UA RUHR

Algorithmen auf Sequenzen

Fehlertolerante Mustersuche: Distanz- und Ähnlichkeitsmaße

Sven Rahmann

Genominformatik
Universitätsklinikum Essen
Universität Duisburg-Essen
Universitätsallianz Ruhr

Einführung

UA RUHR

- Bisher: exakte Mustersuche
  (aber z.B. auch erweiterte Muster: M[ae][iy]er)
- Fehlertolerante Mustersuche wird an vielen Stellen verwendet:
  Rechtschreibkorrektur, Vorschläge passenderer Wörter,
  Bioinformatik: Genomvergleich, DNA-Read-Mapping
- Zur Definition des Problems benötigen wir ein Abstands- oder Ähnlichkeitsmaß zwischen Wörtern

Metrik

UA RUHR

Metriken sind spezielle Abstandsmaße;
sie sind nichtnegativ, null bei Gleichheit und erfüllen die Dreiecksungleichung.

Definition (Metrik)

Sei X eine Menge. Eine Funktion $d : X \times X \to \mathbb{R}_{\geq 0}$ heißt Metrik, genau dann wenn

1. $d(x, y) = 0$ genau dann, wenn $x = y$ (Definitheit),
2. $d(x, y) = d(y, x)$ für alle x, y (Symmetrie),
3. $d(x, y) \leq d(x, z) + d(z, y)$, für alle x, y, z (Dreiecksungleichung).

Hamming-Distanz

UA RUHR

Für Strings gleicher Länge bietet sich die Hamming-Distanz als Abstandsmaß an (nach Richard Wesley Hamming, 1915–1998).

Definition (Hamming-Distanz)

Für jedes Alphabet Σ und jedes $n \geq 0$ ist auf $X := \Sigma^n$ die Hamming-Distanz $d_{\mathrm{H}}(s, t)$ zwischen Strings s, t definiert als die Anzahl der Positionen, in denen sich s und t unterscheiden:

$$d_{\mathrm{H}}(s, t) := \left| \{ i \mid s_i \neq t_i \} \right|$$

Es handelt sich um eine Metrik auf Σ^n (Übung).
Für $|s| \neq |t|$ ist die Hamming-Distanz nicht definiert.

q-Gramm-Distanz

UA RUHR

Für beliebige Strings s, t kann man die Multimengen ihrer q-Gramme (Teilstrings der Länge q) vergleichen.

Definition (q-Gramm-Distanz)

Für einen String $s \in \Sigma^*$ und ein q-gram $x \in \Sigma^q$ sei $N_x(s)$ die Anzahl der Vorkommen von x in s. Dann ist die q-Gramm-Distanz zwischen s und t definiert als

$$d_{\mathrm{q}}(s, t) := \sum_{x \in \Sigma^q} |N_x(s) - N_x(t)|.$$

Dies ist im allgemeinen **keine** Metrik auf Σ^*.

Edit-Distanz

UA RUHR

Eine Metrik auf Σ^* (Nachweis: Übung) erhält man mit der Edit-Distanz (Levenshtein-Distanz, nach Vladimir Iosifovich Levenshtein, geb. 1935).

Definition (Edit-Distanz, Levenshtein-Distanz)

Die Edit-Distanz zwischen zwei Strings s und t ist definiert als die Anzahl der Edit-Operationen, die man **mindestens** benötigt, um einen String in einen anderen zu überführen.
Edit-Operationen sind Löschen, Einfügen und Verändern eines Zeichens.

Beispiele für die Edit-Distanz

UA RUHR

```
- a n a n a s
b a n a n a -
```

(2 Operationen, minimal)

```
d u c k t a l e s
d u c t t a p e -
```

(3 Operationen, minimal)

Visualisierung des Edit-Prozesses

Es gibt viele Möglichkeiten, s in t zu überführen:

```
h a n d    h a n d - - - -    h a n d -
a n d i    - - - - a n d i    - a n d i
```

Visualisierung des Edit-Prozesses

UA RUHR

Es gibt viele Möglichkeiten, s in t zu überführen:

```
h a n d     h a n d - - - -     h a n d -
a n d i     - - - - a n d i     - a n d i
```

Uns interessiert die minimale Anzahl von Edit-Operationen. Da die Edit-Operationen die Reihenfolge der Buchstaben nicht verändern, genügt es, den Prozess von links nach rechts zu betrachten.

Visualisierung des Edit-Prozesses

UA RUHR

Der Edit-Prozess kann, wie in den vorherigen Beispielen gezeigt, durch ein Sequenz-Alignment visualisiert werden.

