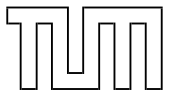
# Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2009



# Übersicht

# Beziehung zwischen Distanz- und Ähnlichkeitsmaßen

## Theorem

*Sei* $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ *eine (sinnvolle) Kostenfunktion für ein Distanzmaß* $d$ *und sei* $C \in \mathbb{R}_+$ *so gewählt, dass* $w' : \overline{\Sigma}_0^2 \to \mathbb{R}$ *mit*

$$\begin{aligned} w'(a,b) &= C - w(a,b) \\ w'(a,-) &= \frac{C}{2} - w(a,-) \end{aligned}$$

*für alle* $a, b \in \Sigma$ *eine (sinnvolle) Kostenfunktion für ein Ähnlichkeitsmaß* $s$ *ist.*
*Dann gilt für alle* $a, b \in \Sigma^*$ *folgende Beziehung zwischen Alignment-Distanz und Alignment-Ähnlichkeit:*

$$\overline{d}_w(a,b) + s_{w'}(a,b) = \frac{C}{2}(|a| + |b|).$$

# Beziehung zwischen Distanz- und Ähnlichkeitsmaßen

## Beweis.

Sei $\mathcal{A}(a, b)$ die Menge aller Alignments von $a$ und $b$ sowie

$$\begin{aligned} M(\overline{a}, \overline{b}) &= \left\{ i \in [1 : |\overline{a}|] \ : \ \overline{a}_i, \overline{b}_i \in \Sigma \right\} \\ I(\overline{a}, \overline{b}) &= [1 : |\overline{a}|] \setminus M(\overline{a}, \overline{b}). \end{aligned}$$

$\mu(\overline{a}, \overline{b}) = |M(\overline{a}, \overline{b})|$ bzw. $\iota(\overline{a}, \overline{b}) = |I(\overline{a}, \overline{b})|$ bezeichne damit die Anzahl von Matches und Substitutionen bzw. von Indel-Operationen eines Alignments $(\overline{a}, \overline{b})$.

Für jedes Alignment $(\overline{a}, \overline{b})$ für $a, b \in \Sigma^*$ gilt:

$$|a| + |b| = 2\mu(\overline{a}, \overline{b}) + \iota(\overline{a}, \overline{b}).$$

# Beziehung zwischen Distanz- und Ähnlichkeitsmaßen

## Beweis.

$$
\begin{aligned}
& s_{w'}(a,b) \\
= & \max_{(\overline{a},\overline{b})\in\mathcal{A}(a,b)} \left\{ \sum_{i\in M(\overline{a},\overline{b})} w'(\overline{a}_i,\overline{b}_i) + \sum_{i\in I(\overline{a},\overline{b})} w'(\overline{a}_i,\overline{b}_i) \right\} \\
= & \max_{(\overline{a},\overline{b})\in\mathcal{A}(a,b)} \left\{ \sum_{i\in M(\overline{a},\overline{b})} (C - w(\overline{a}_i,\overline{b}_i)) + \sum_{i\in I(\overline{a},\overline{b})} \left(\frac{C}{2} - w(\overline{a}_i,\overline{b}_i)\right) \right\} \\
= & \max_{(\overline{a},\overline{b})\in\mathcal{A}(a,b)} \left\{ C\cdot\mu(\overline{a},\overline{b}) + \frac{C}{2}\cdot\iota(\overline{a},\overline{b}) - \sum_{i\in M(\overline{a},\overline{b})} w(\overline{a}_i,\overline{b}_i) - \sum_{i\in I(\overline{a},\overline{b})} w(\overline{a}_i,\overline{b}_i) \right\}
\end{aligned}
$$

# Beziehung zwischen Distanz- und Ähnlichkeitsmaßen

## Beweis.

$$
\begin{aligned}
& s_{w'}(a,b) \\
&= \max_{(\overline{a},\overline{b})\in\mathcal{A}(a,b)} \left\{ \frac{C}{2}(|a|+|b|) - \sum_{i\in M(\overline{a},\overline{b})} w(\overline{a}_i,\overline{b}_i) - \sum_{i\in S(\overline{a},\overline{b})} w(\overline{a}_i,\overline{b}_i) \right\} \\
&= \frac{C}{2}(|a|+|b|) - \min_{(\overline{a},\overline{b})\in\mathcal{A}(a,b)} \left\{ \sum_{i\in M(\overline{a},\overline{b})} w(\overline{a}_i,\overline{b}_i) + \sum_{i\in I(\overline{a},\overline{b})} w(\overline{a}_i,\overline{b}_i) \right\} \\
&= \frac{C}{2}(|a|+|b|) - \overline{d}_w(a,b)
\end{aligned}
$$

□

# Fazit

Unter bestimmten Voraussetzungen kann man also zwischen Edit-Distanz, Alignment-Distanz und Alignment-Ähnlichkeit wechseln, ohne in der Bewertung von Sequenz-Paaren andere Ergebnisse zu erhalten.

