Ausrichtung $(x', y') \in A(x,y)$ mit folgender Eigenschaft:
Es gibt Indizes $0 < i_1 < i_2 < \cdots < i_k$ mit $k = lcs(x,y)$ und $x'[i_j] = y'[i_j]$ für $j = [1:k]$ und $x'[l] = -$ oder $y'[l] = -$ für $l \in [1:m], l \notin \{i_1, \ldots, \imath_k\}, m = |x'|$.

Idee:
Sei $x = abcabba$ und $y = cbabac$, gemeinsame Subsequenz wäre $baba$.

$$\left[\begin{array}{cccccccccc} a & - & \mathbf{b} & c & \mathbf{a} & \mathbf{b} & b & \mathbf{a} & - \\ - & c & \mathbf{b} & - & \mathbf{a} & \mathbf{b} & - & \mathbf{a} & c \end{array} \right]$$ mögliche Ausrichtung.

$a\ c\ b\ c\ a\ b\ b\ a\ c$ Supersequenz.

Dann ist z.B.

$\mu(x'[1..i_1-1]y'[1..i_1-1])x'[i_1]\mu(x'[i_1+1..i_2-1]y'[i_1+1..i_2-1])x'[i_2]y(\cdots)\cdots$
$\cdots x'[i_k]\mu(x'[i_k+1..m], y'[i_k+1..m])$
eine gemeinsame Supersequenz für x und y mit minimaler Länge.
Es ist $(i_1-1)+1+(i_2-i_1+1)+1+\cdots+1+(m-i_k)=m'=|x|+|y|-lcs(x,y)$.

Definition 1.11
Sei $R'=R\cup\{\infty\}$. Dann ist $(R', min, +, \infty, 0)=H$ ein Halbring.
Sei $w:\tilde{\Sigma}\times\tilde{\Sigma}\to R'$ eine elementare Distanzfunktion.
(Anmerkung von SirTobi: $w=\begin{cases}0 & ,a=b\\ 1 & ,a\neq b\end{cases}$ klappt in den Beispielen.)
Dann heißt die abstrakte Distanz d_L^a bezüglich Σ, H und w allgemeine Levenshtein Distanz (GLD).
Ist w eine Metrik auf $\tilde{\Sigma}$, dann heißt d_L^a allgemeine Levenshteinmetrik (GLD).
Ist w definiert durch $w(a,b)=\begin{cases}\infty & \text{für } a\neq b \text{ und } (a=- \text{ oder } b=-)\\ 1 & \text{für } a\neq b \text{ und } a\neq - \text{ und } b\neq -\\ 0 & \text{für } a=b\end{cases}$,
so heißt die abstrakte Distanz d_H Hemming-Distanz.
Ist w definiert durch $w(a,b)=\begin{cases}0 & \text{für } a=b\\ 1 & \text{für } a=- \text{ oder } b=-\\ 2 & \text{für } a\neq b, a\neq - \text{ und } b\neq -\end{cases}$,
dann heißt die abstrakte Distanz d_E die Edit-Distanz.

(Anmerkung von SirTobi: Betrachte w als Kostenfunktion, dann wird alles klar.)

Bemerkung:

1. Es ist $d_L(x,y)$ die minimale Zahl von Einfügungen, Löschungen und Substitutionen von Zeichen um x in y zu überführen. (minimale Zahl von Edit-Operationen)

2. Ist $|x|\neq|y|$, so ist $d_H(x,y)=\infty$. Ist $|x|=|y|$, so ist $d_H(x,y)$ die Zahl der Positionen von x und y, an denen in x und y unterschiedliche Zeichen stehen.

3. Es ist $d_E(x,y)$ die minimale Zahl von Einfügungen oder Löschungen von Zeichen um x in y zu überführen. (Es kann bei Edit-Operationen auf Substitutionen verzichtet werden)

4. Man kann zeigen, daß d_L^a eine Metrik ist, falls w eine Metrik ist. Daraus folgt, daß d_L eine Metrik ist.

Die GLD läßt sich auch über Korrespondenzen definieren.

Satz 1.12
Seien $x,y\in\Sigma^+$ und sei $T(x,y)$ die Menge aller Korrespondenzen zwischen x und y. Sei $\tau\in T(x,y)$ mit $\tau=\{(i_1,j_1),\ldots,(i_k,j_k)\}$ und sei $I_\tau=[1:|x|]\backslash\{i_1,\ldots,i_k\}$ und $J_\tau=[1:|y|]\backslash\{j_1,\ldots,j_k\}$.

Sei $w : \tilde{\Sigma} \times \tilde{\Sigma} \to R'$ elementare Distanzfunktion und sei
$D(\tau) = \Sigma_{(i,j)\in\tau} w(x[i], y[j]) + \Sigma_{i\in I_\tau} w(x[i], -) + \Sigma_{j\in J_\tau} w(-, y[j])$ und
$D_T(x, y) = min\{D(\tau)|\tau \in T(x, y)\}$.
Dann ist $D_T(x, y) = d_L^a(x, y)$ bezüglich w.

Beweis:
Wegen $d_L^a(x, y) = min_{(x',y')\in A(x,y)} \Sigma_{i=1}^{|x'|} w(x'[i], y'[i])$ genügt es zu zeigen:

1. Für jedes $\tau \in T(x, y)$ gibt es $(x', y') \in A(x, y)$ mit $D(\tau) = \Sigma_{i=1}^{|x'|} w(x'[i], y'[i])$.

2. Für jedes $(x', y') \in A(x, y)$ gibt es ein $\tau \in T(x, y)$ mit $D(\tau) = \Sigma_{i=1}^{|x'|} w(x'[i], y'[i])$.

zu 1):
Sei $\tau = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ und sei oBdA $i_1 < i_2 < \cdots < i_k$, dann ist $j_1 < j_2 < \cdots < j_k$ und damit erfüllt die Ausrichtung

$$\begin{bmatrix} x[1] & \cdots & x[i_1-1] & - & \cdots & - & x[i_1] & \cdots & x[i_k] & x[i_k+1] & \cdots & x[n] & - & \cdots & - \\ - & \cdots & - & y[1] & \cdots & y[j_1-1] & y[j_1] & \cdots & y[j_k] & - & \cdots & - & y[j_k+1] & \cdots & y[m] \end{bmatrix}$$

die Aussage.

zu 2):
Sei $(x', y') \in A(x, y)$ gegeben.
Es gelte $x'[i_j] = x[j]$ mit $i_1 < \cdots < i_n$ und $y'[j_k] = y[k]$ mit $j_1 < \cdots < j_m$.
Definiere τ wie folgt:
$(r, s) \in \tau$ gdw. $i_r = j_s$.
τ erfüllt die Aussage.

(24.4.2002)
Will man den Abstandsbegriff über Edit-Operationen definieren, dann muß die elementare Distanzfunktion $w : \tilde{\Sigma} \times \tilde{\Sigma} \to R$ eine Metrik sein !
(speziell: $w(a, b) \leq w(a, c) + w(c, b)$)

Satz 1.13
Seien $x, y \in \Sigma^*, S(x, y)$ die Menge der Edit-Sequenzen (ohne Transpositionen !) die x in y überführen. Sei $\sigma \in S(x, y), \sigma = (r_1 \to s_1, r_2 \to s_2, \ldots, r_k \to s_k)$ und w sei Metrik.
Sei $D(\sigma) = \sum_{i=1}^{k} w(r_i, s_i)$ und setze $D_s(x, y) = min\{D(\sigma)|\sigma \in S(x, y)\}$.
Dann gilt bezüglich w: $D_s(x, y) = d_L^a(x, y)$.

Beweis:
(Notation wie im Beweis zu Satz 1.12)
Genügt zu zeigen:

1. Für jedes $\tau \in T(x, y)$ gibt es $\sigma \in S(x, y)$ mit $D(\sigma) = D(\tau)$

2. Für jedes $\sigma \in S(x, y)$ gibt es $\tau \in T(x, y)$ mit $D(\tau) \leq D(\sigma)$

zu 1)
Sei $\tau = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ und $I_\tau := [1 : |x|]\backslash\{i_1, \ldots, i_k\}$ und $I_\tau := [1 : |y|]\backslash\{j_1, \ldots, j_k\}$.
Setze $\sigma = (x_i \to y_i, i \in [1 : k]; x_j \to \varepsilon, j \in I_\tau; \varepsilon \to y_l, l \in I_\tau)$.
σ ist offensichtlich Edit-Sequenz, die x nach y überführt und es gilt $D(\sigma) = D(\tau)$.

zu 2)
Vorbetrachtung: Sei $z \in \Sigma^+$ und sei $\tau_1 \in T(x,z)$ und $\tau_2 \in T(z,y)$.
Ferner sei $\tau_1 \circ \tau_2 := \{(i,j)|(i,k) \in \tau_1 \text{ und } (k,j) \in \tau_2 \text{ für } k \in N\}$.
$\tau_1 \circ \tau_2$ ist Korrespondenz zwischen x und y und durch (formal) aufwendige Rechnung zeigt man $D(\tau_1 \circ \tau_2) \leq D(\tau_1) + D(\tau_2)$ (hier geht die Metrik-Eigenschaft von w ein !)

Sei $\sigma = (r_1 \to s_1, \ldots, r_k \to s_k)$ Edit-Sequenz, die x nach y überführt.
Es gibt also $z_0, \ldots, z_k \in \Sigma^*$ mit $z_0 = x, z_k = y$ und $z_i \overset{\substack{r_i \to s_i \\ j_i}}{\Longrightarrow} z_i$ für $j \in [1:k]$.
Durch vollständige Induktion über k wird nun 2) gezeigt.
Ist $k = 0$, dann ist σ leer und es ist $x = z_0 = y$.
Es folgt $D(\sigma) = D(\tau)$ für $\tau = \{(i,i)|i \in [1:|x|]\}$. Ist $k > 0$, so betrachte Edit-Sequenz $\tilde{\sigma} = (r_1 \to s_1, \ldots, r_{k-1} \to s_{k-1})$.
$\tilde{\sigma}$ überführt x in z_{k-1} und laut Induktions-Voraussetzung existiert ein $\tilde{\tau} \in T(x, z_{k-1})$ mit $D(\tilde{\tau}) \leq D(\tilde{\sigma})$.
Setzt man $\tau_1 = \{(i,i)|i \in [1:j_{k-1}]\} \cup \{(i, i+\delta)|i \in [j_k + |r_k| : |z_{k-1}|]\} \cup L$ mit $\delta = |s_k| - |r_k|$ und $L = \begin{cases} \{(j_k, j_k)\} & \text{falls } r_k \to s_k \text{ Substitution} \\ \emptyset & \text{sonst} \end{cases}$
Es ist τ_1 Korrespondenz zwischen z_{k-1} und $z_k = y$ mit $D(\tau_1) = w(r_k, s_k)$. Es folgt $D(\tau) = D(\tilde{\tau}_0 \circ \tau_1) \leq D(\tilde{\tau}) + D(\tau_1) \leq D(\tilde{\sigma}) + w(r_k, s_k) = D(\sigma)$.

Will man allgemeine Edit-Operationen $(r,s) \in \Sigma^* \times \Sigma^*$ zulassen, so kann man unter gewissen Voraussetzungen Rekursionsformeln zur Berechnung eines entsprechenden Distanzbegriffs herleiten.
Sei $\sigma \subset \Sigma^* \times \Sigma^*$ eine vorgegebene endliche Menge von Edit-Operationen. Für jedes $(r,s) \in \sigma$ seien Kosten $w(r,s) \in \mathbb{R}_{\geq 0}$ festgelegt.
Weiterhin gelte $w(r,s) = 0$ gdw $r = s$.
(Bemerkung: Manchmal ist es sinnvoll, für $(r,s) \in \Sigma^* \times \Sigma^* \backslash \sigma$ die Kosten $w(r,s) = \infty$ zu setzen)
Seien $x, y \in \Sigma^*$. Dann sei $\tilde{S}(x,y)$ die Menge aller Folgen $t = (r_1 \to s_1, \cdots, r_k \to s_k)$ von Edit-Operationen aus σ, die x in y überführen (☞ Definition 1.3). Die leere Folge t_0 überführt x in x.
Man definiert $\rho(x,y) = max\{\delta_1(x,y), \delta_2(x,y)\}$ mit $\delta_1(x,y) = min_{\tau \in \tilde{S}(x,y)} \sum_{i=1}^{|t|} w(r_i, s_i)$ und $\delta_2(x,y) = \delta_1(y,x)$
Es sei $\delta_1(x,y) = \infty$ falls $\tilde{S}(x,y) = \emptyset$.
Durch $x \sim y$ gdw $\delta(x,y) < \infty$ wird eine Äquivalenzrelation auf Σ^* definiert.

Satz 1.14
Sei A eine Äquivalenzklasse bezüglich $\sim$. Dann ist (A, δ) ein metrischer Raum.

δ ist symmetrisch und nicht negativ. Es ist $\delta(x,x) = 0$ da die leere Folge $t_0 \in \tilde{S}(x,x)$ ist. Für $x, y, z \in A$ gilt weiterhin $\delta(x,z) + \delta(z,y) \geq \delta_1(x,z) + \delta_1(z,y) \geq \delta_1(x,y)$.
Entsprechend ist $\delta(x,z) + \delta(z,y) \geq \delta_2(x,y)$.

Also $\delta(x,z)+\delta(z,y) \geq \delta(x,y)$.

Bemerkung:
Der Abstandsbegriff ist zu allgemein definiert (Term-Ersetzungssystem) und nicht effektiv berechenbar.

Man benötigt also eine Einschränkung bei den möglichen Edit-Sequenzen.
Betrachtet werden sollen Edit-Sequenzen mit der Eigenschaft ($\heartsuit$):
Sei $t=(r_1 \to s_1, \ldots, r_k \to s_k) \in \tilde{S}(x,y)$ eine Edit-Sequenz, die x in y überführt.
Dann gibt es eine Zerlegung von $x = u_0 r_1 u_1 r_2 \cdots r_{k-1} u_{k-1} r_k u_k$
und $y = u_0 s_1 u_1 s_2 \cdots s_{k-1} u_{k-1} s_k u_k$ mit $u_i \in \Sigma^*, i \in [0:k]$.
Jede Edit-Operation kann also unabhängig von jeder anderen ausgeführt werden. (parallele Transformation)
Eine zweite äquivalente Definition wird über aktive Teile der Zeichenkette geführt.
Zu Beginn ist ganz x aktiv. Sei uv eine Zeichenkette, die bei der Transformation von x entsteht und sei v aktiver Teil von uv. Sei $r \to s$ eine Edit-Operation und sei $v = v_1 r v_2$. Dann ist die Edit-Operation anwendbar und das Ergebnis ist $uv_1 s v_2$ und v_2 ist aktiver Teil der Zeichenkette. Man betrachtet nur Edit-Sequenzen, bei denen jede Operation im aktiven Teil durchgeführt wird.
Die Menge aller ($\heartsuit$) erfüllenden Edit-Sequenzen, die x in y überführen, sei $S^*(x,y)$. Sei $\delta(x,y) = min_{t \in S^*(x,y)} \sum_{i=1}^{|t|} w(r_i, s_i), t = (r_1 \to s_1, \ldots r_k \to s_k)$.
Offensichtlich gilt: $\delta(x,y) \geq \rho_1(x,y)$.

(25.4.2002)

Satz 1.15
Sei $x, y \in \Sigma^*, |x| = n, |y| = m$ und sei σ eine endliche Menge von Edit-Operationen.
$w : \Sigma^* \times \Sigma^* \to \mathbb{R}_{\geq 0}$ eine elementare Distanzfunktion mit $w(r,s) = 0$ gdw. $r = s$. Dann gilt:
$\delta(x[1..i], y[1..j]) = min\{p_{ij}, qij\}$ mit

$$p_{ij} = \begin{cases} \delta(x[1..i-1], y[1..j-1]) & \text{falls } x[i] = y[i] \\ \infty & \text{sonst} \end{cases}$$

$q_{ij} = min_* \{\delta(x[1..i-|r|], y[1..j-|s|]) + w(r,s)\}$
$\quad * : (r,s) \in \sigma, r$ Suffix von $x[1..i]$, s Suffix von $y[1..j]$

Beweis:
Seien $i \in [1..n], j \in [1..m]$ fest gewählt. Es sei $\delta(x[1..i], y[1..j]) = \sum_{i=1}^{k} w(r_1, s_i)$ für eine Edit-Sequenz $t = (r_1 \to s_1, \ldots, r_k \to s_k)$, die Bedingung ($\heartsuit$) erfüllt.
Also gilt $x[1..i] = u_0 r_1 u_1 \ldots r_k u_k$ und $y[1..j] = u_0 s_1 u_1 \ldots s_k u_k$.

1. $x[i]$ wird durch eine Edit-Operation transformiert.
   Dann ist $u_k = \varepsilon$ und $t' = (r_1 \to s_1, \ldots, r_{k-1} \to s_{k-1})$ ist minimale Edit-Sequenz,

um $x[1..i-|r_k|]$ in $y[1..j-|s_k|]$ zu transformieren. Also gilt $\delta(x[1..i-|r_k|], y[1..j-|s_k|]) + w(r_k, s_k) = q_{ij}$.

2. $y[j]$ wird durch eine Edit-Operation eingesetzt. wie 1).

3. $x[i]$ und $y[j]$ werden beide nicht verändert. Also $x[i] = y[j]$.
   Es folgt $\delta(x[1..i-1], y[1..j-1]) = \delta(x[1..i], y[1..j])$.

Der Algorithmus benötigt $O(l * n * m)$ Zeiteinheiten, dabei sei $|x| = n, |y| = m$ und $l = \sum_{(r,s)\in\sigma} |r| + |s|$.
Eine geschicktere Lösung ist die Verwendung zweier Aho-Corasick-Automaten zur Erkennung der linken bzw. rechten Seiten.
$\rightarrow$ Laufzeit $O(c*n*m)$ mit c =maximale Zahl gleichzeitig anwendbarer Edit-Operationen.

Betrachte Speziallfall $\hat{\sigma} = \{r \rightarrow s | (r,s) \in \Sigma^* \times \Sigma^*, r, s \in \Sigma$ oder $r = \varepsilon$ und $s \neq \varepsilon$ oder $r \neq \varepsilon$ und $s = \varepsilon\}$.
Erlaubt sind Substitutionen einzelner Zeichen, sowie Einsetzen und Löschen ganzer Wörter.
Die elementare Abstandsfunktion sei durch $w(a,b) = w_0 > 0$ für $a \neq b$ und $w(\varepsilon, u) = w_k$ und $w(u, \varepsilon) = w_{-k}$ und $0 < w_1 \leq w_2 \leq \cdots, 0 < w_{-1} \leq w_{-2} \leq \cdots$ für $u \in \Sigma^+, |u| = k$ festgelegt. (Lückenkosten)

Lemma 1.16
Für die oben definierte Menge $\hat{\sigma}$ von Edit-Operationen und die elementare Abstandsfunktion w gilt ($\heartsuit$). Also ist $\rho_1(x,y) = \delta(x,y)$ für $x, y \in \Sigma^*$.

Beweis:
Man ersetze in einer weiteren Folge von Edit-Operationen minimalen Gewichts zwei Operationen die "nacheinander" dieselben Zeichen verändern durch eine Edit-Operation mit kleinerem Gewicht.

Satz 1.17
Sind die Lückenkosten wie oben definiert, so gilt für alle $x, y \in \Sigma^*$ mit $|x| = n, |y| = m$:
$\delta(x[1..i], y[1..j]) = min\{I_{ij}, D_{ij}, r\}$ mit
$r = \delta(x[1..i-1], y[1..j-1]) + w(x[i], y[j])$
$I_{ij} = min\{\delta(x[1..i], y[1..j-k]) + w_k, k \in [1..j]\}$ (insert)
$D_{ij} = min\{\delta(x[1..i-k], y[1..j]) + w_{-k}, k \in [1..i]\}$ (delete)

Das Verfahren benötigt $O(nm(n+m))$ Zeiteinheiten.

Beweis:
$\surd$

(8.5.2002)

Schreibweise: Eine optimale Korrespondenz (Ausrichtung) von x und y bezüglich eines Distanzbegriffs D ist eine Korrespondenz (Ausrichtung) die den Wert $D(x,y)$ hat.

Definition 1.18
Sei d_L^a eine allgemeine Levenshtein-Metrik mit elementarer
Distanzfunktion $w : \tilde{\Sigma} \times \tilde{\Sigma} \rightarrow \mathbb{R}_{\geq 0}$. Seien $x, y \in \Sigma^*$, $|x| = n$, $|y| = m$ und $n \gg m$.
y stellt das Teilwort $x[i..j]$ am Besten dar, falls für alle Teilworte $x[k..l]$ von x $d_L^a(x[i..j], y) \leq d_L^a(x[k..l], y)$ ist.
Häufig wird eine Fehlerschranke k vorgegeben und man sucht nach allen Teilworten $x[i..j]$ von x mit $d_L^a(x[i..j], y) \leq k$.
Betrachtet man dieses Problem für die einfache Levenshtein-Metrik, dann nennt man es das k-Differenz Problem (k-differences problem). Betrachtet man die Hemmingdistanz, so nennt man es das k-Unterschiede Problem (k-mismatches problem).

2 Algorithmen, die auf der dynamischen Programmierung basieren

Wagner-Fischer Algorithmus

Seien $x, y \in \Sigma^*$ Zeichenketten, für die $d_L^a(x, y)$ (oder $d_L(x, y)$ oder $d_E(x, y)$ oder ...) bestimmt werden soll. Es gelte $|x| = m$, $|y| = n$.
Setzt man $d_{ij} := d_L^a(x[1..i], y[1..j])$, so gilt laut Satz 1.7

$$d_{ij} = min\{d_{i-1,j} + w(x[i], \varepsilon), d_{i,j-1} + w(\varepsilon, y[j]), d_{i-1,j-1} + w(x[i], y[j])\}$$

Man tabelliert die Werte d_{ij} in einer Abstandsmatrix, um die bei der rekursiven Lösung mehrfach auftretenden Teilberechnungen nur einmal zu berechnen. (Prinzip der dynamischen Programmierung)
Es gilt: $d_{0,0} = 0$, $d_{i0} = \sum_{k=1}^{i} w(x[k], \varepsilon)$, $d_{0j} = \sum_{k=1}^{i} w(\varepsilon, y[k])$ (Für die einfache Levenshtein- oder die Edit-Metrik gilt $d_{i0} = i$, $d_{0j} = j$). Dann berechnet man die Matrix Zeilen- oder Spaltenweise und es ist $d_{mn} = d_L^a(x, y)$.

Wie kann man nun eine optimale Korrespondenz (Ausrichtung) erhalten ?
Man geht von der unteren Ecke der Matrix "rückwärts" die Tabellenplätze entlang, bei denen das Minimum auftrat. (nicht notwendig eindeutig !)

Beispiel:
(☞ Folie auf Seite 69)

x	$=$	a	b	c		a		b	b	a		a	b	c	a	b	b	a	$-$	
		|	|		$\diagup$		$\diagup$		$\diagup$	$\hat{=}$	c	b	$-$	a	b	$-$	a	c	
y	$=$	c	b	a		b		a	c			$-$		$-$			$-$		$-$	$Kosten = 4$

Satz 2.1
Die Berechnung des Abstands zweier Zeichenketten $x, y \in \Sigma^*$ mit dem Algorithmus von Wagner und Fischer benötigt $O(nm)$ Speicher- und $O(nm)$ Zeiteinheiten.
Die Berechnung einer optimalen Korrespondenz benötigt $O(m + n)$ Zeiteinheiten.

Bemerkungen:

1. Soll nur der Abstand berechnet werden, kann man den Speicherplatz auf $O(min(m,n))$ reduzieren.
   (Zeilenweise berechnen, alte Zeile löschen)

2. Der Algorithmus ist nicht sehr effektiv, aber er ist der flexibelste, wenn es darum geht unterschiedliche Abstandsfunktionen zu berechnen (Lückenkosten, usw.)

Will man bei den zugeordneten Edit-Operationen Transpositionen zulassen, so kann man dies z.B. mit einer kleinen Modifikation des Wagner-Fischer-Algotithmus berechnen.
Wir betrachten die Menge σ von elementaren Edit-Operationen mit
$\sigma = \{r \to s | r,s \in \tilde{\Sigma}, r * s \neq \varepsilon\} \cup \{ab \to ba, \quad a,b \in \Sigma\}$ und benutzen die Kostenfunktion
$w(r,s) = \begin{cases} 0 & \text{falls } r = s \\ 1 & \text{sonst} \end{cases}$
Weiterhin sollen die berücksichtigten Edit-Sequenzen die Eigenschaft (♡) aus Kapitel 1 erfüllen, also aus $S^*(x,y)$ sein. Sei $\delta(x,y)$ der so definierte Distanzbegriff. Man setze $d_{ij} = \delta(x[1..i], y[1..j])$.
Laut Satz 1.15 gilt

$$d_{ij} = min \left\{ \begin{array}{l} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + w(x[i], y[j]) \\ d_{i-2,j-2} + 1 \text{ falls } x[i-1]x[i] = y[j][y[j-1] \end{array} \right\}$$

falls $x[i-1]x[i] = y[j]y[j-1]$ gilt, ist $w(x[i-1], y[j]) = w(x[i], y[j-1]) = 0$.
Da $d_{ij} \leq d_{i-2,j-2} + 2$ für die einfache Levenshtein Metrik (☞ 3.1) kann man obige Formel umschreiben zu

$$d_{ij} = min \left\{ \begin{array}{l} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i-1,j-1} + w(x[i], y[j]) \\ d_{i-2,j-2} + w(x[i-1], y[j]) + w(x[i], y[j-1]) + 1 \end{array} \right\}$$

und damit kann man mit einer einfachen Erweiterung des Wagner-Fischer-Algorithmus auch Transpositionen berücksichtigen.

