

Bioinformatik

Approximatives Stringmatching
Editabstand



Silke Trißl / Ulf Leser
Wissensmanagement in der
Bioinformatik



Rückblick

- Komplexität
 - O-Notation
- Motivation fürs Stringmatching
 - DNA-Sequenzierung
- Exaktes Stringmatching
 - Naiver Algorithmus
 - Z-Box-Algorithmus

Naiver Ansatz

1. P und T an Position 1 ausrichten
2. Vergleiche P mit T von links nach rechts
 - Zwei ungleiche Zeichen \Rightarrow Gehe zu 3
 - Zwei gleiche Zeichen
 - P noch nicht durchlaufen \Rightarrow Verschiebe Pointer nach rechts, gehe zu 2
 - P vollständig durchlaufen \Rightarrow Merke Vorkommen von P in T
3. Verschiebe P um ein Zeichen nach rechts
4. Wenn P noch nicht über $|T|-|P|$ hinaus, gehe zu 2

```
T   ctgagatcgcgta
P   gagatc
    gagatc
     gagatc
      gagatc
       gatatc
        gatatc
         gatatc
```

Z-Algorithmus: Preprocessing

- Im Folgenden: S
 - (wird gleich aus P und T zusammengesetzt)
- Definition
 - Sei $i > 1$. Dann ist $Z_i(S)$ die Länge des *längsten Substrings* x von S mit
 - $x = S[i..i+|x|-1]$ (x startet an Position i in S)
 - $S[i..i+|x|-1] = S[1..|x|-1]$ (x ist auch Präfix von S)
 - x ist die *Z-Box* von S an Position i mit Länge $Z_i(S)$



Linearer Stringmatching Algorithmus

- Annahme: Z-Boxen lassen sich in $O(|S|)$ berechnen
 - Wie zeigen wir später
- Verwendung der Z-Boxen für String Matching

```
S := P||`$`||T;           // ($ ∉ Σ)
compute Z-Boxes for S;
for i = |P|+2 to |S|
    if (Zi(S)=|P|) then
        print i-|P|-1; // P in T at position i
    end if;
end if;
```

- Komplexität
 - Schleife wird $|S|$ -Mal durchlaufen => $O(m)$

Lineare Berechnung der Z_i Werte

- Trick
 - Verwenden von bereits bekannten Z_i zur Berechnung von Z_k ($k > i$)
 - Lineares Durchlaufen des Strings
 - Kontinuierliches Vorhalten der aktuellen Werte $l=l_i$ und $r=r_i$
 - Größe der Z-Box an Position i ergibt sich mit konstantem Aufwand
- Induktive Erklärung
 - Induktionsanfang: Position $k=2$
 - Berechne Z_2 .
 - Wenn $Z_2 > 0$, setze $r=r_2$ ($=2+Z_2-1$) und $l=l_2$ ($=2$), sonst $r=l=0$
 - Induktionsschritt: Position $k>2$
 - Vorhanden sind die Werte r , l und $\forall j < k: Z_j$

Z-Algorithmus, Fall 1

- Möglichkeit 1: $k > r$
 - D.h., dass es keine Z-Box gibt, die k enthält
 - Wir wissen damit nichts über den Bereich in S ab k
 - Dann gehen wir primitiv vor
 - Berechne Z_k durch Zeichen-für-Zeichen Matching
 - Wenn $Z_k > 0$, setze $r = r_k$ und $l = l_k$

Beispiel

k
CTCGAGTTGCAG
0
1
0
?

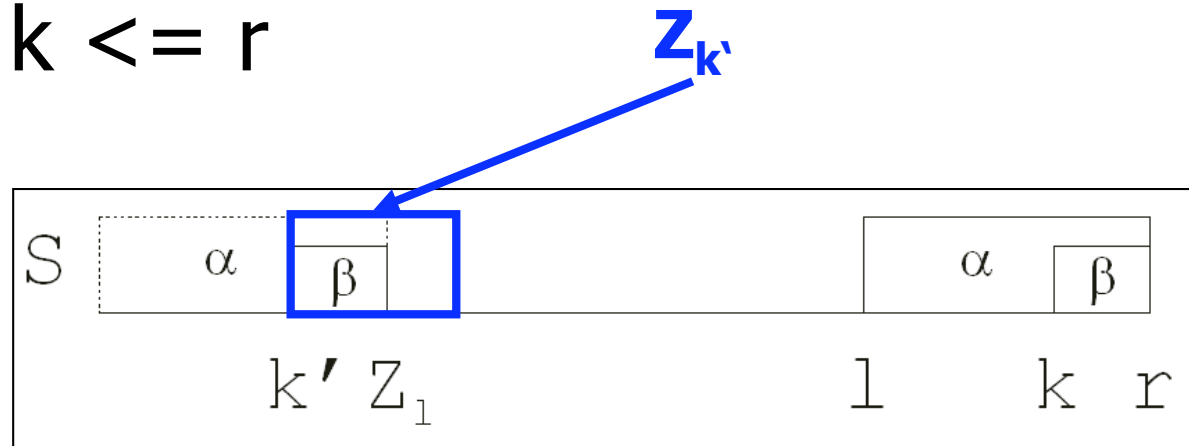
Gegenbeispiel

lk r
CTACTACTTTGCAG
0
0
5
?

Z-Algorithmus, Fall 2

- Möglichkeit 2: $k \leq r$

– Die Situation:



– Also

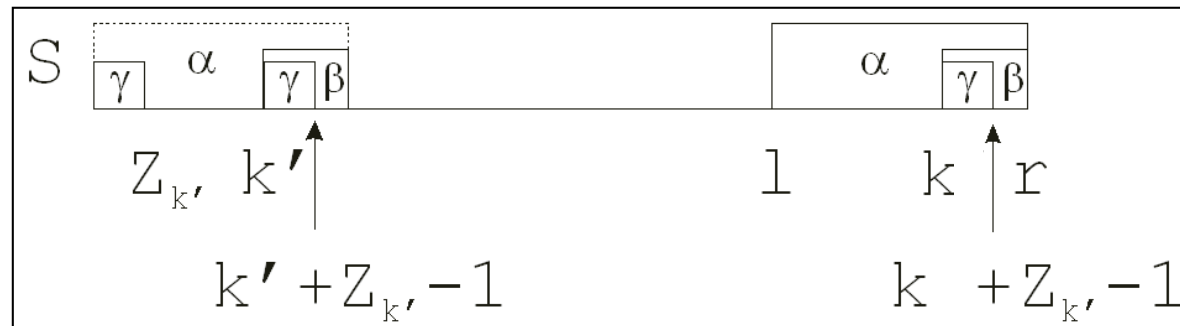
- Z-Box Z_1 ist Präfix von S
- Substring $\beta = S[k..r]$ kommt auch an Position $k' = k - l + 1$ von S vor
- Was wissen wir über diesen Substring? Natürlich: $Z_{k'}$
- D.h., dass $S[k..]$ ist Präfix von S mit mindestens Länge $\min(Z_{k'}, |\beta|)$