# Globale Alignments

## Problem

Globales Alignment

**Eingabe:** $s \in \Sigma^n$, $t = \in \Sigma^m$,
*w: Kostenfunktion für Distanz- oder Ähnlichkeitsmaß* $\mu$.
**Gesucht:** *optimales globales Alignment* $(\overline{s}, \overline{t})$ *für* $s$ *und* $t$, *d.h.*
$\mu(s,t) = w(\overline{s}, \overline{t})$

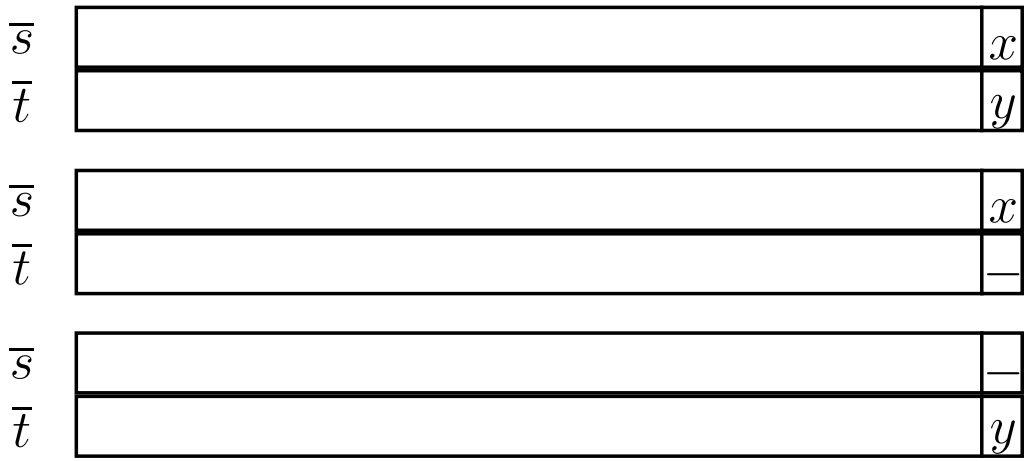
- zunächst mit Distanzmaßen
- Abwandlung für Ähnlichkeitsmaße meist offensichtlich

# Optimale Alignments

- Wir nehmen an, wir kennen schon ein optimales Alignment $(\overline{s}, \overline{t})$ für $s$ und $t$.

- Es gibt jetzt drei 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)

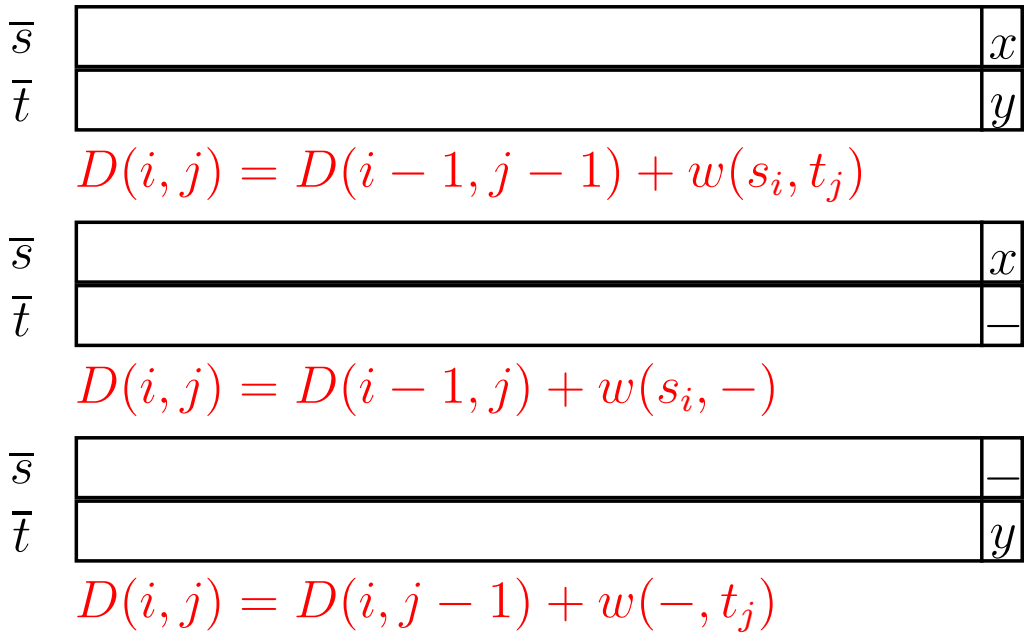
□

# Needleman-Wunsch-Algorithmus

- Das Verfahren von Needleman und Wunsch berechnet ein optimales Alignment für zwei Sequenzen $s = s_1 \cdots s_n \in \Sigma^n$ und $t = t_1 \cdots t_m \in \Sigma^m$.
- Dazu wird eine Matrix $D(i,j) = \mu(s_1 \cdots s_i, t_1 \cdots t_j)$ aufgestellt, in welcher jeweils die Distanz eines optimalen Alignments für $s_1, \ldots s_i$ und $t_1, \ldots t_j$ abgespeichert wird.
- Die Matrix kann rekursiv mit der folgenden Rekursionsformel berechnet werden:

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

# Needleman-Wunsch-Algorithmus

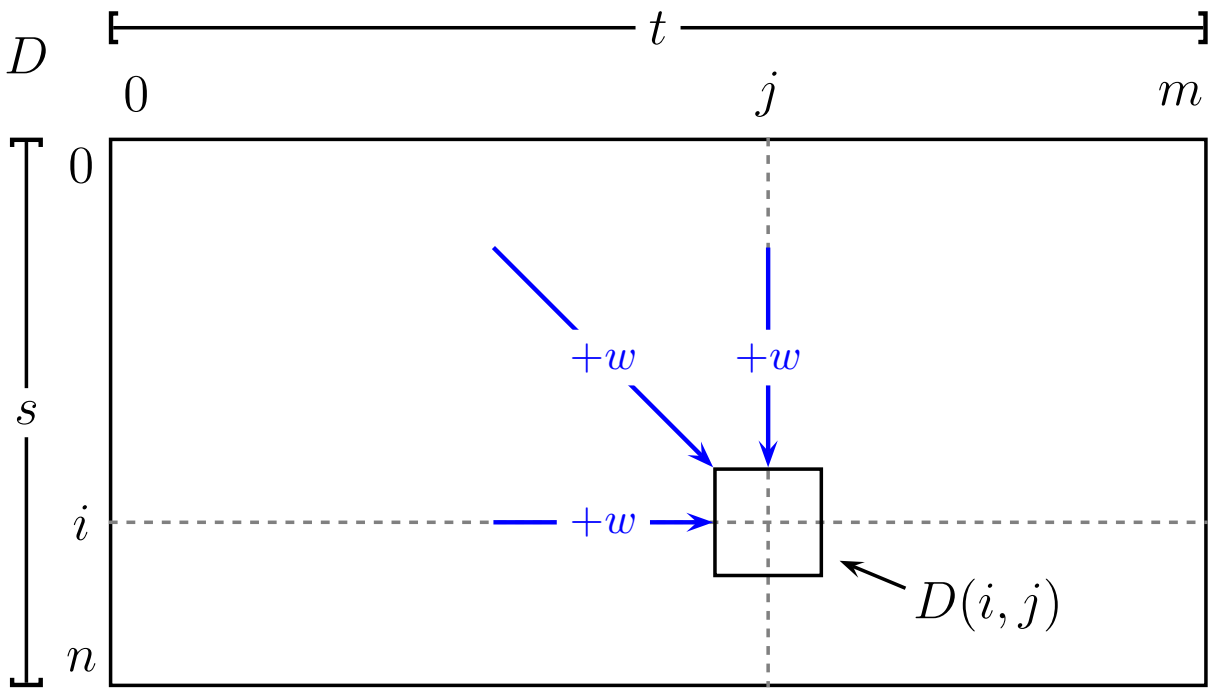



Skizze: Erweiterung eines optimales Alignment zu $s_1 \cdots s_i$ mit $t_1 \cdots t_j$

# 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

**Algorithmus 16** : SequenceAlignment(char $s[]$, int $n$, char $t[]$, int $m$)

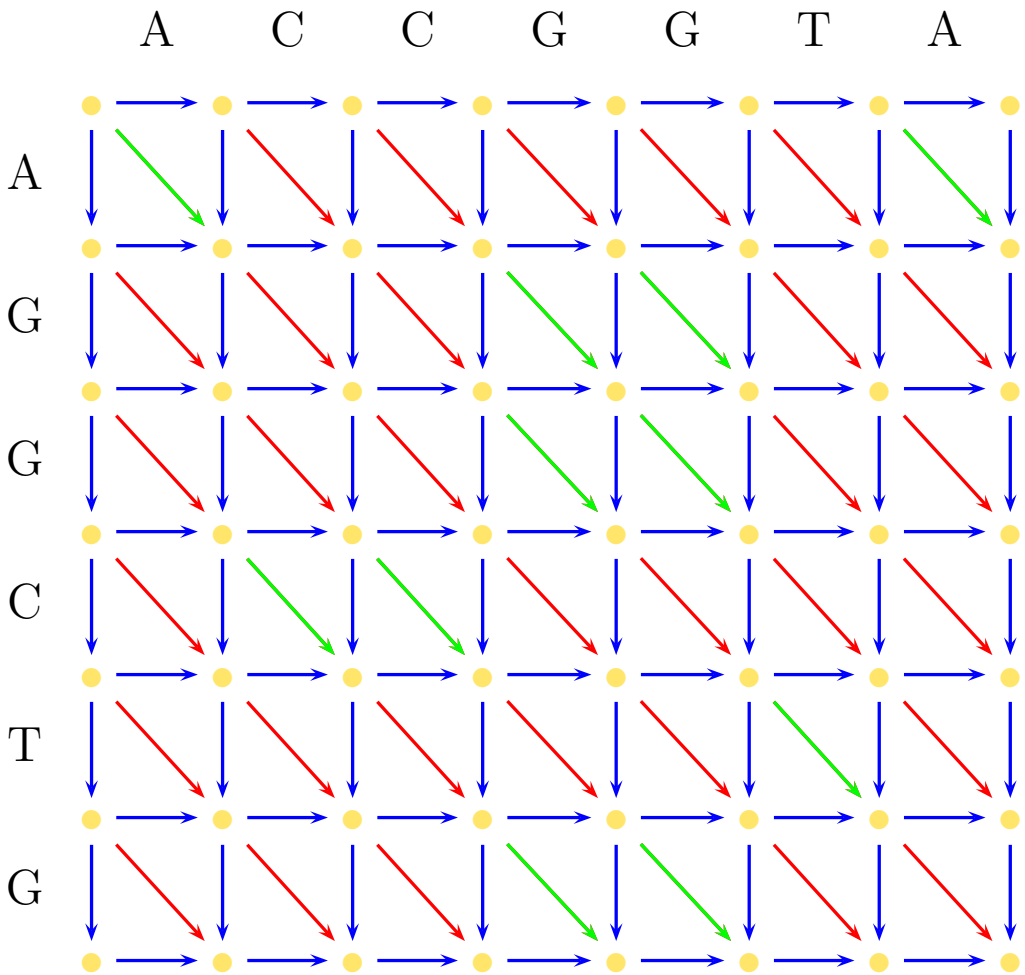
$D[0,0] := 0;$
**for** $(i := 1;\ i \leq n;\ i{+}{+})$ **do**
  $D[i,0] := D[i-1,0] + w(s_i, -);$
**for** $(j := 1;\ j \leq m;\ j{+}{+})$ **do**
  $D[0,j] := D[0,j-1] + w(-, t_j);$
**for** $(i := 1;\ i \leq n;\ i{+}{+})$ **do**
  **for** $(j := 1;\ j \leq m;\ j{+}{+})$ **do**