Beispiel:
$x = abcd$, $y = acbd$

		−1	0	1	2	3	4
				a	c	b	d
−1		∞	∞	∞	∞	∞	∞
0		∞	0	1	2	3	4
1	a	∞	1	0	1	2	3
2	b	∞	2	1	1	1	2
3	c	∞	3	2	1	(1)	2
4	d	∞	4	3	2	2	$\boxed{1}$

( ): hier geht die Transposition ein.
$\boxed{1}$: Der Abstand beträgt 1.
Bemerkung:
Die Spalte und Zeile "1" ist eingeführt worden wegen $d_{i-2,j-2}$.

▲Achtung !
$x = ab$, $y = bc$
Es gilt: x und y haben Abstand 3.

		-1	0	1	2	3
				b	c	a
-1		∞	∞	∞	∞	∞
0		∞	0	1	2	3
1	a	∞	1	1	2	2
2	b	∞	2	1	2	$\boxed{3}$

Der Abstand beträgt also 3, d.h. die optimale Edit-Sequenz zum überführen von x nach y hat 3 Operationen.
Es gilt jedoch:
$ab \overset{[1]}{\Longrightarrow} ba \overset{[2]}{\Longrightarrow} bca$
$[1] : ab \to ba$
$[2] : \varepsilon \to c$
aber Eigenschaft (♡) wird verletzt !

(15.5.2002)

Hirschberg Algorithmus

Lemma 2.2
Seien $x, y \in \Sigma^*$ mit $|x| = m$, $|y| = n$. Ferner sei für $i \in [1 : m]$
$k_i := min\{d_L^a(x[1..i], y[1..j]) + d_L^a(x[i+1..m], y[j+1..n]), j \in [1 : n]\}$
Dann ist $k_i = d_L^a(x, y)$ für $i \in [1 : m]$

Beweis:
☞ Hausübung 4.1 auf Seite 60.

☞ Folie 7 auf Seite 74

Beispiel:
Für $i = 3,\ j = 2$:
$k_3 = min\{3+3, 2+3, 2+2, 2+3, 3+3, 3+4, 3+4\} = 4$
Für $i = 2$:
$k_2 = min\{2+3, 2+3, 1+3, 2+4, 2+4, 3+4, 4+5\} = 4$

$d_L^a(x[1..3], y[1..2]) = 2$ $(j = 2, i = 3)$
Frage: Wie erhalte ich $d_L^a(x[4..7], y[3..6])$?
Idee: Es ist $d_L^a(x^R, y^R) = d_L^a(x, y)$
Also erhalte ich $d_L^a(x[4..7]^R, y[3..6]^R)$.
Also bestimme Abstandsmatrix $\overline{d}$ für $x[4..7]^R$ und y^R.
Man benötigt $\overline{d_{m-i,n-j}}$.
☞ 7 auf Seite 73.

$x \quad = x_1 \dots x_m$
$-^m = - \dots -$

Bemerkung: zum Algorithmus von Hirschberg
Die Prozedur `dist` berechnet die Distanzmatrix d_{ij} zeilenweise. Nach der Initialisierung gilt $Z\_alt[j] = d_{0j}$ und später gilt immer $Z\_alt[j] = d_{i-1,j}$ und $Z\_neu[j] = d_{ij}$.
Es wird die letzte Zeile der Distanzmatrix ausgegeben.

Lemma 2.3
`rec_align(x,y)` berechnet eine optimale Ausrichtung von x und y.

Beweis:
Für die Fälle $n = 0$ oder $m = 1$ ist das sicher der Fall, also sei $n > 0$, $m > 1$ angenommen und weiterhin gelte, daß `rec_align` für kleine m und n korrekt arbeitet.
Es ist $n \geq j$ und damit $n - j \geq 1$.
$L\_1[i] = d_L^a(x[1..j], y[1..i])$ für $i \in [0 : n]$ und, da $d_L^a(x, y) = d_L^a(x^R, y^R)$, folgt $L\_2[n-i] = d_L^a(x[j+1..m], y[i+1..n])$.
Es wird k so gesetzt, daß laut Lemma 2.2 (Seite 19)
$d_L^a(x, y) = d_L^a(x[1..j], y[1..k]) + d_L^a(x[j+1..m], y[k+1..n])$. Also werden x und y so in kürzere Zeichenketten aufgeteilt, daß aus den optimalen Ausrichtungen, die die rekursiven Aufrufe von `rec_align` laut Voraussetzung liefern, durch Konkatenation eine optimale Ausrichtung für x und y erzeugt wird.

Lemma 2.4
Im Algorithmus von Hirschberg wird die Prozedur `rec_align` $2m - 1$ mal aufgerufen.

Beweis:
Ist $m = 1$, dann gibt es einen Aufruf von `rec_align`.
Sei angenommen, daß für $M := 2^r \geq m > 2^{r-1}$, $r > 0$, $2m - 1$ Aufrufe stattfinden. Betrachte m' mit $2^r < m' \leq 2^{r+1} = 2M$.
In `rec_align` wird $j = \lfloor \frac{m'}{2} \rfloor$ gesetzt, also wird `rec_align` aufgerufen mit $x[1..j]$ und $|x[1..j]| = m_1 \leq 2^r$ sowie mit $x[j+1..m']$, wobei $|x[j+1..m']| = m_2 = m' - m_1 \leq 2^r$.
Laut Induktionsvoraussetzung führen diese beiden Aufrufe zu $2m_1 - 1$ bzw. $2m_2 - 1$ Aufrufen.
Es ergibt sich insgesamt für die Zahl der Aufrufe $1 + (2m_1 - 1) + (2m_2 - 1) = 2m' - 1$.

Satz 2.5
Der Algorithmus von Hirschberg benötigt $O(m * n)$ Zeiteinheiten und $O(n + m)$ Speicherplatz.

Beweis:
Sei $T(m, n)$ der Zeitaufwand für den Aufruf von `rec_align(x,y)` mit $|x| = m$, $|y| = n$.
Für $n, m \neq 0$ benötigt `dist` $c_0 * m * n$ Schritte für geeignetes c_0.
Es ist $T(1, n)$ beschränkt durch $c_1 * n + c_2$ für geeignete Konstanten c_1 und c_2.
Annahme: $T(m, n) \leq d_1 m * n + d_2$ für geeignete Konstanten d_1 und d_2.
Betrachte $T(2m, n)$. Es werden für beide Aufrufe von `dist` $c_0 * jn$ und $c_0 * (m-j)n$, also

höchstens c_0mn Zeiteinheiten benötigt. Die restlichen Operationen bis auf die rekursiven Aufrufe benötigen höchstens $c_3n + c_4m + c_5$ Zeiteinheiten für geeignete c_3, c_4, c_5.
Die beiden rekursiven Aufrufe sind bezüglich der Zeit laut Voraussetzung durch $d_1 * jn + d_2$ bzw. $d_1(m-j)n + d_2$ beschränkt.
Also ist $t = (d_1 + c_0)n * m + c_3n + c_4m + c_5 + 2d_2$ eine Schranke für $T(2m, n)$. Dann ist $t \leq (d_1 + c_0 + c_3 + c_4 + c_5 + d_2)nm + d_2$.
Damit dieser Wert konsistent mit der angenommenen Schranke ist, muß $2d_1 = d_1 + c_0 + c_3 + c_4 + c_5 + d_2$ gelten, also $d_1 = c_0 + c_3 + c_4 + c_5 + d_2$.
Also ist $T(m, n) = O(mn)$.
Damit ist der gesamte Hirschberg-Algorithmus durch $O(mn)$ Zeiteinheiten beschränkt.
Die Zeichenketten x und y benötigen $O(n + m)$ Speicherplätze. Die Prozedur `dist` benötigt $O(n)$ Speicherplätze für die Zeilen `Z_alt` und `Z_neu`.
Die Rückgabefelder `L_1` und `L_2` benötigen ebenfalls $O(n)$ Speicherplatz, die Felder können global gespeichert werden.
(x', y') belegt $O(n + m)$ Speicher. Teilzeichenketten können durch Anfangs- und Endindices übergeben werden.
Die rekursiven Aufrufe benötigen also (laut Lemma 2.4 (Seite 20)) auch nur $O(m)$ Speicher. **(29.5.2002)**
Für das Folgende seien x und y Zeichenketten aus Σ^*, $|x| = m$ und $|y| = n$
und $w : \tilde{\Sigma} \times \tilde{\Sigma} \to R_{\geq 0}$ sei die elementare Abstandsfunktion.
Es ist nun möglich, die Bestimmung des Abstands der beiden Zeichenketten x und y auf ein Problem aus der Graphentheorie zurückzuführen.
Man betrachte einen gerichteten Graphen $G = (V, E)$ mit
Knotenmenge $V = \{(i, j) \,|\, i \in [0..m], j \in [0..n]\}$ und der
Kantenmenge $E = E_1 \cup E_2 \cup E_3$ wobei

$$\begin{aligned} E_1 &= \{((i-1, j), (i, j)) \,|\, i \in [1..m], j \in [0..n]\}, \\ E_2 &= \{((i, j-1), (i, j)) \,|\, i \in [0..m], j \in [1..n]\} \text{ und} \\ E_3 &= \{((i-1, j-1), (i, j)) \,|\, i \in [1..m], j \in [1..n]\} \end{aligned}$$

Jede Kante erhält ein Gewicht, und zwar

$$\begin{aligned} \text{eine Kante } ((i-1, j), (i, j)) &\quad \text{das Gewicht} \quad w(x[i], \varepsilon), \\ \text{eine Kante } ((i, j-1), (i, j)) &\quad \text{das Gewicht} \quad w(\varepsilon, y[j]) \text{ und} \\ \text{eine Kante } ((i-1, j-1), (i, j)) &\quad \text{das Gewicht} \quad w(x[i], y[j]). \end{aligned}$$

Die Frage nach dem Abstand zwischen x und y entspricht dann der Frage nach dem "kürzesten Weg" von $(0, 0)$ nach (m, n)!

Beispiel:
☞ Seite 70
Optimale Ausrichtungen:

$$\begin{bmatrix} a & b & c & a & b & b & a \\ c & b & - & a & b & a & c \end{bmatrix} \quad \begin{bmatrix} a & b & c & a & b & b & a & - \\ c & b & - & a & - & b & a & c \end{bmatrix} \quad \begin{bmatrix} a & b & c & a & b & b & a & - \\ c & b & - & a & b & - & a & c \end{bmatrix}$$

(Kosten jeweils 4)

Das Verfahren von Ukkonen

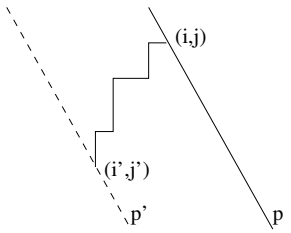
Idee: Nachdem die Distanzmatrix $D = (d_{i,j})$ mit $0 \leq i \leq m$, $0 \leq j \leq n$ für x und y bezüglich der elementaren Abstandsfunktion w aufgebaut wurde, entfernt man im zugeordneten Graphen alle Kanten , die nicht zur Minimumbildung beitragen. Genauer, die Kante vom Knoten $(i-1, j)$ zum Knoten (i, j) wird entfernt, falls $d_{i,j} < d_{i-1,j} + w(x[i], \varepsilon)$ gilt. Entsprechend wird die Kante vom Knoten $(i, j-1)$ zum Knoten (i, j) entfernt, falls $d_{i,j} < d_{i,j-1} + w(\varepsilon, y[j])$ ist bzw. die Kante vom Knoten $(i-1, j-1)$ zum Knoten (i, j), falls $d_{i,j} < d_{i-1,j-1} + w(x[i], y[j])$ gilt. Den so erhaltenen Graphen nennt man Abhängigkeitsgraph.

☞ Seite 75

Lemma 2.6
Für den so definierten Abhängigkeitsgraphen gilt:

1. Die Kosten eines jeden Weges von $(0, 0)$ nach (i, j) sind $d_{i,j}$.
2. Ist d die Summe der Kosten eines Weges von (i, j) nach (i', j'), so gilt $d_{i',j'} = d_{i,j} + d$.
3. Nur Punkte, die auf einem Weg von $(0, 0)$ nach (n, m) liegen, sind für die Bestimmung des Wertes $d_{n,m}$ relevant.

Sei nun $w_{min} = min\{w(a, \varepsilon), w(\epsilon, a) | a \in \Sigma\}$ und sei $w_{min} > 0$. Die Diagonalen der Distanzmatrix D seien mit ganzen Zahlen $p \in [-m..n]$ durchnumeriert, so daß die Diagonale p alle Einträge $d_{i,j}$ mit $j - i = p$ enthält.
Betrachte nun einen Weg von (i, j) nach (i', j') im Abhängigkeitsgraph.
(i, j) liegt auf der Diagonalen $p = j - i$, (i', j') auf $p' = j' - i'$.
Ist $p' \leq p$, dann enthält der Weg mindestens $|p' - p|$ vertikale Kanten; ist $p \leq p'$, dann enthält der Weg mindestens $|p' - p|$ horizontale Kanten.



Also folgt $d_{i'j'} \geq d_{ij} + |(j' - i') - (j - i)| * w_{min}$
Damit ist $d_{ij} \geq |j - i| * w_{min}$ für jeden Punkt (i, j) auf dem Weg von $(0, 0)$ nach (m, n).
Da $d_{ij} \leq d_{mn}$ gelten muß , gilt $|j - i| \leq \frac{d_{ij}}{w_{min}} \leq \frac{d_{mn}}{w_{min}}$
Also reicht es aus, nur Punkte (i, j) im Diagonalband $-\frac{d_{mn}}{w_{min}} \leq j - i \leq \frac{d_{mn}}{w_{min}}$ zu berechnen.

Satz 2.7
Seien x und y Zeichenketten aus Σ^*, $|x| = m$ und $|y| = n$ und sei $D = (d_{i,j})$ die Distanzmatrix zu x und y.

a) Gibt es einen Weg von (i', j') nach (i, j) im Abhängigkeitgraphen, so ist $d_{i,j} \geq d_{i',j'} + |(j - i) - (j' - i')| * w_{min}$.

b) Liegt (i, j) auf einem Weg von $(0, 0)$ nach (m, n) im Abhängigkeitsgraphen von x und y, so gilt

$$-q \leq j - i \leq n - m + q \quad \text{für} \quad m \leq n,$$
$$n - m - p \leq j - i \leq q \quad \text{für} \quad m > n.$$

wobei $q = \left\lfloor \left(\frac{d_{m,n}}{w_{min}} - (n - m) \right) /2 \right\rfloor$

Beweis:
Betrachte Weg von $(0, 0)$ über (i, j) nach (m, n).
Es gilt:
$d_{mn} \geq d_{ij} + |(n - m) - (j - i)| * w_{min} \geq d_{00} + |j - i| * w_{min} + |(n - m) - (j - i)| * w_{min}$
also $\mathbf{d_{mn}} \geq (|\mathbf{j} - \mathbf{i}| + |(\mathbf{n} - \mathbf{m}) - (\mathbf{j} - \mathbf{i})|) * \mathbf{w_{min}}$
Sei $m \leq n$ angenommen.
Für den Fall $j \leq i$ folgt dann

$$\frac{d_{mn}}{w_{min}} \geq -(j - i) + (n - m) - (j - i)$$

also

$$0 \geq j - i \geq -\left\lfloor (\frac{d_{mn}}{w_{min}} - (n - m))/2 \right\rfloor = -q$$

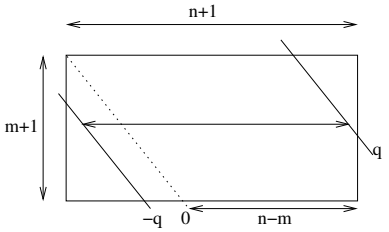
Ist $j \geq i$, dann gibt es zwei Unterfälle:

a) $(n - m) \geq (j - i)$.
Dann gilt $d_{mn} \geq ((j - i) + (n - m) - (j - i)) * w_{min}$,
also $d_{mn} \geq (n - m) \geq (j - i)$
Da $d_{mn} \geq (n - m) * w_{min}$, ist $q \geq 0$ und es folgt die rechte Seite der Ungleichung.

b) $(n - m) < (j - i)$
Dann gilt $d_{mn} \geq ((j - i) + (j - i) - (n - m)) * w_{min}$,
also $\frac{d_{mn}}{w_{min}} \geq 2(j - i) - (n - m)$
Es folgt $\frac{d_{mn}}{w_{min}} - (n - m) \geq 2((j - i) - (n - m))$
und damit $\underbrace{\left\lfloor (\frac{d_{mn}}{w_{min}} - (n - m))/2 \right\rfloor}_{q} + (n - m) \geq j - i$

Der Fall $m > n$ folgt analog.

Will man also testen, ob $d(x, y) \leq h$ gilt für einen vorgegebenen Schwellwert h, kann man die Berechnung der Distanzmatrix auf das Band zwischen den Diagonalen $-q$ und $n - m + q$ beschränken ($m \leq n$ angenommen).



$q = \lfloor (h/w_{min} - (n - m))/2 \rfloor$ ist.
$2q \leq h/w_{min} - (n - m)$, also
$n - m + 2q + 1 \leq 1 + \frac{h}{min}$

Da $2q \leq h/w_{min} - (n-m)$ gilt, folgt, daß die Anzahl der zu berechnenden Werte pro Zeile der Distanzmatrix höchstens $q+1+n-m+q \leq 1+\frac{h}{w_{min}}$, also $n-m+2q+1 = O(h)$ ist.
Folglich kann die Berechnung des Bandes in einer Zeit $O(mh)$ und mit einem Platzbedarf von $O(mh)$ durchgeführt werden, falls nur das Band gespeichert wird. Interessiert nur der Wert $d_{m,n}$, reichen sogar $O(h)$ Speicherplätze.

Um jetzt den Wert $d = d(x,y)$ zu bestimmen, wendet man diesen Schwellwert-Algorithmus nacheinander auf die Werte h_0, $h_1 = 2h_0$, $h_2 = 2^2h_0$, $h_3 = 2^3h_0$, ... an, wobei h_0 der minimal mögliche Wert von $d(x,y) + w_{min}$ ist (etwa $n - m + 1$ für die Edit-Metrik). Ist der so berechnete Wert von $d_{m,n}$ größer als der Schwellwert, muß man das Verfahren mit dem nächst größeren Schwellwert wiederholen. Hat man nach $r+1$ Aufrufen des Schwellwert-Algorithmus $d(x,y)$ bestimmt, ist der gesamte Zeitbedarf $O(m\sum_{k=0}^{r} h_r)$, also $O(m(2h_r - h_0)) = O(mh_r)$. Da $d > \frac{h_r}{2}$ gilt, ist der gesamte Zeitbedarf $O(md)$.
☞ Seite 76
☞ Seite 77
Es folgt

Satz 2.8
Das Verfahren von Ukkonen benötigt zum Test, ob $d(x,y) \leq h$ gilt $O(mh)$ Zeit und $O(mh)$ Speicher zur Speicherung der gesamten Bandmatrix. Will man nur den Wert $d(x,y)$ bestimmen, so benötigt man $O(md)$ Zeit und $O(h)$ Speicher.

(5.6.2002)

Beispiele zum Verfahren von Ukkonen:
☞ Seite 75
☞ Seite 76
☞ Seite 77
$q = \left\lfloor \left(\frac{h}{w_{min}} - (n-m) \right) /2 \right\rfloor$
zum Fall $h = 2$:
$q = 0$
Diagonalen: 0 und 1
Ergebnis: $7 > h$ ↯
zum Fall $h = 4$:
$q = 1$
Diagonalen von -1 bis 2
Ergebnis: $5 > h$ ↯
zum Fall $h = 8$:
$q = 3$
Diagonalen von -3 bis 4
Ergebnis: $5 \leq h$ ✓
☞ Seite 78
zum Fall $h = 2$:
$q = 0$

Diagonalen: 0 und 1
Ergebnis: $4 > h = 2$ ↯
zum Fall $h = 4$:
$q = 1$
Diagonalen von -1 bis 2
Ergebnis: $4 \leq h$ $\surd$
Also betragen die Kosten=4.
Eine Mögliche Ausrichtung wäre etwa:

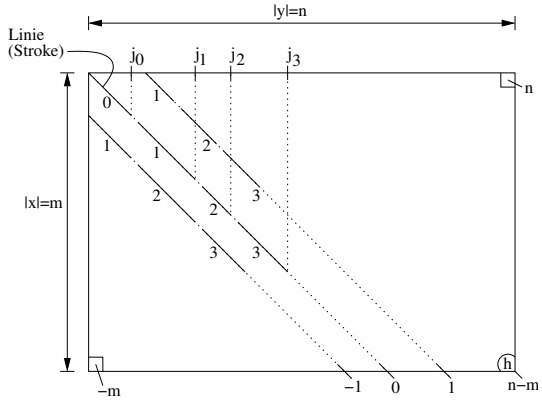
$$\begin{bmatrix} c & b & & a & b & & a & c \\ | & | & & | & | & & | & \\ a & b & c & a & b & b & a & \end{bmatrix}$$

Diagonalen Verfahren

Lemma 2.9
Seien $x, y \in \sum^*$. Dann gilt für die Distanzmatrix $D = (d_{ij})$ bezüglich der **Levenshtein Metrik**

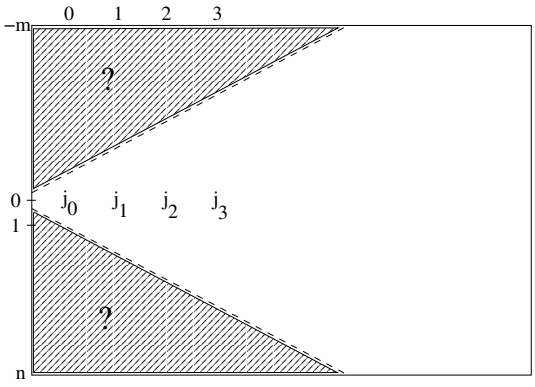
$$d_{ij} \in \{d_{i-1,j-1}, d_{i-1,j-1} + 1\},\ i \in [1..|x|],\ j \in [1..|y|].$$

Beweis:
☞ Aufgabe 1, Aufgabenblatt 3, Seite 58.
Folgt man also den Diagonalen der Distanzmatrix im Fall der Levenshtein-Metrik, so wachsen die Werte monoton um jeweils genau +1.
Also kann man nur die Positionen der Stellen speichern, an denen der Wert auf einer Diagonalen wächst, ohne Informationen zu verlieren.



1 Diagonale beginnt mit 1-Linie u.s.w.
$\Rightarrow$ Nur Kegel der Matrix ist interessant.

Diagonalenmatrix



z.B.: $L_{0,2} = j_2$

Bemerkung:

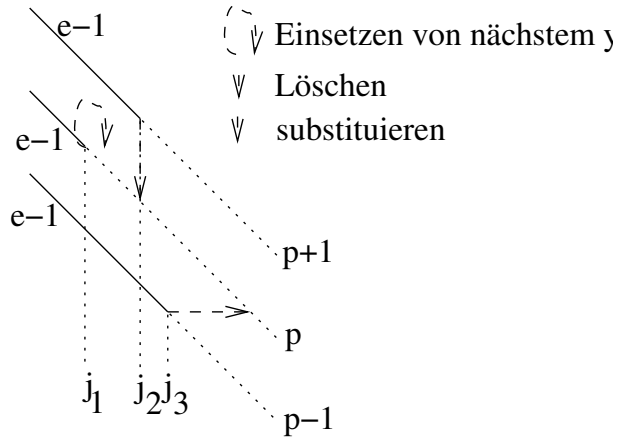
1. Alle Werte auf einer Diagonalen p sind $\geq |p|$

2. Jede Diagonale hat höchstens $min(m,n)+1$ Werte, wobei $m=|x|,\ n=|y|$

Setzt man für $|x|=m$ und $|y|=n$ und

$$L_{p,e} = max\{j \in [0:n] | d_{ij} = e \text{ und } j-i=p\}$$

So hat man für jede Diagonale $p \in |-m,n]$ die "Sprungstellen" festgelegt, bei denen von $e-1$ "Fehlern" auf e Fehler ($\hat{=}$ Abstand) umgeschaltet wird.
Idee:



Mit einer dieser 3 Möglichkeiten muß man von $e-1$ nach e kommen.

