

Algorithmische Bioinformatik

Approximatives Stringmatching
Edit-Abstand



Ulf Leser
Wissensmanagement in der
Bioinformatik



Ziele dieser Vorlesung

- Einstieg in exaktes approximatives Stringmatching
- Bedeutung für die Bioinformatik
- Verständnis Editabstand, Ähnlichkeit, Alignment
- Berechnung mittels dynamischer Programmierung

Inhalt dieser Vorlesung

- Approximative Stringvergleiche
- Edit-Abstand und Alignment
- Effiziente Berechnung
- Varianten

Exakter approximativer Stringvergleich

- Definiere eine **Abstandsfunktion d (distance)**
- Kernproblem: Für zwei Strings A, B : Berechne $d(A,B)$
- Anwendungen
 - Finde alle Substrings B' von B mit $d(A,B') < k$
 - Finde Substring B' von B , so dass $d(A,B')$ minimal ist
 - Gegeben A und eine Menge S von Strings: Finde $s \in S$ mit $d(A,s) = \min$
- Basis: Wie definiert man den Abstand zweier Strings?
 - Oft eine Metrik: $d(A,A)=0$; $d(A,B)=d(B,A)$; $d(A,C) \leq d(A,B), d(B,C)$

Mögliche Maße

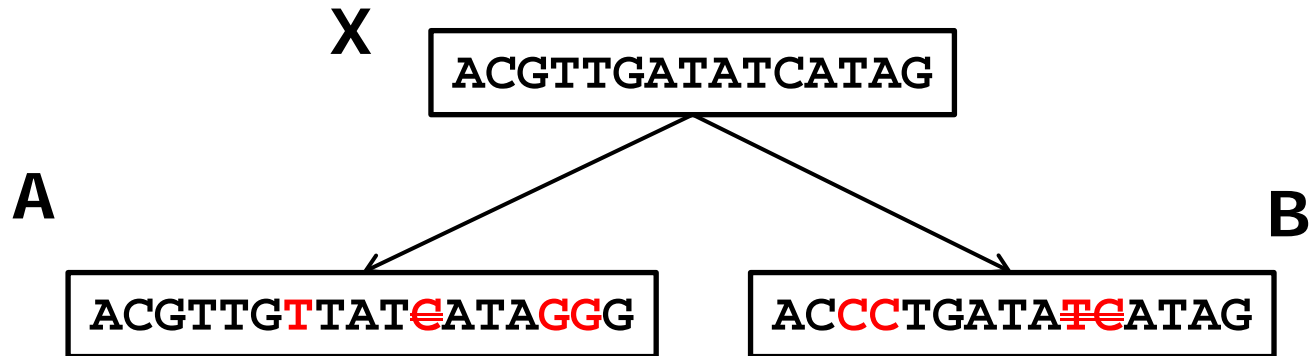
- Hamming-Abstand
 - Voraussetzung: $|A|=|B|$
 - Vergleiche A und B Zeichen für Zeichen
 - Hamming-Abstand = **Anzahl der Mismatches**
 - Verwendung: Korrektur bei digitaler Signalübertragung
 - Beispiel: $ha(CGTGCTCGC, ACGTGCTCG) = 9$
- Jaccard-Abstand (auf q-Grammen)
 - Berechne **alle q-Gramme** von A und B
 - Abstandsfunktion (über Mengen oder Multimengen)

$$d(A, B) = \frac{|qgram(A) \cap qgram(B)|}{|qgram(A) \cup qgram(B)|}$$

Biologie

- **Funktion** von Proteinen wird bestimmt durch Sequenz
 - Sequenz → 3D-Faltung → Domänen → Interaktion/Reaktion
- Proteinsequenz wird bestimmt durch **Sequenz des Gens**
- Sind sich zwei **Proteinsequenzen sehr ähnlich**, dann haben sie sehr wahrscheinlich die gleiche Funktion
- Sind sich zwei (hinreichend lange) Sequenzen sehr ähnlich, dann kann das kein Zufall sein
- Annahme: Sie hängen **evolutionär** zusammen

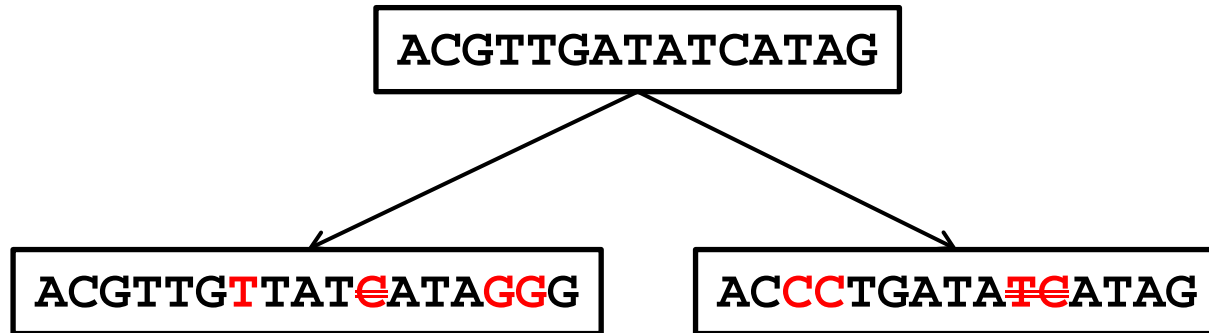
Evolution



- Ähnlichkeit kommt vom **gemeinsamen Ursprung**
- Evolution erfolgt in **kleinen Schritten**
 - Einfügung einer Base (INS), Löschen (DEL), Ersetzen (REP)
- Wie „viel“ Evolution ist bei $X \rightarrow A$ und $X \rightarrow B$ passiert?
 - Wir kennen nur A und B
 - Wenn Evolution zufällig (im Sequenzraum) erfolgt, gilt

$$d(X,A) + d(X,B) \sim d(A,B)$$

Evolution



- Definition von d
 - Wir zählen primitive evolutionäre Ereignisse (INS/DEL/REP)
 - Wir suchen die **kleinste Anzahl Ereignisse**, die X in A/B verwandelt haben kann (Edit-Abstand)
 - **Biologisch motiviertes** Abstandsmaß