Z-Algorithmus, Fall 2.1

- Fallunterscheidung

- $Z_{k'} < |\beta|$: Dann ist das Zeichen an $k'+Z_{k'}$ ein Mismatch bei der Prefixverlängerung. Dann ist das Zeichen $S(k'+Z_{k'})$ der gleiche Mismatch. Also:

$Z_k = Z_{k'}$; r und l unverändert



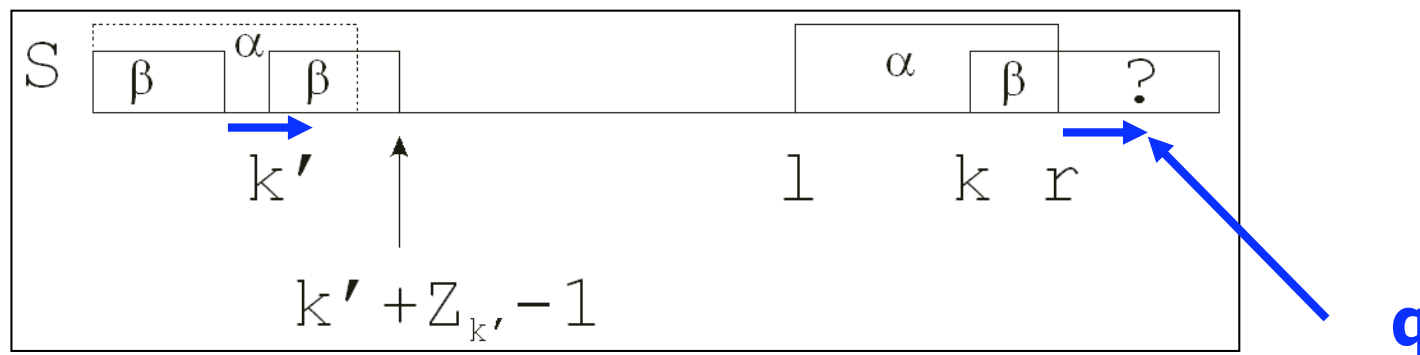
Z-Algorithmus, Fall 2.2

- $Z_{k'} \geq |\beta|$: Dann ist β ein Präfix von S
 - ... dass sich vielleicht noch verlängern lässt
 - Wenn $Z_{k'} > |\beta|$, dann wissen wir: $S(|\beta|+1)=S(k'+|\beta|)$
 - Wir wissen aber nichts über $S(r+1)$ – dieses Zeichen wurde noch nie betrachtet

Matche Zeichen-für-Zeichen $S[r+1.. ?]$ mit $S[|\beta|+1.. ?]$

Sei der erste Mismatch an Position q

Dann: $Z_k = q - k$; $l = k$; $r = q - 1$



Algorithmus

```
match Z2; set l,r;
for k = 3 to |S|
  if k>r then
    match Zk; set r,l;
  else
    k' := k-1+1;
    b := r-k+1; // This is β
    if Zk' < b then
      Zk := Zk';
    else
      match S[r+1.. ] with S[b+1.. ] until q;
      Zk := q-k; l := k; r := q-1;
    end if;
  end if;
end for;
```

Inhalt dieser Vorlesung

- Approximative Stringvergleiche
- Dotplots
- Alignment, Pfade, Editskipte
- Dynamische Programmierung

Motivation

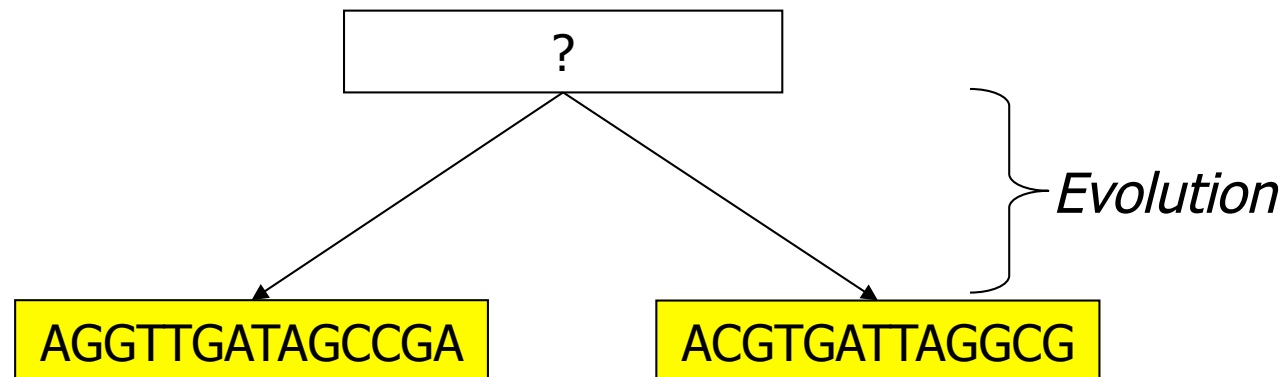
- Relevant ist die **biologische Funktion**, nicht die Sequenz
 - Gleiche Sequenzen – gleiche Funktion
 - Sehr ähnliche Sequenzen – sehr ähnliche Funktion
 - Etwas ähnliche Sequenzen – verwandte Funktion?
- Idee: Funktionsähnlichkeit durch Sequenzähnlichkeit approximieren
 - Comparative Genomics
 - Bestimmung von Funktionen ist extrem aufwändig (wenn überhaupt möglich)
 - Bestimmung von Sequenzen ist vergleichsweise billig
- Außerdem: Hohe Ähnlichkeit kann meistens kein Zufall sein
 - Konservierung von Sequenzen trotz Evolution
 - Hoch ähnliche Sequenzen in evolutionär entfernten Spezies ist starkes Signal für hohen evolutionären Druck auf der Sequenz
 - Billige Methode, um „wichtige“ (funktionstragende) Sequenzabschnitte zu finden

Wie ähnlich sind sich zwei Sequenzen?

- „AGGTAG“ und
 - AGTAG G zu wenig
 - AGGTAG Identisch: Ähnlichkeit=1
 - AGGATAG A zu viel
 - AGTTCAG G durch T ersetzen und C löschen
 - ...TGAGGTAGGTT... **Sehr viel löschen**
- Welche Sequenzen sind sich also besonders ähnlich?
 - „Ähnlichkeit“ muss quantifiziert werden
 - Idee: Wie sehr muss man **eine Sequenz verändern**, um die andere zu erzeugen
 - Man konnte auch Buchstaben zählen, Länge vergleichen, Anzahl GC nehmen, ...