Satz 2.10
Für $L_{p,e}$ mit $p \in [-m..n],\ e \in [0..n]$ gilt

$$L_{p,e} = i + max\{j | x[i-p..i-p+j] = y[i..i+j]\}$$

und $$i = max\{L_{p,e-1}+1, L_{p-1,e-1}+1, L_{p+1,e-1}\}$$

☞ Seite 83 bei 7 ist die Abbruchbedingung erreicht.

Beweis:
Betrachte Distanzmatrix D und betrachte die Werte auf der Diagonalen p. Sei j_1 Spaltenindex des "untersten" Eintrags mit Wert $e-1$ auf p, also $d_{j_1+p,j_1} = e-1$ und $d_{j_1+p+1,j_1+1} = e$.
Sei jetzt j_2 der Spaltenindex des untersten Eintrags mit Wert $e-1$ auf der Diagonalen $p-1$ und j_3 auf der Diagonalen $p+1$.
Zu bestimmen ist der unterste Eintrag auf p mit Wert e.
Es gibt drei Möglichkeiten auf die e-Linie der Diagonalen p zu kommen: (gesucht ist der "unterste" Wert, d.h. der maximale Spaltenindex)

1. von der $e-1$-Linie auf p durch einen diagonalen Schritt (Spaltenindex erhöht sich um 1)

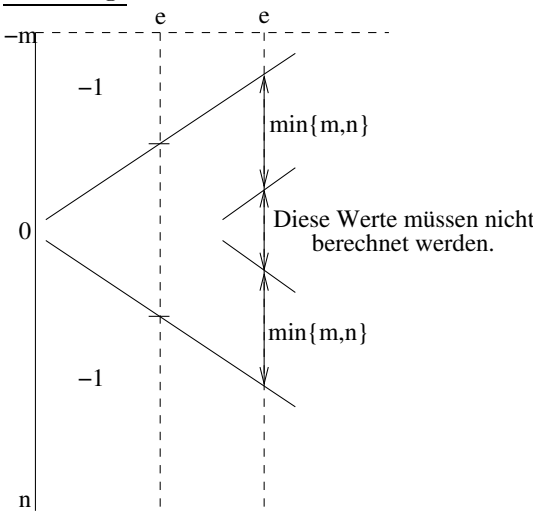
2. von der $e-1$-Linie der Diagonalen $p-1$ durch einen horizontalen Schritt (Spaltenindex erhöht sich)

3. von der $e-1$-Linie der Diagonalen $p+1$ durch einen vertikalen Schritt (Spaltenindex bleibt)

Ist man auf der e-Linie von p angekommen, so muß diese Linie bis zum Ende verfolgt werden, d.h. es muß geprüft werden, wie weit die zugehörigen Teilworte von x und y identisch sind.
☞ Seite 82

Lemma 2.11
Sei $d = d_L(x, y)$ der Abstand von x und y. Dann benötigt der Algorithmus $O(d * min\{m, n\})$ Zeit und $O(d * min\{d, n, m\})$ Speicherplatz.

Beweis:
Die `repeat`-Schleife wird höchstens $min\{d, n, m\} + 1$ mal durchlaufen. Die `for`-Schleife nimmt höchstens $O(d)$ Werte an, also werden $O(d * min\{d, m, n\})$ Werte der Diagonalmatrix berechnet.
Für ein festes e wird bei allen Durchläufen der Test $x[t + p + 1] = y[t + 1]$ höchstens einmal durchgeführt.
Es treten $O(d)$ verschiedene e auf, also erreicht man eine Gesamtlaufzeit
von $O(d * min\{m, n\})$.
Nachtrag:



Weiterhin kann man die Zahl der zu berechnenden Werte der Diagonalenmatrix weiter einschränken, dafür eine Diagonale p nur Werte $L_{p,e}$
mit $e \in \{|p|, |p| + 1, \ldots, |p| + min\{m, n\}\}$ definiert sein können.
Ist $d_L(x, y) = d$, so müssen alle Spalten $\leq d$ berechnet werden.
Also werden pro Spalte $O(min\{d, m, n\})$ Werte berechnet und der Algorithmus benötigt $O(d * min\{d, m, n\})$ Speicherplatz.
Speichert man nur eine Spalte um die nächste zu berechnen, reduziert sich der Platzbedarf auf $O(min\{d, m, n\})$.

Das Verfahren von Sellers für das k-differences Problem

k-differences Problem:
Gegeben: Zeichenketten x, y, Abstandsfunktion d_L, $k \geq 0$.
Gesucht: Die Positionen aller Teilzeichenketten s von y mit $d(s, x) \leq k$.
Idee:
Änderung der Berechnung der Distanzmatrix. d_{ij} soll den minimalen Abstand zwischen $x[1..i]$ und jedem Suffix von $y[1..j]$ angeben.
Man erreicht dies durch eine veränderte Initialisierung der Distanzmatrix !
Es sei $d_{i0} = i,\ i \in [0..m]$ aber $d_{0j} = 0,\ j \in [0 : n]$
☞ Seite 86

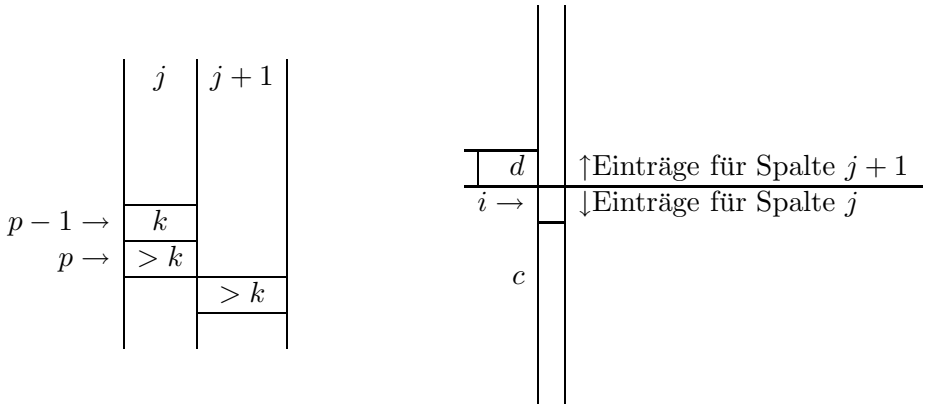
(12.6.2002)

☞ Seite 86

a	b	c	d	e		a	b		c		d	e												
|		|		|		|	|			|			|	|								
a		c		e		a	b	p	c	q	d	e												
a	b	c	d	e		a	b	c	d	e			a	b	c	d	e							
|	|	|				|	|	|		|			|	|	|	|					
a	b	c				a	b	c		r			a	b	c	r								

Satz 2.12
Das Verfahren von Sellers löst das k-Differenzen Problem in einer Zeit $O(mn)$ mit $O(mn)$ Speicherplatz.

Bemerkung:
Ist man nur an einer Position interessiert, kann der Speicherplatzbedarf auf $O(m)$ reduziert werden.
Nutzt man Lemma 2.9 (Seite 25), dann kann man leicht sehen, daß gewisse Teile der Distanzmatrix nicht berechnet werden müssen.
Idee:
Man berechnet die Matrix spaltenweise. Wird in einer Spalte $j \geq 1$ ein Wert $k + 1$ berechnet, dann braucht die zugehörige Diagonale nicht mehr berechnet zu werden !
Man merkt sich, in welcher Zeile letztmalig ein Wert $\geq k$ auftritt.
Sei $p - 1$ diese Zeile. Dann müssen in der nächsten Spalte höchstens p Werte berechnet werden.



☞ Seite 92

Satz 2.13
Das Verfahren von Jokinen, Tarhio und Ukkonen löst das k-Differenzen Problem für zufällige Zeichenketten x und y in $O(km)$ Schritten und $O(m)$ Speicherplatz.

Beweis:
der Zeitkomplexität von Chang und Lampe

Das Verfahren von Chang und Lampe zur Lösung des k-Differenzen Problems

Lemma 2.14
Sei $|x| = m$, $|y| = n$, $x, y \in \Sigma^*$ und sei D Distanzmatrix bezüglich der Levenshtein-Metrik.
Es gilt:

1. $-1 \leq d_{ij} - d_{i-1,j} \leq 1$ (Spalteneigenschaft)
2. $-1 \leq d_{ij} - d_{i,j-1} \leq 1$ (Zeileneigenschaft)

Beweis:
Laut Rekursion gilt

$$d_{ij} = min\{d_{i-1,j} + 1, d_{i,j-1} + 1, d_{i-1,j-1} + w(x[i], y[j])\} \quad (*)$$

also

$$d_{ij} \geq min\{d_{i-1,j} + 1, d_{i,j-1} + 1, d_{i-1,j-1}\} \quad (+)$$

Aus (*) folgt

$$d_{ij} \leq d_{i-1.j} + 1, \text{ also } d_{ij} - d_{i-1,j} \leq 1$$

und (**)

$$d_{ij} \leq d_{i,j-1} + 1, \text{ also } d_{ij} - d_{i,j-1} \leq 1$$

Behauptung:

$$d_{ij} - d_{i-1,j} \geq -1$$

Beweis:
Die Behauptung gilt für $j = 0$, denn $d_{i0} = i$ und $d_{i-1,0} = i - 1$.

Gelte nun $d_{i,j-1} - d_{i-1,j-1} \geq -1$, d.h. $d_{i,j-1} + 1 \geq d_{i-1,j-1}$.
Aus (**) folgt $d_{i-1,j-1} \geq d_{i-1,j} - 1$. Setzt man beides in (+) ein, erhält man

$$d_{ij} \geq min\{d_{i-1,j} + 1, d_{i-1,j} - 1, d_{i-1,j} - 1\}$$

also

$$d_{ij} \geq d_{i-1,j-1}$$

Also folgt (1).
Behauptung:

$$d_{ij} - d_{i,j-1} \geq -1$$

Beweis:
Die Behauptung gilt für $i = 0$, da $d_{0j} = d_{0j-1} = 0$
Gelte nun $d_{i-1,j} - d_{i-1,j-1} \geq -1$, d.h. $d_{i-1,j} + 1 \geq d_{i-1,j-1}$.
Aus (**) folgt $d_{i-1,j-1} \geq d_{i,j-1} - 1$.
Setzt man wieder in (+) ein, so erhält man

$$d_{ij} \geq min\{d_{i,j-1} - 1, d_{i,j-1} + 1, d_{i,j-1} - 1\}$$

also

$$d_{ij} \geq d_{i,j-1} - 1$$

Also folgt (2).

Idee:
Man partitioniert jetzt jede Spalte der Distanzmatrix in Folgen aufsteigender Zahlen (run).
Element d_{ij} gehört zur Folge r, falls $r = i - d_{ij}$.

Beispiel:

0	↓ Folge 0
1	
0	↓ Folge 2
1	
2	
3	

Folge 1 ist leer.
Folge r in Spalte j endet auf Position i, falls i der kleinste Wert ist mit: $d_{i+1,j}$ gehört nicht zur Folge r.

Beispiel:
Folge 0 endet auf Position 1, Folge 1 endet auf Position 1 und Folge 2 endet auf Position 5.

(19.6.2002)

Lemma 2.15
Ist die Folge r in Spalte j leer, so ist die Folge $r + 1$ in Spalte j nicht leer.

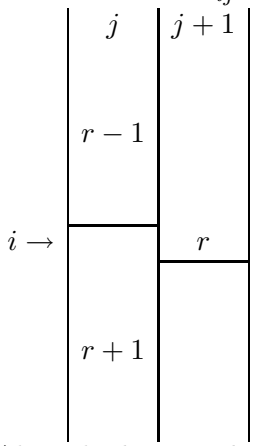
Beweis:
Wenn Folge r in Spalte j auf Position i endet und $i < m$ ist, dann gehört $d_{i+1,j}$ zu einer Folge r' mit $r' = i + 1 - d_{i+1,j}$. Laut Lemma 2.14 (Seite 29) folgt dann $r < r' \leq r + 2$ da $r' = i - d_{i+1,j} + 1 \leq i - d_{ij} + 2 = r + 2$.

Bemerkung:
Die Folge 0 ist niemals leer.

Als nächstes soll versucht werden, aus den Endpositionen der Folgen in Spalte j die Endpositionen der Folgen in Spalte $j + 1$ zu berechnen.

Lemma 2.16
Sei die Folge r in Spalte j leer und sei i Endposition dieser Folge.
Dann endet Folge r in Spalte $j + 1$ auf Position $i + 1$.

Beweis:
Folge $r - 1$ in Spalte j ist also nicht leer und endet auf Position i. Also ist $r - 1 = i - d_{ij}$ also $r = i + 1 - d_{ij}$ und, da Folge r ebenfalls auf Position i endet, $i + 1 - d_{i+1,j} > r$.

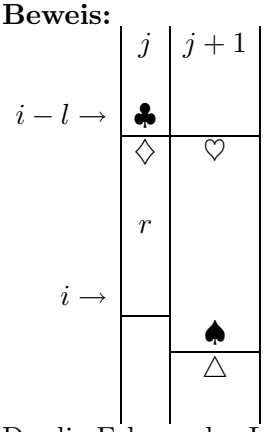


Also gilt $d_{i+1,j} < d_{ij}$ (*)
Laut Lemma 2.9 (Seite 25) gilt $d_{i+1,j+1} \geq d_{ij}$ (Diagonaleneigenschaft)
und laut Lemma 2.14 (Seite 29) gilt
$d_{i+1,j+1} \leq d_{i+1,j} + 1 \leq d_{ij}$ (Zeileneigenschaft und (*)).
Also gehört der Eintrag $d_{i+1,j+1}$ zur Folge $i + 1 - d_{i+1,j+1} = i + 1 - d_{ij} = r$.
Weiter folgt $d_{i+2,j+1} \leq d_{ij}$ (aus (*) mit Diagonaleneigenschaft), also gehört $d_{i+2,j+1}$ zur Folge $i + 2 - d_{i+2,j+1} \geq i + 2 - d_{ij} = r + 1 > r$.
Also endet Folge r in Spalte $j + 1$ auf Position $i + 1$.

Lemma 2.17
Die Folge r der Länge $l > 0$ in Spalte j ende auf Position i.
Ferner sei $s = min\{k | i - l + 2 \leq k \leq i + 1 \text{ und } x[k] = y[j + 1]\} \cup \{\infty\}$.
Dann gilt für die Endposition $\hat{i}$ der Folge r in Spalte $j + 1$

$$\hat{i} = \begin{cases} s-1 & \text{falls } s < \infty \\ i+1 & \text{falls die Folge } r+1 \text{ in Spalte } j \text{ nicht leer ist} \\ i & \text{falls die Folge } r+1 \text{ in Spalte } j \text{ leer ist} \end{cases}$$

Beweis:



Da die Folge r der Länge $l \geq 1$ in Spalte j auf Position i endet, startet diese Folge auf Position $i-l+1$ und Folge $r-1$ endet auf Position $i-l$.
♢ Es folgt $r = i-l+1-d_{i-l+1,j} > i-l-d_{i-l,j} = r-1$ ♣, also $d_{i-l+1,j} < d_{i-l,j}+1$ oder (Spalteneigenschaft) $d_{i-l+1,j} \leq d_{i-l,j}$
Mit Lemma 2.9 (Seite 25) (Diagonaleneigenschaft) folgt $d_{i-l+1,j+1} \geq d_{i-l,j} \geq d_{i-l+1,j}$.
Im Fall $d_{i-l+1,j+1} = d_{i-l+1,j}$ gehört der Eintrag auf Position $i-l+1$ in Spalte $j+1$ zur Folge r.
Im Fall $d_{i-l+1,j+1} > d_{i-l+1,j}$ ist die Anfangsposition der Folge r in Spalte $j+1$ größer $i-l+1$. Also gilt, daß die Position an der Folge r in Spalte $j+1$ enden kann, größer gleich $i-l+1$ ist. (♡).
Lemma 2.9 (Seite 25) zeigt, daß $d_{i+1,j+1} \leq d_{ij}+1$ (Diagonaleneigenschaft).
Ist $d_{i+1,j+1} = d_{ij}+1$ (♠), gehört $d_{i+1,j+1}$ zur Folge r in Spalte $j+1$.
Behauptung: Dann gehört $d_{i+2,j+1}$ △ nicht zur Folge r.
Beweis:
Es gilt $d_{i+2,j+1} \leq d_{i+1,j}+1$ (Diagonaleneigenschaft). Da Folge r auf Position i in Spalte j endet, gilt $i-d_{ij} \leq i+1-d_{i+1,j}$, also $d_{i+1,j} \leq d_{ij}$.
Damit folgt $d_{i+2,j+1} \leq d_{ij}+1 < d_{ij}+2$ und damit $i+2-d_{i+2,j} > i+2-d_{ij}-2 = r$.
Ist $d_{i+1,j+1} = d_{ij}$, dann gehört $d_{i+1,j+1} = d_{ij}$, dann gehört $d_{i+1,j+1}$ zur Folge $r+1$.
Also liegt die Endposition der Folge r in Spalte $j+1$ im Intervall $[i-l+2..i+1]$.
Um die genaue Position zu finden betrachte man Position s in x mit $x[s] = y[j+1]$.
Für die Distanzmatrix folgt $d_{s,j+1} = d_{s-1,j}$.
Gilt $i-l+2 \leq s \leq i+1$, dann gehört $d_{s-1,j}$ zur Folge r in Spalte j, d.h. $s-1-d_{s-1,j} = r$ und es gilt $s-d_{s,j+1} = r+1$.
Also endet die Folge r in Spalte $j+1$ bei Position $s-1$, wobei s wie in der Behauptung definiert ist.
Gibt es kein derartiges s, dann endet die Folge r in Spalte $j+1$ auf der maximal möglichen Position $i+1$, sofern die Länge der Folge $r+1$ in Spalte j nicht 0 ist.
In diesem Fall ist $d_{i+1,j} < d_{ij}$ und damit $d_{i+1,j+1} = d_{ij}$.

Also gehört $d_{i+1,j+1}$ zur Folge $r+1$ und Folge r endet auf Position i in Spalte $j+1$.

Die so entwickelten Rekursionen erlauben es zusammen mit einer Tabelle von Indizes, die Endpositionen einer Folge in konstanter Zeit zu bestimmen. Also kann man die Spalte j der Distanzmatrix in $O(m - d_{mj})$ Zeiteinheiten bestimmen, denn die letzte Folge in dieser Spalte ist die Folge $r - m - d_{mj}$.
Die Zeit zur Bestimmung aller Spalten ist danach $O(n(m - \overline{d_m}))$, wobei $\overline{d_m}$ der Mittelwert aller Einträge in der $m - ten$ Zeile ist.
Für die Tabelle von Indizes gelte

$$loc_i[a] = min\{s | i \le s \le n,\ x[s] = a\} \cup \{m+2\}$$

(hierbei steht $m+2$ im Endeffekt für ∞, geschmackssache, was man nimmt)
Diese Tabelle der Gröβe $O(m|\Sigma|)$ kann in einer Zeit $O(m|\Sigma|)$ erstellt werden.
☞ Seite 87

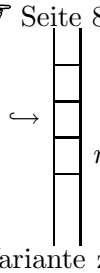
(26.6.2002)

Zeile	$D = (d_{ij})$	
0	0	↓ Folge 0
1	1	
2	2	
3	2	↓ Folge 1
4	3	
5	2	↓ Folge 3
6	3	

Folge $i = i - d_{ij}$

	end	length
0	2	3
1	4	2
2	4	0
3	6	2

☞ Seite 88 $S = min\{k | x[l] = y[j+1]$ und $i - l + 2 \le k \le i+1\}$
$loc_a[i] = min\{k | i \le k \le m,\ x[k] = a\} \cup \{\infty\}$ (oder $\infty = m+2$)
☞ Seite 87



Durch Vorverarbeitung und Verwendung der `loc`-Funktion erspart man sich die `min`-Bildung in jedem Schritt. Es ist zweckmäßig, eine Variante zu benutzen, die die in der Spalte höchste Folgennummer festhält. Wenn also in Spalte j die Folge t auf Position m endet, dann ist die höchste mögliche Folgennummer, die in Spalte $j+1$ auf Position m endet, die $t+1$. Hat also die aktuelle Folge eine Nummer $> t$, so endet die Folge auf Position m.

Satz 2.18
Der Algorithmus von Chang und Lampe löst das k-Differenzen-Problem im Mittel in einer Zeit $O(nm/\sqrt{|\Sigma|})$ und Platzbedarf $O(m|\Sigma|)$.

Beweis:
Es läβt sich zeigen, daß die mittlere Länge einer Folge $O(\sqrt{|\Sigma|})$ ist.
Daraus folgt der Zeitbedarf (gerechnet ohne Initialisierung der $loc_a[i]$)

☞ Seite 86
☞ Seite 89

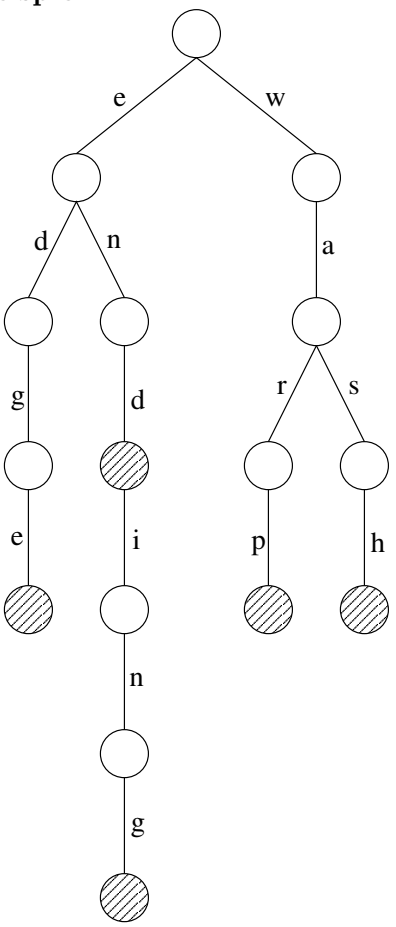
z.B.: $x = abcde$

loc	1	2	3	4	5	6	7
a	1						
b	2	2		∞			
c	3	3	3			∞	∞
d	4	4	4	4			
e	5	5	5	5	5		

Gelangt man bis zu Folge 3, dann ist ein Teilstring gefunden !

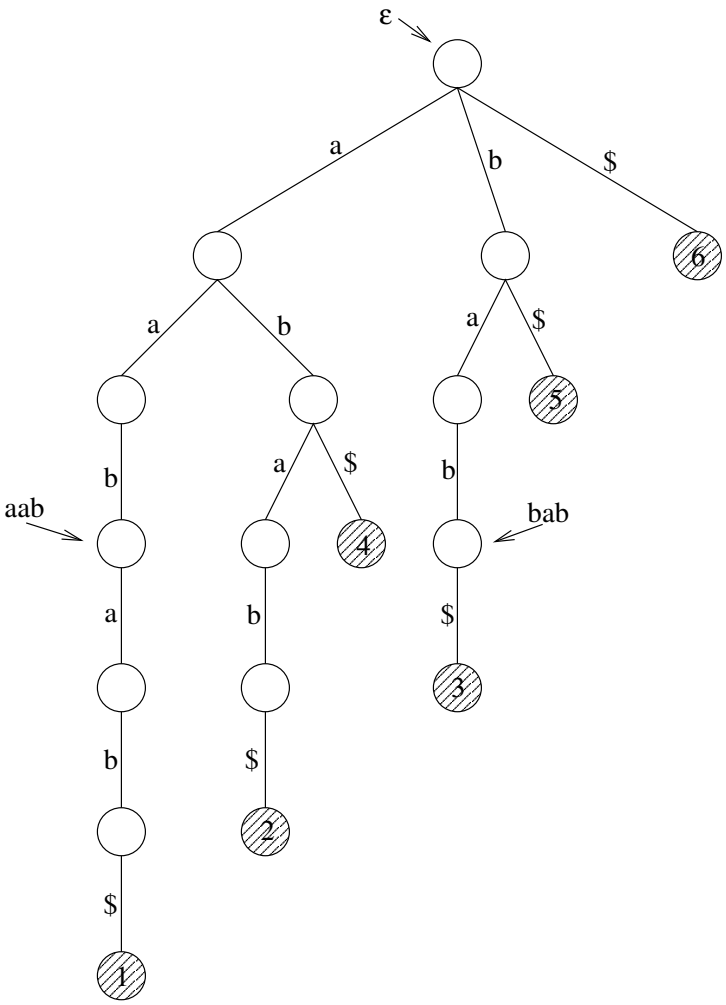
3 Suffix-Bäume

Suffix-Bäume beinhalten eine kompakte Darstellung aller nichtleeren Teilwörter einer Zeichenkette. Sie sind eine Weiterentwicklung sogenannter Patricia-Bäume (Practical Algorithm To Retrieve Information Coded In Alphanumeric) (Morrison 1968) und sind kompakte Darstellung sogenannter Suffix Trie's.
Ein Trie ist ein (digitaler) Suchbaum (Trie stammt von "Information Retrieval", Fredkin 1959)
In einem Trie über einem Alphabet Σ wird jede Kante mit einem Symbol aus Σ markiert und benachbarte Knoten haben unterschiedliche Symbole an den Kanten. Der maximale Verzweigungsgrad ist $|\Sigma|$.
Beispiel:



Blätter sind Endknoten und repräsentieren ein Wort.
Es kann auch passieren, daß interne Knoten ein Wort repräsentieren.
Häufig wird gefordert, daß jedes Schlüsselwort mit einem zusätzlichem "$"-Endmarker versehen wird, der nicht aus Σ ist. (Präfix-Freiheit)
Ein Suffix-Trie einer Zeichenkette x ist ein Trie, in dem alle Suffixe von $x\$$ abgespeichert sind.