Visualisierung des Edit-Prozesses

UA RUHR

Der Edit-Prozess kann, wie in den vorherigen Beispielen gezeigt, durch ein Sequenz-Alignment visualisiert werden.

Definition (Globales Sequenz-Alignment)

Ein globales Alignment A von $s, t \in \Sigma^*$ ist ein String über $(\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$
mit Porjektionen $\pi_1(A) = s$ und $\pi_2(A) = t$,
wobei die Projektion π_1 der String-Homomorphismus
mit $\pi_1\big((a,b)\big) := a$ für $a \in \Sigma$ und $\pi_1\big((-,b)\big) := \epsilon$ ist,
und π_2 derjenige mit $\pi_2\big((a,b)\big) := b$ für $b \in \Sigma$ und $\pi_2\big((a,-)\big) := \epsilon$.

Visualisierung des Edit-Prozesses

UA RUHR

Der Edit-Prozess kann, wie in den vorherigen Beispielen gezeigt, durch ein Sequenz-Alignment visualisiert werden.

Definition (Globales Sequenz-Alignment)

Ein globales Alignment A von $s, t \in \Sigma^*$ ist ein String über $(\Sigma \cup \{-\})^2 \setminus \{(-,-)\}$
mit Porjektionen $\pi_1(A) = s$ und $\pi_2(A) = t$,
wobei die Projektion π_1 der String-Homomorphismus
mit $\pi_1\big((a,b)\big) := a$ für $a \in \Sigma$ und $\pi_1\big((-,b)\big) := \epsilon$ ist,
und π_2 derjenige mit $\pi_2\big((a,b)\big) := b$ für $b \in \Sigma$ und $\pi_2\big((a,-)\big) := \epsilon$.

Das Sequenz-Alignment beschreibt die Reihenfolge der Edit-Operationen.

Berechnung der Edit-Distanz

Beobachtung: Ein Alignment der Strings *sa* und *tb* kann auf genau 3 verschieden Arten enden. Dabei sind $s, t \in \Sigma^*$ und $a, b \in \Sigma$.

s	a
t	b

sa	–
t	b

s	a
tb	–

Aus dieser Beobachtung ergibt sich das folgende Lemma zur Berechnung der Edit-Distanz.

Berechnung der Edit-Distanz

UA RUHR

Lemma (Rekurrenz zur Edit-Distanz)

Seien $s, t \in \Sigma^*$, sei ϵ der leere String, $a, b \in \Sigma$ einzelne Zeichen.
Es sei d die Edit-Distanz auf Σ^*. Dann gilt

$$d(s, \epsilon) = |s|,$$

$$d(\epsilon, t) = |t|,$$

$$d(a, b) = \begin{cases} 1 & \text{falls } a \neq b, \\ 0 & \text{falls } a = b, \end{cases}$$

$$d(sa, tb) = \min \begin{cases} d(s, t) + d(a, b), \\ d(s, tb) + 1, \\ d(sa, t) + 1. \end{cases}$$

Beweis

UA RUHR

$$d(sa, tb) = \min \begin{cases} d(s,t) + d(a,b), \\ d(s, tb) + 1, \\ d(sa, t) + 1. \end{cases}$$

Die elementaren Fälle $d(s, \epsilon), d(\epsilon, t), d(a, b)$ sind trivial.
Für $d(sa, tb)$ gilt die Richtung "$\leq$", da die drei Möglichkeiten zulässige Edit-Operationen darstellen. Zu zeigen ist die Ungleichung "$\geq$".

Beweis

$$d(sa, tb) = \min \begin{cases} d(s,t) + d(a,b), \\ d(s, tb) + 1, \\ d(sa, t) + 1. \end{cases}$$

Die elementaren Fälle $d(s, \epsilon), d(\epsilon, t), d(a, b)$ sind trivial.
Für $d(sa, tb)$ gilt die Richtung "$\leq$", da die drei Möglichkeiten zulässige Edit-Operationen darstellen. Zu zeigen ist die Ungleichung "$\geq$".
Beweis mit Induktion durch Widerspruch: Die Gleichung sei richtig für alle Präfixe x von sa und y von tb mit $|x| + y| < |sa| + |tb|$. Angenommen, $d(sa, tb) < \min(\dots)$.