Abstandsmaße

- Wir suchen ein Maß für die Ähnlichkeit zweier Sequenzen
- Biologischen Hintergrund beachten
 - Situation: Wir haben humane Gensequenz A und suchen ähnliche Sequenzen (B) in anderen Organismen
 - Annahme: A und B haben **gemeinsamen Vorfahren** X und sind aus diesem jeweils durch **evolutionäre Prozesse** entstanden
 - Einfaches Modell evolutionärer Prozesse: **Basenaustausch, Baseneinfügung, Basenlöschen**



Nicht alle Änderungen sind gleich wichtig

Wildtyp

C T T A G T G A C T A C G G T A A A

DNA

Leu Ser Asp Tyr Gly Lys

Protein

Fatale Mutationen

C T T A G T G A C T A G G G T A A A

DNA

Leu Ser Asp **Stop-Codon**

Protein

Leseraster Mutationen

C T T A G T G A A C T A C G G T A A A

DNA

Leu Ser His Asp Leu Thr

Protein

neutrale Mutationen

C T T A G C G A C T A C G G T A A A

DNA

Leu Ser Asp Tyr Gly Lys

Protein

Funktionale Mutationen

C T T A G T G A A T A C G G T A A A

DNA

Leu Ser Glu Tyr Gly Lys

Protein

Approximatives Matchen außerhalb der Bioinformatik

- Anwendungen außerhalb der Bioinformatik
 - Unscharfe Volltextsuche
 - Suche mit „Xylofon“ und finde auch „Xhylophon“
 - Fehler – oder deutsche Rechtschreibreform?
 - Personenabgleich
 - Ist „Herr Müller, 27, Stargarder Str 54“ identisch zu „Hr. Mueller, 27, Stagarder Str. 54“ ?
 - Phonetische Suche
 - Finde alle Meyer, Meier, Maier, Mair, ...
 - Vorschlägen ähnlicher Suchbegriffe



Dotplot

- Definition:
Ein **Dotplot** zweier Strings A, B ist eine Matrix M:
 - Die Spalten entsprechen den Zeichen von A
 - Die Zeilen entsprechen den Zeichen von B
 - $M[a,b]=1$ (blau) gdw. $A[a] = B[b]$

- Beispiel?

	A	T	G	C	G	G	T	G	C	A	A	T	G
A													
T													
G													
G													
T													
G													
C													
A													
T													

Dotplot

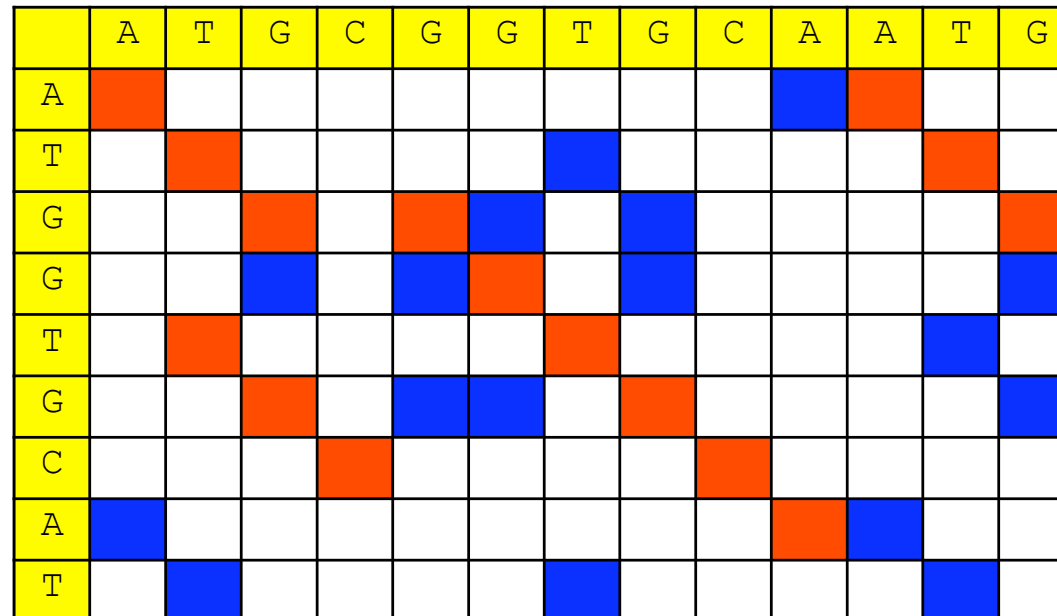
- Definition:
Ein **Dotplot** zweier Strings A, B ist eine Matrix M:
 - Die Spalten entsprechen den Zeichen von A
 - Die Zeilen entsprechen den Zeichen von B
 - $M[a,b]=1$ (blau) gdw. $A[a] = B[b]$

- Beispiel

	A	T	G	C	G	G	T	G	C	A	A	T	G
A	1									1	1		
T		1					1					1	
G			1		1	1		1					1
G			1		1	1		1					1
T		1					1					1	
G			1		1	1		1					1
C				1					1				
A	1									1	1		
T		1					1					1	