Beispiel:
Suffix-Trie für $x = aabab\$$



Suffix $i \hat{=} x[1..n]$.
Jeder Knoten repräsentiert ein Teilwort von x, das aus der Konkatenation der Buchstaben des Weges von der Wurzel zum Knoten besteht.
Diese Beziehung ist eindeutig !
Jedes Blatt repräsentiert einen Suffix, die Nummer gibt die Anfangsposition des Suffix von x an.
Ein derartiger Suffix Trie ist z.B. nützlich bei Fragen wie

- Bestimmung des längsten gemeinsamen Präfixes zweier Teilworte x_1 und x_2 von x.
  (LCA von x_1 und x_2 bestimmen (lowest common ancestor))

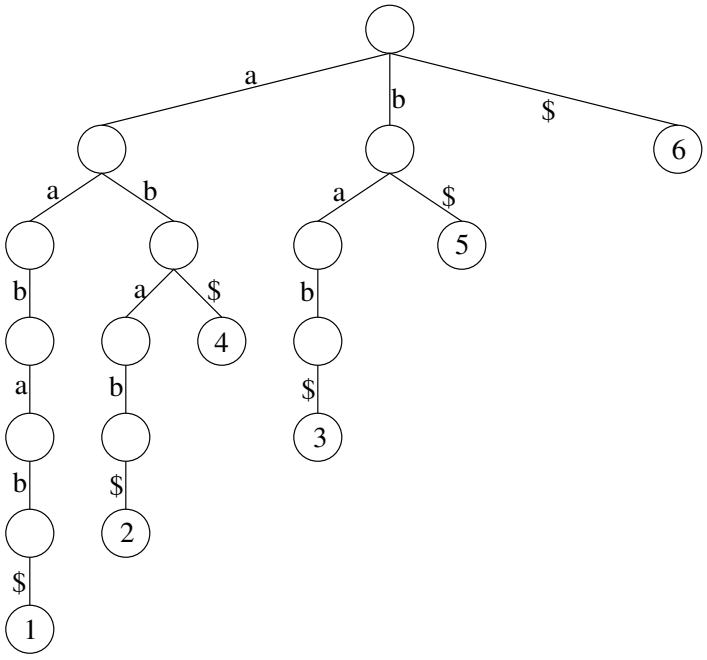
- Prüfung ob y Teilwort von x ist (Suchen in Zeichenketten)
  (verfolgen des Teilwortes)
- Anzahl des Auftretens von y in x
  (Bestimmung der Blätter im Teilbaum mit Wurzel "y"

Nachteile

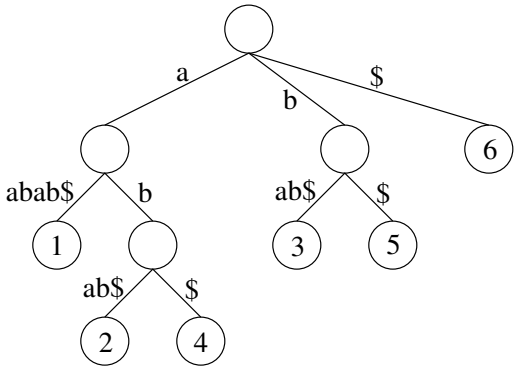
- Die Anzahl der Knoten (und damit auch der Kanten) ist im schlechtesten Fall $O(n^2)$ für $|x| = n$.
  (etwa $a^m b^m$\$, Länge $2m+1$, hat m^2+4m+2 verschiedene Teilwörter)
  $((m+1)(m+1) = m^2+2m+1$ (Alle Teilwörter ohne \$)
  $+(2m+1)$ (Alle Teilwörter mit \$))

(3.7.2002)

Suffix-Trie für $x = aabab$\$

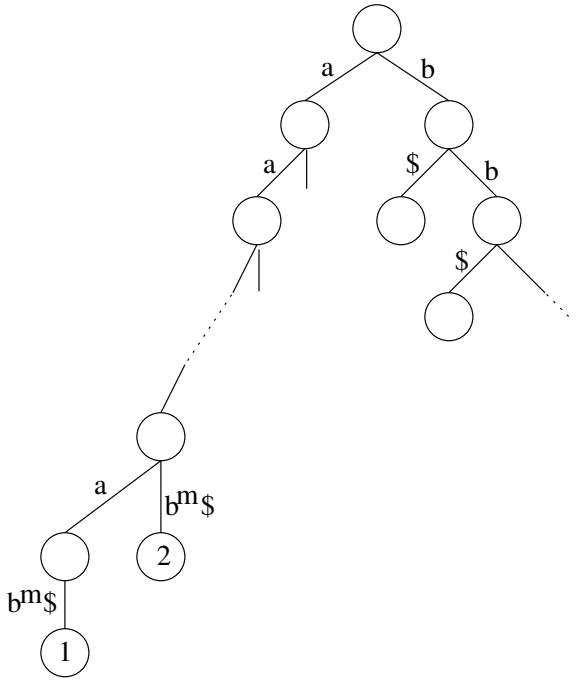


Übergang zum Suffix-Baum:
Zusammenfassen von Wegen, die, bis auf Anfangs- und Endknoten nur aus Knoten mit Verzweigungsgrad 1 bestehen.
Die Kanten sind jetzt mit Teilworten markiert.
Beispiel:



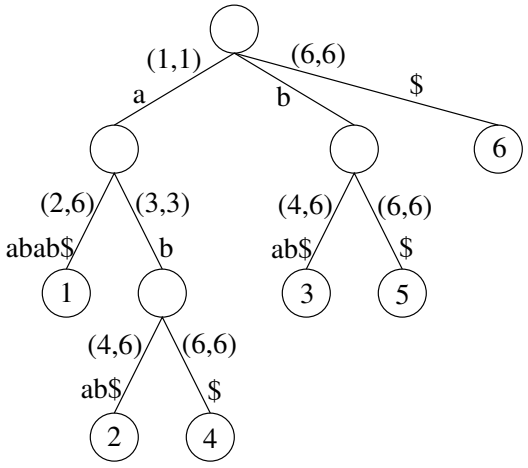
(♡)

$z = a^m b^m \$$



Damit ist noch nichts gewonnen...

Die Teilworte, mit denen die Kanten markiert sind, müssen nicht explizit gespeichert werden - es reichen zwei Indizes für Anfangs- und Endposition in der Zeichenkette. Diese Information kann in den Knoten abgelegt werden !



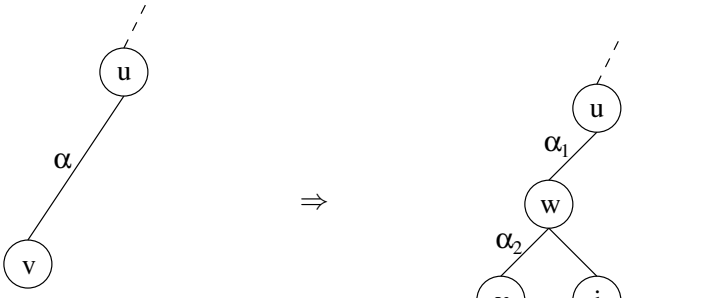
vgl. (♡)

Damit ist der Platzbedarf zur Speicherung eines Suffix-Baumes für eine Zeichenkette x $O(|x|)$.

3.1 Konstruktion von Suffix-Bäumen

Der naive Konstruktions-Algorithmus beginnt mit einem Baum B_0, der nur aus einem Knoten, der Wurzel besteht.
Danach werden nacheinander die Suffixe $x[i..n]$ für $i = 1, \ldots, n$ in den Baum eingefügt; genauer Suffix $x[i..n]$ wird in den Baum B_{i-1} eingefügt und liefert den Baum B_i.
Wie wird eingefügt ?
Man startet an der Wurzel von B_{i-1} und sucht den längsten Pfad, dessen Markierung ein Präfix γ von $x[i..n]$ ist (dieser ist eindeutig bestimmt).

1. Der Pfad endet mitten in einer Kante (u, v).
   Dann wird diese Kante in zwei Kanten aufgebrochen, indem ein neuer Knoten w eingefügt wird.



$\alpha = \alpha_1\alpha_2$ und $x[i..n] = \gamma\beta$
und Knoten u repräsentiert
ein δ mit $\delta\alpha_1 = \gamma$

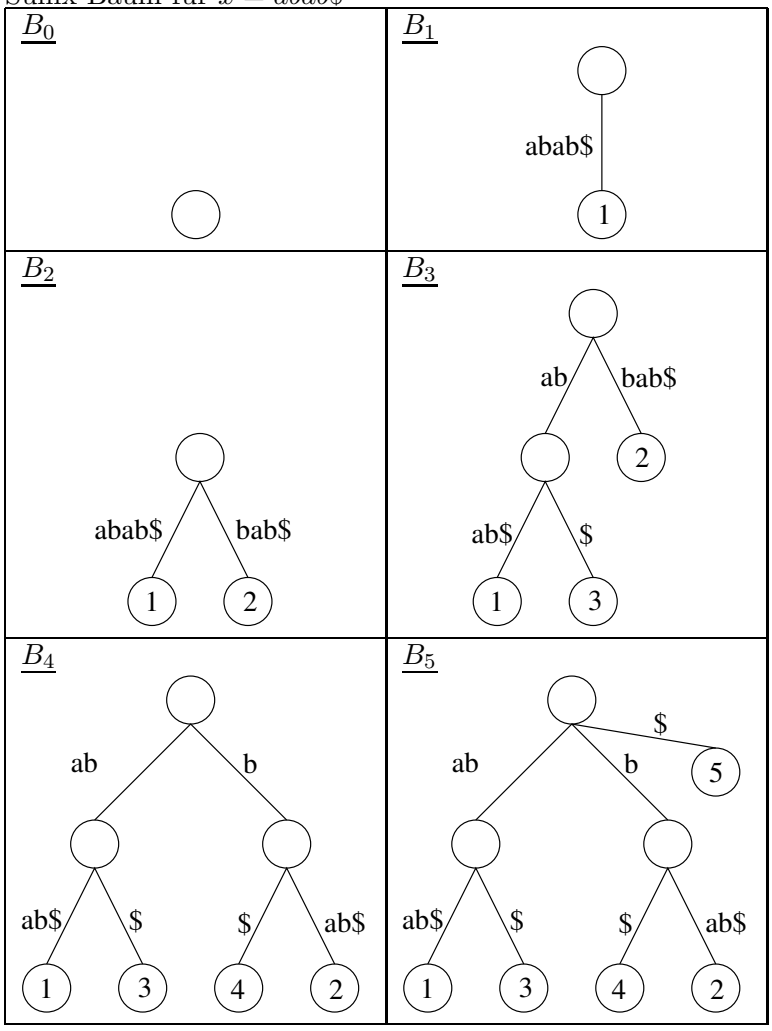
(dabei ist i das neue Blatt)

2. Der Pfad endet in einem Knoten u.

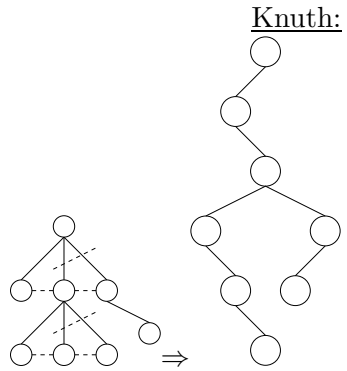
Dann wird eine neue Kante (u, i) zu einem neuen Blatt i eingesetzt, die mit β markiert ist.

Man erhält auf diese Weise einen Suffix-Baum für x. Man benötigt aber $O(n^2)$ Zeiteinheiten (etwa Suffix-Baum für $x = a^n\$$)

Beispiel:
Suffix-Baum für $x = abab\$$

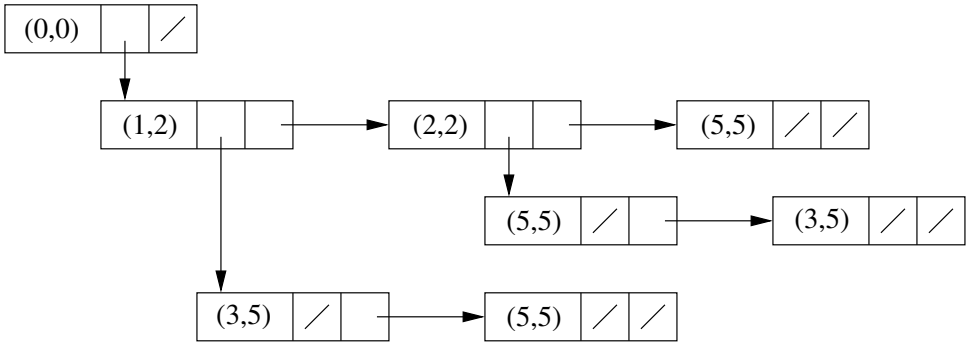


Eine effektive Speicherung eines Suffix-Baumes kann z.B. durch eine Knuth-Transformation in einen binären Baum erreicht werden:



Beispiel:

$\underline{B_5}$



Also Speicherbedarf $O(n)$!

Schreibweisen:

Sei u eine Zeichenkette über Σ.

`knoten(u)` sei der Knoten im Suffix-Baum, der u repräsentiert

`ext(u)`$= \{v | u$ ist der Präfix von $v\}$

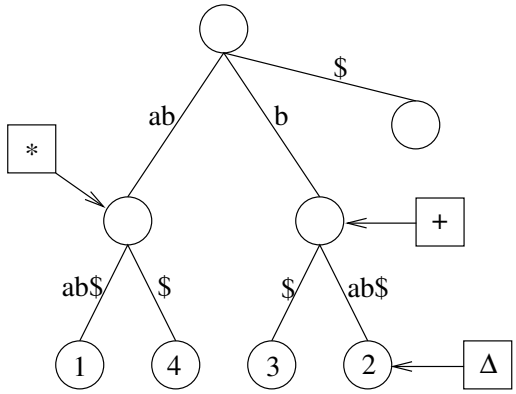
`knoten_min_ext(u)` sei der Knoten im Baum, der das kürzeste Wort aus `ext(u)` repräsentiert

`knoten_max_präfix(u)` sei der Knoten im Baum, der das längste Präfix von u repräsentiert

`head(i)` sei das längste Präfix von $x[i..n]$, das auch Präfix von $x[j..n]$ für $j < i$ ist
(d.h. `head(i)` ist das längste Präfix von $x[i..n]$, für das `knoten_min_ext` in B_{i-1} exisitert)

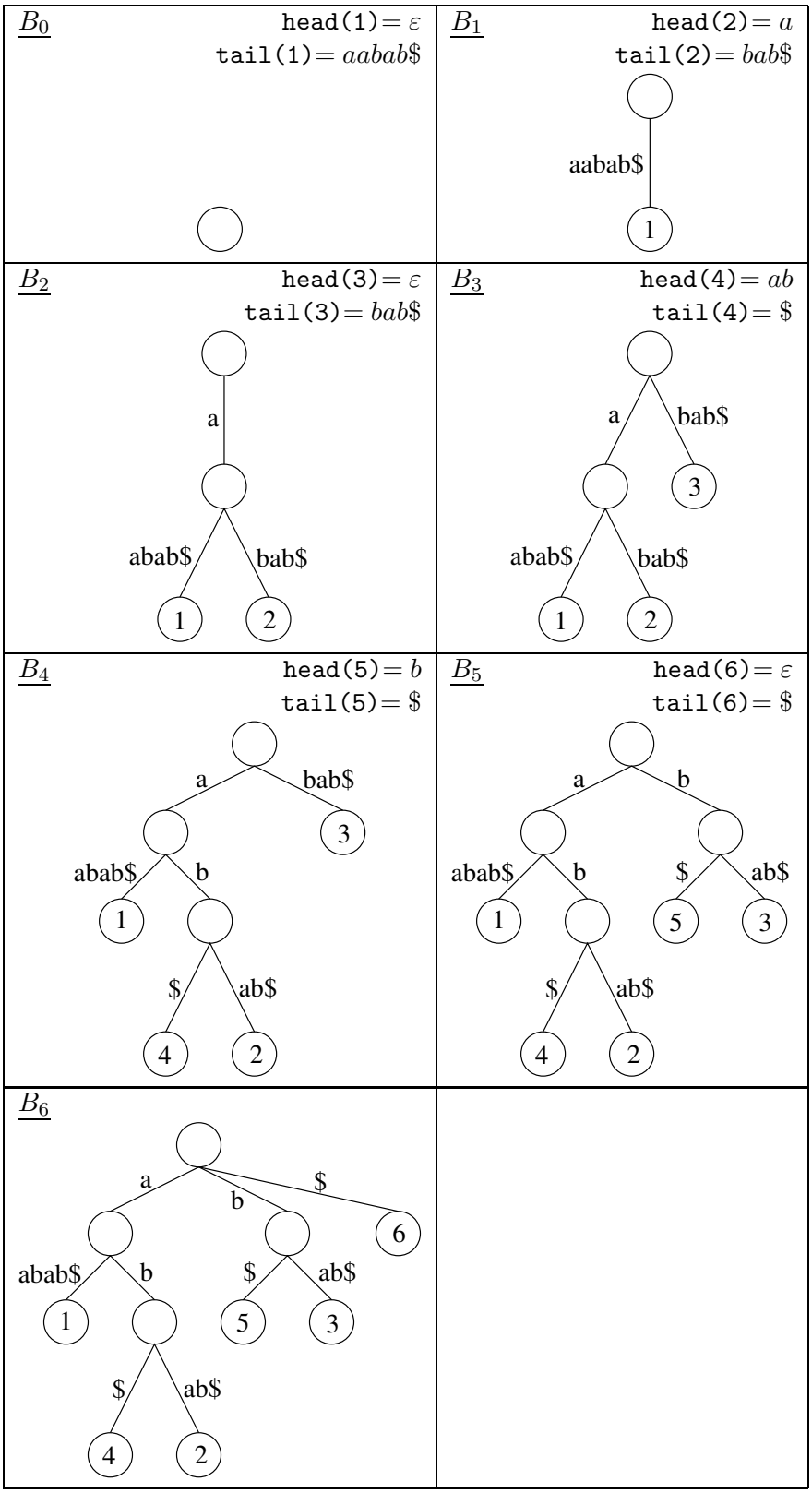
`tail(i)` sei die Zeichenkette, für die $x[i..n] =$`head(i)tail(i)` gilt.
(das \$-Zeichen verhindert `tail(i)`$= \varepsilon$)

Beispiel:



`knoten(a,b)`= *
`knoten(a)` ist undefiniert
`knoten_min_ext(ab)`= *
`knoten_min_ext(a)`= *
`knoten_max_präfix(aba)`= *
`knoten_min_ext(ba)`= Δ
`knoten_max_präfix(b)`= +
`head(3)`= ab
`tail(3)`= $

Der naive Konstruktionsalgorithmus wird nun wie folgt beschrieben:
Sei B_0 der Baum, der nur aus einem Knoten besteht und das leere Wort repräsentiert und sei B_i der Baum, der durch Einfügen von $x[i..n]$ aus B_{i-1} entsteht.
Um diese Einfügung durchzuführen, sucht man zunächst `knoten_min_ext(head(i))` in B_{i-1}.
Repräsentiert dieser Knoten `head(i)`, wird ein neues Blatt i angefügt und die Kante mit `tail(i)` markiert.
Repräentiert der so gefundene Knoten nicht `head(i)`, dann wird die auf ihn zeigende Kante aufgebrochen und ein neuer interner Knoten für `head(i)` eingesetzt. Ein neues Blatt i wird an den eingesetzten Knoten angehängt und die Kante mit `tail(i)` markiert.
Beispiel:
Suffix-Baum für $x = aabab\$$


B0
head(1) = ε
tail(1) = aabab$
B1
head(2) = a
tail(2) = bab$
aabab$
1
B2
head(3) = ε
tail(3) = bab$
a
abab$
bab$
1
2
B3
head(4) = ab
tail(4) = $
a
bab$
3
abab$
bab$
1
2
B4
head(5) = b
tail(5) = $
a
bab$
3
abab$
b
1
$
ab$
4
2
B5
head(6) = ε
tail(6) = $
a
b
abab$
b
$
ab$
1
5
3
$
ab$
4
2
B6
a
b
$
6
abab$
b
$
ab$
1
5
3
$
ab$
4
2

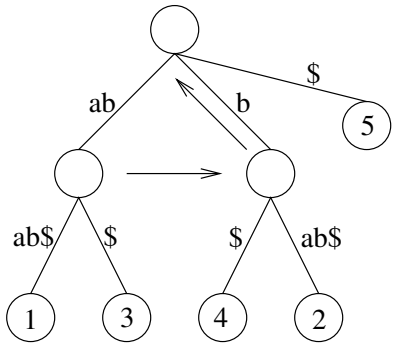
Die Grundlage des Verfahrens von McCreight ist das folgende Lemma:

Lemma 3.1
Wenn `head(i-1)`$= az,\ a \in \Sigma,\ z \in \Sigma^*,\ 1 < i \leq n = |x|$, dann ist z Präfix von `head(i)`

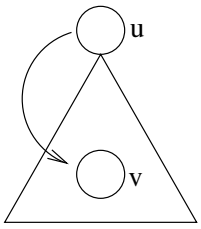
Beweis:
Ist `head(i-1)`$= az$, dann gibt es laut Definition ein $j < i-1$, so daß az Präfix von $x[i-1..n]$ und von $x[j..n]$ ist.
Folglich ist z Präfix von $x[i..n]$ und von $[x[j+1..n]$ und somit Präfix von `head(i)`.

(10.7.2002)

Die Idee von McCreight ist es, den in Entwicklung befindlichen Suffix-Baum für x mit zusätzlichen Zeigern (Suffix-Zeiger) zu versehen. Jeder interne Knoten enthält so einen Zeiger. Vom Knoten, der az repräsentiert ($a \in \Sigma,\ z \in \Sigma^*$) wird ein derartiger Zeiger auf den Knoten verweisen, der z repräsentiert.
Beispiel:
$x = abab\$$



☞ Seite 97
Ein Suffix-Zeiger zeigtz nie in den "eigenen" Teilbaum !



($v \in$ `ext(u)`)

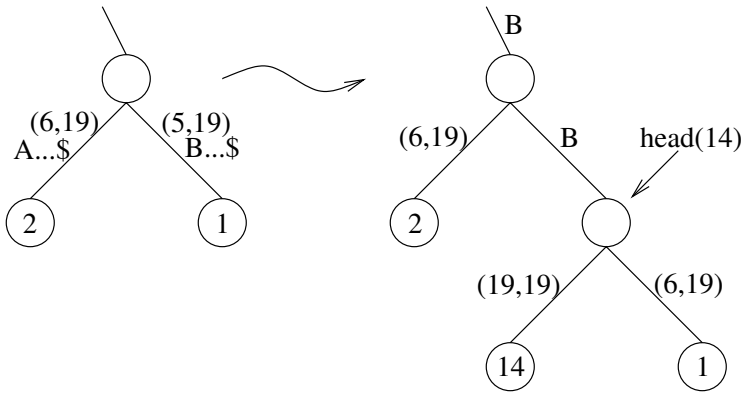
Im Schritt vom Baum B_{i-1} zum Baum B_i ist gegeben: `head(i-1)`
☞ Seite 95 und 97

Beispiel:

$$x = BBBBBAB\underbrace{ABBB}A\overbrace{\underbrace{ABBB}BB\$}^{x[13..19]}$$

Eingetragen werden soll der Suffix $x[14..19] = BBBBB\$$ in B_{13}.
`head(13)`$= ABBB$

`knoten_max_präfix`$(ABBB)$ in B_{12} ist `knoten`(AB)
also $u = A,\ v = B$ und $w = BB$



zu Seite 95, II):
Um diesen Weg zu finden, sucht man die von c ausgehende Kante, die mit einem Wort markiert ist, das mit dem ersten Zeichen von w beginnt. Sei p dieses Wort und sei e der Knoten, auf den diese Kante führt. Ist $|p| < |w|$ wiederholt man das Verfahren rekursiv für das Wort w' mit $w = pw'$ und Startknoten e. Ist $|p| \geq w$ hat man den gesuchten `knoten_min_ext(vw)` gefunden.
hrule
Es gilt:

Satz 3.2
Der Algorithmus von McCreight erzeugt einen Suffix-Baum für eine Zeichenkette x mit $|x| = n$ in einer Zeit $O(n)$ und einem Speicherbedarf $O(n)$.