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

# 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

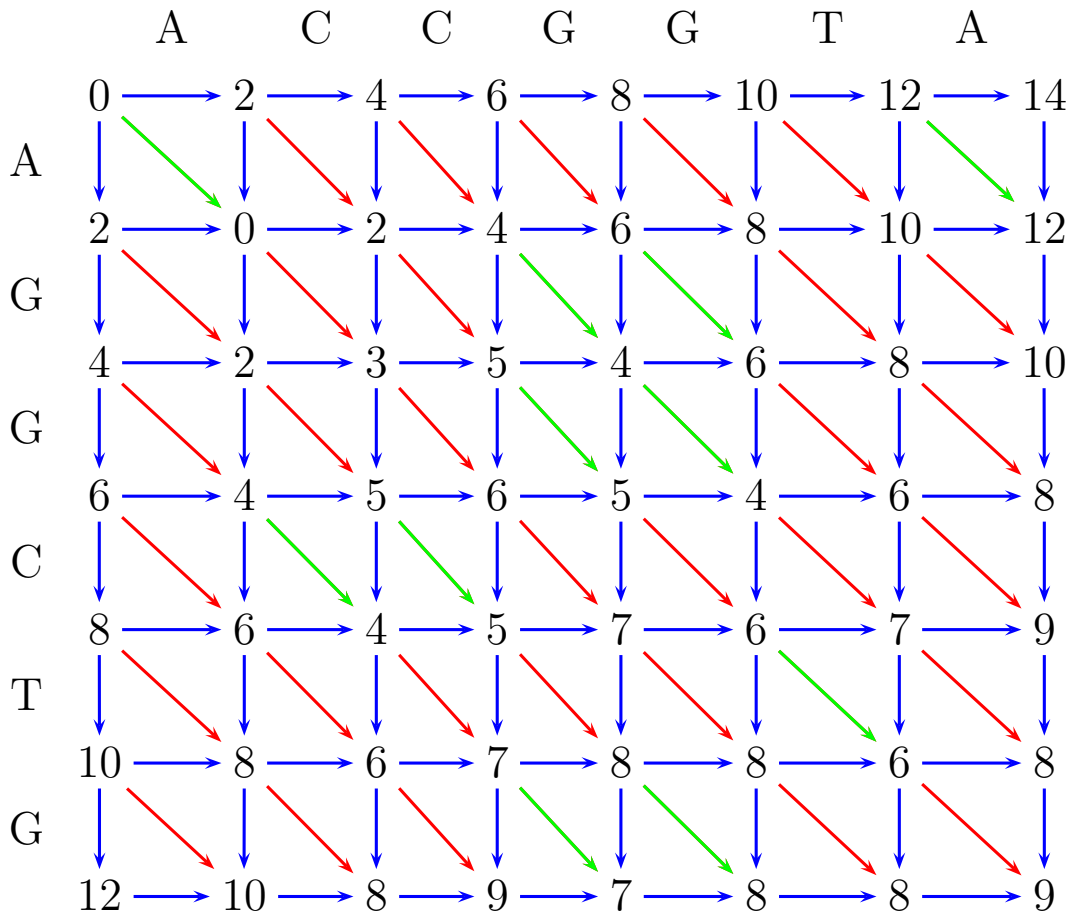



Skizze: Edit-Graph für $s$ und $t$ ohne Distanzen

# 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



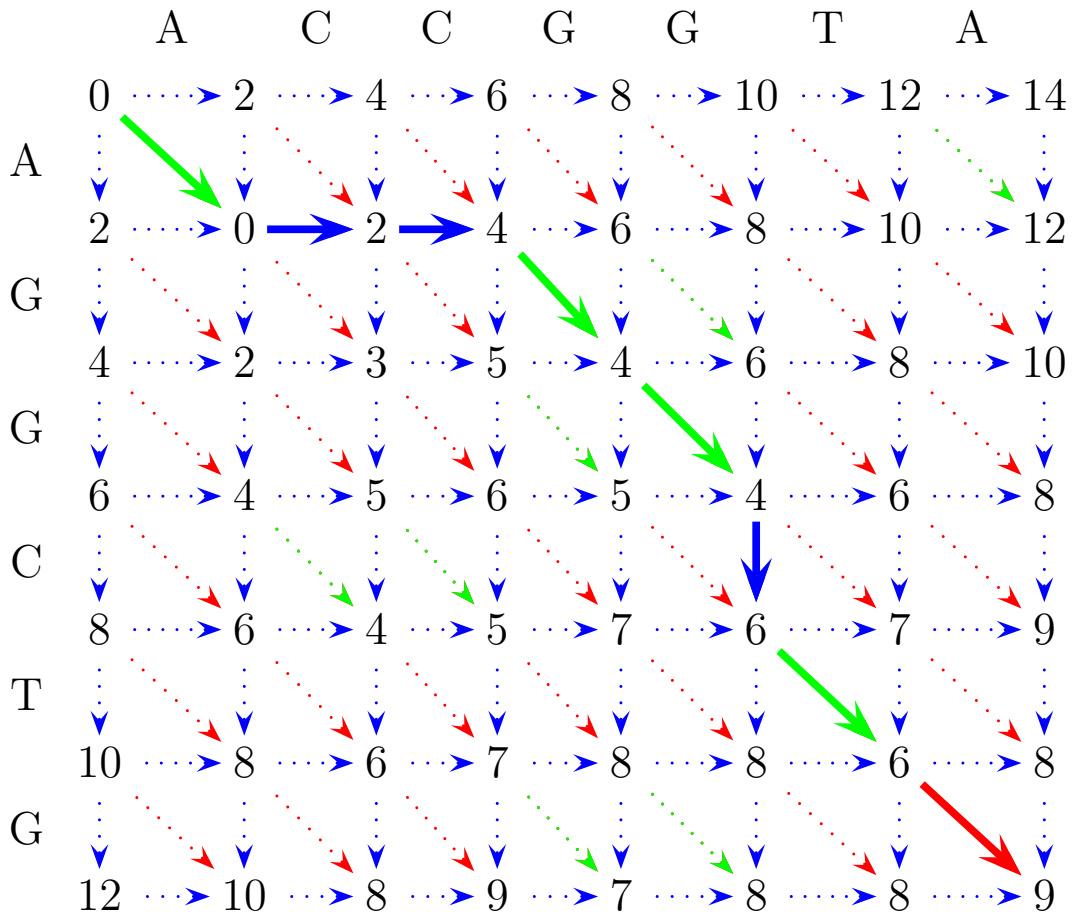

$$\text{Match} = 0, \quad \text{Indel} = 2, \quad \text{Subst} = 3$$

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 in Zeit $\mathcal{O}(nm)$ und mit Platz $\mathcal{O}(nm)$ berechnen.*

# Sequenzen-Alignment mit linearem Platz (Hirschberg)

- Bisher wurde zur Bestimmung eines optimalen Alignments für $s$ und $t$ Platz in der Größenordnung $O(nm)$ benötigt.
- Dies soll nun dahingehend optimiert werden, dass nur noch linear viel Platz gebraucht wird.

- Während der Berechnung der $D(i, j)$ wird immer nur die momentane Zeile $i$ und die unmittelbar darüber liegende Zeile $i - 1$ benötigt.
- Somit bietet es sich an, immer nur diese beiden relevanten Zeilen zu speichern und somit nur linear viel Platz zu beanspruchen.

# Alignment-Distanz mit linearem Platz

**Algorithmus 17** : SequenceAlignment(char $s[]$, int $n$, char $t[]$, int $m$)

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

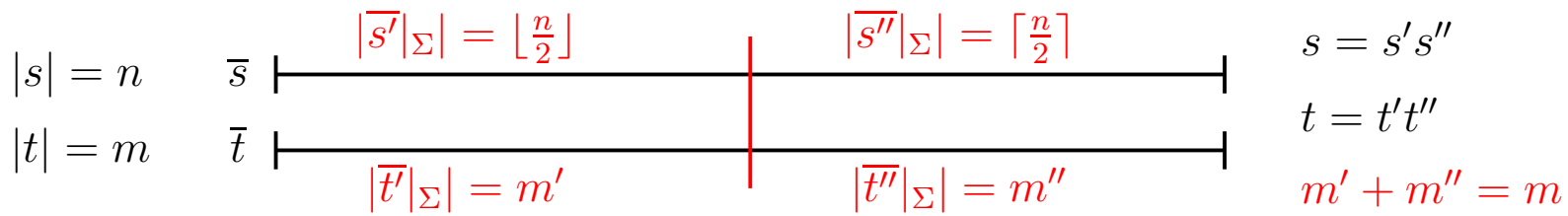
$$D[j] := \min \left\{ \begin{array}{l} D'[j] + w(s_i, -), \\ D[j-1] + w(-, t_j), \\ D'[j-1] + w(s_i, t_j) \end{array} \right\};$$

# Hirschberg-Verfahren

- Mit dem beschriebenen Verfahren lässt sich die Distanz der beiden Sequenzen $s$ und $t$ mit linearem Platz berechnen.
- Nachteil: das Alignment selbst kann nicht mehr einfach anhand des Edit-Graphen aufgebaut werden, da ja die nötigen Zwischenergebnisse nicht gespeichert wurden.