Sequenz und Funktion

- Sequenz und Funktion hängen eng zusammen, sind aber nicht direkt ableitbar
 - Proteine: Ab 20-30% Ähnlichkeit geht man von verwandter Funktion aus
 - Aber: Schon einzelne Mutation kann Funktion verändern
- Bestimmung von Funktion ist extrem aufwändig (wenn überhaupt möglich)
- Bestimmung von Sequenzen dagegen sehr billig
- Idee: Annäherung der **Funktion über Sequenzähnlichkeiten**
 - Geburtsüberlegung der Bioinformatik
 - Basiert auf approximativem Stringmatching

Realität ist komplizierter

- Sequenzen heißen
 - **Homolog**, wenn sie einen gemeinsamen Ursprung haben und von diesem durch Evolution divergiert sind
 - **Ortholog**, wenn sie in verschiedenen Spezies vorkommen, aber vom gleichen „Vorfahren“ abstammen
 - **Paralog**, wenn sie durch Duplikation innerhalb einer Spezies entstanden sind
- Wir ersetzen Homologie durch hohe Sequenzähnlichkeit
- Wir unterscheiden nicht zwischen Paralogen und Orthologen
 - Offen: Sind orthologe Proteine funktional ähnlicher als paraloge?

Positionen sind nicht egal

Wildtyp	CTTAGTGACTACGGTAAA	DNA
	Leu Ser Asp Tyr Gly Lys	Protein
Fatale Mutation	CTTAGTGACTAGGGTAAA	DNA
	Leu Ser Asp Stop-Codon	Protein
Leserastermutation	CTTAGTGAACAACGGTAAA	DNA
	Leu Ser His Asp Leu Thr	Protein
Neutrale Mutation	CTTAGCGACTACGGTAAA	DNA
	Leu Ser Asp Tyr Gly Lys	Protein
Funktionale Mutation	CTTAGTGAAATACGGTAAA	DNA
	Leu Ser Glu Tyr Gly Lys	Protein

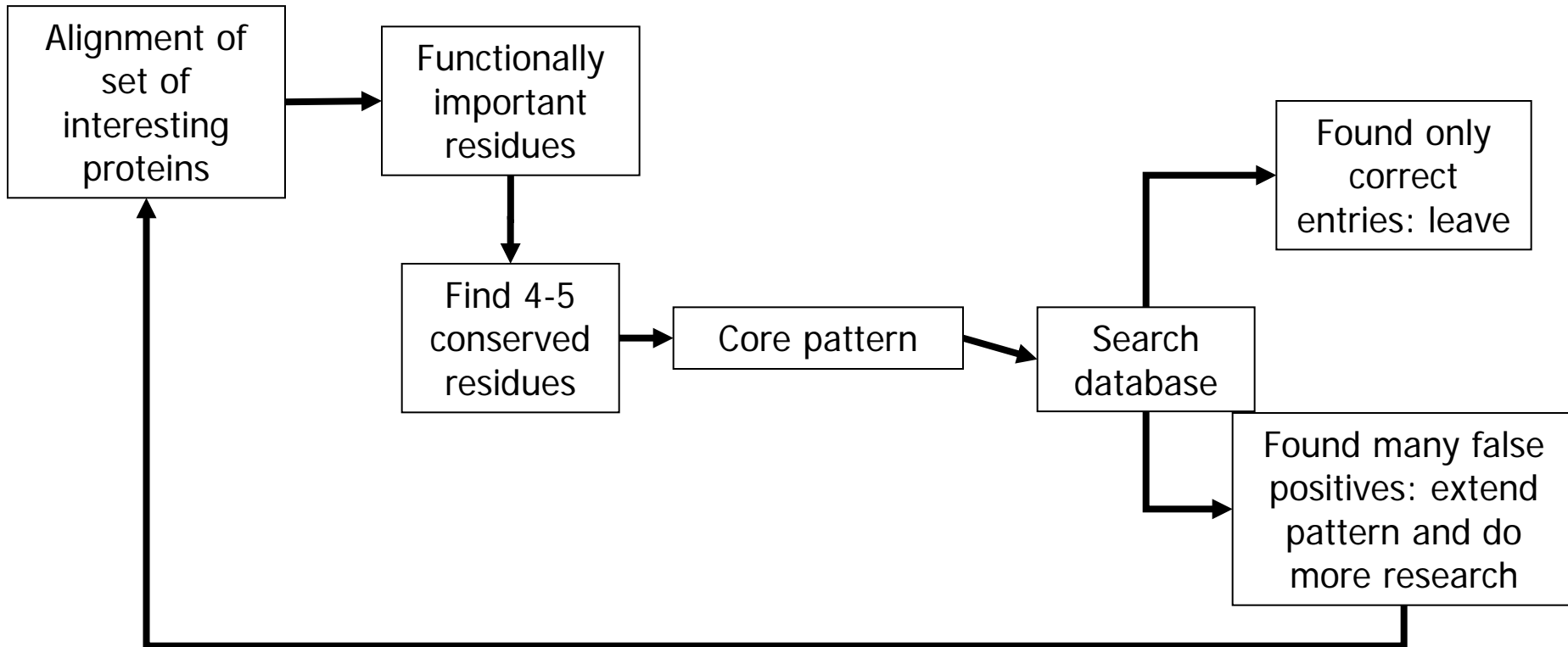
Was wir behandeln

- **Globales Alignment:** Wie ähnlich sind sich zwei Strings?
 - Heisst auch „Needleman-Wunsch“
- **Lokales Alignment:** Wie ähnlich sind sich die zwei ähnlichsten Substrings zweier Strings?
 - Heisst auch „Smith-Waterman“
- Datenbankformulierung. Gegeben Sequenz S und Sequenzdatenbank D
 - Finde die k Sequenzen aus D, die S am global ähnlichsten sind
 - Finde die k Sequenzen aus D, die **S am lokal ähnlichsten sind**
 - BLAST