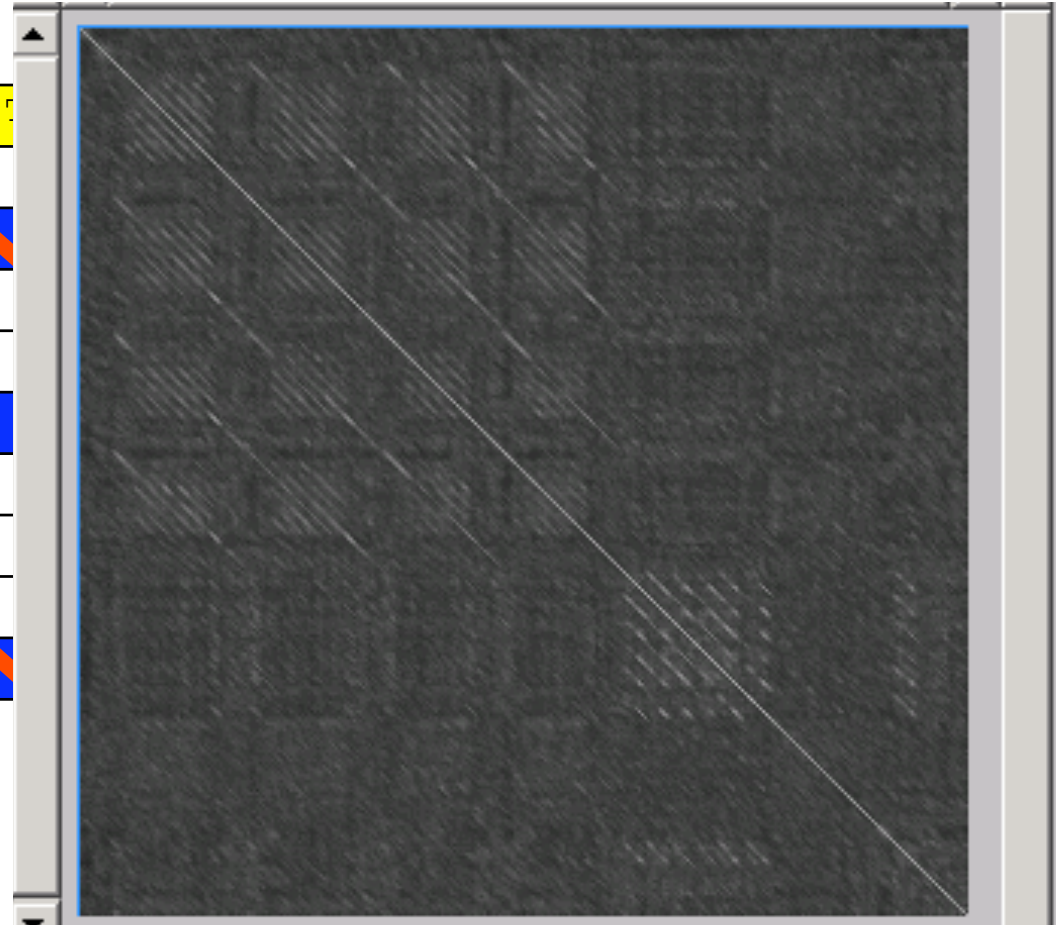
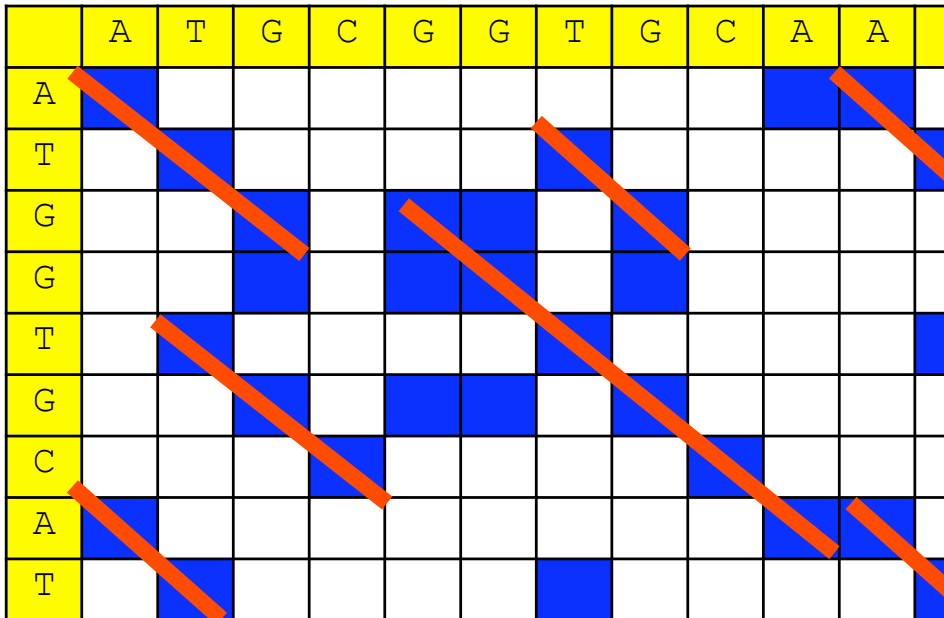
Dotplot und gleiche Teilstrings

- Wie erkennt man gleiche Teilstrings im Dotplot?



- Diagonalen von links-oben nach rechts-unten
 - Größter gemeinsamer Teilstring – längste Diagonale
 - Visuell bei kurzen Strings leicht möglich

Repetitive Sequenzen



- Dotplot mit $A=B$

- Zitat (Genbank, P24014):

- [SIMILARITY] CONTAINS 7 EGF-LIKE DOMAINS.

- [SIMILARITY] Contains 24 leucine-rich (LRR) repeats.

Editskripte

- Definition

Ein *Editskript* e für zwei Strings A, B aus $\Sigma^* = \Sigma \cup \{ _ \}$ ist eine Sequenz von Editieroperationen

- I (*Einfügen* eines Zeichen $c \in \Sigma$ in A)
 - Dargestellt als Lücke in A ; das neue Zeichen erscheint in B
- D (*Löschen* eines Zeichen c in A)
 - Dargestellt als Lücke in B ; das alte Zeichen erscheint in A
- R (*Ersetzen* eines Zeichen in A mit einem anderen Zeichen in B)
- M (*Match*, d.h., gleiche Zeichen in A und B an dieser Stelle)

so, dass $e(A)=B$

- Beispiel: $A = \text{„ATGTA“}$, $B = \text{„AGTGTC“}$

– MIMMR	IRMMMDI
A _TGTA	_ATGTA_
AGTGTC	AGTGT_C

Editabstand

- Offensichtlich gibt es für A, B ziemlich viele Editskripte
- Definition
 - Die *Länge eines Editskript* ist die Anzahl von Operationen o im Skript mit $o \in \{I, R, D\}$
 - Der *Editabstand* zweier Strings A, B ist die Länge des kürzesten Editskript für A, B
- Bemerkung
 - Für den Abstand zählen nur die Änderungen
 - Anderer Name: **Levenshtein-Abstand**
 - Es gibt oft verschiedene kürzeste Editskripte
 - **IMMMMD** ?
 _AGAGAG
 GAGAGA_

Editabstand

- Offensichtlich gibt es für A, B ziemlich viele Editskripte
- Definition
 - Die *Länge eines Editskript* ist die Anzahl von Operationen o im Skript mit $o \in \{I, R, D\}$
 - Der *Editabstand* zweier Strings A, B ist die Länge des kürzesten Editskript für A, B
- Bemerkung
 - Für den Abstand zählen nur die Änderungen
 - Anderer Name: [Levenshtein-Abstand](#)
 - Es gibt oft verschiedene kürzeste Editskripte
 - | | | | | | |
|----------|----------|----------|----------|----------|----------|
| I | M | M | M | M | D |
| _ | A | G | A | G | A |
| G | A | G | A | G | A |
| _ | | | | | _ |

Beobachtungen

- Ziel: Wir benutzen den (leicht berechenbaren) Editabstand als **objektives Maß für Sequenzähnlichkeit**
 - Fußt auf der Annahme, dass ähnliche Sequenzen durch **Evolutionseignisse** aus einem gemeinsamen Vorgänger entstanden sind
 - Beinhaltet eine „**Minimalismusannahme**“ – wir suchen eine Evolutionsgeschichte mit möglichst wenig Ereignissen
 - Sonst wäre das Problem undefiniert
- Handlicher (und gebräuchlicher) als Editskripte sind **Alignments**
- Next steps
 - Definition von Alignments
 - Alignments und Pfade durch Dotplots
 - Berechnung des Editabstandes (=Alignmentsscore)

Alignment

- Definition

- Ein *(globales) Alignment* zweier Strings A, B ist eine Untereinanderanordnung von A und B , jeweils mit beliebigen zusätzlichen Leerzeichen ($_$), ohne dass zwei Leerzeichen untereinander stehen
 - Achtung: Untereinanderstehende Zeichen müssen nicht matchen
- Der *Alignmentscore* eines Alignment ist die Anzahl von Leerzeichen und Mismatches
- Der *Alignmentabstand* zweier Strings A, B ist der minimale Alignmentscore aller Alignments der beiden Strings

- Beispiele

- | | | | | | | |
|---|---|---|---|---|---|---|
| A | _ | T | G | T | _ | A |
| A | G | T | G | T | C | _ |
- | | | | | | | |
|---|---|---|---|---|---|---|
| A | _ | T | _ | G | T | A |
| _ | A | G | T | G | T | C |
- | | | | | | | |
|---|---|---|---|---|---|---|
| _ | A | G | A | G | A | G |
| G | A | G | A | G | A | _ |
- | | | | | | |
|---|---|---|---|---|---|
| A | G | A | G | A | _ |
| _ | G | A | G | A | G |

