Übersicht

Alphabet, Wörter, Wortlänge, Wortmengen

Definition

Ein Alphabet Σ ist eine endliche Menge von Symbolen.

Wörter über Σ sind endliche Folgen von Symbolen aus Σ (meist $w = w_0 \cdots w_{n-1}$ oder $w = w_1 \cdots w_n$).

Notation:

$\lvert w \rvert$	Länge des Wortes w (Anzahl der Zeichen in w)
ε	leeres Wort (Wort der Länge 0)
Σ^*	Menge aller Wörter über Σ
Σ^+	Menge aller Wörter der Länge ≥ 1 über Σ ($\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$)
Σ^k	Menge aller Wörter über Σ der Länge k

Präfix, Suffix, Teilwort

Definition

$[a : b] := \{n \in \mathbb{Z} \mid a \leq n \ \wedge \ n \leq b\}$ für $a, b \in \mathbb{Z}$

Sei $w = w_1 \cdots w_n$ ein Wort der Länge n über Σ, dann heißt

- w' Präfix von w, wenn $w' = w_1 \cdots w_\ell$ mit $\ell \in [0 : n]$
- w' Suffix von w, wenn $w' = w_\ell \cdots w_n$ mit $\ell \in [1 : n+1]$
- w' Teilwort von w, wenn $w' = w_i \cdots w_j$ mit $i, j \in [1 : n]$

Für $w' = w_i \cdots w_j$ mit $i > j$ soll gelten $w' = \varepsilon$.

Das leere Wort ε ist also Präfix, Suffix und Teilwort eines jeden Wortes.

Textsuche

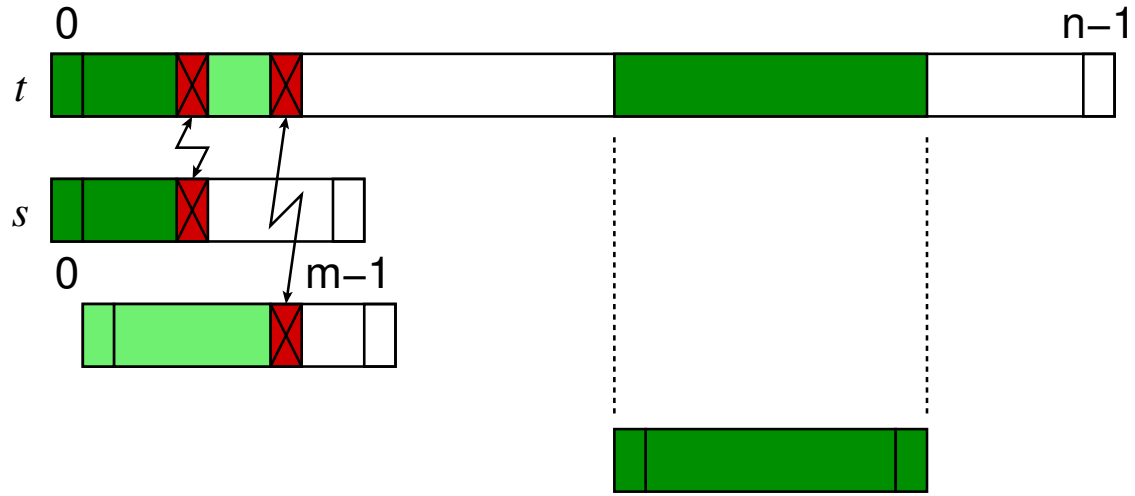
Problem:

Gegeben: Text $t \in \Sigma^*$; $|t| = n$;
Suchwort $s \in \Sigma^*$; $|s| = m \leq n$

Gesucht: $\exists\, i \in [0 : n - m]$ mit $t_i \cdots t_{i+m-1} = s$?
(bzw. alle solchen Positionen i)

Übersicht

Naiver Algorithmus



- Suchwort *s* Zeichen für Zeichen mit Text *t* vergleichen
- wenn zwei Zeichen nicht übereinstimmen (Mismatch), dann *s* um eine Position „nach rechts“ schieben und erneut *s* mit *t* vergleichen
- Vorgang wiederholen, bis *s* in *t* gefunden wird oder bis klar ist, dass *s* in *t* nicht enthalten ist

Naiver Algorithmus: Beispiele

t	=	a	a	a	a	a	a	a	a	a	a
		a	a	a	b					
			a	a	a	b				
				a	a	a	b			

t	=	a	a	b	a	a	b	a	a	b	a	a	b	a	a	b
		a	a	b	a	a	a								
			a	a	b	a	a	a							
				a	a	b	a	a	a						
					a	a	b	a	a	a					

Naiver Algorithmus: Implementation

Funktion NaiveSearch(char $t[\,]$, int n, char $s[\,]$, int m)

```
int i := 0, j := 0;
while (i ≤ n − m) do
    while (t[i + j] = s[j]) do
        j++;
        if (j = m) then
            return TRUE;
    i++;
    j := 0;
return FALSE;
```

Analyse des naiven Algorithmus

- zähle Vergleiche von Zeichen,
- äußere Schleife wird $(n - m + 1)$-mal durchlaufen,
- die innere Schleife wird maximal m-mal durchlaufen.
- maximale Anzahl von Vergleichen: $(n - m + 1)m$,
- Laufzeit: $\mathcal{O}(nm)$

Übersicht

Bessere Idee

- frühere erfolgreiche Zeichenvergleiche ausnutzen

- Idee:

  Suchwort so weit nach rechts verschieben, dass in dem Bereich von t, in dem bereits beim vorherigen Versuch erfolgreiche Zeichenvergleiche durchgeführt wurden, nun nach dem Verschieben auch wieder die Zeichen in diesem Bereich übereinstimmen

Rand und eigentlicher Rand

Definition

Ein Wort r heißt Rand eines Wortes w, wenn r Präfix und Suffix von w ist. (Für jedes Wort w ist das leere Wort ε ein Rand von w, genau wie w selbst.)

Ein Rand r eines Wortes w heißt eigentlicher Rand, wenn $r \neq w$ und wenn es außer w selbst keinen längeren Rand gibt.

Rand und eigentlicher Rand

Beispiel

Das Wort $w = \texttt{aabaabaa}$ besitzt folgende Ränder:

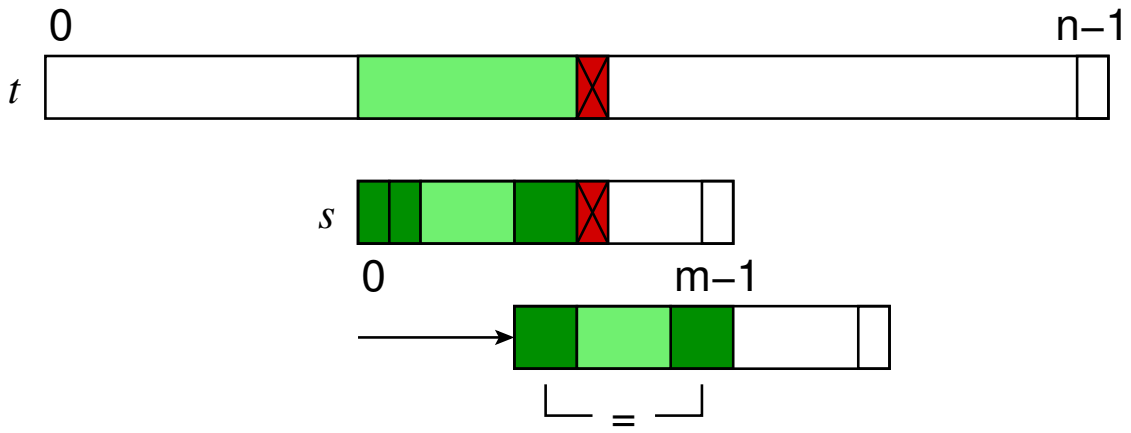
- ε
- $\texttt{a}$
- $\texttt{aa}$
- $\texttt{aabaa}$
- $\texttt{aabaabaa} = w$.

Der eigentliche Rand ist $\texttt{aabaa}$.

Beachte: bei der Darstellung des Rands im Wort können sich Präfix und Suffix in der Wortmitte überlappen.

Shift-Idee

- Pattern s so verschieben, dass im bereits gematchten Bereich wieder Übereinstimmung herrscht.
- Dazu müssen überlappendes Präfix und Suffix dieses Bereichs übereinstimmen.



Shifts und sichere Shifts

Definition

Eine Verschiebung der Anfangsposition i des zu suchenden Wortes (also eine Indexerhöhung $i \to i'$) heißt Shift.