Anwendungsbeispiel: Proteindomänen

- Pattern, Domäne, Motiv, Site: Teile einer Proteinsequenz mit **funktionaler Bedeutung**
 - Bindungsstellen, enzymatische Aktivität, Signal, etc.
 - Beschrieben durch reguläre Ausdrücke, Fingerprints, Profile, ...
- Datenbanken von Domänen
 - PROSITE, Pfam, InterPro, BLOCKS, PRINTS, ...
- Beispiel PROSITE
 - Beginn: Finden einer „interessanten“ Teilsequenz in der Literatur
 - Identifikation ähnlicher Sequenzen in anderen Proteinen
 - **Approximatives Stringmatching**
 - Identifikation der konservierten Aminosäuren
 - **Multiple Sequence Alignment**

Entstehung von Prosite Pattern



Pattern given as kind-of **regular expression**:

[AC]-x-V-x(4)-{ED}

ala/cys-any-val-any-any-any-any-(any except glu or asp)

Außerhalb der Bioinformatik

- Unschärfe Suche in Texten
 - Suche mit „Xylofon“ und finde auch „Xhylophon“
- Personenabgleich / Entity Matching / **Deduplication**
 - Ist „Herr Müller, 27, Stargarder Str 54“ identisch zu „Hr. Mueller, 27, Stagarder Str. 54“ ?
- Phonetische Suche (soundex)
 - Finde alle Meyer, Meier, Maier, Mair, ...



Inhalt dieser Vorlesung

- Approximative Stringvergleiche
- Edit-Abstand und Alignment
- Effiziente Berechnung
- Varianten

Dotplot

- Definition

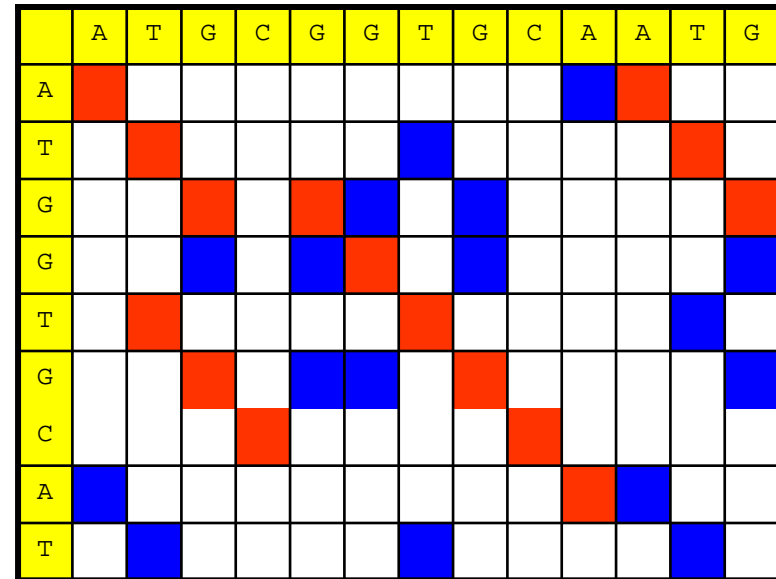
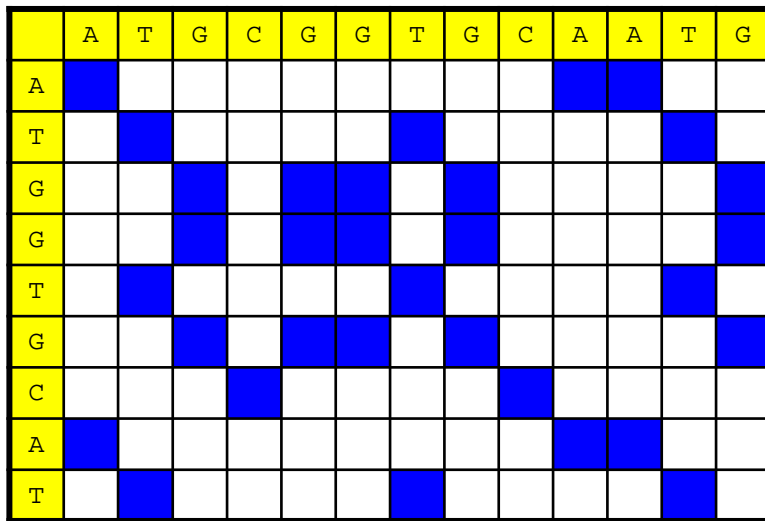
Ein *Dotplot* zweier Strings A , B ist eine Matrix M mit

- Die Spalten entsprechen den Zeichen von A
- Die Zeilen entsprechen den Zeichen von B
- $M[a,b]=1$ gdw. $A[a] = B[b]$; sonst 0

	A	T	G	C	G	G	T	G	C	A	A	T	G
A	1	0	0	0	0	0	0	0	0	1	1	0	0
T	0	1	0	0	0	0	1	0	0	0	0	1	0
G	0	0	1	0	1	1	0	1	0	0	0	0	1
G	0	0	1	0	1	1	0	1	0	0	0	0	1
T	1	0	0	0	0	0	1	0	0	0	0	1	0
G	0	0	1	0	1	1	0	1	0	0	0	0	1
C	0	0	0	1	0	0	0	0	1	0	0	0	0
A	1	0	0	0	0	0	0	0	0	1	1	0	0
T	0	1	0	0	0	0	1	0	0	0	0	1	0