Score:

3

5

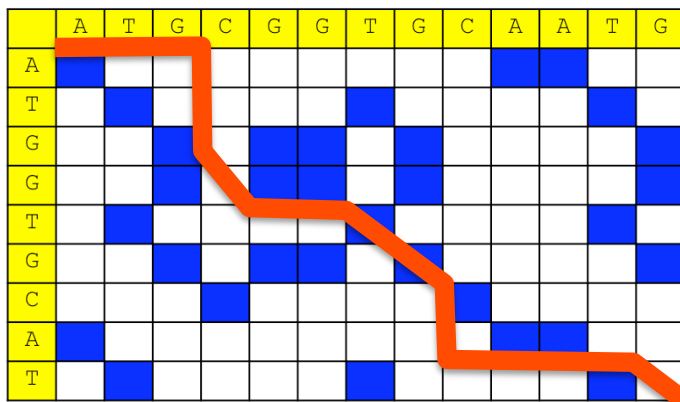
2

2

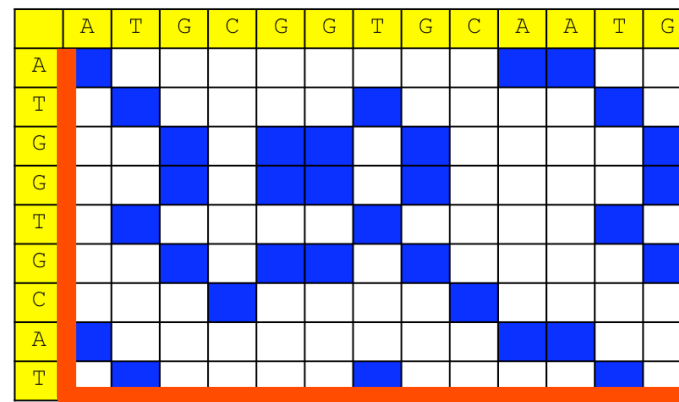
Alignments und Dotplots 2

- Sei $|A|=m$, $|B|=n$
- Betrachte **Pfade** im Dotplot von (1,1) nach (m,n)
 - Alignment: Sei A über B angeordnet
 - Jeder Pfad startet in der linken oberen Ecke
 - Schritt nach rechts: Nächstes Zeichen von A; „_“ in B
 - Schritt nach unten: Nächstes Zeichen von B; „_“ in A
 - Schritt nach rechts-unten: Nächstes Zeichen von A und B

ATG ___ CGGTG ___ CAATG
 ___ ATGG ___ TGCA ___ T

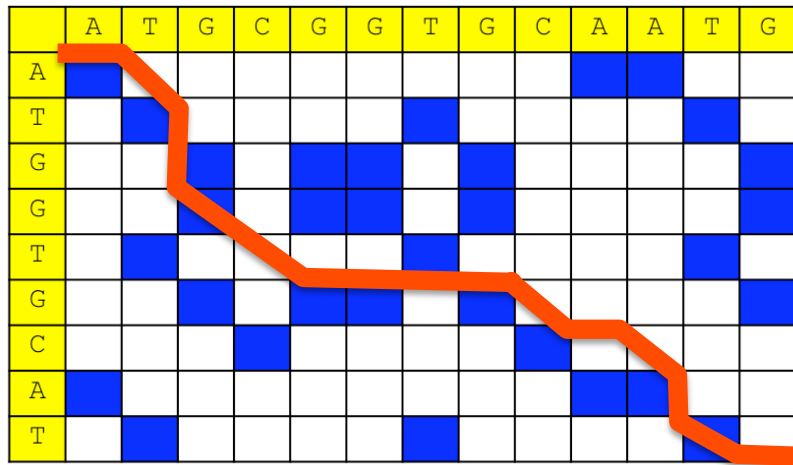


_____ ATGCGGTGCAATG
 ATGGTGCCAT _____

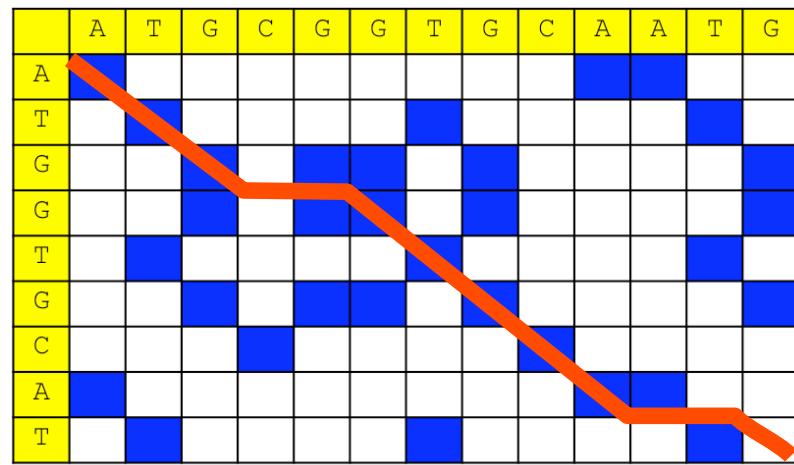


Beispiele

AT__GCGGTGCAA_TG
 _ATGGT_____G_CAT_



ATGCGGTGCAATG
 ATG__GTGCA__T



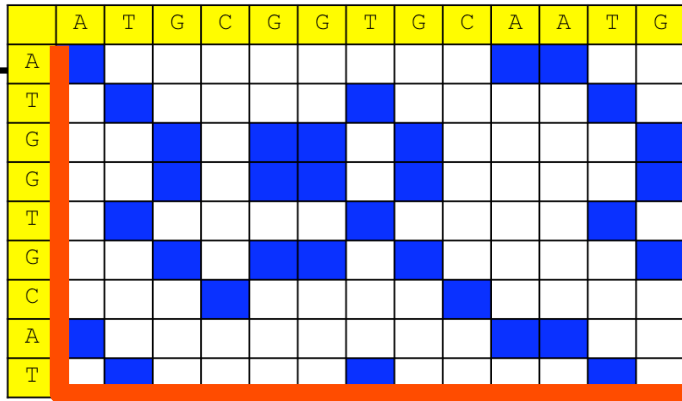
Pfadgüte

- „Gute“ Alignments haben möglichst viele Matches
 - Matches im Dotplot sind die „1er“ Felder
- Definition