Ein Shift von $i \to i'$ heißt sicher, wenn s nicht als Teilwort von t an der Position $k \in [i+1 : i'-1]$ vorkommt, d.h., $s \neq t_k \cdots t_{k+m-1}$ für alle $k \in [i+1 : i'-1]$.

- Sinn eines sicheren Shifts:
  dass man beim Verschieben des Suchworts kein eventuell vorhandenes Vorkommen von s in t überspringt

Sichere Shifts

Definition

Sei $\partial(s)$ der eigentliche Rand von s und sei

$$border[j] = \begin{cases} -1 & \text{für } j = 0 \\ \left|\partial(s_0 \cdots s_{j-1})\right| & \text{für } j \geq 1 \end{cases}$$

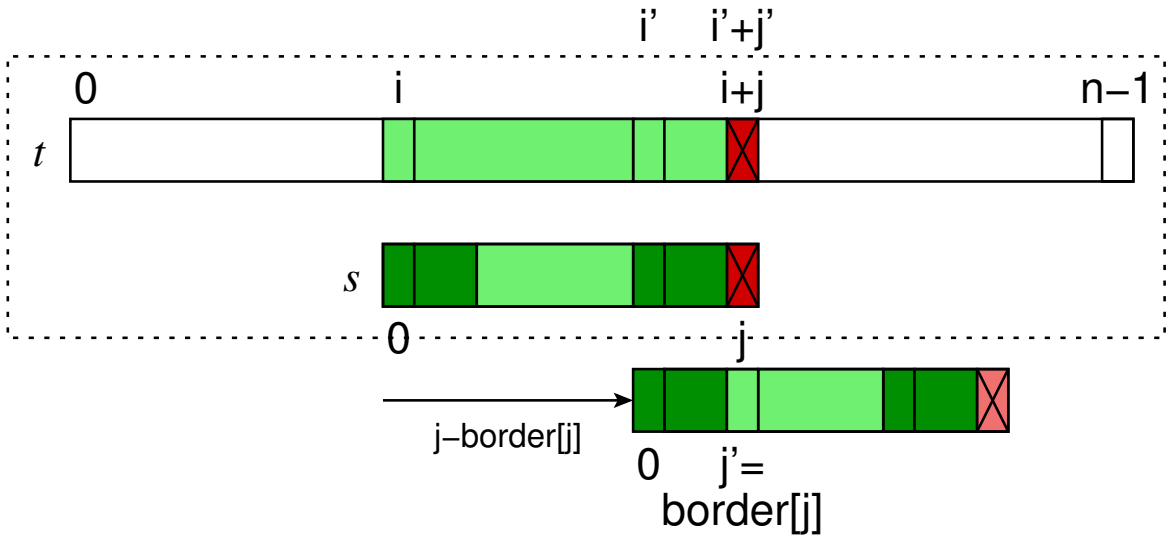
die Länge des eigentlichen Rands des Präfixes der Länge j.

Lemma

Ist das Präfix der Länge j gematcht (also gilt $s_k = t_{i+k}$ für alle $k \in [0 : j-1]$) und haben wir ein Mismatch an der nächsten Position j ($s_j \neq t_{i+j}$), dann ist der Shift $i \to i + j - border[j]$ sicher.

Sichere Shifts

Shift um j – border[j]



Sichere Shifts

Beweis.

- (siehe Skizze)

$$\begin{aligned} s_0 \cdots s_{j-1} &= t_i \cdots t_{i+j-1}, \\ s_j &\neq t_{i+j} \end{aligned}$$

- Der eigentliche Rand von $s_0 \cdots s_{j-1}$ hat die Länge $\mathrm{border}[j]$.
- Verschiebt man s um $j - \mathrm{border}[j]$ nach rechts, so liegt der linke Rand von $s_0 \cdots s_{j-1}$ nun genau da, wo vorher der rechte Rand lag, d.h. im Präfix/Suffix-Überlappungsbereich besteht Übereinstimmung zwischen Präfix, Suffix und Text.
- Da es keinen längeren Rand von $s_0 \cdots s_{j-1}$ als diesen gibt (außer $s_0 \cdots s_{j-1}$ selbst), ist dieser Shift sicher.

□

KMP-Algorithmus

Funktion KMP($t[\,]$, n, $s[\,]$, m)

```
int border[m + 1];
computeBorders(border, m, s);
int i := 0, j := 0;
while i ≤ n − m do
    while t[i + j] = s[j] do
        j++;
        if j = m then
            return TRUE;
    i := i + (j − border[j]) ;          // Es gilt j − border[j] > 0
    j := max{0, border[j]};
return FALSE;
```

Laufzeit des KMP-Algorithmus: erfolglose Vergleiche

Nach erfolglosem Vergleich (Mismatch) wird $(i + j)$ nie kleiner:

- Seien dazu i und j die Werte vor einem erfolglosen Vergleich und i' und j' die Werte nach einem erfolglosen Vergleich.
- Wert vor dem Vergleich: $i + j$
- Wert nach dem Vergleich:
  $i' + j' = (i + j - border[j]) + (max\{0, border[j]\})$.
- Fallunterscheidung: $border[j]$ negativ oder nicht.
  - $border[j] < 0$, also $border[j] = -1$, dann muss $j = 0$ sein.
    Das bedeutet $i' + j' = i' + 0 = (i + 0 - (-1)) + 0 = i + 1$.
  - $border[j] \geq 0$, dann gilt $i' + j' = i + j$
- Also wird $i + j$ nach einem erfolglosen Vergleich nicht kleiner.

Laufzeit des KMP-Algorithmus

- Nach jedem erfolglosen Vergleich wird $i \in [0 : n - m]$ erhöht.
- i wird nie verkleinert.

$\Rightarrow$ maximal $n - m + 1$ erfolglose Vergleiche

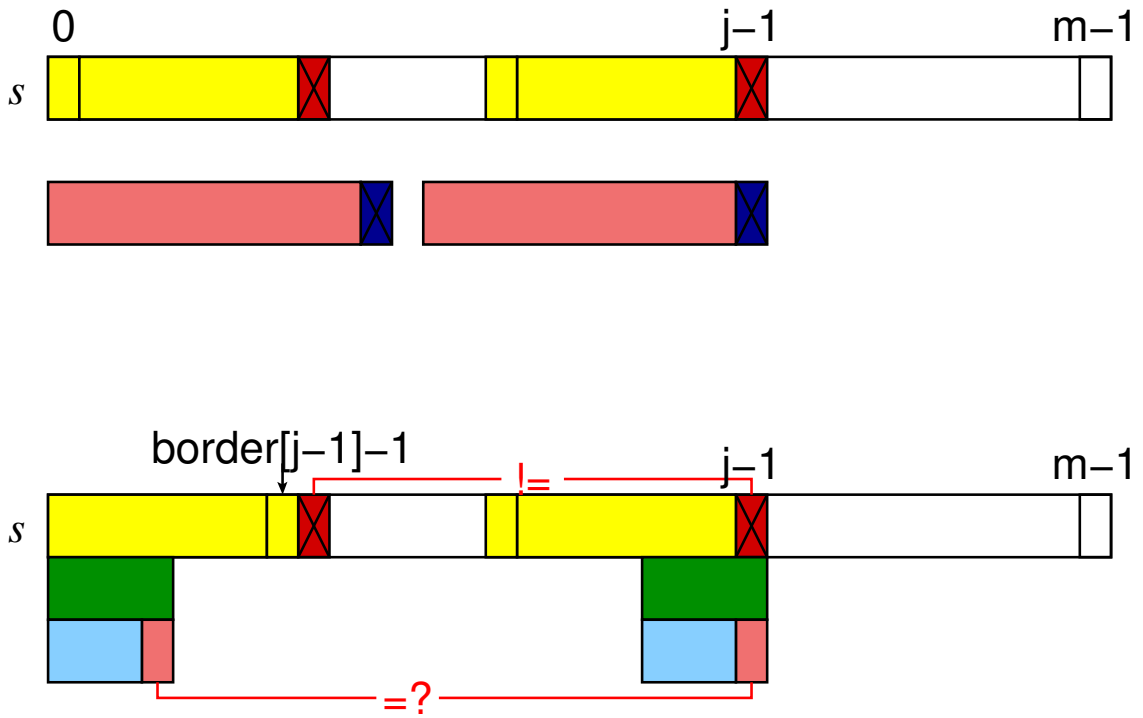
- Nach einem erfolgreichen Vergleich wird $i + j$ um 1 erhöht.
- maximal n erfolgreiche Vergleiche, da $i + j \in [0 : n - 1]$.