- Mit dem Verfahren nach Hirschberg kann ein optimales Sequenzen-Alignment selbst konstruiert werden, so dass nur linear viel Platz benutzt werden muss.
- Dazu betrachtet man zunächst einmal ein optimales paarweises Alignment von $s$ mit $t$, wie in der folgenden Abbildung angegeben.

# Hirschberg-Verfahren




Skizze: Optimales Alignment für $s$ und $t$

- Wir teilen nun dieses Alignment so in zwei Teil-Alignments auf, dass beide Teile in etwa die Hälfte der Zeichen aus $s$ enthalten: der erste Teil enthalte $\lfloor n/2 \rfloor$ und der zweite Teil $\lceil n/2 \rceil$ Zeichen aus $s$.
- Dieser Aufteilungsschritt muss nicht eindeutig sein, da das Alignment $\overline{s}$ von $s$ Leerzeichen zwischen dem Zeichen $s_{\lfloor n/2 \rfloor}$ und dem Zeichen $s_{\lfloor n/2 \rfloor + 1}$ enthalten kann.

# Hirschberg-Verfahren

- Im Folgenden bezeichnen wir mit $m'$ die Anzahl der Zeichen aus $t$ die bei der Aufteilung in der ersten Hälfte des Alignments sind und mit $m''$ die Anzahl der Zeichen in der zweiten Hälfte.
- Es gilt also $m = m' + m''$.
- Weiter bezeichne $\overline{s'}$ und $\overline{s''}$ bzw. $\overline{t'}$ und $\overline{t''}$ die Teile des Alignments nach der Aufteilung, d.h. $\overline{s} = \overline{s'} \cdot \overline{s''}$ und $\overline{t} = \overline{t'} \cdot \overline{t''}$.
- Ferner gilt $s' = \overline{s'}|_\Sigma$ und $s'' = \overline{s''}|_\Sigma$ bzw. $t' = \overline{t'}|_\Sigma$ und $t'' = \overline{t''}|_\Sigma$.
- Es gilt also $s = s' \cdot s''$ und $t = t' \cdot t''$ sowie $|s'| = \lfloor n/2 \rfloor$ bzw. $|s''| = \lceil n/2 \rceil$ und $|t'| = m'$ bzw. $|t''| = m'' = m - m'$.