4 Anwendungen von Suffix-Bäumen

4.1 Das "Longest Repeated Substring Problem"

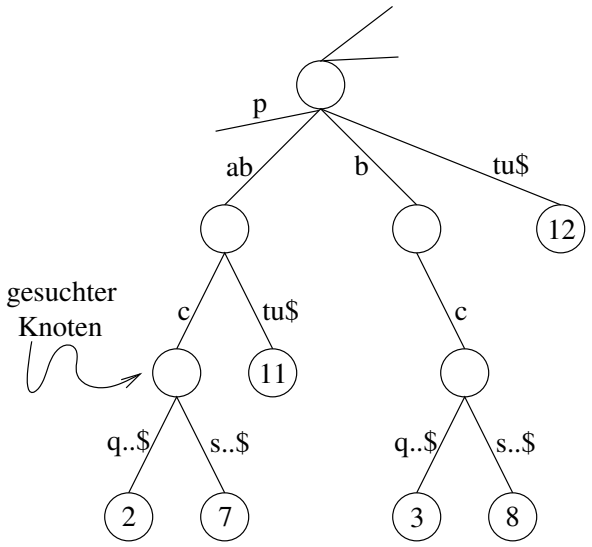
Gegeben: Zeichenkette $y \in \Sigma^*$ mit $|y| = n > 0$. Man bestimme die längste Teilzeichenkette x von y, die an mindestens zwei verschiedenen Positionen auftritt.
Beispiel:
$y = abababa$.
Dann ist $x = ababa$ die gesuchte Zeichenkette, die an Position 1 und 3 in y auftritt.
Beispiel:
$y = pabcqrabcsabtu$
$x = abc$ ist längste Teilzeichenkette, die an Position 2 und 7 auftritt.

Naheliegender Ansatz: Konstruiere $n \times n$ Matrix M mit

$$M_{ij} = \begin{cases} 1 & \text{falls } y[i] = y[j]x \\ 0 & \text{sonst} \end{cases}$$

(eigentlich nur obere (untere) Dreiecksmatrix - M ist symmetrisch)
Man sucht dann maximale Diagonalen in M außerhalb der Hauptdiagonalen.
☞ Seite 94 und 93
Dieser einfache Algorithmus löst das Problem in einer Zeit $O(n^2)$.
Die Elemente von M müssen nicht gespeichert werden $\rightarrow O(n)$.

Effektivere Lösung: Man konstruiert einen Suffix-Baum zu y. Dann muß man nur den internen Knoten finden, der die längste Zeichenkette repräsentiert. Dies ist die längste Teilzeichenkette von y, die mehrfach auftritt.
Man erhält die Antwort in $O(n)$ Zeiteinheiten.
Beispiel:
(partieller) Suffix-Baum für $y = p\,\underbrace{abc}\,qr\,\underbrace{abc}\,sabtu\$$

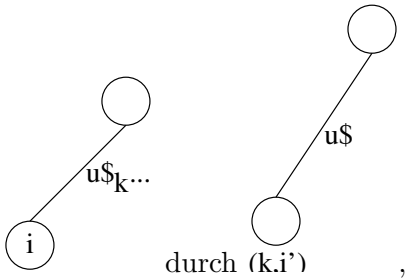


4.2 Das "Longest Common Substring"-Problem

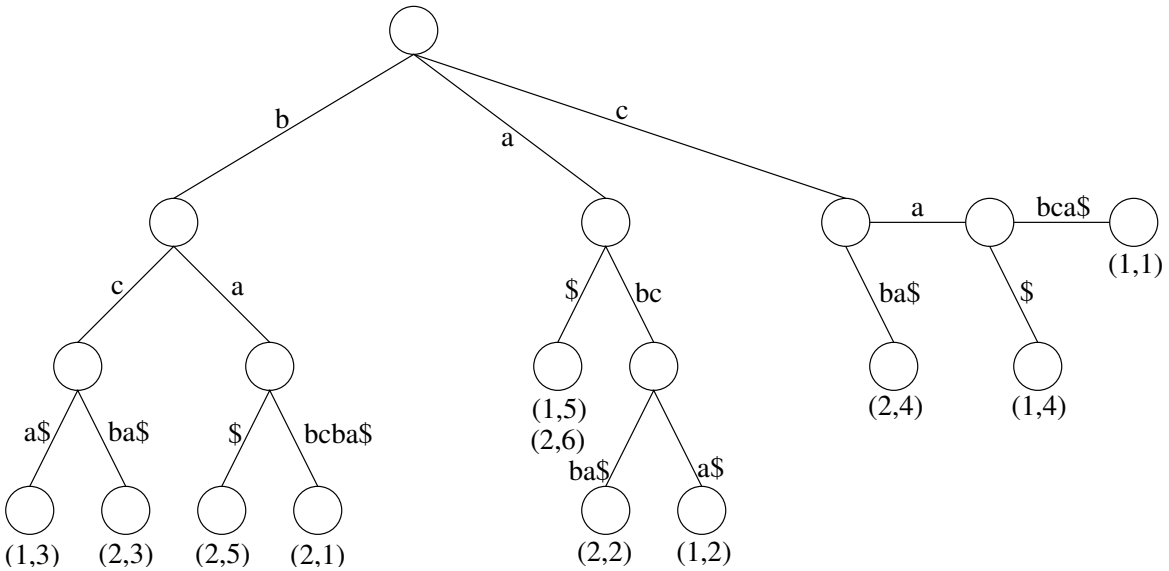
indexProblem!Common Substring Gesucht ist die längste Zeichenkette, die in zwei gegebenen Zeichenketten x und y auftritt.
Lösung: Konstruiere Suffix-Baum für $x\sharp y\$$, wobei $\sharp$ und $\$$ nicht in x oder y auftreten.
Man sucht interne Knoten, die eine längste Zeichenkette repräsentieren und Blätter als Nachfolger besitzen, von denen mindestens eins zu einem Suffix gehört, das vor dem $\sharp$ beginnt und eines, das nach dem $\sharp$ beginnt.

4.3 Verallgemeinerte Suffix-Bäume

repräsentieren die Suffixe einer Menge $\{x_1, \ldots, x_r\}$ von Zeichenketten. Konzeptionell kann man einen derartigen Baum konstruieren, in dem man einen Suffix-Baum für $y = x_1\$_1 x_2\$_2 \cdots x_r\$_r$, $\$_i$ spezielle Endmarkierungen, aufbaut.



Ersetze durch ,
d.h. man streicht die langen "künstlichen" Suffixe in den Markierungen der Kanten, die auf Blätter verweisen, und erweitert die Information in den Blättern um die Angabe aus welchen Zeichenketten das Suffix stammt.
Beispiel:
$x_1 = cabca,\ x_2 = babcba$
Suffix-Baum für $y = x_1\$_1 x_2\$_2$:



Damit ist das Teilstring Problem effektiv lösbar:
Gegeben seien Zeichenketten $x_1, \ldots, x_r$ sowie y.
Gesucht ist die Menge aller i mit: y ist Zeichenkette von x_i.
Konstruiere verallgemeinerten Suffix-Baum und suche nach y (`knoten_min_ext(y)`).
Gibt es einen derartigen Knoten, so betrachte alle Blätter unterhalb des Knotens.
Vorverarbeitung: $O(\sum |x_i|)$ Schritte
Suche nach y: $O(|y|)$ Schritte

5 Aufgaben

UNIVERSITÄT HANNOVER
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
10.04.2002

1. Aufgabenblatt

Aufgabe 1
Seien x, $y \in \Sigma^*$ mit $|x| = n$ und $|y| = m$ gegeben. Man beweise, dass für die Menge $A(x, y)$ der Ausrichtungen von x und y gilt:

$$|A(x,y)| = \sum_{l=\max\{n,m\}}^{n+m} \frac{l!}{(l-n)!(l-m)!(n+m-l)!}$$

Aufgabe 2
Seien x, $y \in \Sigma^*$ mit $|x| = n$ und $|y| = m$. $A(x, y)$ sei die Menge der Ausrichtungen von x und y und es sei $|A(x, y)| = f(n, m)$. Man beweise die folgende Rekusion:

$$f(n,m) = \begin{cases} 1 & \text{falls } n = 0 \text{ oder } m = 0 \\ (f(n-1,m) + f(n-1,m-1) + f(n,m-1) & \text{sonst} \end{cases}$$

Aufgabe 3
Gegeben sei eine Programmiersprache, in der eine Integerarithmetik mit beliebiger Genauigkeit implementiert ist (z.B. Scheme). Konstruieren Sie ein Programm zur Berechnung von $|A(x, y)|$, dessen Zeit- und Speicherplatzbedarf möglichst klein ist. Von welcher Zeitkomplexität ist Ihre Lösung? Wie groß ist der Speicherplatzbedarf? Wieviel Dezimalstellen hat $|A(x, y)|$ für $|x| = |y| = 100$?

Aufgabe 4
Seien x, $y \in \Sigma^*$ mit $|x| = n$ und $|y| = m$ gegeben. Man versuche wie in den Aufgaben 1 oder 2 für die Menge $T(x, y)$ der Korrespondenzen entsprechende Anzahlformeln zu finden.

Aufgabe 5
Sei $T = (\mathbb{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ ein Halbring, wobei $\cdot$ das Produkt von Zeichenkettenmengen bezeichnet. Sei w die durch

$$w(a,b) = \begin{cases} \{a\} & \text{falls } a = b \\ \varepsilon & \text{sonst} \end{cases}$$

gegebene elementare Distanzfunktion auf $\tilde{\Sigma}$ und D die dadurch gegebene abstrakte Distanzfunktion. Man beschreibe für $x, y \in \Sigma^*$ die Menge $D(x, y)$.

ABGABETERMIN: Mittwoch, 17.04.2001 in der Vorlesung.

UNIVERSITÄT HANNOVER
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
17.04.2002

2. Aufgabenblatt

Aufgabe 1
Man berechne die Länge einer längsten gemeinsamen Subsequenz der beiden Zeichenketten $x = cbabacaccbabac$ und $y = abcabbabaabcababbba$. Geben Sie eine längste gemeinsame Subsequenz an.

Aufgabe 2
Für die beiden Zeichenketten aus der vorigen Aufgabe bestimme man eine kürzeste gemeinsame Supersequenz.

Aufgabe 3
Man beweise:
Ist die elementare Distanzfunktion $w : \tilde{\Sigma} \times \tilde{\Sigma} \rightarrow \mathbb{R}$ eine Metrik, so ist die allgemeine Levenshtein Distanz d_L^a ebenfalls eine Metrik.

ABGABETERMIN: Mittwoch, 17.04.2001 in der Vorlesung.

UNIVERSITÄT HANNOVER
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
2.05.2002

3. Aufgabenblatt

Aufgabe 1
Es seien $x, y \in \Sigma^*$ mit $|x| = n$ und $|y| = m$. Ferner sei $d_{i,j} = d_L(x[1..i], y[1..j])$, $1 \leq i \leq n$, $1 \leq j \leq m$ der Abstand zwischen den beiden Präfixen $x[1..i]$ und $y[1..j]$ von x und y bezüglich der einfachen Levenshtein-Metrik. Man beweise, dass dann

$$d_{i,j} \in \{d_{i-1,j-1}, d_{i-1,j-1} + 1\}$$

gilt.

Aufgabe 2
Man spezialisiere das rekursive Verfahren zur Berechnung der Lückenkosten aus Satz 1.18 für den Fall affiner Lückenkosten und zeige, dass das Verfahren dann nur noch eine Zeitkomplexität von $O(nm)$ hat. Dabei sei die affine Lückenkostenfunktion durch

$$w_k = w_{-k} = g + h(|k| - 1) \text{ mit } g, h \in \mathbb{R}^+, k > 0$$

gegeben.

Aufgabe 3
Für $x = ywcqpgk$ und $y = lawyqqkpgka$ bestimme man den Abstand für eine affinen Lückenkostenfunktion mit $g = 3$ und $h = 1$. Es gelte außerdem:

$$w(a, b) = \begin{cases} 0 & \text{falls } a = b \\ 3 & \text{sonst} \end{cases}$$

Geben sie zwei Ausrichtungen von x und y an, die bezüglich dieses Abstandsbegriffs optimal sind.

ABGABETERMIN: Mittwoch, 8.05.2001 in der Vorlesung.

UNIVERSITÄT HANNOVER
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
8.05.2002

4. Aufgabenblatt

Aufgabe 1
Man beweise:
Es seien $x, y \in \Sigma^*$ mit $|x| = n$ und $|y| = m$. Ferner sei für $1 \leq i \leq n$

$$k_i = \min\{d_L^a(x[1..i], y[1..j]) + d_L^a(x[i+1..n], y[j+1..m]), 1 \leq j \leq m\}.$$

Dann gilt $k_i = d_L^a(x, y)$ für ein $i, 1 \leq i \leq n$.

Aufgabe 2
Man ändere den Algorithmus von Wagner und Fischer, so dass am Ende nicht eine optimale Korrespondenz, sondern eine optimale Ausrichtung bezüglich des verwendeten Distanzbegriffs ausgegeben wird.

Aufgabe 3
Man entwickle einen Algorithmus, der aus der Distanzmatrix für zwei Zeichenketten x und y aus Σ^* **alle** optimalen Ausrichtungen bezüglich der allgemeinen Levenshtein Distanz ausgibt.

ABGABETERMIN: Mittwoch, 15.05.2001 in der Vorlesung.

UNIVERSITÄT HANNOVER
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
29.05.2002

5. Aufgabenblatt

Das *local assignment problem* ist aus biologischer Sicht besonders wichtig. Man sucht in den gegebenen zwei Zeichenketten x und y Teilzeichenketten, die möglichst ähnlich sind. Die üblichen Abstandsbegriffe kann man in dieser Situation nicht so richtig verwenden, denn wählt man jeweils die leere Zeichenkette als Teilzeichenkette, so erhält man bereits den Abstand 0.
Besser ist es, die elementare Gewichtsfunktion so zu wählen, dass $w(a,a) > 0$ und $w(a,b) < 0$ für $a \neq b$ ist und mit einer Distanzfunktion über dem Halbring $(\mathbb{R}, \max, +, 0, 0)$ zu arbeiten.
Abstrakt kann man das Problem etwa wie folgt beschreiben:
Es seien $x, y \in \Sigma^*$ mit $|x| = m$ und $|y| = n$. Ferner sei $w : \tilde{\Sigma} \times \tilde{\Sigma} \to \mathbb{R}$ eine elementare Gewichtsfunktion. Gesucht sind Teilwörter $\tilde{x}$ und $\tilde{y}$ von x bzw. y, für die der Abstand $D(\tilde{x}, \tilde{y})$ maximal ist. Sei $V(x,y)$ dieser maximal Wert.

Beispiel: Sei $x =$ *pqraxabcstvq* und $y =$ *deaxbacsll* und sei $w(a,a) = 2$, $w(a,b) = -2$ für $a \neq b$ und $w(a,\varepsilon) = w(\varepsilon,a) = -1$. Dann ist $\begin{smallmatrix} \text{a} & \text{x} & \text{a} & \text{b} & - & \text{c} & \text{s} \\ \text{a} & \text{x} & - & \text{b} & \text{a} & \text{c} & \text{s} \end{smallmatrix}$ eine Ausrichtuing der Teilzeichenketten $\tilde{x} =$ *axabcs* und $\tilde{y} =$ *axbacs* mit $D(\tilde{x}, \tilde{y}) = 8$ und für alle anderen Teilzeichenketten erhält man keine größeren Werte. Also ist $V(x,y) = 8$.

Aufgabe 1
Seien nun $i \leq m$ und $j \leq n$ und sei $v(i,j) = \max\{D(\alpha,\beta) \,|\, \alpha$ Suffix von $x[1..i]$, β Suffix von $y[1..j]\}$. Es gilt offensichtlich $v(i,j) \geq 0$, da $\alpha = \beta = \varepsilon$ möglich ist.
Man beweise: $V(x,y) = \max\{v(i,j) \,|\, 0 \leq i \leq m,\, 0 \leq j \leq n\}$.

Aufgabe 2
Man beweise: Für $i > 0$, $j > 0$ gilt

$$v(i,j) = max\{0, v(i-1,j-1) + w(x[i], y[j]), v(i-1,j) + w(x[i], \varepsilon), v(i,j-1) + w(\varepsilon, y[j])\}$$

Aufgabe 3
Für die elementare Gewichtsfunktion $w(a,a) = 1, w(a,b) = -3$ für $a \neq b$ und $w(a,\varepsilon) = w(\varepsilon,a) = -1$ bestimme man $V($*zwcqpgk*, *lawzqqkpgka*$)$. Wie lauten die beiden Teilzeichenketten größter Ähnlichkeit?

Aufgabe 4
Man entwerfe einen Algorithmus, der für zwei Zeichenketten x und y den Wert $V(x,y)$ berechnet und Teilzeichenketten größter Ähnlichkeit von x und y ausgibt.

ABGABETERMIN: Mittwoch, 5.06.2001 in der Vorlesung.

Universität Hannover
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
12.06.2002

6. Aufgabenblatt

Aufgabe 1
Entwerfen Sie einen Algorithmus, der aus der Diagonalenmatrix L für zwei Zeichenketten $x, y \in \Sigma^*$ mit $|x| = m$ und $|y| = n$ eine Ausrichtung konstruiert. Dazu überlege man sich zunächst, wie man aus L die Distanzmatrix D rekonstruieren könnte.

Aufgabe 2
Man wende das Verfahren von Sellers zur Lösung des k-Differenzen Problems auf die beiden Zeichenketten x^R und y^R an. Welche Bedeutung haben jetzt Einträge $\leq k$ in der letzten Zeile der Distanzmatrix?

Aufgabe 3
Man übertrage das Verfahren von Sellers auf die Diagonalenmatrix und modifiziere den Algorithmus zur Berechnung der Diagonalenmatrix, um das k-Differenzen Problem zu lösen.

Abgabetermin: Mittwoch, 19.06.2001 in der Vorlesung.

UNIVERSITÄT HANNOVER
Institut für Informatik
Prof. Dr. R. Parchmann

Approximative
Zeichenkettensuche
SS 2002
19.06.2002

7. Aufgabenblatt

Aufgabe 1
Modifizieren Sie den in der Vorlesung angegebenen Algorithmus von Chang und Lampe, so dass die Cut-Off-Methode des Jokinen, Tarhio, Ukkonen Algorithmus implementiert wird, also die Spalten der Distanzmatrix nicht vollständig entwickelt werden.

Aufgabe 2
Für zwei Zeichenketten x und $y \in \Sigma^*$ mit $|x| = m$ und $|y| = n$ besteht das lokale Assignment Problem darin, zwei Teilzeichenketten von x und y zu finden, die bezüglich eines vorgegebenen Ähnlichkeitsbegriffs am ähnlichsten sind (siehe Aufgabenblatt 5). Ein damit zusammenhängendes Problem ist es, zwei Teilzeichenketten x_1 und x_2 in *einer* Zeichenkette zu finden, die eine hohe Ähnlichkeit haben (*inexakt repeated substring problem*).
Es liegt daher nahe, um angenäherte Wiederholungen in einer Zeichenkette x zu finden, lokale Assignments der Zeichenkette mit sich selbst zu suchen. Leider ist in diesem Fall die Zeichenkette x eine beste Lösung. Selbst wenn man alle Werte in der Tabelle nutzt, wird der Wert $d_{i,j}$ mit $i \neq j$ stark durch die Werte in der Diagonalen beeinflusst. Gibt es eine einfache Lösung dieses Problems?
Man teste das gefundene Verfahren und suche nach den angenäherten Wiederholungen in der Zeichenkette $x =$ *pqraxabcstvqdeaxbacsll*. Dabei sei die elementare Gewichtsfunktion gegeben durch $w(a,a) = 2$, $w(a,b) = -2$ für $a \neq b$ und $w(a,\varepsilon) = w(\varepsilon,a) = -1$.

ABGABETERMIN: Mittwoch, 26.06.2002 in der Vorlesung.

6 Lösungen der Aufgaben

6.1 1. Hausübung

Aufgabe 1.1
Es ist $|x| = n$ und $|y| = m$.
Sei $(x', y') \in A(x, y), (x', y')$ habe die Länge l.
Da $(x'[i], y'[i]) \neq (-, -)$ für $i \in [1 : l]$ folgt $l \in [max\{n, mi\} : n + m]$.
x' enthält $k = l - n$ neutrale Zeichen.
Es gilt $k \in [max\{o, m - n\} : m]$.
Auf die $l = n + k$ Positionen in $\overline{x}$ können die k neutralen Zeichen auf $\binom{l}{k} = \binom{l}{l-n}$ Arten verteilt werden.

y' enthält $l - m = n - m + k$ neutrale Zeichen. Diese dürfen nicht "unter" neutralen Zeichen in x' stehen. Also haben wir $l - k = n$ freie Positionen für $l - m$ neutrale Zeichen.
Diese kann man auf $\binom{n}{l-m}$ Arten verteilen.
Also gibt es $\binom{l}{l-n}\binom{n}{l-m} = \frac{l!n!}{(l-n)!n!(l-m)!(n-l+m)!}$ Ausrichtungen von x und y der Länge l.

Aufgabe 1.2
Ist $n = 0$ oder $m = 0$, so ist $x = \varepsilon$ oder $y = \varepsilon$, folglich ist $|A(x, y)| = 1$. Seien jetzt $n > 0$ und $m > 0$. Für jede Ausrichtung $(x', y' \in A(x, y)$ der Länge l gilt entweder

1. $x'[l] = -$ und $y'[l] = y[m]$ oder
2. $x'[l] = x[n]$ und $y'[l] = -$ oder
3. $x'[l] = x[n]$ und $y'[l] = y[m]$

Damit erhält man alle Ausrichtungen von x und y rekursiv aus den Ausrichtungen von x und $y[1..m - 1], x[1..n - 1]$ und y und $x[1..n - 1]$ und $y[1..m - 1]$.

Aufgabe 1.3
$|A(x, y)|$, $f(100, 100) = \underbrace{2053 \ldots 041}_{76\ Stellen}$
Entweder die Formel aus 1) benutzen, oder:
Die rekursive Form aus Aufgabe 2 benutzen, dann sollten jedoch alle bereits berechneten Werte in einer Tabelle zwischen gespeichert werden, um doppelte Berechnungen zu vermeiden.

Auf diese Weise ergibt sich:
Zeitkomplexität: $O(|x| * |y|)$
Speicherplatzkomplexität: $O(|x| + |y|)$

Aufgabe 1.4
Sei $|x| = n, |y| = m$. Jede Korrespondenz τ_{xy} zwischen x und y enthält zwischen 0 und $min\{|x|, |y|\}$ Elemente.
Sei l die Anzahl der Elemente einer Korrespondenz. Offensichtlich hat man $\binom{n}{l}$ Möglichkeiten, die l Zeichen in x auszuwählen, die mit l Zeichen in y in Korrespondenz stehen. Also

$$|T(x,y)| = \sum_{l=0}^{min\{|x|,|y|\}} \binom{n}{l}\binom{m}{l}$$

Für $|x| = |y| = 100$ ergibt sich $|T(x,y)| = \underbrace{905\ldots}_{59\ Stellen}$.

Aufgabe 1.5
$T = (\mathbb{P}(\Sigma^*), \cup, *, \emptyset, \{\varepsilon\})$ $w(a,b) = \begin{cases} \{a\} & a = b \\ \{\varepsilon\} & \text{sonst} \end{cases}$

$$D(x,y) = \begin{cases} \{\varepsilon\} & \text{falls } x = y = \varepsilon \\ \bigcup_{A(x,y)} \underbrace{\prod_{i=1}^{|x'|} w(x'[i], y'[i])}_{*} & \text{sonst} \end{cases}$$

$(*)$ ist eine ein-elementige Menge, die eine Zeichenkette enthält, die Subsequenz von x und y ist.
Also ist $D(x,y)$ die Menge der gemeinsamen Subsequenzen von x und y.

6.2 2. Hausübung

Aufgabe 2.1
$x = cbabacaccbabac$
$y = abcabbabaabcababbba$
$lcs(x,y) = 11$
Eine längste gemeinsame Subsequenz ist etwa *cbabacababa*.
Wie ist dies aus der Tabelle direkt abzulesen ?

- rechts unten beginnen
- rückwärts schauen auf welchem Weg man zu diesem Wert gekommen ist (woher kam dies Maximum?)
- Änderungen zwischen den Elementen ?
- Daraus Korrespondenzen bilden
- $\Rightarrow lcs$

Aufgabe 2.2
$scs(x,y) = |x| + |y| - lcs(x,y) = 14 + 19 - 11 = 22$
Eine Supersequenz wäre etwa
abcabbabaabcaccbabbbac

Aufgabe 2.3
Sei $w : \tilde{\Sigma} \times \tilde{\Sigma} \rightarrow \mathbb{R}$ elementare Distanzfunktion. Ist w Metrik, dann ist d_L^a eine Metrik.

Beweis:
Es ist für $x, y \in \Sigma^*$ $d_L^a(x,y) = min_{(x',y') \in A(x,y)} \sum_{i=1}^{|x'|} w(x'[i], y'[i])$.