- insgesamt maximal $2n - m + 1$ Vergleiche

Berechnung der border-Tabelle

- *border*[ ]-Tabelle:
  speichert für jedes Präfix $s_0 \cdots s_{j-1}$ der Länge $j \in \{0 \ldots m\}$ von Suchstring s die Größe des eigentlichen Rands
- Initialisierung: $border[0] = -1$ und $border[1] = 0$
- Annahme: $border[0], \ldots, border[j-1]$ sind schon berechnet
- Ziel: Berechnung von $border[j]$
  (Länge des eigentlichen Randes von Präfix der Länge j)

Berechnung der border-Tabelle



Berechnung der border-Tabelle

- Der eigentliche Rand $s_0 \cdots s_k$ von $s_0 \cdots s_{j-1}$ kann um maximal ein Zeichen länger sein als der eigentliche Rand von $s_0 \cdots s_{j-2}$, denn $s_0 \cdots s_{k-1}$ ist auch ein Rand von $s_0 \cdots s_{j-2}$ (oberer Teil der Abbildung).
- Ist $s_{border[j-1]} = s_{j-1}$, so ist $border[j] = border[j-1] + 1$.
- Andernfalls müssen wir ein kürzeres Präfix von $s_0 \cdots s_{j-2}$ finden, das auch ein Suffix von $s_0 \cdots s_{j-2}$ ist.
- Der nächstkürzere Rand eines Wortes ist offensichtlich der eigentliche Rand des zuletzt betrachteten Randes dieses Wortes.
- Nach Konstruktion der Tabelle *border* ist das nächstkürzere Präfix mit dieser Eigenschaft das der Länge $border[border[j-1]]$.

Berechnung der border-Tabelle

- teste nun, ob sich dieser Rand von $s_0 \cdots s_{j-2}$ zu einem eigentlichen Rand von $s_0 \cdots s_{j-1}$ erweitern lässt
- solange wiederholen, bis wir einen Rand gefunden haben, der sich zu einem Rand von $s_0 \cdots s_{j-1}$ erweitern lässt
- Falls sich kein Rand von $s_0 \cdots s_{j-2}$ zu einem Rand von $s_0 \cdots s_{j-1}$ erweitern lässt, so ist der eigentliche Rand von $s_0 \cdots s_{j-1}$ das leere Wort und wir setzen $\mathit{border}[j] = 0$.

Algorithmus zur Berechnung der border-Tabelle

Prozedur computeBorders(int *border*[ ], int *m*, char *s*[ ])

```
border[0] := −1;
border[1] := 0;
int i := 0;
for (int j := 2; j ≤ m; j++) do
    // Hier gilt: i = border[j − 1]
    while (i ≥ 0) && (s[i] ≠ s[j − 1]) do
        i := border[i];
    i++;
    border[j] := i;
```

Laufzeit der Berechnung der border-Tabelle

- maximal $m-1$ erfolgreiche Vergleiche, da jedes Mal $j \in [2:m]$ um 1 erhöht und nie erniedrigt wird

- Betrachte für die Anzahl erfolgloser Vergleiche den Wert i.
  Zu Beginn ist $i = 0$.
- i wird genau $(m-1)$ Mal um 1 erhöht, da die for-Schleife $(m-1)$ Mal durchlaufen wird.
- Bei einem erfolglosen Vergleich wird i um mindestens 1 erniedrigt.
- i kann maximal $(m-1)+1 = m$ Mal erniedrigt werden, da immer $i \geq -1$ gilt. Es gibt also höchstens m erfolglose Vergleiche.

- Gesamtzahl der Vergleiche $\leq 2m-1$

Laufzeit des KMP-Algorithmus

Theorem

Der Algorithmus von Knuth, Morris und Pratt benötigt maximal $2n + m$ Vergleiche, um festzustellen, ob ein Muster s der Länge m in einem Text t der Länge n enthalten ist.

Der Algorithmus lässt sich leicht derart modifizieren, dass er alle Positionen der Vorkommen von s in t ausgibt, ohne dabei die asymptotische Laufzeit zu erhöhen.

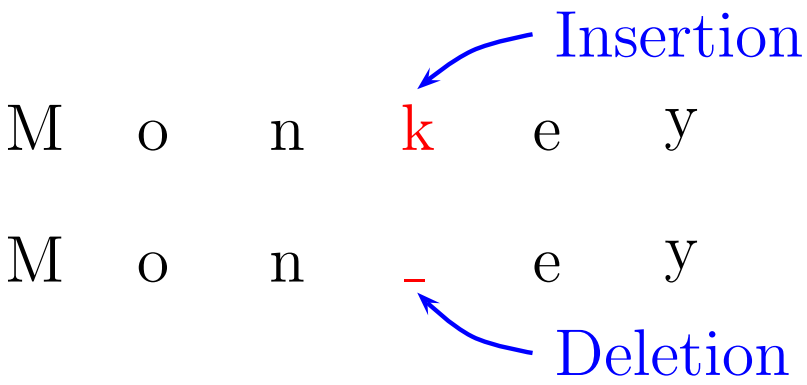
Donald E. Knuth, James H. Morris, Jr. and Vaughan R. Pratt
Fast Pattern Matching in Strings
SIAM Journal on Computing 6(2):323–350, 1977.

Übersicht

Distanz und Ähnlichkeit von Sequenzen

Paarweises Sequenzenalignment

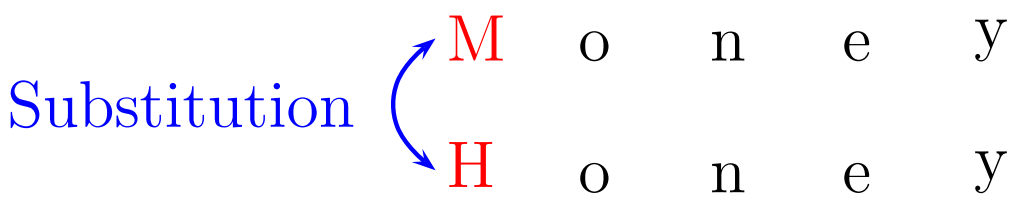
- Ähnlichkeit von 2 Sequenzen bzw.
- Wie kann die eine Sequenz aus der anderen hervorgegangen sein?



Beispiel: Veränderung durch Insertion bzw. Deletion

Distanz und Ähnlichkeit von Sequenzen

- Neben Einfügen und Löschen ist das Ersetzen von Zeichen eine weitere Möglichkeit.



Beispiel: Veränderung durch Substitution

Edit-Distanz

Definition

Sei Σ ein Alphabet und sei $-$ ein neues Zeichen, d.h. $- \notin \Sigma$.

Dann bezeichne

$$\overline{\Sigma} := \Sigma \cup \{-\}$$

das um $-$ erweiterte Alphabet.

Außerdem sei

$$\overline{\Sigma}_0^2 := \overline{\Sigma} \times \overline{\Sigma} \setminus \{(-,-)\} \quad .$$

Das Zeichen $-$ werden wir auch als *Leerzeichen* bezeichnen.

Edit-Operationen

Definition

Eine Edit-Operation ist ein Paar $(x, y) \in \overline{\Sigma}_0^2$ und (x, y) heißt

- *Match*, wenn $x = y \in \Sigma$;
- *Substitution*, wenn $x \neq y$ mit $x, y \in \Sigma$;
- *Insertion*, wenn $x = -$, $y \in \Sigma$;
- *Deletion*, wenn $x \in \Sigma$, $y = -$.

Als *InDel-Operation* bezeichnet man eine Edit-Operation, die entweder eine Insertion oder Deletion ist.

Eine neutrale (NoOp)-Operation $(x, x) \in \Sigma \times \Sigma$ ist hier als Edit-Operation zugelassen. Manchmal wird dies auch nicht erlaubt.