Beweis

UA RUHR

$$d(sa, tb) = \min \begin{cases} d(s,t) + d(a,b), \\ d(s,tb) + 1, \\ d(sa,t) + 1. \end{cases}$$

Die elementaren Fälle $d(s, \epsilon), d(\epsilon, t), d(a, b)$ sind trivial.
Für $d(sa, tb)$ gilt die Richtung "$\leq$", da die drei Möglichkeiten zulässige Edit-Operationen darstellen. Zu zeigen ist die Ungleichung "$\geq$".
Beweis mit Induktion durch Widerspruch: Die Gleichung sei richtig für alle Präfixe x von sa und y von tb mit $|x| + y| < |sa| + |tb|$. Angenommen, $d(sa, tb) < \min(\dots)$.
Ein Alignment von sa, tb muss jedoch auf eine der drei Arten enden: Entweder a steht über b, oder a steht über $-$, oder $-$ steht über b.

Beweis

UA RUHR

$$d(sa, tb) = \min \begin{cases} d(s,t) + d(a,b), \\ d(s,tb) + 1, \\ d(sa,t) + 1. \end{cases}$$

Die elementaren Fälle $d(s, \epsilon), d(\epsilon, t), d(a, b)$ sind trivial.
Für $d(sa, tb)$ gilt die Richtung "$\leq$", da die drei Möglichkeiten zulässige Edit-Operationen darstellen. Zu zeigen ist die Ungleichung "$\geq$".
Beweis mit Induktion durch Widerspruch: Die Gleichung sei richtig für alle Präfixe x von sa und y von tb mit $|x| + y| < |sa| + |tb|$. Angenommen, $d(sa, tb) < \min(\dots)$.
Ein Alignment von sa, tb muss jedoch auf eine der drei Arten enden: Entweder a steht über b, oder a steht über –, oder – steht über b. Je nachdem, welcher Fall im optimalen Alignment von sa und tb eintritt, müsste bereits $d(s, t)$ oder $d(s, tb)$ oder $d(sa, t)$ kleiner als optimal gewesen sein. ↯

Berechnung der Edit-Distanz

UA RUHR

- Die Edit-Distanz kann (statt rekursiv) mit einem **Dynamic-Programming-Algorithmus** berechnet werden.
- Dynamic Programming (DP) ist eine algorithmische Technik, deren Anwendung sich anbietet, wenn Probleme im Prinzip rekursiv gelöst werden können, dabei aber (bei naiver Implementierung) dieselben Instanzen des Problems wiederholt gelöst werden müssten.
- So kann ggf. exponentielle Laufzeit auf polynomielle Laufzeit reduziert werden.

DP-Berechnung der Edit-Distanz

UA RUHR

- Seien $m := |s|$ und $n := |t|$.
- Eine $(m+1) \times (n+1)$ Matrix $D = (D[i,j])$ wird berechnet.
  $D[i,j]$ sei die Edit-Distanz zwischen dem i-Präfix von s und dem j-Präfix von t.

DP-Berechnung der Edit-Distanz

UA RUHR

- Seien $m := |s|$ und $n := |t|$.
- Eine $(m+1) \times (n+1)$ Matrix $D = (D[i,j])$ wird berechnet.
  $D[i,j]$ sei die Edit-Distanz zwischen dem i-Präfix von s und dem j-Präfix von t.
- Initialisierungen laut Lemma:
  - $D[0,0] = 0$.
  - $D[i,0] = i$.
  - $D[0,j] = j$.

DP-Berechnung der Edit-Distanz

UA RUHR

- Seien $m := |s|$ und $n := |t|$.
- Eine $(m+1) \times (n+1)$ Matrix $D = (D[i,j])$ wird berechnet.
  $D[i,j]$ sei die Edit-Distanz zwischen dem i-Präfix von s und dem j-Präfix von t.
- Initialisierungen laut Lemma:
  - $D[0,0] = 0$.
  - $D[i,0] = i$.
  - $D[0,j] = j$.
- Für alle weiteren Einträge gilt laut Lemma:

$$D[i,j] = \min \begin{cases} D[i-1,j-1] + d(s[i-1],t[j-1]), \\ D[i-1,j] + 1, \\ D[i,j-1] + 1. \end{cases}$$

DP-Berechnung der Edit-Distanz

- Seien $m := |s|$ und $n := |t|$.
- Eine $(m+1) \times (n+1)$ Matrix $D = (D[i,j])$ wird berechnet.
  $D[i,j]$ sei die Edit-Distanz zwischen dem i-Präfix von s und dem j-Präfix von t.
- Initialisierungen laut Lemma:
  - $D[0,0] = 0$.
  - $D[i,0] = i$.
  - $D[0,j] = j$.
- Für alle weiteren Einträge gilt laut Lemma:

$$D[i,j] = \min \begin{cases} D[i-1,j-1] + d(s[i-1], t[j-1]), \\ D[i-1,j] + 1, \\ D[i,j-1] + 1. \end{cases}$$

- Das Ergebnis (Edit-Distanz von s,t) steht in $D[m,n]$.
- Laufzeit: $\mathcal{O}(mn)$

Berechnung der Edit-Distanz

Beispiel einer Edit-Matrix mit $s = \texttt{andi}$ und $t = \texttt{handy}$.