1. $d_L^a(x,y) \geq 0 \quad \surd$

2. Sei $x = y$. Dann ist $(x,y) \in A(x,y)$ und $d_L^a(x,y) \leq \sum_{i=1}^{|x|} w(x[i], y[i]) = 0$.
   Ist $d_L^a(x,y) = 0$, dann gibt es eine Ausrichtung $(x', y') \in A(x,y)$ mit $\sum_{i=1}^{|x'|} w(x'[i], y'[i]) = 0$. Also $w(x'[i], y'[i]) = 0$ für alle i, d.h. $x = y$.

3. **Behauptung:** $d_L^a(x,y) \leq d_L^a(x,z) + d_L^a(z,y)$
   Es gibt $(x'', z'') \in A(x,z)$ mit $d_L^a(x,z) = \sum_{i=1}^{|x''|} w(x''[i], z''[i])$ und es gibt $(\hat{z}, \hat{y}) \in A(z,y)$ mit $d_L^a(z,y) = \sum_{i=1}^{|\hat{z}|} w(\hat{z}[i], \hat{y}[i])$.
   Man konstruiert eine kürzeste Supersequenz z' von z'' und $\hat{z}$ mit $\mu(z') = \mu(z'') = \mu(\hat{z}) = z$ durch Einfügen von "-".

   Beispiel:
   $z'' = a - b - cd$, $\hat{z} = -a - -bcd-$, dann $z' = -a - -b - cd-$
   Man füllt x'' bzw. $\hat{y}$ an den entsprechenden Positionen ebenfalls mit "-" auf und erhält x' und y'. Wegen $w(-,-) = 0$ (Metrik-Eigenschaft) gilt:
   $\sum_{i=1}^{|x''|} w(x''[i], z''[i]) = \sum_{i=1}^{|z'|} w(x'[i], z'[i])$ und entsprechend
   $\sum_{i=1}^{|\hat{z}|} w(\hat{z}[i], \hat{y}[i]) = \sum_{i=1}^{|z'|} w(z'[i], y'[i])$.
   Damit ist $d_L^a(x,z) + d_L^a(z,y) = \sum_{i=1}^{|z'|} w(x'[i], z'[i]) + \sum_{i=1}^{|z'|} w(z'[i], y'[i]) \geq \sum_{i=1}^{|z'|} w(x'[i], y'[i])$
   Streicht man in x' und y' korrespondierende Positionen j mit $x'[j] = y'[j] = -$, dann erhält man eine Ausrichtung von x und y. Also folgt die Behauptung, denn $\sum_{i=1}^{|x'|} w(x'[i], y'[i]) \geq d_L^a(x,y)$.

Tip(p) zum nächsten Aufgabenblatt, Aufgabe 2:
$\delta(x[1..i], y[1..j]) = min\{I_{ij}, D_{ij}, r\}$
$r = \delta(x[1..i-1], y[1..j-1]) + w(x[i], y[j])$
$I_{ij} = min\{\delta(x[1..i], y[1..j-k]) + w_k,\ k \in [1..j]\}$
$D_{ij} = min\{\delta(x[1..i-k], y[1..j]) + w_{-k},\ k \in [1..i]\}$
Tip(p):
I_{ij}, D_{ij} über Tabellen bestimmen und rekursive Formeln finden.
I_{ij}, D_{ij} müssen von k unabhängig werden !

6.3 3. Hausübung

Aufgabe 3.1
$d_{ij} \in \{d_{i-1,j-1}, d_{i-1,j-1} + 1\}$
Beweis:
Es ist $w(a,b) = \left\{ \begin{array}{ll} 1 & \text{für } a = b \\ 0 & \text{sonst} \end{array} \right\}$ mit $a, b \in \tilde{\Sigma}$
und es ist $d_{ij} = min\{d_{i-1,j} + 1, d_{i,j-1} + 1, d_{i-1,j-1} + w(x[i], y[j])\}$ und damit auch

$d_{ij} \leq d_{i-1,j-1} + 1$.
über vollständige Induktion über $k = i + j$ bleibt zu zeigen: $d_{i-1,j-1} \leq d_{ij}$:
Für $i = 1, j = 1$ ist $D_{i-1,j-1} = 0$ und $d_{i-1,j} = d_{i,j-1} = 1$ also ist $d_{ij} = 0$ oder $d_{ij=1}$.
Also $d_{i-1,j-1} \leq d_{ij}$.
Gelte $d_{i-1,j-1} \leq d_{ij}$ für alle $i + j \leq k$ und sei $i + j = k + 1$.
Ist $d_{ij} = d_{i-1,j-1} + w(x[i], y[j])$, dann ist $d_{i-1,j-1} \leq d_{ij}$.
Ist $d_{ij} = d_{i-1,j} + 1$, so ist laut Induktionsvoraussetzung $d_{i-2,j-1} \leq d_{i-1,j}$, also $d_{ij} \geq d_{i-2,j-1} + 1$.
Nach der Rekusionsformel ist $d_{i-2,j-1} + 1 \geq d_{i-1,j-1}$, also $d_{i-1,j-1} \leq d_{ij}$.
Ist $d_{ij} = d_{i,j-1} + 1$, wird entsprechend geschlossen.

Aufgabe 3.2
Laut Satz 1.17 (☞ Seite 16) ist $\delta(x[1..i], y[1..j]) = min\{I_{ij}, D_{ij}, r\}$ mit

$$r = \delta(x[1..i-1], y[1..j-1]) + w(x[i], y[j])$$
$$I_{ij} = min\{\delta(x[1..i], y[1..j-k]) + w_k, \quad k \in [1..j]\}$$
$$D_{ij} = min\{\delta(x[1..i-k], y[1..j]) + w_{-k}, \quad k \in [1..i]\}$$

Was passiert bei affinen Lückenkosten, d.h. $w_k = w_{-k} = g + h(k-1), \quad g, h \in R_{>0}$?
Es ist $I_{ij} = min\{\delta(x[1..i], y[1..j-1]) + g, \delta(x[1..i], y[1..j-k]) + g + h(k-1), k \in [2..j]\}$
$= min\{\delta(x[1..i], y[1..j-1] + g, min(\delta(x[1..i], y[1..(j-1)-k]) + g + h(k-1), k \in [1..j-1]) + h\}$
Also gilt $I_{i,j} = min\{\delta(x[1..i], y[1..j-1]) + g, I_{i,j-1} + h\}$ für $i, j \geq 1$, falls man $I_{i0} = \infty$ setzt.
Entsprechend zeigt man:

$$D_{ij} = min\{\delta(x[1..i-1], y[1..j]) + g, D_{i-1,j} + h\}$$

mit $D_{0,j} = \infty$.
Diese Werte können wieder tabelliert werden und der Algorithmus benötigt $O(nm)$ Zeiteinheiten und $O(n + m)$ Speicherplätze.
(Speicherung von jeweils einer Zeile oder Spalte der Mtrizen I, D, δ)
☞ Seite 71:
$-- = \infty$

Beispiel:
$i = 4, j = 5$

$$I_{45} = min\{\overbrace{d_{44} + 3}^{13}, \overbrace{I_{44} + 1}^{13}\} = 13$$
$$D_{45} = min\{\underbrace{d_{35} + 3}_{14}, \underbrace{d_{35} + 1}_{12}\} = 12$$
$$d_{45} = min\{\underbrace{I_{45}}_{13}, \underbrace{D_{45}}_{12}, \underbrace{d_{34} + 0}_{9}\} = 9$$

Die optimalen Ausrichtungen haben jeweils die Kosten 16
(Lücke aufmachen:3, Lücke "verlängern":1, Substitution: 3)

Aufgabe 3.3
$w_k = 3 + (k-1) = w_{-k}$, Substitutionskosten $w(a, b) = \begin{cases} 3 & \text{falls } a \neq b \\ 0 & \text{sonst} \end{cases}$

Hinweis:
Das Berechnen optimaler Ausrichtungen ist nicht (einfach) durch Rückverfolgen der Minima möglich.
Man erhielte sonst etwa:

$-$	$-$	$-$	y	w	c	q	$-$	p	g	k	$-$
l	a	w	y	$-$	q	q	k	p	g	k	a
3	1	1		3	3		3				3

Kosten: $5(=3+1+1)+3+3+3+3=17>16$ also nicht optimal !

6.4 4. Hausübung

Aufgabe 4.1
$x, y \in \Sigma^*,\ |x|=n,\ |y|=m$. Es ist für alle $0 \le i \le n$

$$k_i = min\{d_L^a(x[1..i], y[1..j]) + d_L^a(x[i+1..n], y[j+1..m])\ 0 \le j \le m\}$$

Dann gilt: $k_i = d_L^a(x,y)$ für $1 \le i \le n$.
Beweis:
Sei w die allgemeine elementare Distanzfunktion.

1. Sei $j \in [0..n]$ so gewählt, daß
   $$k_i = d_L^a(x[1..i], y[1..j]) + d_L^a(x[i+1..n], y[j+1..m])$$
   Sei (x', y') optimale Ausrichtung für $x[1..i]$ und $y[1..j]$
   und (x'', y'') optimale Ausrichtung für $x[i+1..n]$ und $y[j+1..m]$.
   Dann ist $(x'x'', y'y'')$ Ausrichtung für x und y und es gilt:
   $$k_i = \sum_{i=1}^{|x'|} w(x'[i], y'[i]) + \sum_{i=1}^{|x''|} w(x''[i], y''[i]) \ge d_L^a(x,y)$$

2. Sei (x', y') optimale Ausrichtung für x und y und sei $|x'| = r$.
   Dann existiert ein $s \in [1..r]$ mit $\mu(x'[1..s]) = x[1..i]$ und $\mu(x'[s+1..r]) = x[i+1..n]$.
   Damit ist ein $j \in [0:m]$ bestimmt mit $\mu(y'[1..s]) = y[1..j]$ und $\mu(y'[s+1..r]) = y[j+1..m]$.
   Also ist $(x'[1..s], y'[1..s])$ eine Ausrichtung von $x[1..i]$ und $y[1..j]$, und $(x'[s+1..r], y'[s+1..r])$ Ausrichtung von $x[i+1..n]$ und $y[j+1..m]$.
   Es gilt:
   $k_i \le d_L^a(x[1..i], y[1..j] + d_L^a(x[i+1..n], y[j+1..m])$
   $\le \sum_{l=1}^{s} w(x'[l], y'[l]) + \sum_{l=s+1}^{r} w(x'[l], y'[l])$
   $= d_L^a(x,y)$

Aufgabe 4.2
Seien a und b zwei Felder von Zeichen, in denen die Ausrichtungen für x und y in umgekehrter Reihenfolge abgespeichert werden. `print_anordnung` druckt die beiden Felder "rückwärts" aus.

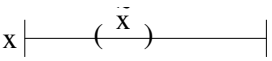
```
anordnung(i,j,r) {
  if i>0 then
    if j > 0 then {
      if (d[i,j]=d[i-1.j-1]+w(x[i],y[j])) then {
        a[r]=x[i];
```

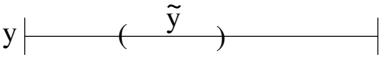
```
        b[r]=y[j];
        anordnung(i-1,j-1,r+1)
      }
      [else /* fuer die Berechnung EINER Anordnung */]
      if(d[i,j]=d[i-1,j]+w(x[i],-)) then {
        a[r]=x[i];
        b[r]='-';
        anordnung(i-1,j,r+1);
      }
      [else /* fuer die Berechnung EINER Anordnung */]
      if(d[i,j]=d[i,j-1]+w(-,y[j])) then {
        a[r]='-';
        b[r]=y[j];
        anordnung(i,j-1,r+1);
      }
}
    else { /* j<=0 */
      a[r]=x[i];
      b[r]='-';
      anordnung(i-1,j,r+1);
    }
  }
  else { /* i<=0 */
    if j != 0 then {
      a[r]='-';
      b[r]=y[j];
      anordnung(i,j-1,r+1);
    }
    else {
      print_anordnung(r);
    }
  }
}
```

▲Achtung !
Dieser Algorithmus läuft nicht bei Lückenkosten !

Aufgabe 4.3
☞ Aufgabe 4.2

6.5 5. Hausübung





$w(a,a) > 0$ "sind ähnlinch"
$w(a,b) < 0$ "sind unähnlinch"
$w(a,\varepsilon) < 0$
$w(\varepsilon,a) < 0$

$D(x,y) = max_{(x',y')\in A(x,y)} \sum_{i=1}^{|x'|} w(x[i],y[i])$

z.B.:

a	x	a	b	$-$	c	s
a	x	$-$	b	a	c	s
2	2	-1	2	-1	2	2

Ähnlichkeit: $2+2-1+2-1+2+2=8$

Aufgabe 5.1
$v(i,j) = max\{D(\alpha,\beta)|\alpha$ Suffix von $x[1..i],\ \beta$ Suffix von $y[1..j]\}$
Man zeige $V(x,y) = max\{v(i,j),\ i\in[0..m],\ j\in[0..n]\}$
Beweis:
Offensichtlich gilt

$$V(x,y) \geq max\{v(i,j)|i\in[0..m],\ j\in[0..n]\}$$

Sei $x = x_1\tilde{x}x_2$ und $y = y_1\tilde{y}y_2$ mit $D(\tilde{x},\tilde{y}) = V(x,y)$.
Es gelte $r = |x_1\tilde{x}|$ und $s = |y_1\tilde{y}|$.
Dann gilt

$$D(\tilde{x},\tilde{y}) = V(x,y) \leq v(r,s) \leq max\{v(i,j)|i\in[0..m],\ j\in[0..n]\}$$

Aufgabe 5.2
Behauptung:

$$v(i,j) = max\left\{\begin{array}{l} 0 \\ v(i-1,j-1)+w(x[i],y[j]) \\ v(i-1,j)+w(x[i],\varepsilon) \\ v(i,j-1)+w(\varepsilon,y[i])\} \end{array}\right\}$$

Beweis:
Seien α und β Suffixe von $x[1..i]$ bzw. $y[1..j]$ mit $D(\alpha,\beta) = v(i,j)$.
Da $\alpha = \beta = \varepsilon$ erlaubt ist, gilt $v(i,j) \geq 0$.
Ist $\alpha \neq \varepsilon$, so gilt für die optimale Ausrichtung von α und β

1. $x[i]$ steht $y[j]$ gegenüber
   $$v(i,j) \geq v(i-1,j-1) + w(x[i],y[j])$$
2. $x[i]$ steht "-" gegenüber
   $$v(i,j) \geq v(i-1,j) + w(x[i],\varepsilon)$$

3. Ist $\beta \neq \varepsilon$, dann
   "$-$" steht $y[j]$ gegenüber
   $$v(i,j) \geq v(i,j-1) + w(\varepsilon, y[j])$$

Also $v(i,j) \geq max\{\dots\}$.
Es muß aber einer der obigen Fälle auftreten → Behauptung.

Aufgabe 5.3
$x = zwcqpgk$
$y = lawzqqkpgka$
$w(a,a) = 1$
$w(a,b) = -4,\ a \neq b$
$w(\varepsilon,a) = w(a,\varepsilon) = -1$

Lösung:
$\tilde{x} = pgk$
$\tilde{y} = pgk$
$\Rightarrow D(\tilde{x},\tilde{y}) = 3$
oder
$\tilde{x} = qpgk$
$\tilde{y} = qkpgk$

Bemerkung:
$w(a,a) = 2$
$w(a,b) = -2,\ a \neq b$
$w(\varepsilon,a) = w(a,\varepsilon) = -1$
Dann wäre
$\tilde{x} = qpgk$
$\tilde{y} = qkpgk$
mit $D(\tilde{x},\tilde{y}) = 7$.

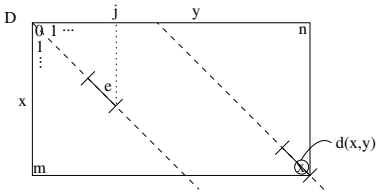
Aufgabe 5.4

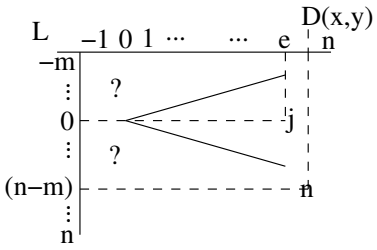
$v(i,j)$	0	1	2	$\cdots$	$\cdots$	$\cdots$	n
1	0			•			
2	0				↘		
3	0					•	
$\vdots$	$\vdots$						
m	0						

Diese Matrix wird mit dem Algorithmus aus Aufgabe 2 gefüllt.
Die Wege von • nach • entsprechen den gesuchten Teilzeichenketten.
☞ Seite 79 mit der Matrix von Seite 81

6.6 6. Hausübung

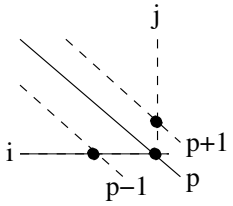
Aufgabe 6.1





1. Rücktransformation der Distanzmatrix D aus der Diagonalenmatrix L.
   Es gilt: Für alle $e \in [0:n]$ gilt:
   ist $L_{p,e-1} < i \leq L_{p,e}$, dann ist $d_{i,p+i} = e$.

2. Konstruktion einer Ausrichtung direkt aus L.

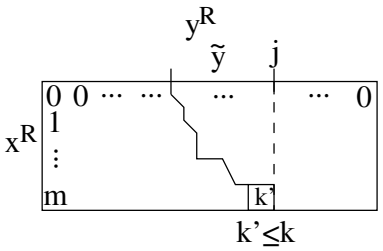


   Idee:
   Ausgehend von der Diagonalen $(m-n)$ wird der Weg zurückverfolgt. Man konstruiert rekursiven Algorithmus. Beim Aufruf des Algorithmus "befindet" man sich in der gedachten Distanzmatrix auf der Position i und j, d.h. auf der Diagonalen $p = j - i$.
   Gilt $d_{ij} = d_{i-1,j} + 1$, dann folgt $L_{p+1,e-1} \geq j$.
   Gilt $d_{ij} = d_{i,j-1} + 1$, dann folgt $L_{p-1,e-1} \geq j - 1$.
   Gilt $d_{ij} = d_{i-1,j-1}$, dann folgt $L_{p,e-1} < j - 1$ und $x[i] = y[j]$.
   Gilt $d_{ij} = d_{i-1,j-1} + 1$, dann folgt $L_{p,e-1} = j - 1$ und $x[i] \neq y[j]$.

```
ass(i,j,e,r) {
  if (i=0) and (j=0) { print_ass(r); return }
  p:=j-i
  if L[p+1,e-1] >= j then {
    a[r] := x[i], b[r] := "-", ass(i-1,j,e-1,r+1)
  } else
  if L[p-1,e-1] >= j-1 then {
    a[r] := "-"; b[r] := y[j]; ass(i,j-1,e-1,r+1)
  } else
  if L[p,e-1] < j-1 and x[i] = y[j] then {
    a[r] := x[i], b[r] := y[j]; ass(i-1,j-1,e,r+1)
```

```
    } else /* L[p,e-1] = j-1 und x[i] != y[j] */ {
      a[r] := x[i], b[r] := y[j]; ass(i-1,j-1,e-1,r+1)
    }
  }
```

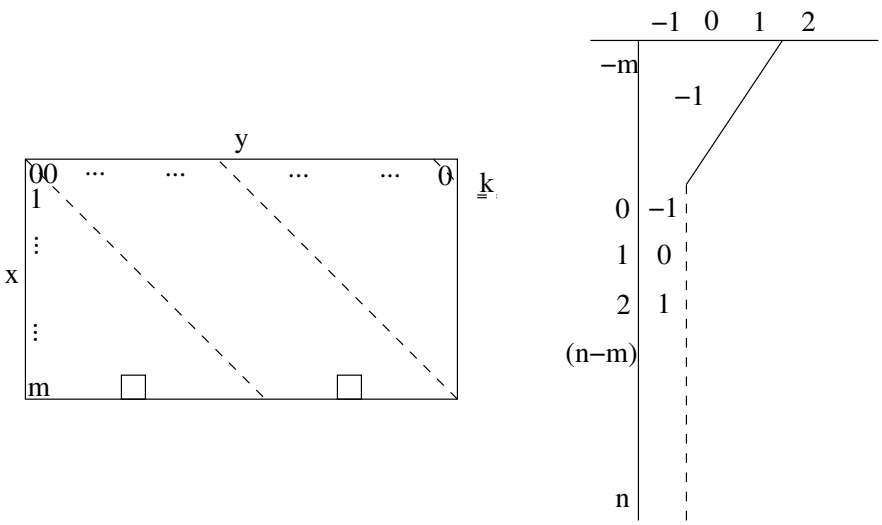
Aufgabe 6.2



$\tilde{y}$ ist Teilzeichenkette in y^R mit $d(x^R, \tilde{y}) = k' \leq k$. Es gilt offensichtlich $d(x^R, \tilde{y}) = d(x, \tilde{y}^R)$.
Folglich ist $\tilde{y}^R$ Teilzeichenkette von y mit Anfangsindex $n - j + 1$.

Satz:
Gilt $d_{mj} \leq k$ in der Matrix, so ist $n - j + 1$ der Anfangsindex des gesuchten Teilwortes von y.
☞ Seite 98

Aufgabe 6.3



Grundidee: Man berechnet die Diagonalmatrix nur bis zur Spalte k.
Gilt $L_{p,k} = m + p$ oder $L_{p,k} =$ "?" für $-m \leq p \leq n - m$, so hat man die Endposition $m + p$ einer Teilzeichenkette $\tilde{y}$ von y mit $d_L(x, \tilde{y}) \leq k$ gefunden.
Wegen der veränderten Initialisierung der Distanzmatrix muß hier ebenfalls die Initialisierung verändert werden.

☞ Seite 83
z.B.

-2-Diagonale: $3 = \overbrace{5}^{|x|} + (-2)$

5-Diagonale: $10 = \overbrace{5}^{|x|} + 5$

6.7 7. Hausübung

Aufgabe 7.1

☞ Seite 99f

j	
0	↓ Folge 0
1	
2	
2	↓ Folge 1
3	
4	
3	↓ Folge 3

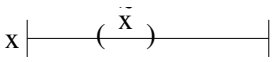
Folge 2 ist leer.

Eine Variable p legt die Position m der aktuellen Spalte fest, bis zu der die Folgen berechnet werden müssen.

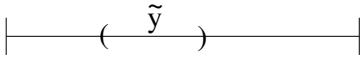
Zu Beginn ist $p = k + 1$, das ist der Wert für Spalte 1.
Wird bei der Berechnung der Endposition der Folge r ein Wert $\geq p$ erreicht, kann die Berechnung der Spalte abgebrochen werden.
Um die cut-off-Position für die nächste Spalte zu bestimmen, muß man die berechnete Spalte "zurücklaufen" und den ersten Eintrag mit Wert k finden.

	j	
		↓ Folge 0
	.	
	.	
	.	
	.	
		↓ Folge r
	d	
$p \rightarrow$		

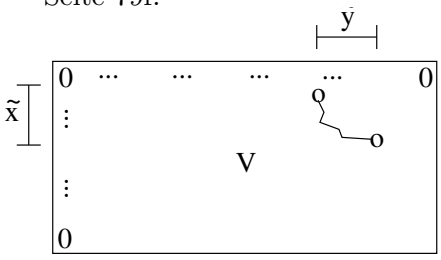
```
r:=t
d:=k+1
while r>=0 and d>k do {
  l:=0
  while l<length[r] and d>k do {
    p:=end[r]-l
    d:=p-r  /* das ist der Wert von d[p,j] */
    l:=l+1
  }
  r:=r-1
}
if p=m then print "Pattern gefunden, Abstand m-t, Position j"
else
  p:=p+1
```



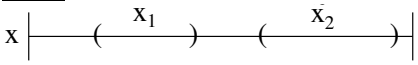
Aufgabe 7.2



☞ 6.5 Seite 62:
$w(a,a) = 2,\ w(a,b) = -2,\ w(\varepsilon,a) = w(a,\varepsilon) = -1$
☞ Seite 79f.