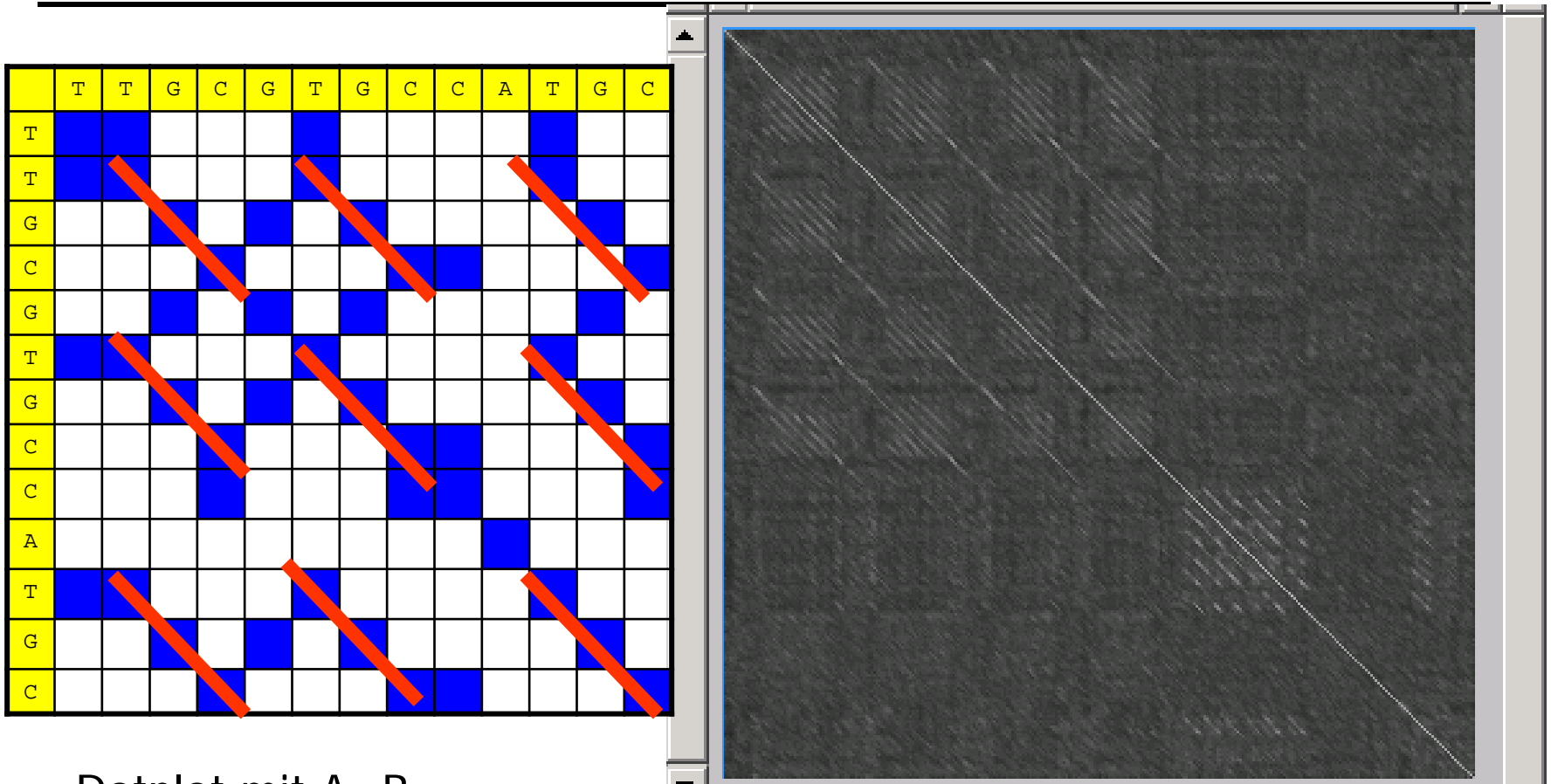
Dotplot und gleiche Teilstrings

- Wie erkennt man **gleiche Teilstrings** im Dotplot?



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

Repetitive Sequenzen



- Dotplot mit $A=B$
 - Zitat (Genbank, P24014):
[SIMILARITY] CONTAINS 7 EGF-LIKE DOMAINS.
[SIMILARITY] Contains 24 leucine-rich (LRR) repeats.

Finden längster gemeinsamer Teilstrings

- Gegeben Dotplot M zweier Strings A, B
- Gesucht: Längster gemeinsamer Teilstring
- Naives Verfahren ($|A| = |B| = m$)
 - Prüfe jede der $2 \cdot m$ Diagonalen: $O(m)$
 - Suche zusammenhängende Sequenzen von 1'ern: $O(m)$
 - Merke das längste zusammenhängende Stück
 - Komplexität: $O(m^2)$
 - (Zusätzlich: Konstruktion von M - wie komplex?)
- Wir kennen schon lineare Algorithmen
- Außerdem wollen wir approximativ matchen

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

– MIMMMR	IRMMMDI
A_TGTA	_ATGTA_
AGTGTC	AGTGT_C

Editabstand

- Offensichtlich gibt es immer unendlich viele Editskripte
- Definition
 - Die *Länge eines Editskript* ist die Anzahl von Operationen o im Skript mit $o \in \{I, R, D\}$
 - Der *Editabstand* (oder *Levenshtein-Abstand*) zweier Strings A, B ist die Länge des *kürzesten Editskript* für A, B
- Bemerkung
 - Matches zählen nicht – interessant sind nur die Änderungen
 - Es gibt oft verschiedene kürzeste Editskripte
 - | | |
|----------------|----------------|
| IMMMMD | DMMMMMI |
| _AGAGAG | AGAGAG_ |
| GAGAGA_ | _GAGAGA |

Alignment

- Definition

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

- Beispiele

– A_TGT_A	A_T_GTA	_AGAGAG	AGAGAG_
AGTGTC_	_AGTGTC	GAGAGA_	_GAGAGA

Score:

3

5

2

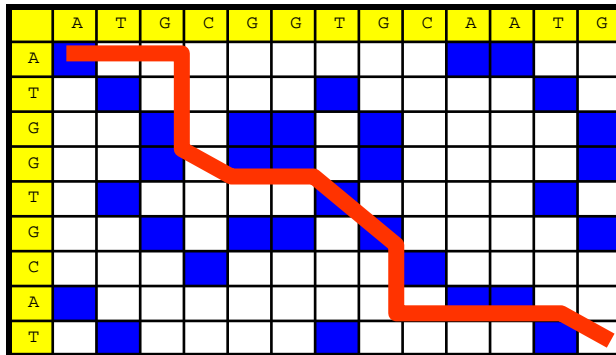
2

Alignments und Dotplots

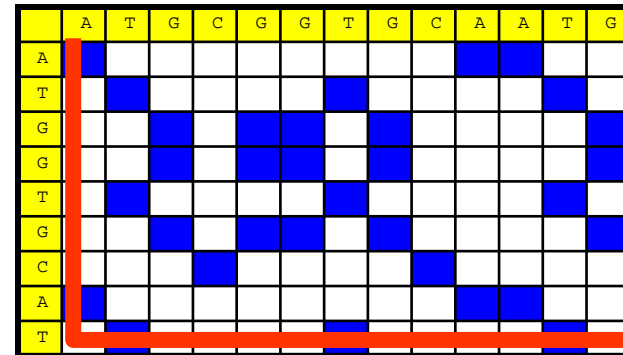
- Übersetzung von **Pfaden im Dotplot in Alignments**

- Dotplot: Sei A horizontal und B vertikal aufgetragen
- Alignment: Sei A über B angeordnet
- 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



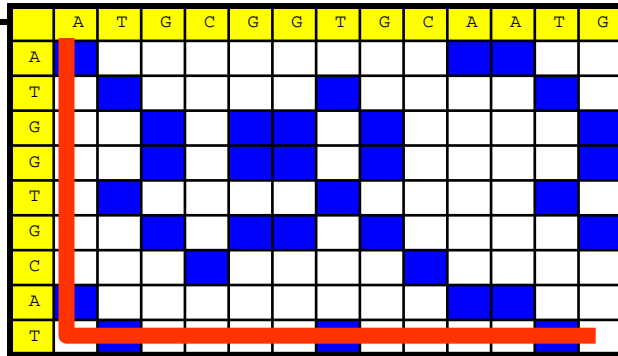
_____ATGCGGTGCAATG
 ATGGTGCCAT_____



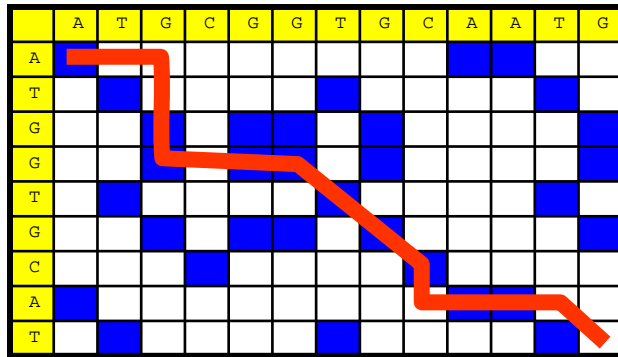
Pfadgüte

- „Gute Pfade“ haben viele Matches (1'er Felder)
- Definition
 - Die *Güte eines Pfades* P durch einen Dotplot M ist die Anzahl an diagonal durchquerten 1'er Feldern
 - Die *Länge eines Pfades* P durch einen Dotplot M ist die Anzahl an Schritten, die nicht diagonal durch 1'er Felder laufen
- Bemerkung
 - Der beste Pfad kann also höchstens Güte $\min(m,n)$ haben

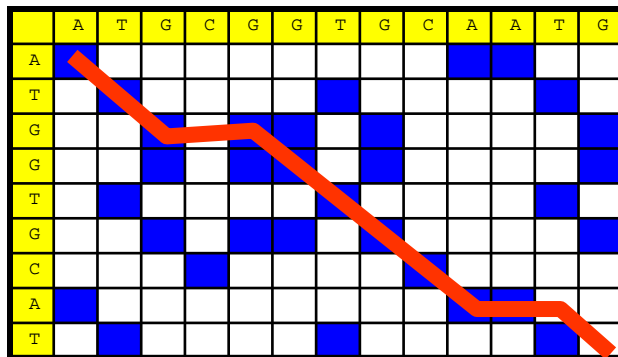
Beispiele



Pfadgüte: 0



Pfadgüte: 4



Pfadgüte: 9

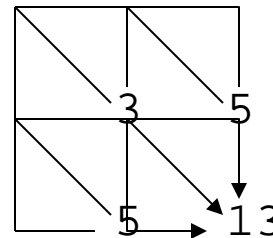
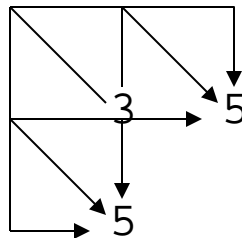
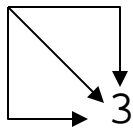
Maximale Güte?

Äquivalenzen

- Gegeben zwei Strings A,B und deren Dotplot M
- Die folgenden Probleme sind **äquivalent**
 - Finde das **optimale Alignment** von A und B
(= Alignmentabstand)
 - Finde die **minimale Menge an Editoroperationen** von A nach B
(= Editabstand)
 - Finde in M den **Pfad mit minimaler Länge**
(= Pfadlänge)
- Beweis: Einfach
- Wir verwenden im Folgenden meistens Alignments
 - Einfacher zu lesen, weniger redundant, platzsparend

Algorithmus

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



- Nur Pfade „um“ die Hauptdiagonale: $> 3^{\min(m,n)}$
 - Genaue Anzahl Pfade: Übungsaufgabe
 - Inakzeptable Laufzeit
- Tatsächliche **Komplexität des Problems**: $O(m \cdot n)$

Inhalt dieser Vorlesung

- Approximative Stringvergleiche
- Edit-Abstand und Alignment
- Effiziente Berechnung
- Varianten