Edit-Operationen

Definition

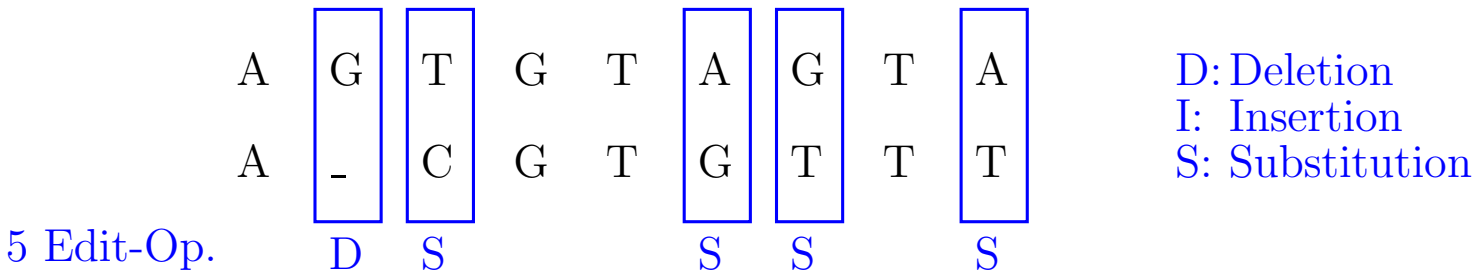
Ist (x, y) eine Edit-Operation und sind $a, b \in \Sigma^*$, dann gilt $a \xrightarrow{(x,y)} b$ (a kann durch die Edit-Operation (x, y) in b umgeformt werden), wenn

- $x, y \in \Sigma \ \wedge \ \exists\, i \in [1 : |a|] : \ (a_i = x) \wedge$
  $(b = a_1 \cdots a_{i-1} \cdot y \cdot a_{i+1} \cdots a_{|a|})$ (Substitution oder Match)
- $x \in \Sigma \ \wedge \ y = - \ \wedge \ \exists\, i \in [1 : |a|] : (a_i = x) \wedge$
  $(b = a_1 \cdots a_{i-1} \cdot a_{i+1} \cdots a_{|a|})$ (Deletion)
- $x = - \ \wedge \ y \in \Sigma \ \wedge \ \exists\, i \in [1 : |a| + 1] :$
  $(b = a_1 \cdots a_{i-1} \cdot y \cdot a_i \cdots a_{|a|})$ (Insertion)

Sei $s = ((x_1, y_1), \ldots, (x_m, y_m))$ eine Folge von Edit-Operationen mit $a_{i-1} \xrightarrow{(x_i,y_i)} a_i$, wobei $a_i \in \Sigma^*$ für $i \in [0 : m]$ und $a := a_0$ und $b := a_m$.
Dann schreibt man auch kurz $a \overset{s}{\Rightarrow} b$.

Sequenz-Transformationen

$$AGTGTAGTA \overset{s}{\Rightarrow} ACGTGTTT \text{ mit } s = ((G,-),(T,C),(A,G),(G,T),(A,T))$$
$$\text{oder mit } s = ((G,T),(A,G),(G,-),(A,T),(T,C))$$



Beispiel: Transformation mit Edit-Operationen

Anmerkung: Es kommt nicht unbedingt auf die Reihenfolge der Edit-Operationen an.

Sequenz-Transformationen

$$AGTGTAGTA \overset{s'}{\Rightarrow} ACGTGTTT \text{ mit } s' = ((-, C), (A, -), (G, -), (A, T))$$

A	_	G	T	G	T	A	G	T	A	D: Deletion
A	C	G	T	G	T	_	_	T	T	I: Insertion
4 Edit-Op.	I					D	D		S	S: Substitution

Beispiel: Transformation mit anderen Edit-Operationen

Dieselben Sequenzen können auch mit Hilfe anderer Edit-Operationen ineinander transformiert werden:

Kosten

- Kosten einer Transformationsfolge setzen sich zusammen aus den Kosten der einzelnen Edit-Operationen.
- Man verwendet eine Kostenfunktion $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$
- Bsp.: Match-Kosten 0, Substitution/Insertion/Deletion-Kosten 1
- In der Biologie (wo die Sequenzen Basen oder insbesondere Aminosäuren repräsentieren) wird man jedoch intelligentere Kostenfunktionen wählen.

Kostenfunktion, Edit-Distanz

Definition

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine Kostenfunktion. Seien $a, b \in \Sigma^*$ und sei $s = (s_1, \ldots, s_\ell)$ eine Folge von Edit-Operationen mit $a \overset{s}{\Rightarrow} b$. Dann sind die Kosten der Edit-Operationen s definiert als

$$w(s) := \sum_{j=1}^{\ell} w(s_j).$$

Die Edit-Distanz von $a, b \in \Sigma^*$ ist definiert als

$$d_w(a, b) := \min_s \left\{ w(s) \ : \ a \overset{s}{\Rightarrow} b \right\}.$$

Dreiecksungleichung

- Folgende Beziehung soll gelten:

$$\forall x, y, z \in \overline{\Sigma} : \; w(x, y) + w(y, z) \geq w(x, z)$$

- Betrachte zum Beispiel eine Mutation (x, z), die als direkte Mutation relativ selten (also teuer) ist, sich jedoch sehr leicht (d.h. billig) durch zwei Mutationen (x, y) und (y, z) ersetzen lässt.
- Dann sollten die Kosten für diese Mutation durch die beiden billigen beschrieben werden, da man in der Regel nicht feststellen kann, ob eine beobachtete Mutation direkt oder über einen Umweg erzielt worden ist.
- Diese Bedingung ist beispielsweise erfüllt, wenn w eine Metrik ist.

Metrik

Definition

Sei M eine beliebige Menge.
Eine Funktion $w : M \times M \to \mathbb{R}_+$ heißt Metrik auf M, wenn die folgenden Bedingungen erfüllt sind:

(M1) $\forall x, y \in M : w(x, y) = 0 \Leftrightarrow x = y$ (Definitheit)

(M2) $\forall x, y \in M : w(x, y) = w(y, x)$ (Symmetrie)

(M3) $\forall x, y, z \in M : w(x, z) \leq w(x, y) + w(y, z)$ (Dreiecksungleichung)

Lemma

Ist $w : \overline{\Sigma} \times \overline{\Sigma} \to \mathbb{R}_+$ *eine Metrik, dann ist auch* $d_w : \overline{\Sigma}^* \times \overline{\Sigma}^* \to \mathbb{R}_+$ *eine Metrik.*

Restriktion

Definition (Restriktion)

Sei $u \in \overline{\Sigma}^*$. Dann sei die Restriktion von u auf Σ mit Hilfe eines Homomorphismus h wie folgt definiert

$$u|_\Sigma = h(u), \text{ wobei}$$

$$\begin{aligned} h(a) &= a && \text{für alle } a \in \Sigma, \\ h(-) &= \varepsilon, \\ h(u'u'') &= h(u')h(u'') && \text{für alle } u', u'' \in \Sigma^*. \end{aligned}$$

Die Restriktion von $u \in \overline{\Sigma}^*$ auf Σ ist also nichts anderes als das Löschen aller Leerzeichen (–) aus u.

Alignment

Definition (Alignment)

Ein (paarweises) Alignment ist ein Paar $(\overline{a}, \overline{b}) \in \overline{\Sigma}^* \times \overline{\Sigma}^*$ mit

- $|\overline{a}| = |\overline{b}|$ und
- $\overline{a}_i \neq - \neq \overline{b}_i$ für alle $i \in [1 : |\overline{a}|]$ mit $a_i = b_i$.

$(\overline{a}, \overline{b})$ ist ein Alignment für $a, b \in \Sigma^*$, wenn $\overline{a}|_\Sigma = a$ und $\overline{b}|_\Sigma = b$.

Beispiel

$$\begin{array}{cccccccc} A & - & G & G & C & A & T & T \\ A & G & C & G & C & - & T & T \end{array}$$

Alignment von *AGGCATT* mit *AGCGCTT*
Dieses Alignment hat Distanz 3,
es gibt jedoch ein besseres mit Distanz 2.

Alignment-Kosten und Alignment-Distanz

Definition

Sei $\overline{w} : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ eine Kostenfunktion (für einzelne Zeichen).

Die Notation von $\overline{w}$ wird wie folgt auf Sequenzen erweitert, um die (Gesamt-)Kosten eines Alignments $(\overline{a}, \overline{b})$ für (a, b) zu definieren:

$$\overline{w}(\overline{a}, \overline{b}) = \sum_{i=1}^{|\overline{a}|} \overline{w}(\overline{a}_i, \overline{b}_i).$$

Die Alignment-Distanz von $a, b \in \Sigma^*$ ist definiert als

$$\overline{d}_{\overline{w}}(a, b) := \min\left\{\overline{w}(\overline{a}, \overline{b}) \ : \ (\overline{a}, \overline{b}) \text{ ist Alignment für } a, b\right\}.$$

Beziehung zwischen Edit- und Alignment-Distanz

Lemma

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ *eine Kostenfunktion (die hier sowohl für die Edit-Operationen als auch für das Alignment benutzt wird, also* $w = \overline{w}$*) und seien* $a, b \in \Sigma^*$.