Nun:



Man benutzt zur Lösung dieser Aufgabe den Algorithmus zur Lösung des "local assignement Problems" aus Aufgabenblatt 5 (☞ Seite 62). Um die in der Aufgabenstellung beschriebenen Probleme zu umgehen, muß die Gewichtsfunktion w bei identischen Zeichen, also auf der Diagonalen der v-Matrix den Wert 0 zurückgeben:

$$v_{ij} = max\{0, v_{i-1,j}+w(x[i],\varepsilon), v_{i,j-1}+w(\varepsilon,x[j]), v_{i-1,j-1}+\left\{\begin{array}{ll} 0 & \text{falls } i=j \\ w(x[i],x[j]) & \text{sonst} \end{array}\right.\}$$

Diese Änderung muß beim Rückverfolgen der Maxima berücksichtigt werden !
☞ Seite 101

7 Folien

Algorithmus zur Bestimmung des minimalen Abstands zweier Zeichenketten und einer optimalen Spur (Wagner-Fischer)

```
/* x[1..m] ist die Ausgangszeichenkette */
/* y[1..n] ist die Zielzeichenkette */
/* d[0..m,0..n] ist die Abstandsmatrix */
/* w ist die Gewichtsfunktion */
/* eps bezeichnet das leere Wort */

/* Initialisierung */
d[0,0] := 0;

for i := 1 to m do
    d[i,0] = d[i-1,0] + w(x[i],eps);

for j := 1 to n do
    d[0,j] = d[0,j-1] + w(eps,y[j]);

/* Berechnung der Abstandsmatrix */

for i := 1 to m do
    for j := 1 to n do
        d[i,j] := min ( d[i-1,j]  +  w(x[i],eps),
                        d[i,j-1]  +  w(eps,y[j]),
                        d[i-1,j-1] + w(x[i],y[j]) );

print "Minimaler Abstand ist d[m,n]"

/* Berechnung einer optimalen Spur T durch "Rückverfolgen" */
/* der Minima von d[m,n] bis d[0,0] */


i := m;
j := n;

while i>0 and j>0  do
    if d[i,j] = d[i-1,j] + w(x[i],eps) then
        i := i-1;   /* Zeichen x[i] gelöscht */

    else if d[i,j] = d[i,j-1] + w(eps,y[j]) then
        j := j-1;   /* Zeichen j[j] eingesetzt */
    else begin
        print " (i,j) "
                /* Zeichen x[i] durch y[j] substituiert */
        i := i-1;
        j := j-1;
        end;
```

Wagner-Fischer Beispiel

Es ist: x = abcabba
und y = cbabac

Die Distanzmatrix d für die Levenshtein–Metrik:

	y		c	b	a	b	a	c
x		0	1	2	3	4	5	6
	0	0	1	2	3	4	5	6
a	1	1	1	2	2	3	4	5
b	2	2	2	1	2	2	3	4
c	3	3	2	2	2	3	3	3
a	4	4	3	3	2	3	3	4
b	5	5	4	3	3	2	3	4
b	6	6	5	4	4	3	3	4
a	7	7	6	5	4	4	3	4

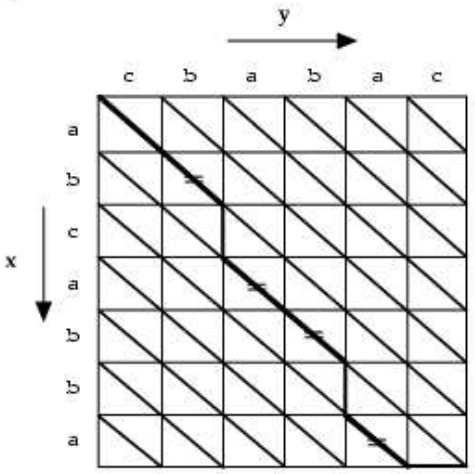
Eine optimale Spur wäre z.B. T = {(1,1),(2,2),(4,3),(5,4),(7,5)}.

Die Distanzmatrix d für die Edit–Metrik:

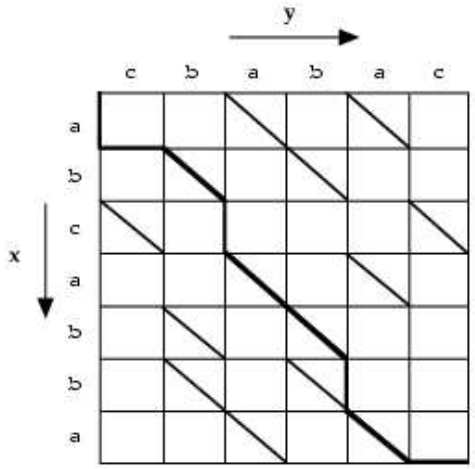
	y		c	b	a	b	a	c
x		0	1	2	3	4	5	6
	0	0	1	2	3	4	5	6
a	1	1	2	3	2	3	4	5
b	2	2	3	2	3	2	3	4
c	3	3	2	3	4	3	4	3
a	4	4	3	4	3	4	3	4
b	5	5	4	3	4	3	4	5
b	6	6	5	4	5	4	5	6
a	7	7	6	5	4	5	4	5

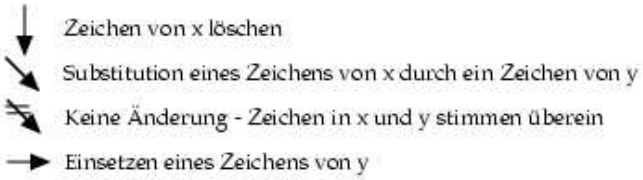
Eine optimale Spur wäre z.B. T = {(2,2),(4,3),(5,4),(7,5)}

Kürzester Weg bzgl. Levenshtein-Metrik:



Kürzester Weg bzgl. Edit-Metrik





Mac OS 9:Vorlesungen:Appr. Zeichenkettensuc...cher:WF-Luecke.prog:affine Lueckenkosten.out Page: 1
Donnerstag, 30. Mai 2002 / 18:07 Uhr

```
x =  ywcqpgk
y =  lawyqqkpgka
Abstand = 16

Die Matrix Ins:
       0  1  2  3  4  5  6  7  8  9 10 11
          l  a  w  y  q  q  k  p  g  k  a
    ---------------------------------------
 0   |--  3  4  5  6  7  8  9 10 11 12 13
 1 y |--  6  6  7  8  8  9 10 11 12 13 14
 2 w |--  7  8  9  9 10 11 12 13 14 15 16
 3 c |--  8  9 10 11 12 13 14 15 16 17 18
 4 q |--  9 10 11 12 13 12 13 14 15 16 17
 5 p |-- 10 11 12 13 14 15 15 16 16 17 18
 6 g |-- 11 12 13 14 15 16 17 18 19 16 17
 7 k |-- 12 13 14 15 16 17 18 18 19 19 16

Die Matrix Del:
       0  1  2  3  4  5  6  7  8  9 10 11
          l  a  w  y  q  q  k  p  g  k  a
    ---------------------------------------
   0 |-- -- -- -- -- -- -- -- -- -- -- --
y  1 |  3  6  7  8  9 10 11 12 13 14 15 16
w  2 |  4  6  8  9  8 11 12 13 14 15 16 17
c  3 |  5  7  9  9  9 11 13 14 15 16 17 18
q  4 |  6  8 10 10 10 12 14 15 16 17 18 19
p  5 |  7  9 11 11 11 12 14 16 17 18 19 20
g  6 |  8 10 12 12 12 13 15 17 16 19 20 21
k  7 |  9 11 13 13 13 14 16 18 17 16 19 20

Die Matrix d:
       0  1  2  3  4  5  6  7  8  9 10 11
          l  a  w  y  q  q  k  p  g  k  a
    ---------------------------------------
   0 |  0  3  4  5  6  7  8  9 10 11 12 13
y  1 |  3  3  6  7  5  8  9 10 11 12 13 14
w  2 |  4  6  6  6  8  8 11 12 13 14 15 16
c  3 |  5  7  9  9  9 11 11 14 15 16 17 18
q  4 |  6  8 10 10 10  9 11 13 14 15 16 17
p  5 |  7  9 11 11 11 12 12 14 13 16 17 18
g  6 |  8 10 12 12 12 13 15 15 16 13 16 17
k  7 |  9 11 13 13 13 14 16 15 17 16 13 16

Optimale Ausrichtungen:
- y w c q - - p g k -
l a w y q q k p g k a

y - w c q - - p g k -
l a w y q q k p g k a

- - - y w c q - - p g k -
l a w y - - q q k p g k a
```

Algorithmus zur Bestimmung des Abstands und einer optimalen Ausrichtung zweier Zeichenketten mit linearem Platzbedarf (Hirschberg)

```
/* x[1..m] ist die Ausgangszeichenkette , m > 0 */
/* y[1..n] ist die Zielzeichenkette */
/* w ist die elementare Gewichtsfunktion */
/* eps bezeichnet das leere Wort */


/* die Prozedur rec_align(x,y) liefert eine optimale */
/* Ausrichtung der beiden Zeichenketten x und y */

procedure rec_align(x,y)
{
m := |x|; n := |y|;

if n = 0 then
     (x',y') = (x, -m);  /* y = eps */

else if m = 1 then {        /* x = x[1], y ≠ eps */

     wähle i mit w(x[1],y[i]) = min{w(x[1],y[j]) | 1 ≤ j ≤ n};

     (x',y') = (-(i-1)x[1]-(n-i), y);

     }

else {

     j := floor(m/2);

     L_1 := dist(x[1..j],y);
     L_2 := dist(x[j+1..m]^R, y^R);

     Minimum := min{L_1[i] + L_2[n-i] | 1 ≤ i ≤ n}
     k := min{ i | 1 ≤ i ≤ n und L_1[i] + L_2[n-i] = Minimum }

     (x_1',y_1') := rec_align(x[1..j], y[1..k]);
     (x_2',y_2') := rec_align(x[j+1..m], y[k+1..n]);

     (x',y') := (x_1'x_2', y_1'y_2');
     }

return (x',y');
}
```

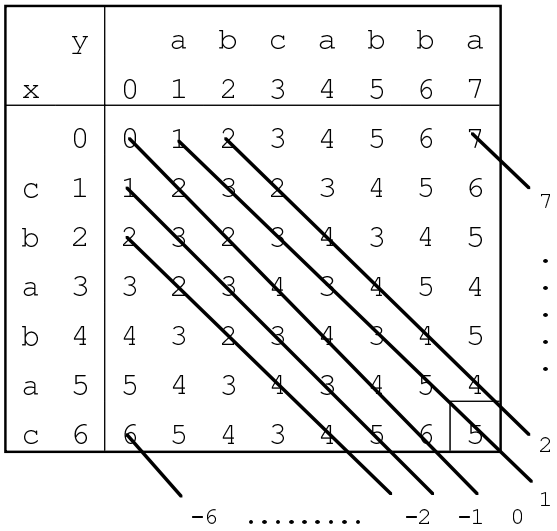
```
/* Die Prozedur dist(x,y) berechnet durch zeilenweise */
/* Auswertung der Distanzmatrix den Abstand zweier    */
/* Zeichenketten x und y. Z_alt[0..n] und Z_neu[0..n] sind */
/* dabei die jeweils aktiven Zeilen. Der Rückgabewert ist
/* die letzte Zeile der Distanzmatrix */

procedure dist(x,y)
{
m := |x|; n := |y|;

/* Initialisierung der 0. Zeile */
Z_alt[0] := 0;
for j := 1 to n do
    Z_alt[j] := Z_alt[j-1] + w(eps,y[j]);

/* zeilenweises Berechnen der Distanzmatrix */
for i := 1 to m do |
    Z_neu[0] := Z_alt[0] + w(x[i],eps);
    for j := 1 to n do
        Z_neu[j] := min (Z_alt[j-1] + w(x[i],y[j]),
                         Z_alt[j]   + w(x[i],eps),
                         Z_neu[j-1] + w(eps,y[j]));
    Z_alt := Z_neu;
    |
return(Z_neu);
}


/* Der eigentlich Algorithmus zur Bestimmung des Abstands */
/* und einer optimalen Ausrcihtung ist dann sehr einfach: */

(x', y') := rec_align(x,y);

abstand := 0;
for i := 1 to |x'| do
    abstand := abstand + w(x'[i], y'[i]);
```

```
x = abcabba
y = cbabac
Abstand = 4
       0  1  2  3  4  5  6
          c  b  a  b  a  c
    ------------------------
 0   | 0  1  2  3  4  5  6
 1  a| 1  1  2  2  3  4  5
 2  b| 2  2  1  2  2  3  4
 3  c| 3  2  2  2  3  3  3
 4  a| 4  3  3  2  3  3  4
 5  b| 5  4  3  3  2  3  4
 6  b| 6  5  4  4  3  3  4
 7  a| 7  6  5  4  4  3  4
-------------------------------
Zerlegung nach Lemma 2.2
x = abc
y = cbabac

       0  1  2  3  4  5  6
          c  b  a  b  a  c
    ------------------------
 0   | 0  1  2  3  4  5  6
 1  a| 1  1  2  2  3  4  5
 2  b| 2  2  1  2  2  3  4
 3  c| 3  2  2  2  3  3  3

x = abba
y = cababc

       0  1  2  3  4  5  6
          c  a  b  a  b  c
    ------------------------
 0   | 0  1  2  3  4  5  6
 1  a| 1  1  1  2  3  4  5
 2  b| 2  2  2  1  2  3  4
 3  b| 3  3  3  2  2  2  3
 4  a| 4  4  3  3  2  3  3
-------------------------------
andere Zerlegung nach Lemma 2.2
x = ab
y = cbabac

       0  1  2  3  4  5  6
          c  b  a  b  a  c
    ------------------------
 0   | 0  1  2  3  4  5  6
 1  a| 1  1  2  2  3  4  5
 2  b| 2  2  1  2  2  3  4

x = abbac
y = cababc

       0  1  2  3  4  5  6
          c  a  b  a  b  c
    ------------------------
 0   | 0  1  2  3  4  5  6
 1  a| 1  1  1  2  3  4  5
 2  b| 2  2  2  1  2  3  4
 3  b| 3  3  3  2  2  2  3
 4  a| 4  4  3  3  2  3  3
 5  c| 5  4  4  4  3  3  3
```

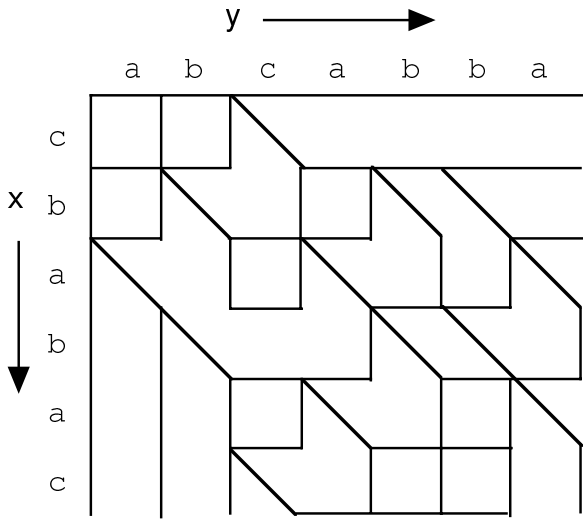
Abhängigkeitsgraph zu einer Distanzmatrix (Ukkonen Algorithmus)

Es ist: `x = cbabac`
und `y = abcabba`

Die Distanzmatrix D für die Edit–Metrik mit Diagonalennumerierung:



`Der Abhängigkeitsgraph:`



Algorithmus zur Bestimmung des Abstands zweier Zeichenketten (Ukkonen)

```
/* x[1..m] ist die Ausgangszeichenkette, */
/* y[1..n] ist die Zielzeichenkette und es  gelte m <= n */
/* d[0..m,0..n] ist die Abstandsmatrix */
/* als Initialwert sei d[i,j] = n+m gesetzt */

/* w ist die Gewichtsfunktion, eps das leere Wort */
/* wmin = min {w(eps,a),w(a,eps) | a in Σ} */


distance_test(h)

/* ist d[m,n]>h, wird -1, sonst wird d[m,n] zurückgegeben */

begin
/* trivialer Fall */
if h/wmin < n-m then    return (-1);

/* Initialisierung */
q = floor((h/wmin - (n-m))/2);
d[0,0] = 0;
for j = 1 to min(n,n-m+q) do
    d[0,j] = d[0,j-1] + w(eps,y[j]);

/* Berechnung von d[m,n] */

for i = 1 to m do
    for j = max(0,i-q) to min(n,i+n-m+q) do
        if j = 0 then
            d[i,0] = d[i-1,0] + w(x[i],eps);
        else
            d[i,j] = min (d[i-1,j] + w(x[i],eps),
                          d[i,j-1] + w(eps,y[j]),
                          d[i-1,j-1] + w(x[i],y[j]));

if (d[m,n] <= h)
    return (d[m,n]);
else
    return (-1);
end;



/* Die eigentliche Abstandsbestimmung */

h = (n-m+1)*wmin;

while (dist := distance_test(h)) < 0 do
    h = 2*h;
```

Ukkonen Beispiel 1

Es ist: `x = cbabac`
und `y = abcabba`

Die Distanzmatrix d für die Edit–Metrik:

h = 2:

	y		a	b	c	a	b	b	a
x		0	1	2	3	4	5	6	7
	0	0	1						
c	1		2	3					
b	2			2	3				
a	3				4	3			
b	4					4	3		
a	5						4	5	
c	6							6	7

h = 4:

	y		a	b	c	a	b	b	a
x		0	1	2	3	4	5	6	7
	0	0	1	2					
c	1	1	2	3	2				
b	2		3	2	3	4			
a	3			3	4	3	4		
b	4				5	4	3	4	
a	5					5	4	5	4
c	6						5	6	5

`h = 8:`

	y		a	b	c	a	b	b	a
x		0	1	2	3	4	5	6	7
	0	0	1	2	3	4			
c	1	1	2	3	2	3	4		
b	2	2	3	2	3	4	3	4	
a	3	3	2	3	4	3	4	5	4
b	4		3	2	3	4	3	4	5
a	5			3	4	3	4	5	4
c	6				3	4	5	6	5

Ukkonen Beispiel 2

Es ist: `x = cbabac`
und `y = abcabba`

Die Distanzmatrix d für die Levenshtein–Metrik:

h = 2:

	y		a	b	c	a	b	b	a
x		0	1	2	3	4	5	6	7
	0	0	1						
c	1		1	2					
b	2			1	2				
a	3				2	2			
b	4					3	2		
a	5						3	3	
c	6							4	4

h = 4:

	y		a	b	c	a	b	b	a
x		0	1	2	3	4	5	6	7
	0	0	1	2					
c	1	1	1	2	2				
b	2		2	1	2	3			
a	3			2	2	2	3		
b	4				3	3	2	3	
a	5					3	3	3	3
c	6						4	4	4

Algorithmus zur Bestimmung zweier Teilzeichenketten größter Ähnlichkeit

```
/* x[1..m] und y[1..n] sind die beiden Zeichenketten */
/* v[0..m,0..n] ist die rekusiv bestimmte Matrix */
/* w ist die elementare Gewichtsfunktion */
/* eps bezeichnet das leere Wort */
/* vmax, maxi und maxj markieren den Maximalwert */

/* Initialisierung */
v[0,0] := 0;
vmax := maxi := maxj := 0;

for i := 1 to m do
    v[i,0] = 0;

for j := 1 to n do
    v[0,j] = 0;

/* Berechnung der Matrix v[i,j]*/

for i := 1 to m do
    for j := 1 to n do {
        v[i,j] := max (0, v[i-1,j-1] + w(x[i],y[j]),
                          v[i-1,j]   +  w(x[i],eps),
                          v[i,j-1]   +  w(eps,y[j]) );

        if v[i,j] > vmax then {
            vmax := v[i,j]; maxi := i; maxj := j;}
    }

print "Die größte Ähnlichkeit zweier Teilzeichenketten
       von x und y ist vmax"
```

```
/* Die Berechnung einer optimalen Ausrichtung der beiden */
/* ähnlichsten Teilzeichenketten geschieht durch */
/* "Rückverfolgen" der Maxima von v[maxi,maxj] aus. */

/* Die Ausrichtung selbst wird durch zwei Felder a und b */
/* repräsentiert, die "umgekehrt" aufgebaut werden.


procedur print-assignment (i, j, r);

if v[i,j] > 0 then {

        if v[i,j] = v[i-1,j-1] + w(x[i],y[j]) then {
            a[r] = x[i];
            b[r] = y[j];
            print_assignment(i-1,j-1,r+1);
            }
        else if v[i,j] = v[i-1,j] + w(x[i],eps) then {
            a[r] = x[i];
            b[r] = '-';
            print_assignment(i-1,j,r+1);
            }
        else if v[i,j] = v[i,j-1] + w(eps,y[j]) then {
            a[r] = '-';
            b[r] = y[j];
            print_assignment(i,j-1,r+1);
            }
        }
else print_ausrichtung(r);


print_assignment(maxi, maxj, 1)
```

y-Matrix zur Beispiellösung zu Aufgabe 4 vom 5. Aufgabenblatt.

```
x = pqraxabcstvq
y = deaxbacsll
maximale Aehnlichkeit = 8
       0  1  2  3  4  5  6  7  8  9 10
          d  e  a  x  b  a  c  s  l  l
       ---------------------------------
 0    | 0  0  0  0  0  0  0  0  0  0  0
 1  p| 0  0  0  0  0  0  0  0  0  0  0
 2  q| 0  0  0  0  0  0  0  0  0  0  0
 3  r| 0  0  0  0  0  0  0  0  0  0  0
 4  a| 0  0  0  2  1  0  2  1  0  0  0
 5  x| 0  0  0  1  4  3  2  1  0  0  0
 6  a| 0  0  0  2  3  2  5  4  3  2  1
 7  b| 0  0  0  1  2  5  4  3  2  1  0
 8  c| 0  0  0  0  1  4  3  6  5  4  3
 9  s| 0  0  0  0  0  3  2  5  8  7  6
10  t| 0  0  0  0  0  2  1  4  7  6  5
11  v| 0  0  0  0  0  1  0  3  6  5  4
12  q| 0  0  0  0  0  0  0  2  5  4  3

Optimale Ausrichtung:
a x - a b c s
a x b a - c s
```

Algorithmus zur Bestimmung der Diagonalenmatrix

```
/* x[1..m] ist die Ausgangszeichenkette */
/* y[1..n] ist die Zielzeichenkette */
/* L[-m..n,-1..n] ist die Diagonalenmatrix */

/* Initialisierung */
for p := -m to n do
    for e := 1 to n do
        L[p,e] = -1;

/* Berechnung der Diagonalenmatrix */

e := -1;
repeat
    e := e + 1;
    for p := -e to e do {
        t := max{L[p,e-1] + 1, L[p-1,e-1] + 1, L[p+1,e-1]}

        /* Liegt die so bestimmte Position eines Punktes  */
        /* der e-Linie auf der Diagonalen p außerhalb der */
        /* Matrix D und endet die (e-1)-Linie vor der     */
        /* letzten Zeile bzw. Spalte, dann endet die      */
        /* e-Linie auf der letzten Zeile bzw. Spalte.     */

        if t > n and L[p,e-1] < n then t = n;
        if t > m+p and L[p,e-1] < m+p then t = m+p;

        while t < n and t-p < m and x[t+1-p] = y[t+1] do
                t := t + 1;
        if t > n or t > m+p then L[p,e] := "?"
                else L[p,e] := t;
    }
until L[n-m,e] = n;
print ("der Abstand von x und y ist e");
```

Diagonalen Beispiel

Es ist: `x = cbabac`
und `y = abcabba`

Die Distanzmatrix D für die Levenshtein–Metrik:

	y		a	b	c	a	b	b	a
x		0	1	2	3	4	5	6	7
	0	0	1	2	3	4	5	6	7
c	1	1	1	2	2	3	4	5	6
b	2	2	2	1	2	3	3	4	5
a	3	3	2	2	2	2	3	4	4
b	4	4	3	2	3	3	2	3	4
a	5	5	4	3	3	3	3	3	3
c	6	6	5	4	3	4	4	4	4

Die Diagonalenmatrix L ergibt sich zu:

	-1	0	1	2	3	4	5	6
-6								
-5			..					
-4		..	**-1**	..		2		
-3			..		3	?		
-2				2	3	4		
-1			0	2	4	5		
0		0	2	3	5	6		
1			1	5	6	7		
2				3	7	?		
3			..		5	7		
4		..	**-1**	..		7		
5			..					
6								
7								

Der Abstand beträgt also 4.

Diagonalenbeispiel 2:

Distanzmatrix für
x = cbabacaccbabac
y = abcabbabaabcababbba

Abstand = 10

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
			a	b	c	a	b	b	a	b	a	a	b	c	a	b	a	b	b	b	a
0		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	c	1	1	2	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
2	b	2	2	1	2	3	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	a	3	2	2	2	2	3	4	4	5	6	7	8	9	10	11	12	13	14	15	16
4	b	4	3	2	3	3	2	3	4	4	5	6	7	8	9	10	11	12	13	14	15
5	a	5	4	3	3	3	3	3	3	4	4	5	6	7	8	9	10	11	12	13	14
6	c	6	5	4	3	4	4	4	4	4	5	5	6	6	7	8	9	10	11	12	13
7	a	7	6	5	4	3	4	5	4	5	4	5	6	7	6	7	8	9	10	11	12
8	c	8	7	6	5	4	4	5	5	5	5	5	6	6	7	7	8	9	10	11	12
9	c	9	8	7	6	5	5	5	6	6	6	6	6	6	7	8	8	9	10	11	12
10	b	10	9	8	7	6	5	5	6	6	7	7	6	7	7	7	8	8	9	10	11
11	a	11	10	9	8	7	6	6	5	6	6	7	7	7	7	8	7	8	9	10	10
12	b	12	11	10	9	8	7	6	6	5	6	7	7	8	8	7	8	7	8	9	10
13	a	13	12	11	10	9	8	7	6	6	5	6	7	8	8	8	7	8	8	9	9
14	c	14	13	12	11	10	9	8	7	7	6	6	7	7	8	9	8	8	9	9	10

Diagonalenmatrix für

x = cbabacaccbabac
y = abcabbabaabcababbba
Abstand = 10

```
      -1  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
     ------------------------------------------------------------------
-14  |
-13  |
-12  |
-11  |
-10  |                                    4
 -9  |                                 5  ?
 -8  |                              6  ?  ?
 -7  |                           7  ?  ?  ?
 -6  |                        7  8  ?  ?  ?
 -5  |                     5  9  ?  ?  ?  ?
 -4  |                  4  9 10  ?  ?  ?  ?
 -3  |               4  5  6 10 11  ?  ?  ?
 -2  |            2  3  5  6  9 12  ?  ?  ?
 -1  |         0  2  4  5  7  8 11 13  ?  ?
  0  |      0  2  3  5  7  8  9 11 13 14  ?
  1  |         1  5  6  7  9 11 12 15  ?  ?
  2  |            3  7  9 10 11 15 16  ?  ?
  3  |               5  8 10 12 13 16 17  ?
  4  |                  9 10 12 16 17 18  ?
  5  |                    10 11 13 17 18 19
  6  |                       13 14 16 19  ?
  7  |                          14 15 17 19
  8  |                             15 16 19
  9  |                                16 17
 10  |                                   17
 11  |
 12  |
 13  |
...
```

Sellers Beispiel

Es ist: `x = abcde`
und `y = aceabpcqdeabcr`

Gesucht sind alle Positionen in y, an denen Teilzeichenketten auftreten, deren Levenshtein-Abstand zu x kleiner gleich 2 ist.