	ϵ	h	a	n	d	y
ϵ	0	1	2	3	4	5
a	1	1	1	2	3	4
n	2	2	2	1	2	3
d	3	3	3	2	1	2
i	4	4	4	3	2	2

Die Edit-Distanz zwischen s und t beträgt also 2.

Berechnung der Edit-Distanz

UA RUHR

- Die Berechnung kann zeilenweise, spaltenweise oder diagonal erfolgen.
- Da für Feld $D[i,j]$ jeweils nur seine direkten "linken" und "oberen" Nachbarn notwendig sind, genügt es, die aktuelle Zeile/Spalte und die Vorgängerzeile/spalte zu speichern.
- Der Speicherverbrauch sinkt somit auf $\mathcal{O}(\min(m,n))$.
- Für die Rekonstruktion des optimalen Alignments ist aber die ganze Matrix erforderlich (später).

Berechnung der Edit-Distanz

UA RUHR

```
def edit_distance(s, t):
    m, n = len(s), len(t)
    # Spalte 0
    Dp = [0] * (m + 1)  # vorherige Spalte
    Dc = list(range(m + 1)) # aktuelle Spalte
    # Spalten j = 1, ...
    for j, tj in zip(count(1), t):
        Dl, Dc = Dc, Dl
        Dc[0] = j
        for i, si in zip(count(1), s):
            Dc[i] = min(Dl[i - 1] + (si != tj),
                        Dl[i] + 1,
                        Dc[i - 1] + 1)
    return Dc[m]
```

Ähnlichkeitsmaß: Longest Common Subsequence

UA RUHR

- Sei $lcs(s, t)$ die Länge der längsten gemeinsamen Teilsequenz von s und t.
- Dies ist ein **Ähnlichkeitsmaß** und schon deswegen keine Metrik.

Ähnlichkeitsmaß: Longest Common Subsequence

UA RUHR

- Sei $lcs(s, t)$ die Länge der längsten gemeinsamen Teilsequenz von s und t.
- Dies ist ein **Ähnlichkeitsmaß** und schon deswegen keine Metrik.
- Der DP-Algorithmus zur Edit-Distanz kann angepasst werden, um $lcs(s, t)$ zu berechnen.
- Umkehrung der Logik:
  Insertionen, Deletionen und Substitutionen tragen 0 zur Länge bei.
  Eine Übereinstimmung trägt 1 zur Länge bei.

Ähnlichkeitsmaß: Longest Common Subsequence

UA RUHR

- Sei $lcs(s, t)$ die Länge der längsten gemeinsamen Teilsequenz von s und t.
- Dies ist ein **Ähnlichkeitsmaß** und schon deswegen keine Metrik.
- Der DP-Algorithmus zur Edit-Distanz kann angepasst werden, um $lcs(s, t)$ zu berechnen.
- Umkehrung der Logik:
  Insertionen, Deletionen und Substitutionen tragen 0 zur Länge bei.
  Eine Übereinstimmung trägt 1 zur Länge bei.
- Folglich werden nullte Zeile und Spalte in der Matrix mit 0 initialisiert.
- Bei der Reukkrenz wird der größte Vorgängerwert genommen.

Longest Common Subsequence

UA RUHR

Sei $L[i,j]$ die Länge der längsten gemeinsamen Teilsequenz von $s[:i]$ und $t[:j]$:

$$
\begin{aligned}
L[i,0] &= 0, \\
L[0,j] &= 0, \\
L[i,j] &= \max \begin{cases} L[i-1,j-1] + (s[i-1] == t[j-1]), \\ L[i-1,j], \\ L[i,j-1]. \end{cases}
\end{aligned}
$$

Longest Common Subsequence

UA RUHR

Sei $L[i,j]$ die Länge der längsten gemeinsamen Teilsequenz von $s[:i]$ und $t[:j]$:

$$
\begin{aligned}
L[i,0] &= 0,\\
L[0,j] &= 0,\\
L[i,j] &= \max \begin{cases} L[i-1,j-1] + (s[i-1] == t[j-1]),\\ L[i-1,j],\\ L[i,j-1]. \end{cases}
\end{aligned}
$$

Auch hierbei benötigt man $\mathcal{O}(\min(m,n))$ Speicherplatz und $\mathcal{O}(mn)$ Zeit.