Editabstände

- Definition

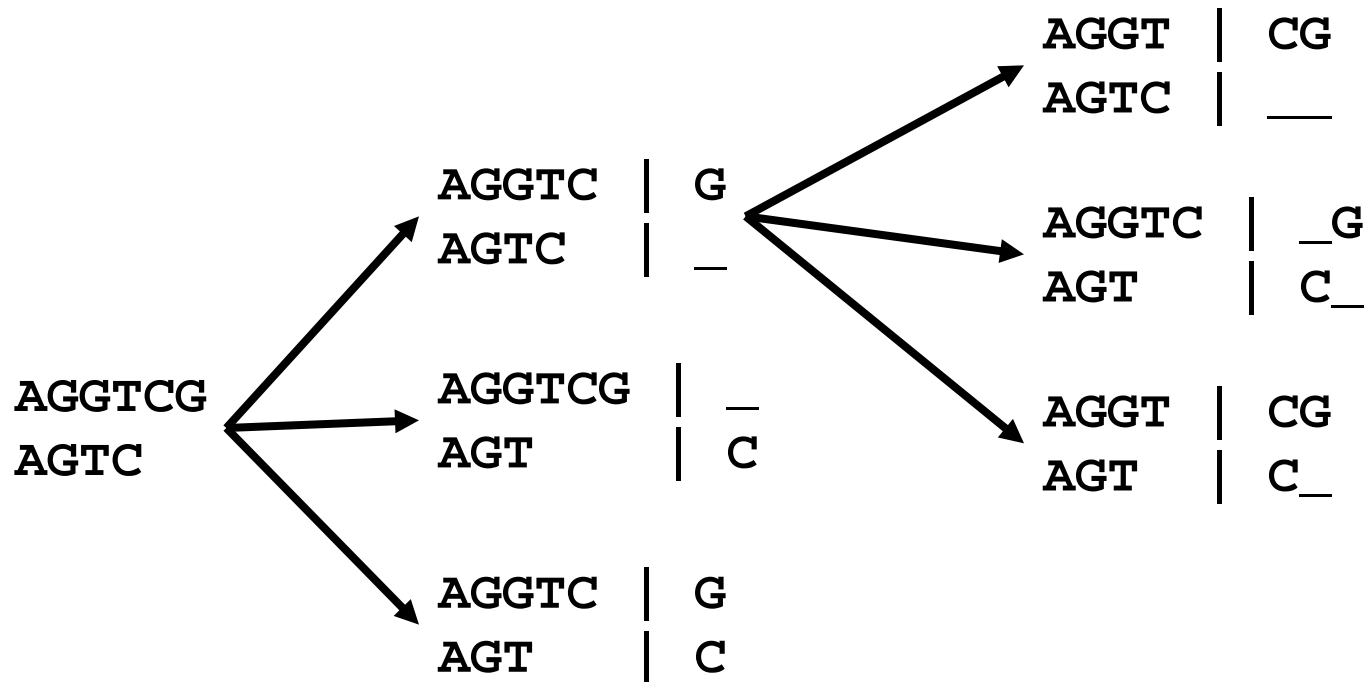
Gegeben zwei Strings A , B mit $|A|=n$, $|B|=m$

- *Funktion $\text{dist}(A,B)$ berechne den Editabstand von A , B*
- *Funktion $d(i,j)$, $0 \leq i \leq n$ und $0 \leq j \leq m$, berechne den Editabstand zwischen $A[1..i]$ und $B[1..j]$*

- Bemerkungen

- Offensichtlich: $d(n,m) = \text{dist}(A,B)$
- $d(i,j)$ dient zur rekursiven Berechnung von $\text{dist}(A,B)$
- Divide-and-Conquer: Wie kann man $d(i,j)$ aus „kleineren“ $d(x,y)$ Werten berechnen?

Rekursive Betrachtung



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*

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

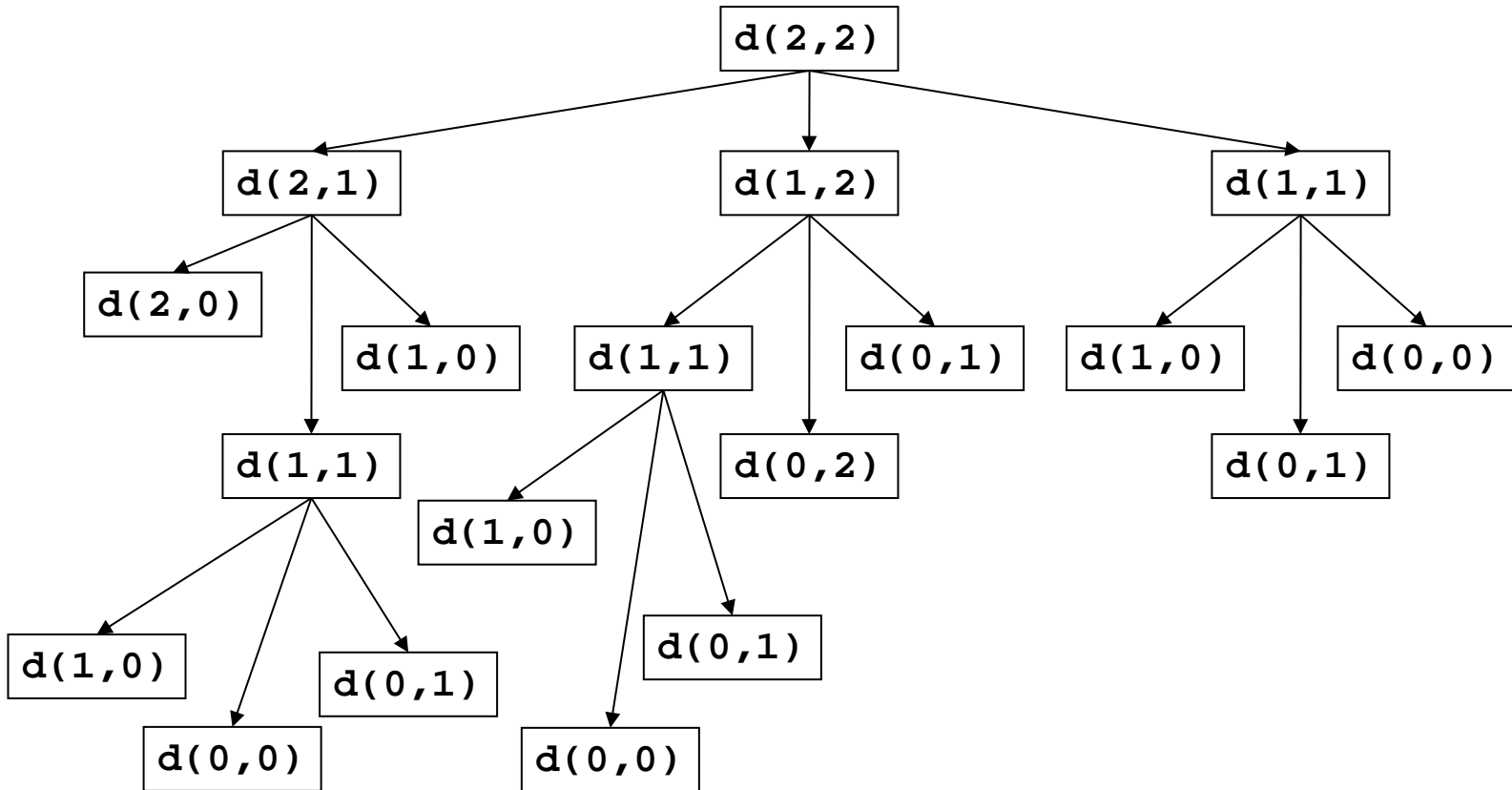
wobei $t(i, j) = 0$ wenn $A[i]=B[j]$ sonst 1