Die modifizierte Distanzmatrix D für die Levenshtein–Metrik:

	y		a	c	e	a	b	p	c	q	d	e	a	b	c	r
x		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	1	1	0	1	1	0	1	1	1	1	1	1	0	1	1	1
b	2	2	1	1	2	1	0	1	2	2	2	2	1	0	1	2
c	3	3	2	1	2	2	1	1	1	2	3	3	2	1	0	1
d	4	4	3	2	2	3	2	2	2	2	2	3	3	2	1	1
e	5	5	4	3	2	3	3	3	3	3	3	2	3	3	2	2

Die vier Teilzeichenketten sind:

Endposition 3 : `ace`
Endposition 10: `abpcqde`
Endposition 13: `abc`
Endposition 14; `abcr`

Algorithmus zur Lösung des k-Differenzen Problems
(Chang - Lampe)

```
/* x[1..m] ist die Zeichenkette, nach der gesucht wird */
/* y[1..n] ist die Eingabezeichenkette und es sei m ≤ n */

/* loc[0..m+2,1..|Σ|] wird in der Vorverarbeitungsphase */
/* berechnet. Hierbei wird angenommen, dass jedes Zeichen */
/* aus dem Alphabet Σ geeignet auf das Intervall [1..|Σ|] */
/* abgebildet wird */


/* Vorverarbeitungsphase */

for a∈ Σ do
    loc[m+1,a] := m + 2; /* Randbedingungen, um den */
    loc[m+2,a] := m + 2;     /* Algorithmus einfacher */
                          /* schreiben zu können */

    for s := m downto 1 do
        if x[s] = a then
            loc[s,a] := s;
        else
            loc[s,a] := loc[s+1,a];
```

```
/* der eigentliche Suchalgorithmus */


/* end[0..m] sind die aktuelle Endpositionen der Folgen */
/* length[0..m] sind die aktuelle Längen der Folgen */
/* t ist die höchste gültige Folgennummer */

/* Initialisierung für Folge 0 in Spalte 0 */

end[0] := m;
length[0] := m+1;
t := 0;

/* die Suchphase */

for j := 1 to n do {
    r := 0;
    while true do {
        if r > t then
                end[r] := m;
        else {
            if length[r] = 0 then
                end[r] := end[r] + 1;
            else {
                s := loc[end[r]-length[r]+2,y[j]]
                if s ≤ end[r] + 1 then
                    end[r] := s - 1;
                else
                    if r+1 ≤ t and length[r+1] ≠ 0 then
                        end[r] := end[r] + 1;
                }
            }
        if end[r] ≥ m then {
            end[r] := m;
            break;
            }
        r := r + 1;
        }
    t := r;


/* Berechnung der Folgenlängen */

length[0] := end[0] + 1;
for r := 1 to t do
    length[r] := end[r] - end[r-1];
if t ≥ m-k then
    print "Pattern gefunden, Abstand m-t, Endpos. j";

}
```

```
Es ist x = abcde und y = aceabpcqdeabcr

(das gleiche Beispiel wurde beim Sellers-Algorithmus verwendet!)
-----------------------------------------------------------------

Spalte j = 1

i =       0  1
end[i]    0  5
length[i] 1  5

 Spalte j = 2

i =       0  1  2
end[i]    1  2  5
length[i] 2  1  3

 Spalte j = 3

i =       0  1  2  3
end[i]    2  3  4  5
length[i] 3  1  1  1
Teilstring mit Abstand 2 endet in Position 3

 Spalte j = 4

i =       0  1  2
end[i]    0  4  5
length[i] 1  4  1

 Spalte j = 5

i =       0  1  2
end[i]    1  1  5
length[i] 2  0  4

 Spalte j = 6

i =       0  1  2
end[i]    1  2  5
length[i] 2  1  3

 Spalte j = 7
```

```
i =       0  1  2
end[i]    2  2  5
length[i] 3  0  3

 Spalte j = 8

i =       0  1  2
end[i]    2  3  5
length[i] 3  1  2

 Spalte j = 9

i =       0  1  2
end[i]    3  3  5
length[i] 4  0  2

 Spalte j = 10

i =       0  1  2  3
end[i]    3  4  4  5
length[i] 4  1  0  1
Teilstring mit Abstand 2 endet in Position 10

 Spalte j = 11

i =       0  1  2
end[i]    0  4  5
length[i] 1  4  1

 Spalte j = 12

i =       0  1  2
end[i]    1  1  5
length[i] 2  0  4

 Spalte j = 13

i =       0  1  2  3
end[i]    1  2  2  5
length[i] 2  1  0  3
Teilstring mit Abstand 2 endet in Position 13

 Spalte j = 14
```

```
i =       0  1  2  3
end[i]    2  2  3  5
length[i] 3  0  1  2
Teilstring mit Abstand 2 endet in Position 14
```

Algorithmus zur Lösung des k-differences Problems (Jokinen, Tarhio und Ukkonen)

```
/* x[1..m] ist die Zeichenkette, nach der gesucht wird */
/* y[1..n] ist die Eingabezeichenkette und es sei m ≤ n */
/* c[0..m] ist die aktuelle Spalte der Abstandsmatrix */

/* p ist die Position in der Spalte, bis zu der die */
/* Abstandswerte berechnet werdet müssen */
/* Es wir der Abstand nach der Levenshtein-Metrik benutzt */

/* Initialisierung der ersten Spalte der Distanzmatrix */

p := k+1;
for i := 0 to m do c[i] := i;


/* Berechnung der j-ten Spalte */

for j := 1 to n do

    begin
    d := 0;
    for i := 1 to p do
        begin
        if x[i] = y[j] then
            e := d;
        else
            e := 1 + min{ c[i-1], c[i], d };
        d := c[i];
        c[i] := e;
        end;

    /* Suche maximale Position in Spalte c mit Wert ≤ k */

    while c[p] > k do p := p-1;

    if p = m then
        print "Pattern gefunden, Abstand c[m], Endpos. j";
    else
        p := p + 1;
    end;
```

7.1 Algorithmus zur Lösung des "Longest Repeated Substring" Problems

```
/* y[1..n] ist die vorgegebene Zeichenkette */
/* Wegen der Symmetrie der Match-Matrix wird nur die */
/* Diagonalen deroberen Dreiecksmatrix durchsucht */

pos1 := 0; /* Position des ersten Auftretens */
pos2 := 0; /* Position des zweiten Auftretens */
mmax := 0; /* Länge der Teilzeichenkette */

for d:= 1 to n-1 do {
  j := n-d+1;
  m := 0; /* Länge der Teilzeichenkette */
  for i := 1 to d do {
    if y[i] = y[j] then
      m := m+1;
    else
      if m > 0 then {
        /* das Ende einer Teilzeichenkette wurde */
        /* erreicht - es gilt */
        /* y[i-m..i-1] = y[j-m..j-1] */
        if m > mmax then {
          mmax := m;
          pos1 := i-m;
          pos2 := j-m;
        }
        m := 0;
      }
    j := j+1;
  }
  if m > mmax then {
    /* das Ende einer Teilzeichenkette wurde erreicht */
    /* es gilt y[d-m..d] = y[j-m..j-1] */
    mmax := m;
    pos1 := d-m;
    pos2 := j-m;
  }
}
print "auf Positionen pos1 und pos2 beginnt die
                maximale Teilzeichenkette der Länge m"
```

Match-Matrix zur Lösung des „Longest Repeated Substring“ Problems

Betrachtet wird die Zeichenkette y = `pabcqrabcsabtu`.

	j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
i		p	a	b	c	q	r	a	b	c	s	a	b	t	u
1	p	1													
2	a		1					1				1			
3	b			1					1				1		
4	c				1					1					
5	q					1									
6	r						1								
7	a		1					1				1			
8	b			1					1				1		
9	c				1					1					
10	s										1				
11	a		1					1				1			
12	b			1					1				1		
13	t													1	
14	u														1

7.2 Algorithmus zur Konstruktion eines Suffix-Baums (McCreight)

Konstruktion eines Suffix-Baumes für eine Zeichenkette $x \in \sum^* \$,\ |x| = n$.

Ausgehend von einem Baum B_0 mit nur einem Knoten werden nacheinander Suffixe $x[i..n]$ von x in den Baum B_{i-1} eingefügt und so ein Baum B_i erzeugt. B_n ist dann der gesuchte Suffix-Baum.
Zwei Invarianten des Algorithmus:

(1) Im Baum B_i hat nur der Knoten, der `head(i)` repräsentiert eventuell keinen gültigen Suffix-Zeiger.

(2) Beim Einfügen von $x[i..n]$ wird der durch `knoten_max_präfix(head(i))` gegebene Knoten B_{i-1} besucht.

Die Invarianten sind für $i = 1$ offensichtlich erfüllt.

Für das Einfügen von $x[i..n]$ in B_{i-1} werden die folgenden Schritte ausgeführt:

(I) Es werden drei Zeichenketten u, v, und w bestimmt, so daß

  (a) `head(i-1)`$= uvw$

  (b) Sei k =`knoten_max_präfix(head(i-1))` in B_{i-2}. Ist k die Wurzel, so sei $v = \varepsilon$. Ist k nicht die Wurzel, dann sei uv die Zeichenkette, die k repräsentiert.

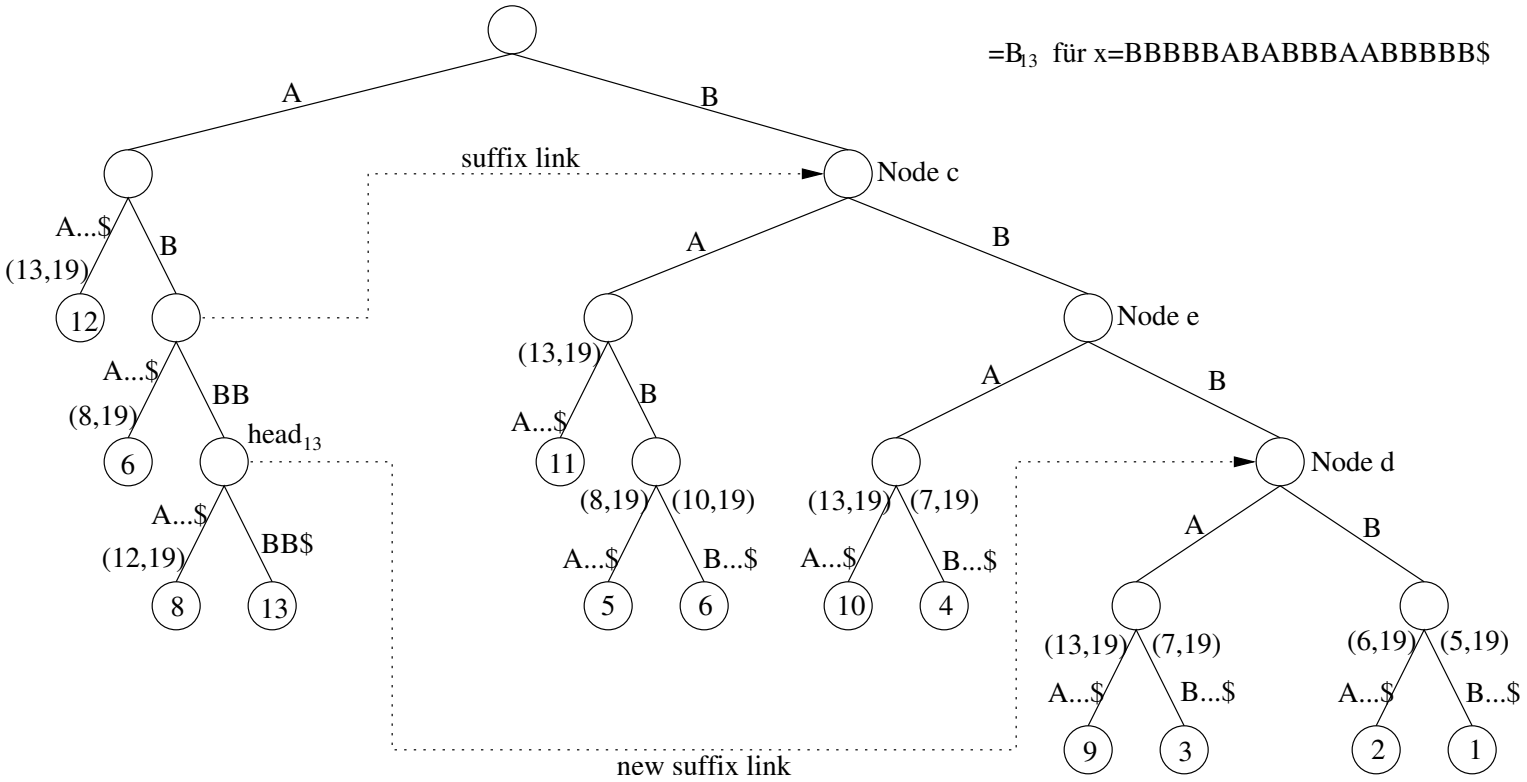
  (c) Es sei $u = \varepsilon$ gdw. `head(i-1)`$= \varepsilon$. Ansonsten gelte $|u| = 1$.

  Bemerkung: Da `head(i-1)`$= uvw$, folgt mit Lemma 3.1 (☞ Seite 44), daß `head(i)`$=vwz$ für ein $z \in \Sigma^*$.

  Ist $v = \varepsilon$, setze c auf die Wurzel des Baumes und gehe zu (II).
  Ist $v \neq \varepsilon$, dann muß der Knoten, der uv repräsentiert bereits in B_{i-2} exisitiert haben (siehe (b)!). Invariante (1) besagt, daß der zugehörige Suffix-Zeiger in B_{i-1} korrekt definiert sein muß , da dieser Knoten vor dem Einsetzen von $x[i-1,..,n]$ eingesetzt wurde. Invariante (2) zeigt, daß dieser Knoten beim Einsetzen von $x[i-1,..,n]$ besucht wurde.
  Der Suffix-Zeiger dieses Knotens wird nun benutzt, um zu einem internen Knoten c zu kommen. (c repräsentiert c). Gehe zu (II).

(II) ("rescanning Phase")
  Aus (I) folgt, daß `head(i)`$= vwz$ ist. Es existiert also der Knoten `knoten_min_ext(vw)` in B_{i-1}. Es gibt daher eine Folge von Kanten ausgehend vom Knoten c zu einem Knoten, der eine kürzeste Zeichenkette von `ext(uv)` repräsentiert. Dieses "rescanning" von w kann in einer Zeit proportional zur Anzahl der durchlaufenen Knoten durchgeführt werden. Ein neuer Knoten d wird erzeugt, falls es einen Knoten gibt, der uv repräsentiert (dies tritt nur auf, wenn $z = \varepsilon$, d.h. `head(i)`$= uv$ ist). Ansonsten sei d der so gefundene Knoten. Gehe zu (III).

(III) ("scanning Phase")
Ist der Suffix-Zeiger des Knotens `knoten(uvw)` (d.h. `head(i-1)`) nicht definiert, so setzte ihn auf d. Damit ist Invariante (1) erfüllt !
Man sucht jetzt ausgehend vom Knoten d den Knoten `knoten_min_ext(vwz)`. Dies wird durch die Zeichen in `tail(i-1)` gesteuert. Der Knoten `knoten_min_ext(vwz)` ist gefunden, wenn dieser Weg nicht weiter geht. Der vorliegende Knoten ist `knoten_max_präfix(head(i))`, also ist die Invariante (2) erfüllt. Falls notwendig wird ein neuer interner Knoten eingeführt, der vwz =`head(i)` repräsentiert und dann eine neue Kante mit Markierung `tail(i)` und ein neues Blatt i einfügt.


=B13 für x=BBBBBABABBBAABBBBB$
A
B
suffix link
Node c
A...$
(13,19)
12
B
A
B
Node e
A...$
(8,19)
6
BB
head13
(13,19)
B
A
B
A...$
11
(8,19)
(10,19)
(13,19)
(7,19)
Node d
A...$
(12,19)
8
BB$
13
A...$
B...$
5
6
A...$
B...$
10
4
A
B
(13,19)
(7,19)
(6,19)
(5,19)
A...$
B...$
A...$
B...$
9
3
2
1
new suffix link

Sellers Beispiel

Es ist: `x = abcde, also` x^R `= edcba`
und `y = aceabpcqdeabcr also` y^R `= rcbaedqcpbaeca`

Gesucht sind alle Positionen in y, an denen Teilzeichenketten auftreten, deren Levenshtein-Abstand zu x kleiner gleich 2 ist.

Die modifizierte Distanzmatrix D für die Levenshtein–Metrik:

	y		r	c	b	a	e	d	q	c	p	b	a	e	c	a
x		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	1	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1
d	2	2	2	2	2	2	1	0	1	2	2	2	2	1	1	2
c	3	3	3	2	3	3	2	1	1	1	2	3	3	2	1	2
b	4	4	4	3	2	3	3	2	2	2	2	2	3	3	2	2
a	5	5	5	4	3	2	3	3	3	3	3	3	2	3	3	2

Die vier Teilzeichenketten in x und y sind:

Anfangsposition 14-14+1 = 1 : `ace`
Anfangsposition 14-11+1 = 4: `abpcqde`
Anfangsposition 14-4+1 = 11: `abc`
oder auch `abcr`

Algorithmus zur Lösung des k-Differenzen Problems mit cut-off Strategie (Chang - Lampe)

```
/* x[1..m] ist die Zeichenkette, nach der gesucht wird */
/* y[1..n] ist die Eingabezeichenkette und es sei m ≤ n */

/* loc[0..m+2,1..|Σ|] wird in der Vorverarbeitungsphase */
/* berechnet. Hierbei wird angenommen, dass jedes Zeichen */
/* aus dem Alphabet Σ geeignet auf das Intervall [1..|Σ|] */
/* abgebildet wird */


/* Vorverarbeitungsphase */

for a∈ Σ do
    loc[m+1,a] := m + 2; /* Randbedingungen, um den */
    loc[m+2,a] := m + 2;     /* Algorithmus einfacher */
                         /* schreiben zu können */

    for s := m downto 1 do
        if x[s] = a then
            loc[s,a] := s;
        else
            loc[s,a] := loc[s+1,a];


/* in der Suchphase wird ausgenutzt, dass die Werte auf */
/* den Diagonalen der Distanzmatrix monoton wachsen und */
/* daher die Berechnung auf den Diagonalen nach Über- */
/* schreiten des Schwellwerts k abgebrochen werden kann.*/
/* Die Variable p legt diese cut-off Position für die */
/* aktuelle Spalte fest.*/
```

```
/* der eigentliche Suchalgorithmus */
/* k sei der vorgegebene Schwellwert */

/* end[0..m] sind die aktuelle Endpositionen der Folgen */
/* length[0..m] sind die aktuelle Längen der Folgen */
/* t ist die höchste gültige Folgennummer */

/* Initialisierung für Folge 0 in Spalte 0 */

end[0] := k+1; length[0] := k+2; t := 0;
p := k+1;  /* die cut-off position */

/* die Suchphase */

for j := 1 to n do {
    r := 0;
    while true do {
        if r > t then
                end[r] := p+1;
        else {
            if length[r] = 0 then
                end[r] := end[r] + 1;
            else {
                s := loc[end[r]-length[r]+2,y[j]]
                if s ≤ end[r] + 1 then
                    end[r] := s - 1;
                else
                    if r+1 ≤ t and length[r+1] ≠ 0 then
                        end[r] := end[r] + 1;
                }
            }
        if end[r] ≥ p then {
            if end[r] > m then
                end[r] := m;
            break;
            }
        r := r + 1;
        }
    t := r;

/* Berechnung der Folgenlängen */

length[0] := end[0] + 1;
for r := 1 to t do
    length[r] := end[r] - end[r-1];

/* Berechnung der neuen cut-off Position */

r := t; d := k+1;
while r ≥ 0 and d > k do {
    l := 0
    while l < length[r] and d > k do {
        p := end[r]-l; d := p-r; l := l+1;}
    r := r-1;
}
if p = m then
    print "Pattern gefunden, Abstand m-t, Endpos. j";
else
    p := p+1;
}
```

x = pqraxabcstvqdeaxbacsll

maximale Aehnlichkeit = 8

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
			p	q	r	a	x	a	b	c	s	t	v	q	d	e	a	x	b	a	c	s	l	l
0	|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	p|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	q|	0	0	0	0	0	0	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0
3	r|	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
4	a|	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	2	1	0	2	1	0	0	0
5	x|	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	4	3	2	1	0	0	0
6	a|	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	2	3	2	5	4	3	2	1
7	b|	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	2	5	4	3	2	1	0
8	c|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	3	6	5	4	3
9	s|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	5	8	7	6
10	t|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	1	4	7	6	5
11	v|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	3	6	5	4
12	q|	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	5	4	3
13	d|	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	3	2
14	e|	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	2	1
15	a|	0	0	0	0	2	1	2	1	0	0	0	0	0	0	0	0	0	0	2	1	2	1	0
16	x|	0	0	0	0	1	4	3	2	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0
17	b|	0	0	0	0	0	3	2	5	4	3	2	1	0	0	0	0	0	0	0	0	0	0	0
18	a|	0	0	0	0	2	2	5	4	3	2	1	0	0	0	0	2	1	0	0	0	0	0	0
19	c|	0	0	0	0	1	1	4	3	6	5	4	3	2	1	0	1	0	0	0	0	0	0	0
20	s|	0	0	0	0	0	0	3	2	5	8	7	6	5	4	3	2	1	0	0	0	0	0	0
21	l|	0	0	0	0	0	0	2	1	4	7	6	5	4	3	2	1	0	0	0	0	0	0	2
22	l|	0	0	0	0	0	0	1	0	3	6	5	4	3	2	1	0	0	0	0	0	0	2	1

Optimale Ausrichtung:
a x - a b c s
a x b a - c s

8 Literaturliste für die Vorlesung "Approximative Zeichenkettensuche" (SS 2002)

1. A. Apostolico und Z. Galil, (eds), *Pattern matching algorithms*, Oxford University Press, 1997
2. 2. A.V. Aho, *Algorithms for Finding Patterns in Strings*, in: *Handbook of Theoretical Computer Science*, J. van Leeuwen (ed), Elsevier Science Publishers, 1990
3. 3. B. Baehr, *Vergleichsmethoden für Zeichenketten*, Diplomarbeit am Institut für Informatik der Universität Hannover, 1990.
4. 4. M. Crochemore und W. Rytter,*Text Algorithms,* Oxford University Press, 1994.
5. 5. C. Charras und T. Lecroq, *Sequence Comparison*, Université de Rouen, Manuskript, http://www-igm.univ-mlv.fr/ lecroq/seqcomp/seqcomp.ps
6. 6. D. Gusfield, *Algorithms on Strings, Trees and Sequences*, Cambridge University Press, 1997.
7. 7. G. Navarro, *A Guided Tour to Approximate String Matching*, ACM Computing Surveys, Vol. 33, No. 1, 2001, pp. 31-88.
8. 8. D. Sankoff und J.B. Kruskal, *Time warps, string edits and macromolecules: the theory and practice of sequence comparisons*, Addison Wesley, 1983.
9. 9. J.C. Setubal und J. Meidanis, *Introduction to Computional Molecular Biology*, PWS Publishing Company, 1997.
10. 10. G.A. Stephen, *String Searching Algorithms*, World Scientific, 1994.
11. 11. M.S. Waterman, *Introduction to Computational Biology - Maps, Sequences and Genomes*, Chapman & Hall, 1995.
12. 12. E. Ukkonen, *Algorithms for Approximate String Matching,* Information and Control 64 (1985), 100-118

Index