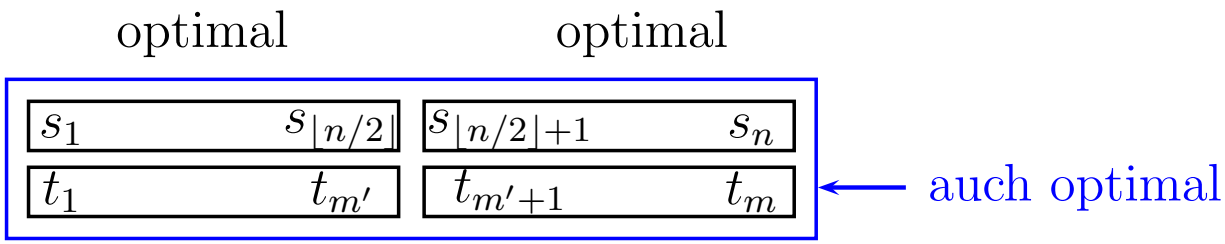
# Hirschberg-Verfahren

- letztes Lemma: $(\overline{s'}, \overline{t'})$ muss ein optimales Alignment für $s'$ mit $t'$ sein, und $(\overline{s''}, \overline{t''})$ ein optimales Alignment für $s''$ mit $t''$.
- Auch hier könnte man andererseits aus besseren Alignments für $s'$ mit $t'$ bzw. $s''$ mit $t''$ ein besseres Alignment für $s$ mit $t$ konstruieren.

- *algorithmische Idee*: Berechne optimale Alignments für $s_1 \cdots s_{\lfloor n/2 \rfloor}$ mit $t_1 \cdots t_{m'}$ sowie für $s_{\lfloor n/2 \rfloor + 1} \cdots s_n$ mit $t_{m'+1} \cdots t_m$.
- Dieser Ansatz führt uns auf einen Divide-and-Conquer-Algorithmus, da wir nun rekursiv für kleinere Eingaben ein Problem derselben Art lösen müssen.

# Hirschberg-Verfahren

- Der *Conquer-Schritt* ist dabei trivial, da man einfach die beiden erhaltenen Alignments für beide Teile zu einem großen Alignment für $s$ mit $t$ zusammenhängen muss (wie in der folgenden Abbildung)