Für jedes Alignment $(\overline{a}, \overline{b})$ *von* a *und* b *gibt es eine Folge* s *von Edit-Operationen, so dass* $a \overset{s}{\Rightarrow} b$ *und* $w(s) = w(\overline{a}, \overline{b})$

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Sei $(\overline{a}, \overline{b})$ ein Alignment für a und b.
Betrachte die Folge $s = (s_1, \ldots, s_{|\overline{a}|})$ von Edit-Operationen mit $s_i = (\overline{a}_i, \overline{b}_i)$.
Offensichtlich gilt: $a \overset{s}{\Rightarrow} b$. Für die Edit-Kosten erhalten wir somit:

$$w(s) = \sum_{i=1}^{|\overline{a}|} w(s_i) = \sum_{i=1}^{|\overline{a}|} w(\overline{a}_i, \overline{b}_i) = w(\overline{a}, \overline{b})$$

□

Beziehung zwischen Edit- und Alignment-Distanz

Aus diesem Lemma folgt sofort, dass die Edit-Distanz von zwei Zeichenreihen höchstens so groß ist wie die Alignment-Distanz.

Folgerung

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ *eine Kostenfunktion, dann gilt für alle* $a, b \in \Sigma^*$*:*

$$d_w(a,b) \leq \overline{d}_w(a,b)$$

Beziehung zwischen Edit- und Alignment-Distanz

Lemma

Sei $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ *eine metrische Kostenfunktion und seien* $a, b \in \Sigma^*$.

Für jede Folge von Edit-Operationen mit $a \overset{s}{\Rightarrow} b$ *gibt es ein Alignment* $(\overline{a}, \overline{b})$ *von* a *und* b, *so dass* $w(\overline{a}, \overline{b}) \leq w(s)$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

(durch Induktion über $n = |s|$)

Induktionsanfang ($n = 0$):

- Aus $|s| = 0$ folgt, dass $s = \epsilon$.
- Also ist $a = b$ und $w(s) = 0$.
- Wir setzen nun $\overline{a} = a = b = \overline{b}$ und erhalten ein Alignment $(\overline{a}, \overline{b})$ für a und b mit $w(\overline{a}, \overline{b}) = 0 \leq w(s)$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Induktionsschritt ($n \to n+1$):

- Sei $s = (s_1, \ldots, s_n, s_{n+1})$ eine Folge von Edit-Operationen mit $a \overset{s}{\Rightarrow} b$.
- Sei nun $s' = (s_1, \ldots, s_n)$ und $a \overset{s'}{\Rightarrow} c \xrightarrow{s_{n+1}} b$ für ein $c \in \Sigma^*$.
- Aus der Induktionsvoraussetzung folgt nun, dass es ein Alignment $(\overline{a}, \overline{c})$ von a, c gibt, so dass $w(\overline{a}, \overline{c}) \leq w(s')$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Betrachte zuerst den Fall, dass die letzte Edit-Operation $s_{n+1} = (x, y)$ eine Substitution, ein Match oder eine Deletion ist, d.h. $x \in \Sigma$ und $y \in \overline{\Sigma}$.
- Wir können dann (wie in der Abbildung) ein Alignment für a und b erzeugen, indem wir die Zeichenreihe $\overline{b}$ geeignet aus $\overline{c}$ unter Verwendung der Edit-Operation(x, y) umformen.

$\overline{a}$		$*$	
$\overline{c}$		x	
$\overline{b}$		y	

$$\overline{a}|_\Sigma = a$$
$$\overline{c}|_\Sigma = c$$
$$\overline{b}|_\Sigma = b$$

Skizze: $s_{n+1} = (x, y)$ ist eine Substitution, Match oder Deletion

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Es gilt dann:

$$
\begin{aligned}
w(\overline{a},\overline{b}) &= w(\overline{a},\overline{c}) \underbrace{-w(\overline{a}_i,\overline{c}_i) + w(\overline{a}_i,\overline{b}_i)}_{\leq w(\overline{c}_i,\overline{b}_i)} \\
&\quad \text{aufgrund der Dreiecksungleichung} \\
&\quad \text{d.h., } w(\overline{a}_i,\overline{b}_i) \leq w(\overline{a}_i,\overline{c}_i) + w(\overline{c}_i,\overline{b}_i) \\
&\leq \underbrace{w(\overline{a},\overline{c})}_{\leq w(s')} + \underbrace{w(\overline{c}_i,\overline{b}_i)}_{= w(s_{n+1})} \\
&\leq w(s') + w(s_{n+1}) = w(s)
\end{aligned}
$$

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

Den Fall $\overline{a}_i = \overline{b}_i = -$ muss man gesondert betrachten.
Hier wird das verbotene Alignment von Leerzeichen im Alignment $(\overline{a}, \overline{b})$ eliminiert:

$$\begin{aligned} w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) - w(\overline{a}_i, \overline{c}_i) \\ &\leq w(\overline{a}, \overline{c}) + w(\overline{c}_i, \overline{b}_i) \\ &\leq w(s') + w(s_{n+1}) = w(s) \end{aligned}$$

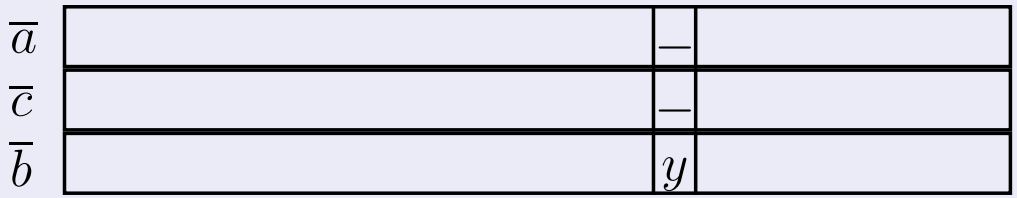
Beziehung zwischen Edit- und Alignment-Distanz

Beweis.

- Es bleibt noch der Fall, wenn $s_{n+1} = (-, y)$ mit $y \in \Sigma$ eine Insertion ist.
- Dann erweitern wir das Alignment $(\overline{a}, \overline{c})$ von a und c zu einem eigentlich *unzulässigen Alignment* $(\overline{a'}, \overline{c'})$ von a und c wie folgt.
- Es gibt ein $i \in [0 : |b|]$ mit $b_i = y$ und $b = c_1 \cdots c_i \cdot y \cdot c_{i+1} \cdots c_{|a|}$.
- Sei j die Position, nach der das Symbol y in $\overline{c}$ eingefügt wird.
- Dann setzen wir $\overline{a} = \overline{a}_1 \cdots \overline{a}_j \cdot - \cdot \overline{a}_{j+1} \cdots \overline{a}_{|\overline{c}|}$, $\overline{c} = \overline{c}_1 \cdots \overline{c}_j \cdot - \cdot \overline{c}_{j+1} \cdots \overline{c}_{|\overline{c}|}$ und $\overline{b} = \overline{c}_1 \cdots \overline{c}_j \cdot y \cdot \overline{c}_{j+1} \cdots \overline{c}_{|\overline{c}|}$.

Beziehung zwischen Edit- und Alignment-Distanz

Beweis.



Skizze: $s_{n+1} = (x, y)$ ist eine Insertion

- $(\overline{a}, \overline{c})$ ist jetzt wegen der Spalte $(-, -)$ kein Alignment mehr.
- jedoch nur noch interessant: Alignment $(\overline{a}, \overline{b})$ von a und b

$$\begin{aligned} w(\overline{a}, \overline{b}) &= w(\overline{a}, \overline{c}) + w(-, y) \\ &\quad \text{Nach Induktionsvoraussetzung} \\ &\leq w(s') + w(s_{n+1}) = w(s) \end{aligned}$$

□

Beziehung zwischen Edit- und Alignment-Distanz

Also ist die Alignment-Distanz durch die Edit-Distanz beschränkt, falls die zugrunde liegende Kostenfunktion eine Metrik ist:

Folgerung

Ist $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ *eine metrische Kostenfunktion, dann gilt für alle* $a, b \in \Sigma^*$*:*

$$\overline{d}_w(a, b) \leq d_w(a, b).$$

Zusammengefasst ergibt sich für den Fall einer metrischen Kostenfunktion die Gleichheit von Edit- und Alignment-Distanz:

Theorem

Ist w *eine Metrik, dann gilt für* $a, b \in \Sigma^*$*:* $d_w(a, b) = \overline{d}_w(a, b)$.

Globale Alignments

Problem

GLOBALES ALIGNMENT

Eingabe: $s \in \Sigma^n$, $t = \in \Sigma^m$,
w: Kostenfunktion für Distanz- oder Ähnlichkeitsmaß μ.