Ähnlichkeitsmaß: Longest Common Factor

UA RUHR

- Mit Hilfe des DP-Ansatzes lässt sich auch die Länge des längsten gemeinsamen Teilstrings (LCF) ermitteln.
- Die Laufzeit $\mathcal{O}(mn)$ ist aber asymptotisch schlechter als die der Algorithmen mit Suffixbaum und -array!

Ähnlichkeitsmaß: Longest Common Factor

UA RUHR

- Mit Hilfe des DP-Ansatzes lässt sich auch die Länge des längsten gemeinsamen Teilstrings (LCF) ermitteln.
- Die Laufzeit $\mathcal{O}(mn)$ ist aber asymptotisch schlechter als die der Algorithmen mit Suffixbaum und -array!
- Hier sei $L[i,j]$ die Länge des längesten gemeinsamen Teilstrings, der mit $s[i-1]$ und $t[j-1]$ endet.
- Damit wieder $L[i,0] = 0$ für alle i und $L[0,j] = 0$ für alle j.

Ähnlichkeitsmaß: Longest Common Factor

UA RUHR

- Mit Hilfe des DP-Ansatzes lässt sich auch die Länge des längsten gemeinsamen Teilstrings (LCF) ermitteln.
- Die Laufzeit $\mathcal{O}(mn)$ ist aber asymptotisch schlechter als die der Algorithmen mit Suffixbaum und -array!
- Hier sei $L[i,j]$ die Länge des längesten gemeinsamen Teilstrings, der mit $s[i-1]$ und $t[j-1]$ endet.
- Damit wieder $L[i,0]=0$ für alle i und $L[0,j]=0$ für alle j.

$$L[i,j] = \begin{cases} L[i-1,j-1]+1, & \text{falls } s[i-1]=t[j-1], \\ 0 & \text{sonst.} \end{cases}$$

Ähnlichkeitsmaß: Longest Common Factor

UA RUHR

- Mit Hilfe des DP-Ansatzes lässt sich auch die Länge des längsten gemeinsamen Teilstrings (LCF) ermitteln.
- Die Laufzeit $\mathcal{O}(mn)$ ist aber asymptotisch schlechter als die der Algorithmen mit Suffixbaum und -array!
- Hier sei $L[i,j]$ die Länge des längesten gemeinsamen Teilstrings, der mit $s[i-1]$ und $t[j-1]$ endet.
- Damit wieder $L[i,0]=0$ für alle i und $L[0,j]=0$ für alle j.

$$L[i,j] = \begin{cases} L[i-1,j-1]+1, & \text{falls } s[i-1]=t[j-1], \\ 0 & \text{sonst.} \end{cases}$$

- Damit $lcf(s,t) = \max\{L[i,j] \mid 0 \leq i \leq m,\ 0 \leq j \leq n\}$.

Beispiel: Longest Common Factor

UA RUHR

$L[i,j]$: Länge des längesten gemeinsamen Teilstrings, der mit $s[i-1]$ und $t[j-1]$ endet.

L	ϵ	a	b	a	b
ϵ	0	0	0	0	0
b	0	0	1	0	1
a	0	1	0	2	0
b	0	0	2	0	3
a	0	1	0	3	0

Edit-Graph

UA RUHR

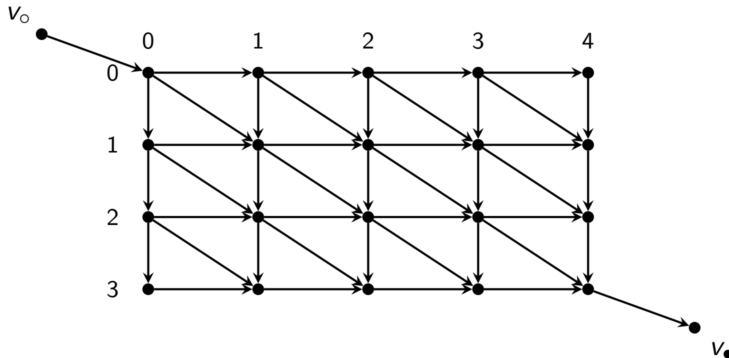
Die Berechnung der Editdistanz kann mit einem Graphen dargestellt werden.