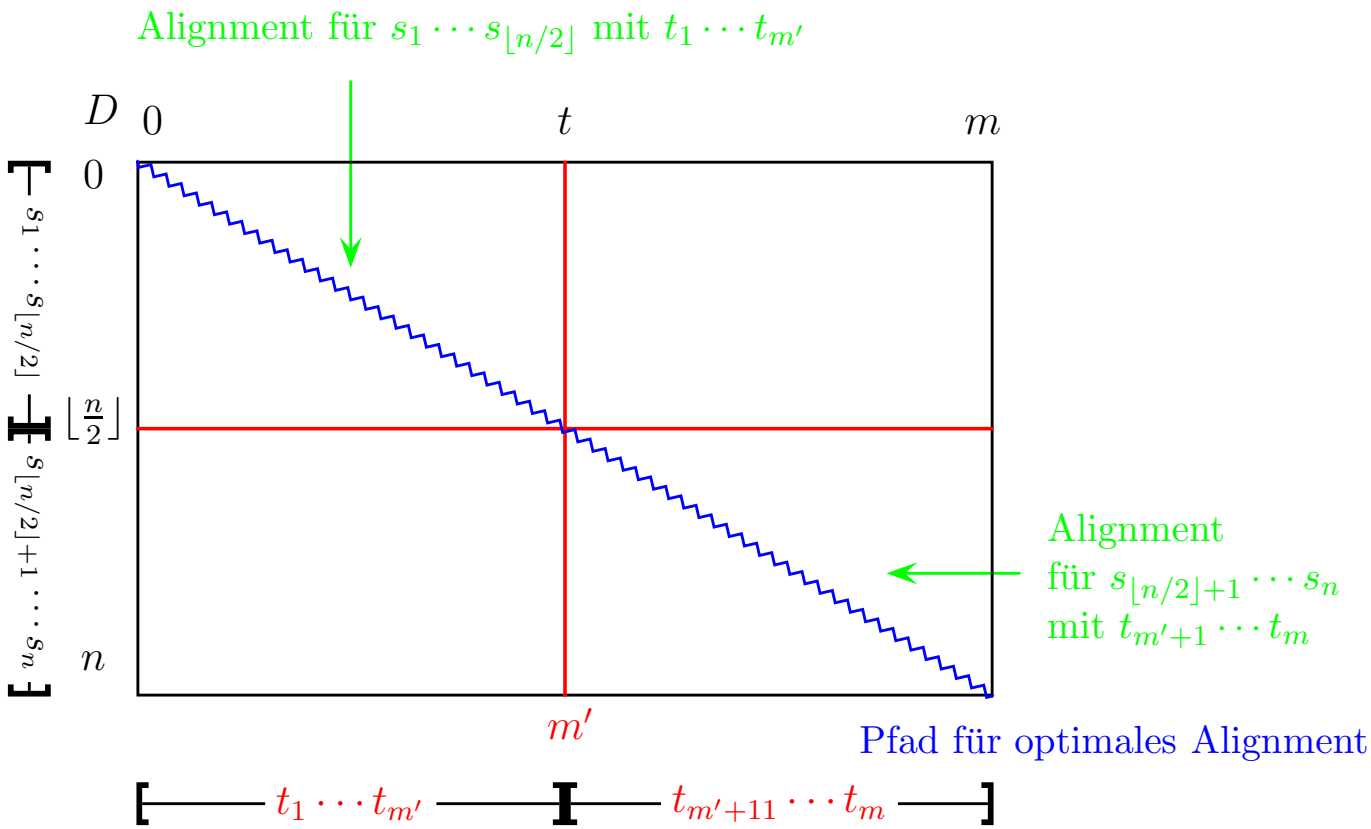

Conquer-Schritt des Hirschberg-Algorithmus

# Hirschberg-Verfahren

- Problem: Wir kennen $m'$ noch nicht.
- Wir haben $m'$ ja über ein optimales Alignment für $s$ mit $t$ definiert. Wenn wir also erst das optimale Alignment für $s$ mit $t$ berechnen wollen, kennen wir $m' = |t'|$ ja noch nicht.

- Wie findet man $m'$?
- Naiv: alle möglichen $m' \in [0 : m]$ ausprobieren
- Dies sind allerdings recht viele, die uns ja dann auch noch $m + 1$ rekursiv zu lösende Teilprobleme zur Aufgabe stellen.
- Allerdings wollen wir ja gar nicht die Alignments selbst berechnen, sondern nur wissen, wie $m'$ für ein optimales Alignment von $s$ mit $t$ aussieht.

# Hirschberg-Verfahren




Auffinden von $m'$ (Divide-Schritt bei Hirschberg)

# Hirschberg-Verfahren

- In dieser Abbildung ist die Tabelle $D(i, j)$ wieder als Edit-Graph bildlich dargestellt.
- Ein optimales Alignment entspricht in diesem Graphen einem Weg von $(0, 0)$ nach $(n, m)$.
- Offensichtlich muss dieser Weg die Zeile $\lfloor n/2 \rfloor$ an einem Punkt $m'$ schneiden.
- Dieser Punkt muss wie gesagt nicht eindeutig sein, da aufgrund von Insertionen der Pfad waagerecht innerhalb der Zeile $\lfloor n/2 \rfloor$ verlaufen kann.

# Hirschberg-Verfahren

- Der Pfad von $(0,0)$ nach $(n,m)$ zerfällt also in zwei Teile, nämlich in $(0,0)$ bis $(\lfloor n/2 \rfloor, m')$ und in $(\lfloor n/2 \rfloor, m')$ nach $(n,m)$.
- Diese beiden Teile entsprechen dann genau den vorher diskutierten optimalen Alignments für $s'$ mit $t'$ und $s''$ mit $t''$.
- Können wir die beiden Teilpfade jetzt schnell finden?
- Den ersten auf jeden Fall. Wir berechnen die optimalen Alignment-Distanzen von $s_1 \cdots s_{\lfloor n/2 \rfloor}$ mit $t_1 \cdots t_{m'}$ für alle $m' \in [0:m]$.
- Dies können wir mit unserem vorhin vorgestellten Algorithmus in linearem Platz berechnen.
- Dort haben wir als Endergebnis das Feld $D[j]$ erhalten, dass die Alignment-Distanzen von $s$ zu allen Präfixen von $t$ enthielt.

# Hirschberg-Verfahren

- Jetzt brauchen wir noch den zweiten Teil des Pfades.
- Dazu benötigen wir insbesondere die Alignment-Distanzen von $s_{\lfloor n/2 \rfloor+1} \cdots s_n$ mit $t_{m'+1} \cdots t_m$ für alle $m' \in [0 : m]$.
- Diese können wir jedoch mit demselben Algorithmus berechnen.
- Wir stellen uns die Tabelle nur um 180 Grad um den Mittelpunkt gedreht vor.
- Wir berechnen dann alle Alignment-Distanzen von $(s'')^R = s_n \cdots s_{\lfloor n/2 \rfloor+1}$ mit $(t'')^R = t_m \cdots t_{m'+1}$, wobei hier $x^R$ für eine Zeichenreihe $x = x_1 \cdots x_n$ die gespiegelte oder reversierte Zeichenreihe $x^R = x_n \cdots x_1$ bezeichnet.

# Hirschberg-Verfahren

**Theorem**

*Sei* $w : \overline{\Sigma}_0^2 \to \mathbb{R}_+$ *eine Kostenfunktion für ein Distanzmaß* $\overline{d}_w$ *bzw. für ein Ähnlichkeitsmaß* $s_w$ *und seien* $s, t \in \Sigma^*$.