Gesucht: *optimales globales Alignment* $(\overline{s}, \overline{t})$ *für s und t, d.h.*
$\mu(s,t) = w(\overline{s}, \overline{t})$

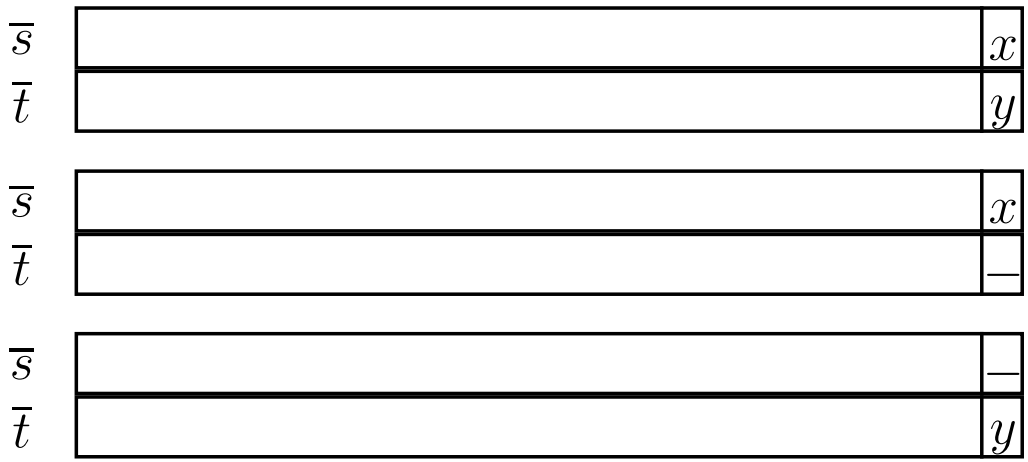
- zunächst mit Distanzmaßen
- Abwandlung für Ähnlichkeitsmaße meist offensichtlich

Optimale Alignments

- Annahme: sei $(\overline{s}, \overline{t})$ ein optimales Alignment für s und t

- 3 Möglichkeiten, wie die letzte Spalte $(\overline{t}_{|\overline{t}|}, \overline{s}_{|\overline{s}|})$ dieses optimalen Alignments aussehen kann:
  - Entweder wurde $x = s_n$ durch $y = t_m$ substituiert (oben)
  - oder es wurde das letzte Zeichen $x = s_n$ in s gelöscht (Mitte)
  - oder es wurde das letzte Zeichen $y = t_m$ in t eingefügt (unten).

Optimale Alignments



Skizze: Optimales Alignment mit Substitution/Match, Insertion bzw. Deletion am Ende

- In allen drei Fällen ist das Alignment, das durch Streichen der letzten Spalte entsteht, also $(\overline{s}_1 \cdots \overline{s}_{|\overline{s}|-1}, \overline{t}_1 \cdots \overline{t}_{|\overline{t}|-1})$, ebenfalls ein optimales Alignment für
  - $s_1 \cdots s_{n-1}$ mit $t_1 \cdots t_{m-1}$,
  - $s_1 \cdots s_{n-1}$ mit $t_1 \cdots t_m$ bzw.
  - $s_1 \cdots s_n$ mit $t_1 \cdots t_{m-1}$.

Optimale Alignments

Lemma

Sei $(\overline{a}, \overline{b})$ ein optimales Alignment für $a, b \in \Sigma^*$ für ein gegebenes Distanz- oder Ähnlichkeitsmaß.
Für $i \leq j \in [1 : |\overline{a}|]$ ist dann $(\overline{a}_i \cdots \overline{a}_j, \overline{b}_i \cdots \overline{b}_j)$ ein optimales Alignment für $a' = \overline{a}_i \cdots \overline{a}_j|_\Sigma$ und $b' = \overline{b}_i \cdots \overline{b}_j|_\Sigma$.

Optimale Alignments

Beweis.

- Sei $(\overline{a}, \overline{b})$ ein optimales Alignment für $a, b \in \Sigma^*$.
- Für einen Widerspruchsbeweis nehmen wir an, dass $(\overline{a}_i \cdots \overline{a}_j, \overline{b}_i \cdots \overline{b}_j)$ kein optimales Alignment für $a', b' \in \Sigma$ ist.
- Sei also $(\widetilde{a'}, \widetilde{b'})$ ein optimales Alignment für a' und b'.
- Dann ist aber nach Definition der Kosten eines Alignments (unabhängig, ob Distanz- oder Ähnlichkeitsmaß) das Alignment

$$(\overline{a}_1 \cdots \overline{a}_{i-1} \cdot \widetilde{a'} \cdot \overline{a}_{j+1} \cdots \overline{a}_n, \overline{b}_1 \cdots \overline{b}_{i-1} \cdot \widetilde{b'} \cdot \overline{b}_{j+1} \cdots \overline{b}_n)$$

ein besseres Alignment als $(\overline{a}, \overline{b})$ (Widerspruch)

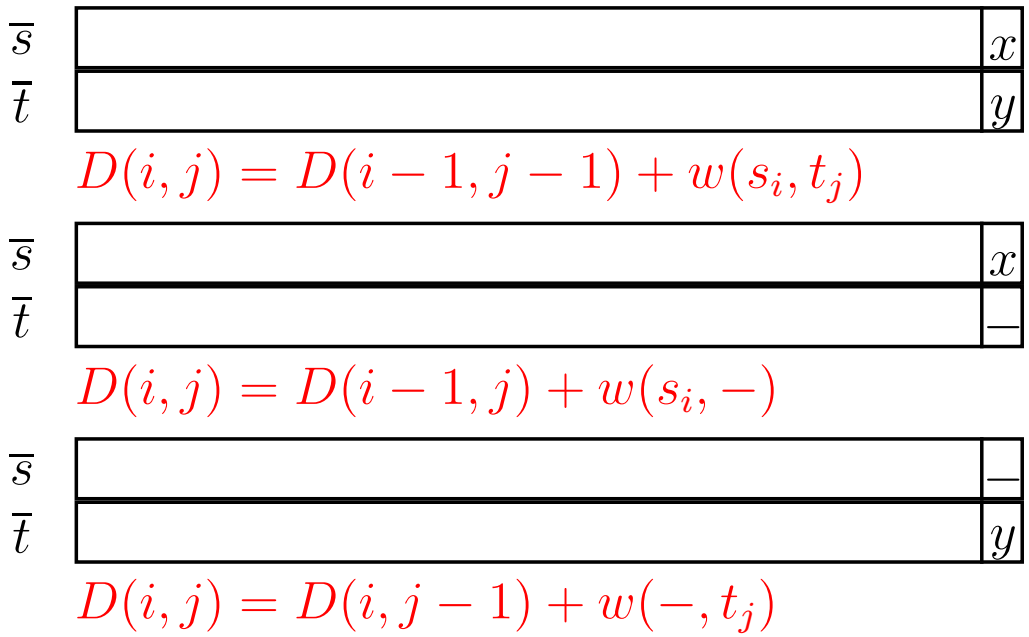
□

Needleman-Wunsch-Algorithmus

- berechnet ein optimales Alignment für zwei Sequenzen $s = s_1 \cdots s_n \in \Sigma^n$ und $t = t_1 \cdots t_m \in \Sigma^m$
- benutzt Matrix $D(i,j) = \mu(s_1 \cdots s_i, t_1 \cdots t_j)$, in der jeweils die Distanz eines optimalen Alignments für $s_1, \ldots s_i$ und $t_1, \ldots t_j$ gespeichert wird
- rekursive Berechnung der Matrix:

$$D(i,j) \quad = \quad \min \left\{ \begin{array}{lcl} D(i-1,j-1) & + & w(s_i,t_j), \\ D(i-1,j) & + & w(s_i,-), \\ D(i,j-1) & + & w(-,t_j) \end{array} \right\}.$$