Rekursiver Algorithmus

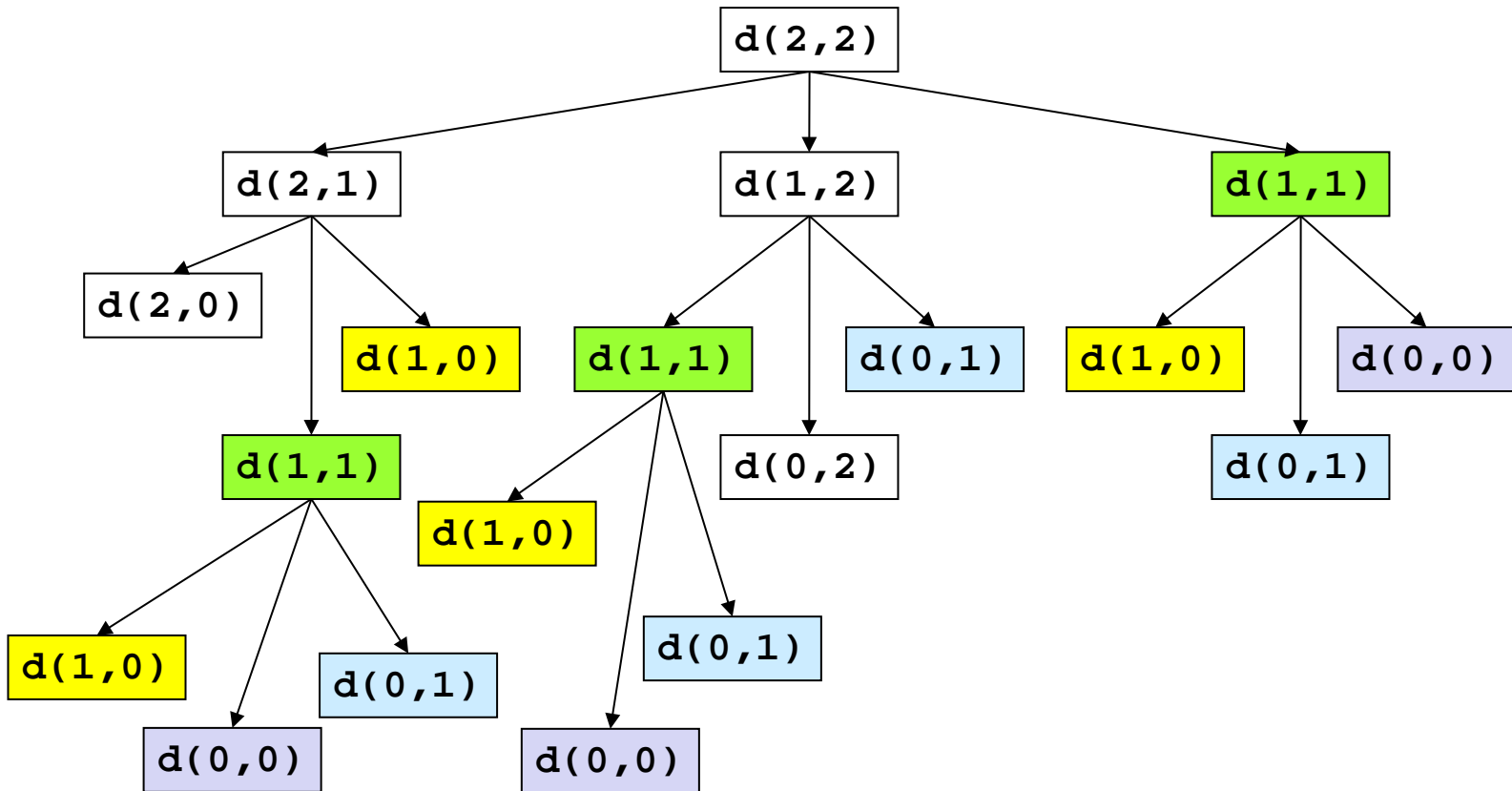
```
function d(i,j) {
    if (i = 0)          return j;
    else if (j = 0)     return i;
    else
        return min (   d(i-1,j) + 1,
                       d(i,j-1) + 1,
                       d(i-1,j-1) + t(A[i],B[j]));
}
function t(c1, c2) {
    if (c1 = c2)     return 0;
    else              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

- Komplexität?
 - Für (n,m) erfolgen 3 Aufrufe, die wiederum jeweils 3 Aufrufe auslösen, die ...
 - Komplexität damit **mindestens** $O(3^{\min(n,m)})$

Aufrufbaum



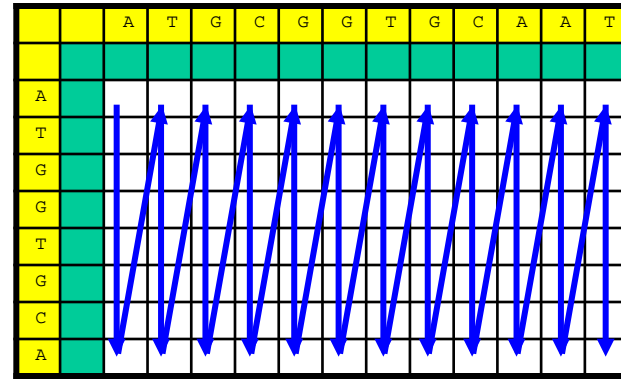
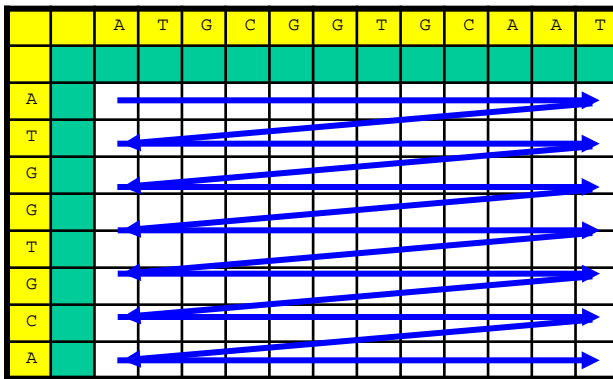
Redundanz



Es gibt nur $(n+1) * (m+1)$ verschiedene Aufrufe

Tabellarische Berechnung

- Grundidee
 - Speichern der Teillösungen in Tabelle
 - Bei Berechnung: Wiederverwendung wo immer möglich
- Aufbau der Tabelle: Bottom-Up (statt rekursiv Top-Down)
 - **Initialisierung** mit festen Werten $d(i,0)$ und $d(0,j)$
 - **Sukzessive Berechnung** von $d(i,j)$ mit steigendem i,j
 - Für $d(i,j)$ brauchen wir $d(i,j-1)$, $d(i-1,j)$ und $d(i-1,j-1)$
 - Verschiedene Reihenfolgen möglich



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

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

- Editabstand von ATGG, ATGCGGT ist 3
- Wo ist das Alignment?
- **Traceback**
 - Bei Berechnung von $d(i,j)$ behalte Pointer auf Vorgängerzelle(n) mit Minimum
 - Die muss nicht eindeutig sein
 - Alle Wege führen zu gültigen Alignments

		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 $(0,0)$ ist ein optimales Alignment
 - Starte von (n,m)
 - Nach links: Deletion in A
 - Nach oben: 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_G__

	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_

Komplexität

- Aufbau der Tabelle
 - Berechnung einer Zelle betrachtet genau drei andere Zellen
 - $m \cdot n$ Zellen
 - Insgesamt: $O(m \cdot n)$
- Traceback (für ein Alignment)
 - Man kann einen beliebigen Pfad wählen
 - Es muss einen Pfad von (n, m) nach $(0, 0)$ geben
 - Jede Zelle hat mindestens einen Pointer
 - Keine Zelle zeigt aus der Tabelle hinaus