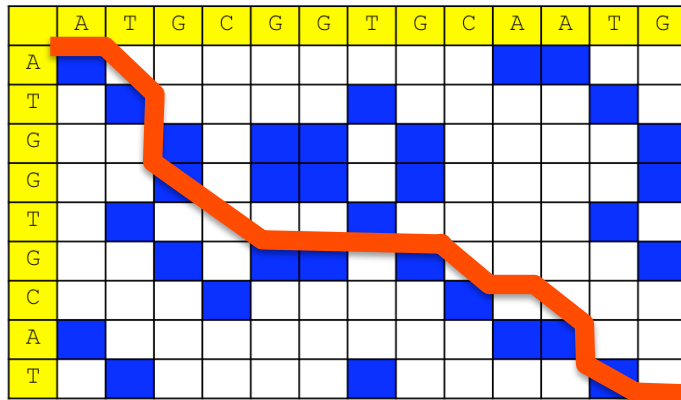
*Die **Güte eines Pfades** P durch einen Dotplot M für Strings A, B ist die Anzahl an durchquerten 1'er Feldern*

 - Bzw. blauen Feldern
- Bemerkung
 - Schritte nach rechts oder unten können nicht zählen
 - Der beste Pfad kann also höchstens Güte $\min(m,n)$ haben

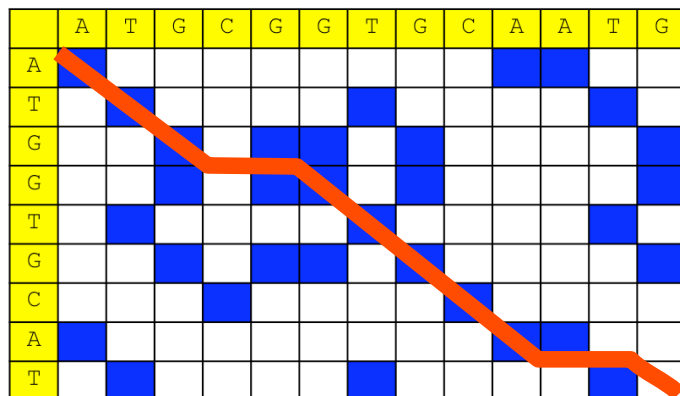
Beispiele



Pfadgüte: 0



Pfadgüte: 2



Pfadgüte: 8

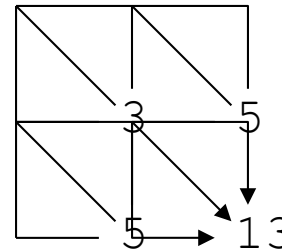
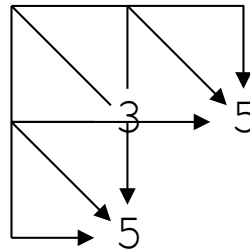
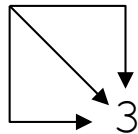
Maximale Güte?

Pfadgüte und Alignmentsscore

- „Gute“ Alignments haben viele Matches
 - Pfadgüte: Matches im Dotplot
 - Alignmentabstand: Anzahl von `_` plus Anzahl von Mismatches
- Berechnung der Güte eines Pfades ist (offensichtlich) fast das gleiche Problem wie das Finden des optimalen Alignments
 - Einziger (oberflächlicher) Unterschied:
 - Pfadgüten sollen gross sein
 - Alignmentsscores sollen klein sein

Algorithmus

- Naives Verfahren um den besten Pfad zu finden
 - Alle Pfade aufzählen
 - Das sind **exponentiell viele**

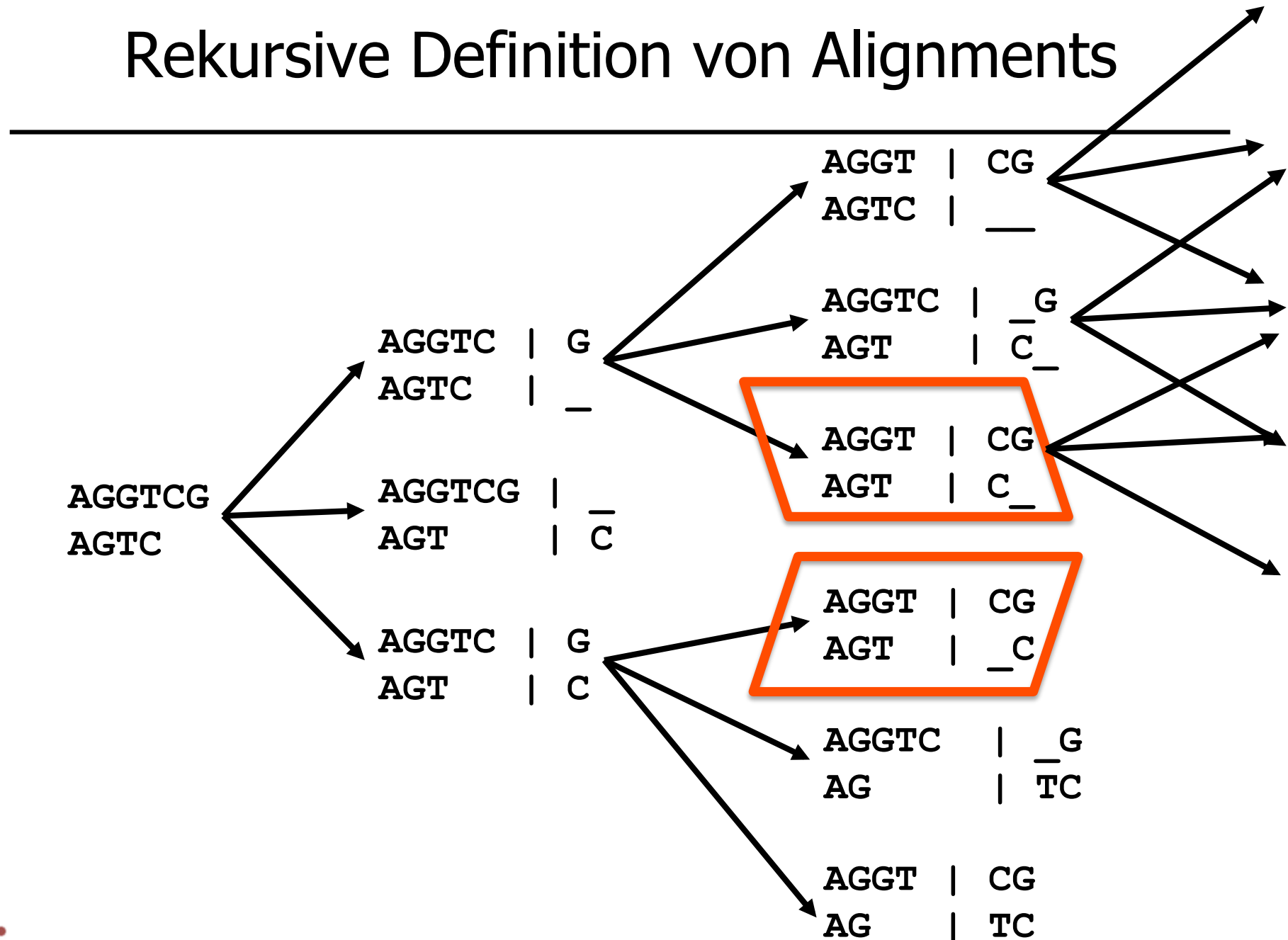


- Nur Pfade „um“ die Hauptdiagonale: $> 3^{\min(m,n)}$
 - **Inakzeptable Laufzeit**
- Tatsächliche Komplexität des Problems: $O(m*n)$
 - Das erfordert noch einige Arbeit