*Dann gilt*

$$\overline{d}_w(s,t) = \overline{d}_w(s^R, t^R) \text{ bzw. } s_w(s,t) = s_w(s^R, t^R)$$

(Beweis trivial)

# Hirschberg-Verfahren

Bezeichne

- $V(\lfloor n/2 \rfloor, k)$ die minimale Alignment-Distanz von $s' = s_1 \cdots s_{\lfloor n/2 \rfloor}$ mit $t_1 \cdots t_k$ und
- $V'(\lfloor n/2 \rfloor, k)$ die minimale Alignment-Distanz von $s'' = s_{\lfloor n/2 \rfloor + 1} \cdots s_n$ mit $t_{k+1} \cdots t_m$
  (was nach dem letzten Satz gleichbedeutend mit der minimalen Alignment-Distanz von $(s'')^R$ mit $t_m \cdots t_{k+1}$ ist).

$$
\begin{aligned}
V\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) &= \overline{d}(s_1 \cdots s_{\lfloor n/2 \rfloor}, t_1 \cdots t_k), \\
V'\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) &= \overline{d}(s_{\lfloor n/2 \rfloor + 1} \cdots s_n, t_{k+1} \cdots t_m) \\
&= \overline{d}(s_n \cdots s_{\lfloor n/2 \rfloor + 1}, t_m \cdots t_{k+1}).
\end{aligned}
$$

# Hirschberg-Verfahren

Für das $m'$ mit optimaler Alignment-Distanz gilt:

$$\overline{d}(s,t) = V(\lfloor n/2 \rfloor, m') + V'(\lfloor n/2 \rfloor, m')$$

Wir können also $\overline{d}(s,t)$ und $m'$ wie folgt berechnen:

$$\begin{aligned} \overline{d}(s,t) &= \min_{k \in [0:m]} \left\{ V\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) + V'\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) \right\}, \\ m' &= \operatorname*{argmin}_{k \in [0:m]} \left\{ V\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) + V'\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) \right\}. \end{aligned}$$

Hierbei bezeichnet $\operatorname{argmin}$ einen Index-Wert, für den in der Menge das Minimum angenommen wird, d.h. es gilt für eine Menge $M = \{e_i : i \in I\}$ mit der zugehörigen Indexmenge $I$:

$$\min\{e_i : i \in I\} = e_{\operatorname{argmin}\{e_i : i \in I\}}.$$

# Hirschberg-Verfahren

- Somit können wir also für zwei Zeichenreihen $s$ und $t$ den Schnittpunkt $m'$ berechnen, der zwei optimale Teil-Alignments angibt, aus dem ein optimales Alignment für $s$ und $t$ berechnet wird:
  - Wir bestimmen also zunächst den Mittelpunkt des Pfades eines optimalen Alignments $(\lfloor n/2 \rfloor, m')$,
  - dann lösen wir rekursiv die beiden entstehenden Alignment-Probleme und
  - konstruieren zum Schluss aus diesen beiden Alignments ein neues optimales Alignment für $s$ und $t$.

# Hirschberg-Verfahren

1. Berechne die Werte optimaler Alignments für $s' = s_1 \cdots s_{\lfloor n/2 \rfloor}$ mit $t_1 \cdots t_k$ für alle $k \in [0 : m]$, d.h. $V(\lfloor n/2 \rfloor, k)$ für alle $k \in [0 : m]$.
   (In Wirklichkeit $s_1 \cdots s_{\lfloor n/2 \rfloor}$ mit $t_1 \cdots t_m$.)
2. Berechne die Werte optimaler Alignments für $s'' = s_{\lfloor n/2 \rfloor + 1} \cdots s_n$ mit $t_{k+1} \cdots t_m$ für alle $k \in [0 : m]$, d.h. $V'(\lfloor n/2 \rfloor, k)$ für alle $k \in [0 : m]$.
   (In Wirklichkeit $s_n \cdots s_{\lfloor n/2 \rfloor + 1}$ mit $t_m \cdots t_1$.)
3. Bestimme $m'$ mittels
   $m' = \operatorname{argmin} \left\{ V\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) + V'\left(\left\lfloor \frac{n}{2} \right\rfloor, k\right) \; : \; k \in [0 : m] \right\}$.
4. Löse rekursiv die beiden Alignment-Probleme für $s' = s_1 \cdots s_{\lfloor n/2 \rfloor}$ mit $t_1 \cdots t_{m'}$ sowie $s'' = s_{\lfloor n/2 \rfloor + 1} \cdots s_n$ mit $t_{m'+1} \cdots t_m$.

# Hirschberg-Verfahren

Offene Fragen:

- Wann brechen wir die Rekursion ab?
- Lassen sich diese Teilprobleme dann trivial lösen?

- Man bricht die Rekursion ab, wenn die erste Zeichenreihe, d.h. das Teilwort von $s$, die Länge 1 erreicht.

# Hirschberg-Verfahren

- Sei also $(s_\ell, t_p \cdots t_q)$ eine solche Eingabe, in der die Rekursion abbricht.
- Ist $q < p$, d.h. $t_p \cdots t_q = \epsilon$, dann wird eine Deletion von $s_\ell$ als Alignment

$$\begin{pmatrix} s_\ell \\ - \end{pmatrix}$$

zurückgeliefert.

# Hirschberg-Verfahren

- Ist andernfalls $p \leq q$, dann bestimmen wir ein Zeichen $t_i$ aus $t_p \cdots t_q$ (inklusive dem Leerzeichen) das mit $s_\ell$ den besten Score besitzt.
- Dazu bestimmen wir zuerst $k = \underset{i \in [p:q]}{argmin}\{w(s_\ell, t_i)\}$.
- Falls $w(s_\ell, t_k) \leq w(s_\ell, -) + w(t_k, -)$, dann liefern wir als Alignment

$$\begin{pmatrix} - & \cdots\cdots & - & s_\ell & - & \cdots\cdots & - \\ t_p & \cdots\cdots & t_{k-1} & t_k & t_{k+1} & \cdots\cdots & t_q \end{pmatrix}.$$

Andernfalls wird das Alignment

$$\begin{pmatrix} s_\ell & - \cdots\cdots & - \\ - & t_p & \cdots\cdots & t_q \end{pmatrix}$$

zurückgegeben.