Needleman-Wunsch-Algorithmus

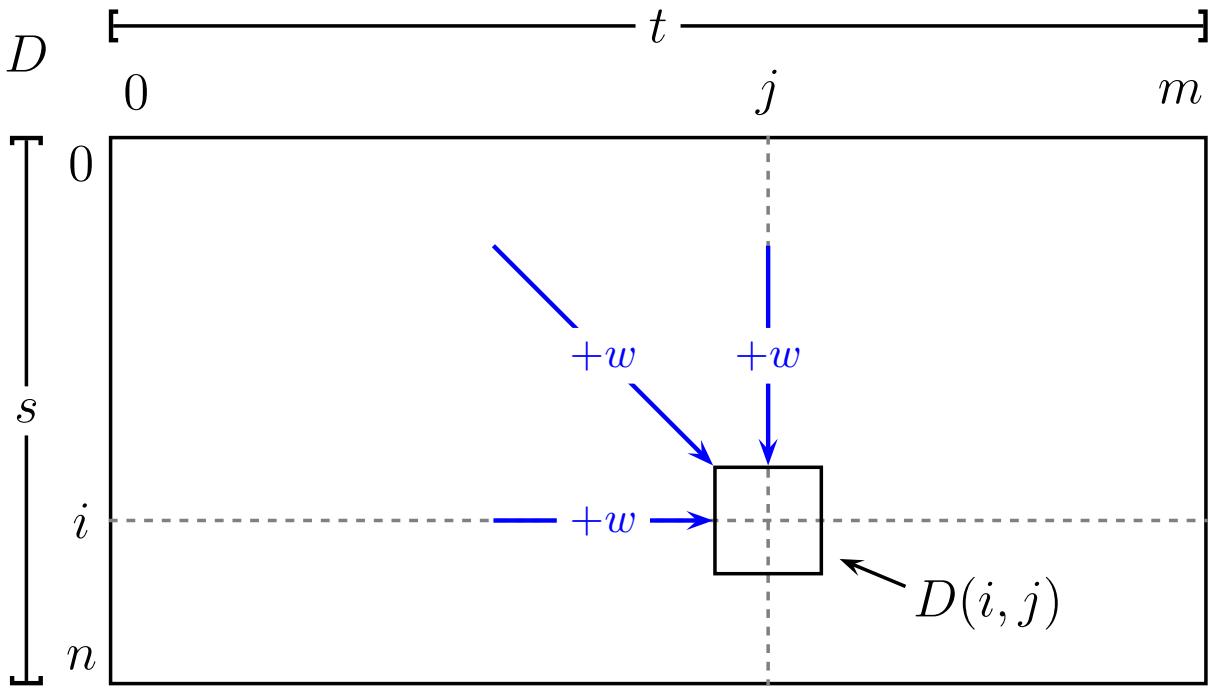


Skizze: Erweiterung eines optimales Alignment zu $s_1 \cdots s_i$ mit $t_1 \cdots t_j$

Needleman-Wunsch-Algorithmus

- 1. Fall: optimales Alignment für $s_1 \cdots s_{i-1}$ und $t_1 \cdots t_{j-1}$ ist bereits berechnet und in $D(i-1, j-1)$ abgelegt; für die Distanz eines Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_j$ müssen noch die Substitutionskosten von s_i durch t_j hinzuaddiert werden.
- 2. Fall: ein Zeichen in t wurde gelöscht. Distanz eines Alignments von $s_1 \cdots s_i$ mit $t_1 \cdots t_j$ besteht aus Kosten dieser Löschung und der Distanz des bereits berechneten optimalen Alignments für $s_1 \cdots s_{i-1}$ und $t_1 \cdots t_j$.
- 3. Fall: ein Zeichen wurde in die Sequenz t eingefügt. Zur Distanz des bereits berechneten optimalen Alignments für $s_1 \cdots s_i$ und $t_1 \cdots t_{j-1}$ müssen noch die Kosten für die Einfügung hinzuaddiert werden.
- Da das Optimum einer dieser Fälle ist, genügt es, aus allen drei möglichen Werten das Minimum auszuwählen (bei Ähnlichkeitsmaßen das Maximum).

Needleman-Wunsch-Algorithmus



Skizze: Berechnung optimaler Alignments nach Needleman-Wunsch

Needleman-Wunsch-Algorithmus

Prozedur SeqAlign(char $s[]$, int n, char $t[]$, int m)

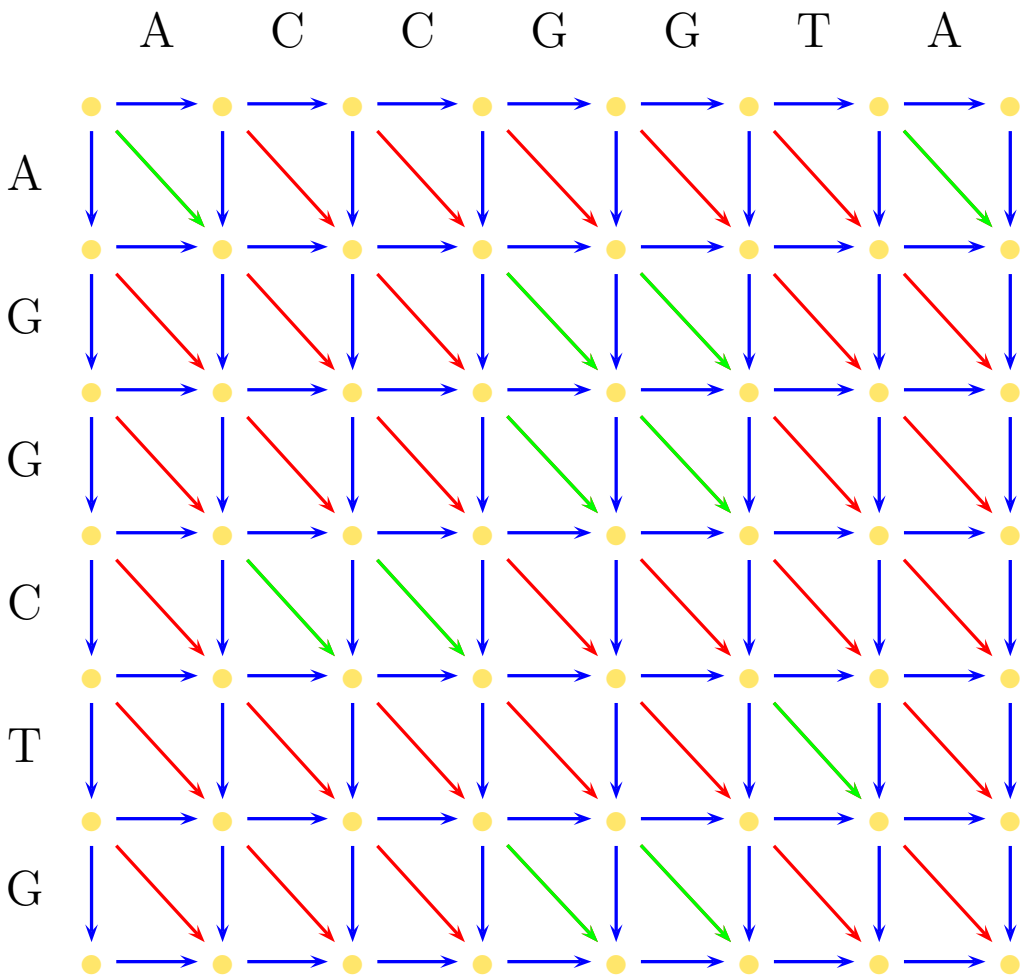
$D[0,0] := 0$;
for $(i := 1; i \leq n; i{+}{+})$ **do**
  $D[i,0] := D[i-1,0] + w(s_i, -)$;
for $(j := 1; j \leq m; j{+}{+})$ **do**
  $D[0,j] := D[0,j-1] + w(-, t_j)$;
for $(i := 1; i \leq n; i{+}{+})$ **do**
  for $(j := 1; j \leq m; j{+}{+})$ **do**

$$D[i,j] := \min \left\{ \begin{array}{l} D[i-1,j] + w(s_i, -), \\ D[i,j-1] + w(-, t_j), \\ D[i-1,j-1] + w(s_i, t_j) \end{array} \right\};$$

Needleman-Wunsch-Algorithmus: Beispiel

- Visualisierung am Beispiel: $s = AGGCTG$ und $t = ACCGGTA$
- 1. Schritt: Aufstellen des Edit-Graphen
- In Abhängigkeit von der jeweiligen Operation werden unterschiedliche Pfeile eingefügt:
  - blaue horizontale bzw. vertikale Pfeile: Insertionen bzw. Deletionen
  - rote diagonale Pfeile: Substitutionen
  - grüne diagonale Pfeile: Matches