Berechnung von Editabständen

- Definition.
Gegeben zwei Strings A, B mit $|A|=n, |B|=m$
 - Sei $\text{dist}(A,B)$ der *Editabstand* von A und B
 - Sei $d(i,j)$, $0 \leq i \leq n$ und $0 \leq j \leq m$, der Editabstand zwischen $A[1..i]$ und $B[1..j]$
- Bemerkungen
 - Offensichtlich gilt: $d(n, m) = \text{dist}(A,B)$
 - Definition von $d(i,j)$ dient zur *rekursiven Berechnung* von $\text{dist}(A,B)$

Rekursive Definition von Alignments



Rekursionsgleichung

- Wir leiten das nächste Symbol im Editskript aus schon bekannten Editabständen ab
- Wir suchen das kürzeste Skript, also

$$d(i, j) = \min \left\{ \begin{array}{l} d(i, j-1) + 1 \\ d(i-1, j) + 1 \\ d(i-1, j-1) + t(i, j) \end{array} \right\}$$

$$t(i, j) = \begin{cases} 1 : \text{wenn } A[i] \neq B[j] \\ 0 : \text{sonst} \end{cases}$$

Randbedingungen

- **Randbedingungen** nicht vergessen
 - $d(i,0) = i$
 - Um $A[1..i]$ zu „“ zu transformieren braucht man i Deletions
 - $d(0,j) = j$
 - Um $A[1..0]$ zu $B[1..j]$ zu transformieren braucht man j Insertions

Zusammen

- Theorem

- Der *Editabstand zweier Strings* A, B mit $|A|=n$, $|B|=m$ berechnet sich mit Startbedingung

$$d(i,0) = i \quad d(0,j) = j$$

als $d(n,m)$ mit folgender *Rekursionsgleichung*

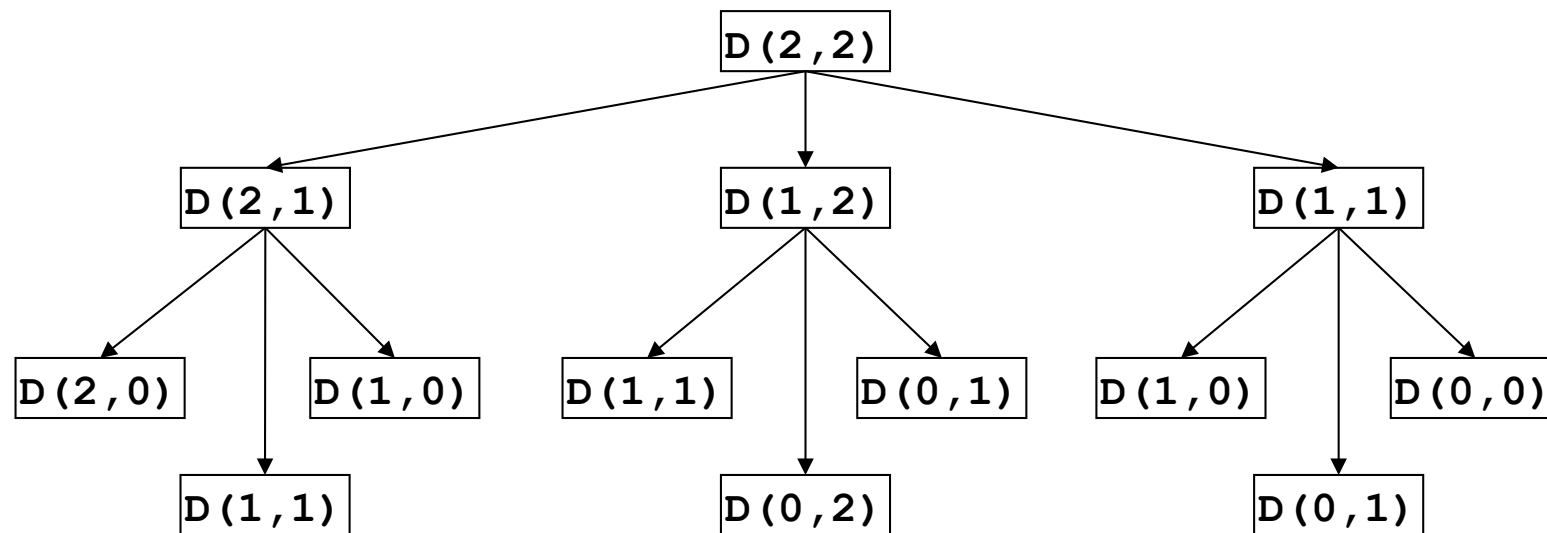
$$d(i,j) = \min \left\{ \begin{array}{l} d(i,j-1) + 1 \\ d(i-1,j) + 1 \\ d(i-1,j-1) + t(i,j) \end{array} \right\}$$

Rekursiver Algorithmus

```
function d(i,j) {
    if (i = 0)           return j;
    else if (j = 0)      return i;
    else
        return min (    d(i-1,j) + 1,
                        d(i,j-1) + 1,
                        d(i-1,j-1) + t(A[i],B[j]));
}
function t(c1, c2) {
    if (c1 = c2)      return 0;
    else                 return 1;
}
```