Definition (globaler Alignment-Graph, Edit-Graph)

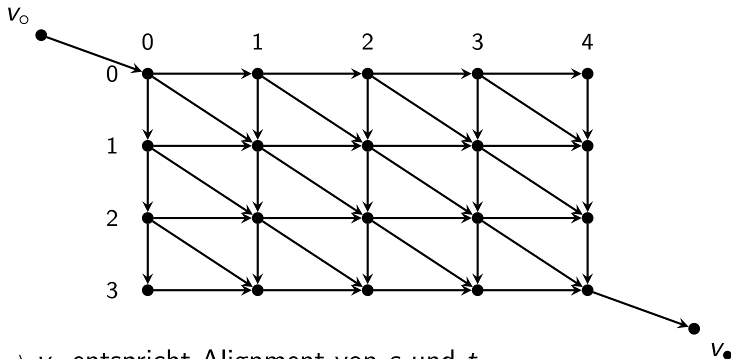
- Knotenmenge $V := \{(i,j) : 0 \leq i \leq m, 0 \leq j \leq n\} \cup \{v_\circ, v_\bullet\}$
- Kanten:

	Kante	Label	Kosten
horizontal	$(i,j) \to (i,j+1)$	$\left[\begin{smallmatrix}-\\t_j\end{smallmatrix}\right]$	1
vertikal	$(i,j) \to (i+1,j)$	$\left[\begin{smallmatrix}s_i\\-\end{smallmatrix}\right]$	1
diagonal	$(i,j) \to (i+1,j+1)$	$\left[\begin{smallmatrix}s_i\\t_j\end{smallmatrix}\right]$	$[s_i \neq t_j]$
Initialisierung	$v_\circ \to (0,0)$	ϵ	0
Finalisierung	$(m,n) \to v_\bullet$	ϵ	0

Edit-Graph



Edit-Graph



- Pfad $v_\circ \to v_\bullet$ entspricht Alignment von s und t (Konkatenation der Kantenlabel)
- Edit-Distanz: Länge des kürzesten Weges von $v_\circ$ nach $v_\bullet$.
- $D[i, j]$: Länge des kürzesten Weges $v_\circ \to (i, j)$.

Anzahl der Pfade

UA RUHR

- Anzahl $N(m, n)$ im Edit-Graphen von Strings der Längen m und n:
  Anzahl der Möglichkeiten, eine Sequenz in die andere zu "editieren",
  Anzahl der globalen Alignments der Sequenzen

Anzahl der Pfade

UA RUHR

- Anzahl $N(m, n)$ im Edit-Graphen von Strings der Längen m und n:
  Anzahl der Möglichkeiten, eine Sequenz in die andere zu "editieren",
  Anzahl der globalen Alignments der Sequenzen
- Berechnung:

$$
\begin{aligned}
N(0,0) &= 1,\\
N(m,0) &= 1 \text{ für alle } m,\\
N(0,n) &= 1 \text{ für alle } n,\\
N(m,n) &= N(m-1,n-1) + N(m,n-1) + N(m-1,n) \text{ für } m > 1,\ n > 1.
\end{aligned}
$$

Anzahl der Pfade

UA RUHR

Anzahl der Pfade/Alignments $N(m, n)$ für $0 \leq m, n \leq 4$.

$m \backslash n$	0	1	2	3	4	...
0	1	1	1	1	1	...
1	1	3	5	7	9	...
2	1	5	13	25	41	...
3	1	7	25	63	129	...
4	1	9	41	129	321	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Anzahl der Pfade

UA RUHR

Anzahl der Pfade/Alignments $N(m, n)$ für $0 \le m, n \le 4$.

$m \backslash n$	0	1	2	3	4	...
0	1	1	1	1	1	...
1	1	3	5	7	9	...
2	1	5	13	25	41	...
3	1	7	25	63	129	...
4	1	9	41	129	321	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

- Schranke: $N(n, n) > 3N(n-1, n-1)$ und damit $N(n, n) > 3^n$.
- Asymptotisch $N(n, n) = \Theta\left(\sqrt{n} \cdot (1 + \sqrt{2})^{2n+1}\right)$,
  d.h. das Wachstum von $N(n, n)$ ist exponentiell mit Basis $(1 + \sqrt{2})^2 \approx 5.8$.