Needleman-Wunsch-Algorithmus: Beispiel

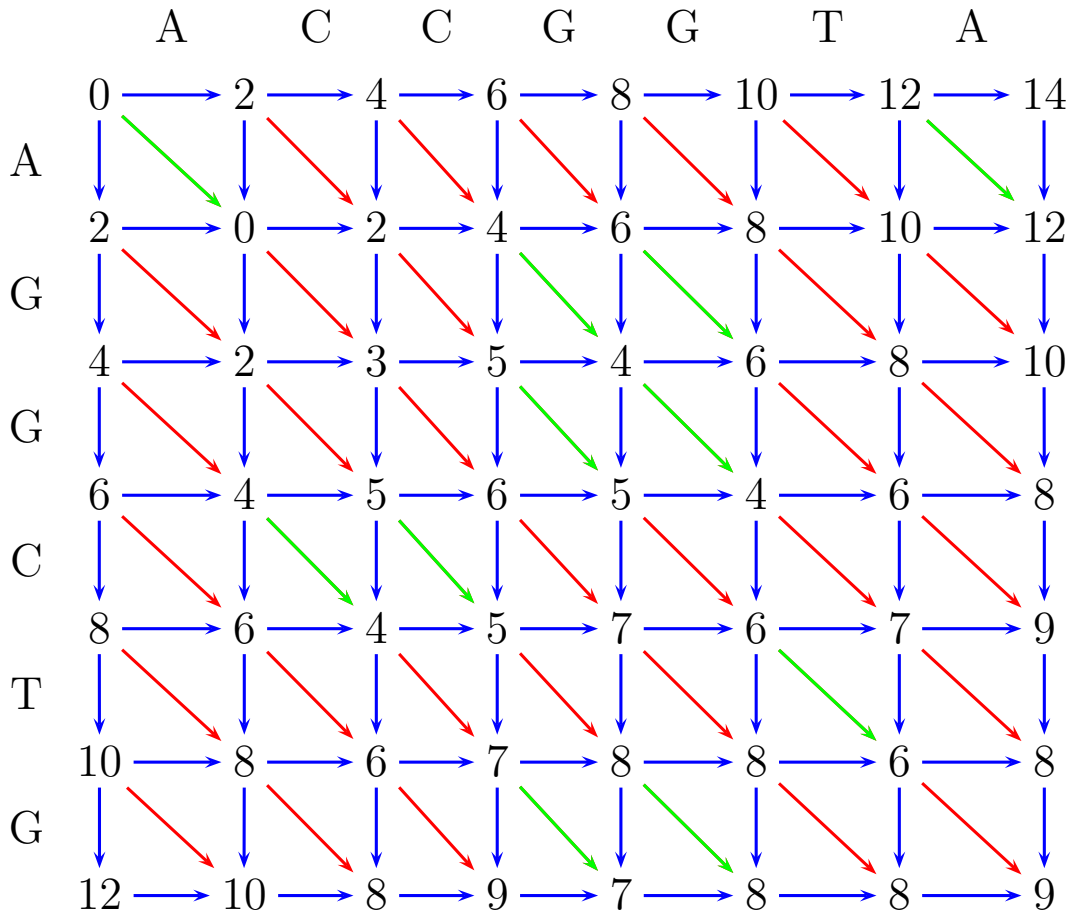


Skizze: Edit-Graph für s und t ohne Distanzen

Needleman-Wunsch-Algorithmus

- Nun werden die jeweiligen Distanzen des aktuellen Alignments (mit Hilfe der Rekursionsformel) eingetragen.
- In diesem Beispiel verursachen Einfügungen und Löschungen Kosten 2.
- Substitutionen verursachen hier Kosten 3.
- Ein Match verursacht keine Kosten (also 0).
- In der rechten unteren Ecke ($D(n, m)$) findet sich zum Schluss die Distanz eines optimalen Alignments.

Needleman-Wunsch-Algorithmus: Beispiel



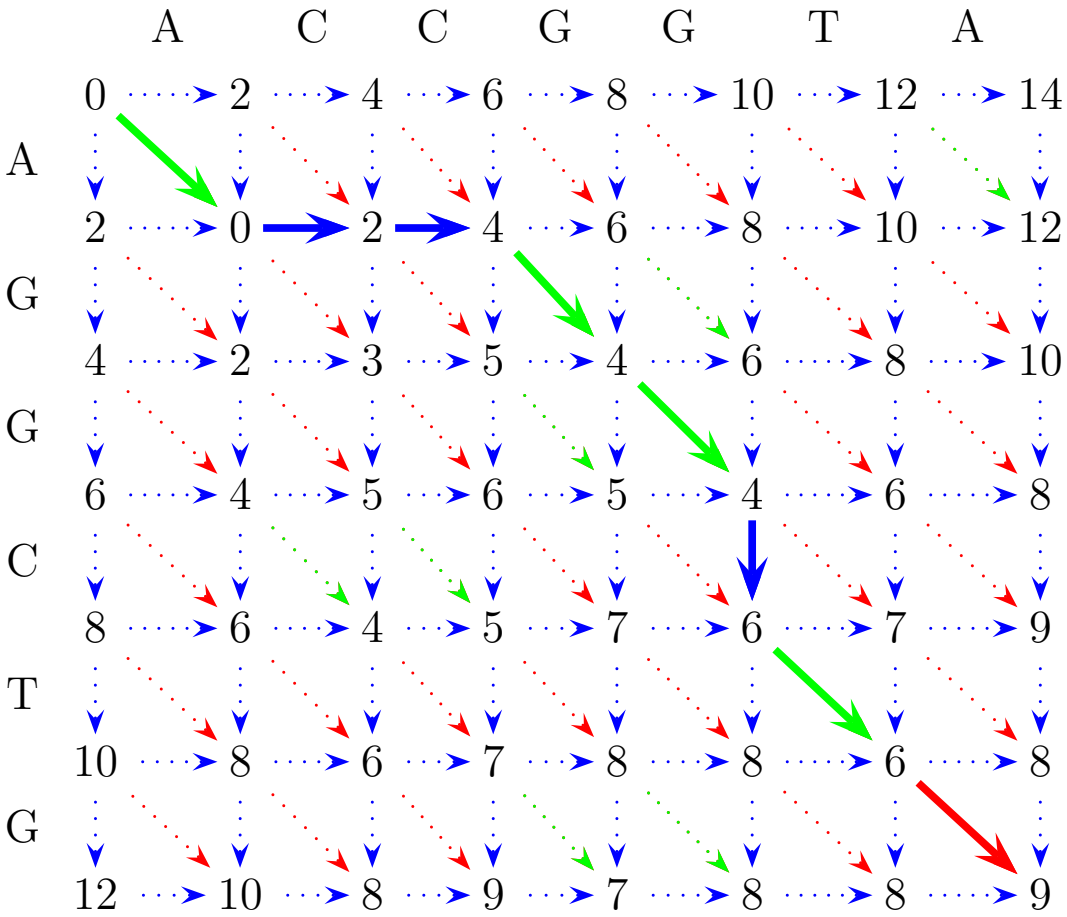
$\text{Match} = 0, \quad \text{Indel} = 2, \quad \text{Subst} = 3$

Skizze: Edit-Graph für s und t mit Distanzen

Needleman-Wunsch-Algorithmus

- Damit haben wir zwar den Wert eines optimalen Alignments für s und t bestimmt, kennen das Alignment an sich jedoch noch nicht.
- Dafür wird nun ein Pfad im Graphen von rechts unten nach links oben gesucht, der minimale Kosten verursacht.
- Gestartet wird in der rechten unteren Ecke.
- Als Vorgängerknoten wird nun der Knoten gewählt, der zuvor als Sieger bei der Minimum-Bildung hervorging. (Liefern mehrere Knoten die gleichen minimalen Kosten, kann einer davon frei gewählt werden. Meist geht man hier in einer vorher fest vorgegeben Reihenfolge bei Unentschieden vor, z.B. Insertion vor Substitution vor Deletion.)
- So verfährt man nun immer weiter, bis man in der linken oberen Ecke ankommt.

Needleman-Wunsch-Algorithmus: Beispiel



$\text{Match} = 0, \quad \text{Indel} = 2, \quad \text{Subst} = 3$

Skizze: Pfad im Edit-Graphen zur Bestimmung des Alignments

Needleman-Wunsch-Algorithmus: Beispiel

Nun kann man das optimale Alignment für s und t angeben. Dieses muss nur noch aus dem Edit-Graphen (entlang des gefundenen Pfades) abgelesen werden:

s :	A	–	–	G	G	C	T	G
t :	A	C	C	G	G	–	T	A

Beispiel: Optimales globales Alignment von s mit t

Needleman-Wunsch-Algorithmus

Theorem

Das optimale globale paarweise Sequenzen-Alignment für s und t mit $n = |s|$ und $m = |t|$ sowie die zugehörige Alignment-Distanz lassen sich mit dem Prinzip der Dynamischen Programmierung in Zeit $O(nm)$ und mit Platz $O(nm)$ berechnen.