 - Worst-Case Pfadlänge ist $O(m+n)$
- Zusammen
 - $O(m \cdot n)$ (für $m \cdot n > m+n$)

Needleman-Wunsch

- S.B. Needleman, C.D. Wunsch, „A general method applicable to the search for similarities in the amino acid sequence of two proteins“, J. of Molecular Biology, 48, 1970
- Der originale Algorithmus hat **Komplexität $O(n^3)$**
 - Einen Trick übersehen
- „Needleman-Wunsch“ wird trotzdem oft als Synonym für die Idee der dynamischen Programmierung zur Berechnung des Edit-Abstands verwendet

Analogie: Kürzeste Wege in Editgraphen

- Definition. Ein *Editgraph* für A, B mit $|A|=n$, $|B|=m$ ist
 - Graph mit $(m+1) \cdot (n+1)$ Knoten
 - Jeder Knoten ist beschriftet mit Label (i, j) für $0 \leq i \leq n$, $0 \leq j \leq m$
 - Kanten und Gewichte
 - $(i, j-1) \rightarrow (i, j)$ mit Gewicht 1
 - $(i-1, j) \rightarrow (i, j)$ mit Gewicht 1
 - $(i-1, j-1) \rightarrow (i, j)$ mit Gewicht 0 gdw. $A[i]=B[j]$; sonst 1
- Es gilt
 - „Leichteste“ Wege von $(1, 1)$ bis (n, m) sind optimale Alignments
 - Dijkstra's Algorithmus läuft in $O(n \cdot \log(n) + m)$ (n/m : Knoten/Kanten)
 - Also schnellere Lösung?
 - Nein: Wir haben $m \cdot n$ Knoten
 - Aber: Oft muss nicht der ganze Graph traversiert werden

Zusammenfassung

- Berechnung des optimalen globalen Alignments ist $O(m \cdot n)$
- Platzbedarf ist auch $O(m \cdot n)$
 - Wir werden Algorithmen mit linearem Platzbedarf kennen lernen
- Diverse Erweiterungen
 - **Biologie:** Mutationswahrscheinlichkeiten, Aminosäureähnlichkeiten, Evolutionsmodell, ...
 - Erst diese Erweiterungen machen approximatives Stringmatching sinnvoll verwendbar in der Bioinformatik
 - Ändern aber oftmals wenig am Algorithmus

Selbsttest

- Warum interessiert uns in der Bioinformatik gerade der Editabstand zweier Sequenzen?
- Wie ist der Editabstand zweier Strings definiert?
- Berechnen Sie den Editabstand von AGGTC und GCTA mittels dynamische Programmierung auf einem Blatt Papier
- Wiederholen Sie die Berechnung, wenn $c_{I/D}=2$ und $c_R=1$
- Was ist der Needleman-Wunsch Algorithmus?
- Wie viele optimale Alignments kann es durch eine $m \times n$ große Tabelle geben?
- Ist Editabstand eine Metrik?