Sicher nicht optimal

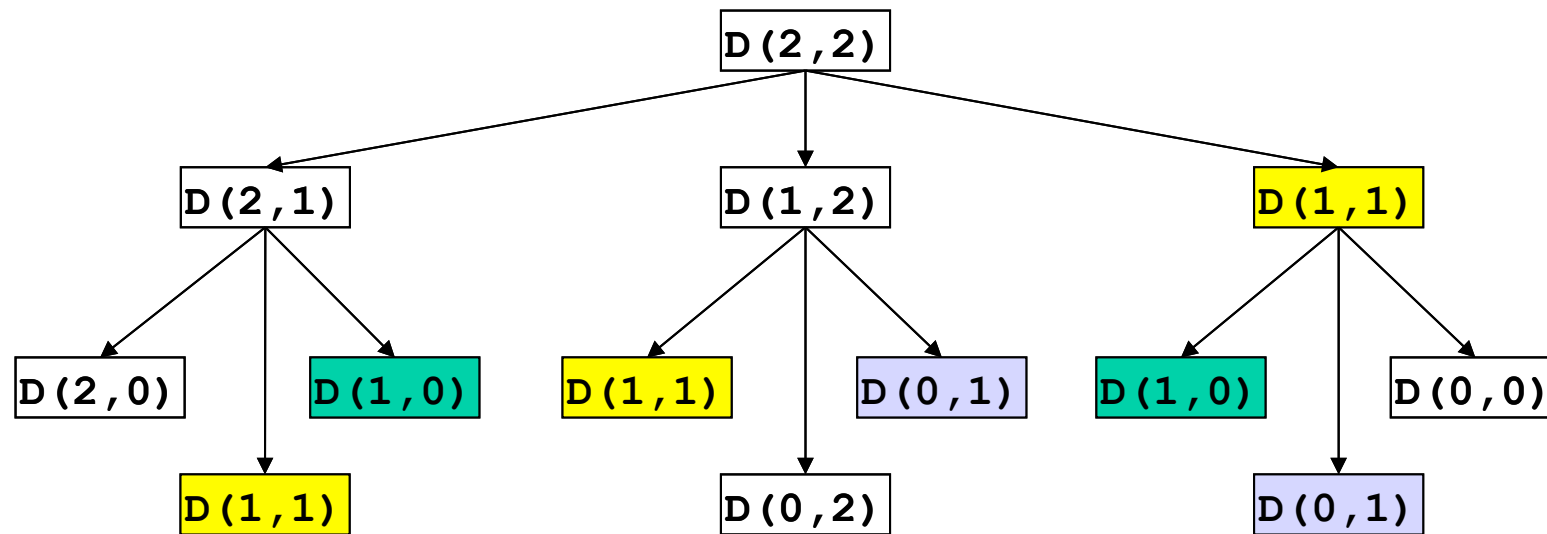
- Aufrufbaum mit Parametern



- Optimierungspotential?

Sicher nicht optimal

- Durch die Rekursionsgleichung werden viele Teillösungen mehrfach berechnet



- Es gibt nur $(n+1) \cdot (m+1)$ verschiedene Aufrufe
- Wie kann man die redundanten Berechnungen sparen?

Beispiel

$$d(i, j) = \min \left\{ \begin{array}{l} d(i, j-1) + 1 \\ d(i-1, j) + 1 \\ d(i-1, j-1) + t(i, j) \end{array} \right\}$$

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1							
T	2							
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0						
T	2							
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2							
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3							
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4							

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

Was ist gewonnen?

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

- Editabstand von ATGG, ATGCGGT ist 3
- Wir suchen auch das **Alignment**, nicht nur den Abstand
- Extraktion aus der Tabelle durch „Traceback“
 - Bei Berechnung von $d(i,j)$ behalte Pointer auf minimale Vorgängerzelle(n)
 - Die muss nicht eindeutig sein

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

		A	T	G	C	G	G	T
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

Vom Pfad zum Alignment

	A	T	G	C	G	G	T	
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

- Jeder Pfad von (n,m) nach $(1,1)$ ist ein optimales Alignment
 - Starte von $(1,1)$
 - Nach rechts: Deletion in A
 - Nach unten: Insertion in A
 - Diagonal: Match/Replace

	A	T	G	C	G	G	T	
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

ATGCGGT
ATG _ _

	A	T	G	C	G	G	T	
	0	1	2	3	4	5	6	7
A	1	0	1	2	3	4	5	6
T	2	1	0	1	2	3	4	5
G	3	2	1	0	1	2	3	4
G	4	3	2	1	1	1	2	3

ATGCGGT
AT _ _ GG _

Beispiel 2

		0	1	2	3	4	5	6	7	
		w	r	i	t	e	r	s		
0		0	1	2	3	4	5	6	7	
1	v	1	1	2	3	4	5	6	7	
2	i	2	2	2	2	3	4	5	7	
3	n	3	3	3	3	3	4	5	6	
4	t	4	4	4	4	3	4	5	6	
5	n	5	5	5	5	4	4	5	6	
6	e	6	6	6	6	5	4	5	6	
7	r	7	7	6	7	6	5	4	5	

		0	1	2	3	4	5	6	7	
		w	r	i	t	e	r	s		
0		0	1	2	3	4	5	6	7	
1	v	1	1	2	3	4	5	6	7	
2	i	2	2	2	2	3	4	5	7	
3	n	3	3	3	3	3	4	5	6	
4	t	4	4	4	4	3	4	5	6	
5	n	5	5	5	5	4	4	5	6	
6	e	6	6	6	6	5	4	5	6	
7	r	7	7	6	7	6	5	4	5	

		0	1	2	3	4	5	6	7	
		w	r	i	t	e	r	s		
0		0	1	2	3	4	5	6	7	
1	v	1	1	2	3	4	5	6	7	
2	i	2	2	2	2	3	4	5	7	
3	n	3	3	3	3	3	4	5	6	
4	t	4	4	4	4	4	4	5	6	
5	n	5	5	5	5	4	4	5	6	
6	e	6	6	6	6	5	4	5	6	
7	r	7	7	6	7	6	5	4	5	

		0	1	2	3	4	5	6	7	
		w	r	i	t	e	r	s		
0		0	1	2	3	4	5	6	7	
1	v	1	1	2	3	4	5	6	7	
2	i	2	2	2	2	3	4	5	7	
3	n	3	3	3	3	3	4	5	6	
4	t	4	4	4	4	3	4	5	6	
5	n	5	5	5	5	4	4	5	6	
6	e	6	6	6	6	5	4	5	6	
7	r	7	7	6	7	6	5	4	5	

WRIT_ERS
VINTNER_

WRI_T_ERS
V_INTNER_

WRI_T_ERS
VINTNER

Komplexität

- Aufbau der Tabelle
 - Zur Berechnung einer Zelle muss man genau drei andere Zellen betrachten
 - Konstante Zeit pro Betrachtung
 - $m \cdot n$ Zellen
 - Insgesamt: $O(m \cdot n)$
- Traceback
 - Man kann einen beliebigen Pfad wählen
 - Es muss einen Pfad von (n, m) nach $(1, 1)$ geben
 - Jede Zelle hat mindestens einen Pointer
 - Keine Zelle zeigt aus der Tabelle hinaus
 - Worst-Case Pfadlänge ist $O(m+n)$
 - Für ein Alignment
- Zusammen
 - $O(m \cdot n)$ (für $m \cdot n > m+n$)

Prinzip des dynamischen Programmierens

- Ziel: Optimierung einer Zielfunktion
 - Hier: Editabstand
- Prinzip
 - Berechnung von Lösungen für „große“ Problemen aus Lösungen für „kleinere“ Probleme
 - Weiteres Beispiel: Kürzeste Wege in Graphen
- Drei Bestandteile
 - Berechnung aus **optimalen Teillösungen**
 - Hier: Rekursionsgleichung
 - **Zwischenspeichern der Teillösungen**
 - Hier: Tabelle mit Werten $d(i,j)$
 - **Rückverfolgung** der Lösung aus Tabelle
 - Hier: Traceback des Alignments

Zusammenfassung

- Editabstand und Alignierung ineinander überführbar
- Berechnung eines optimalen Alignments hat **quadratische Komplexität** (wenn $m=n$)
 - Mittels dynamischer Programmierung
 - Tabelle aufbauen, Pfad zurückverfolgen
- **Platzbedarf ist auch quadratisch**
 - Das ist kritisch
 - Es gibt Erweiterungen mit linearem Platzbedarf
- Wichtige Erweiterungen: Substitutionsmatrizen
 - Berücksichtigen die **Biologie der Aufgabe**
 - Mutationswahrscheinlichkeiten, Aminosäureähnlichkeiten, Evolutionsmodell, ...
 - Substitutionsmatrizen machen approximatives Stringmatching erst anwendbar in der Bioinformatik