

Bioinformatik

Zeichenketten und Stringalgorithmen

Ulf Leser

Wissensmanagement in der
Bioinformatik

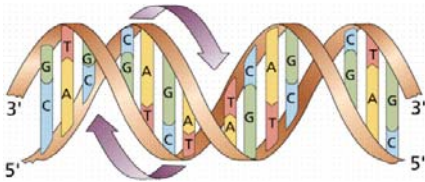


Inhalt dieser Vorlesung

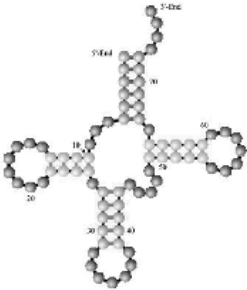
- Warum Stringmatching?
- Strings und Matching
- Naiver Algorithmus

Biomoleküle

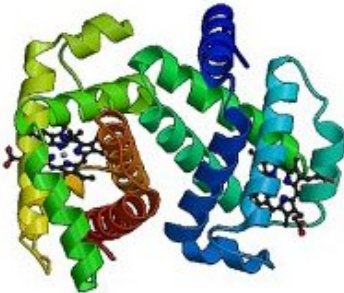
- DNA, RNA und Proteine lassen sich als Zeichenkette über festem Alphabet darstellen



DNA
A C G T



RNA
A C G U



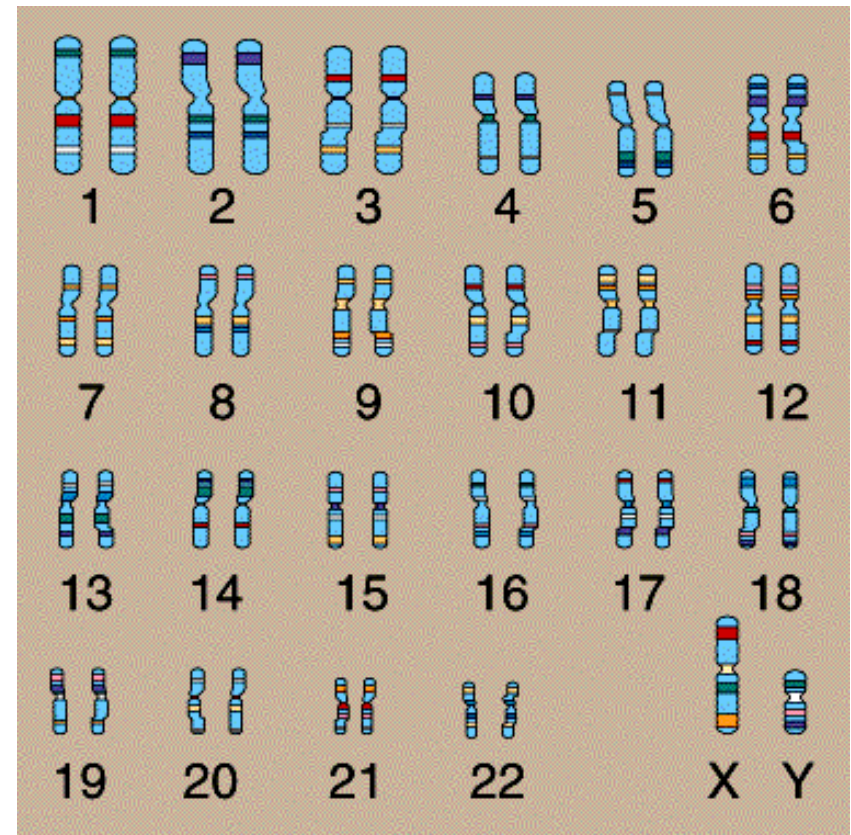
Protein
A C D E F G H I K L M N P Q R S T V W Y

Drei Anwendungen

- Sequenzierung
 - Assembly von Teilsequenzen
- cDNA Clustering
 - All-against-all Sequenzvergleiche
- Funktionale Annotation
 - Schnelle Suche in Sequenzdatenbanken

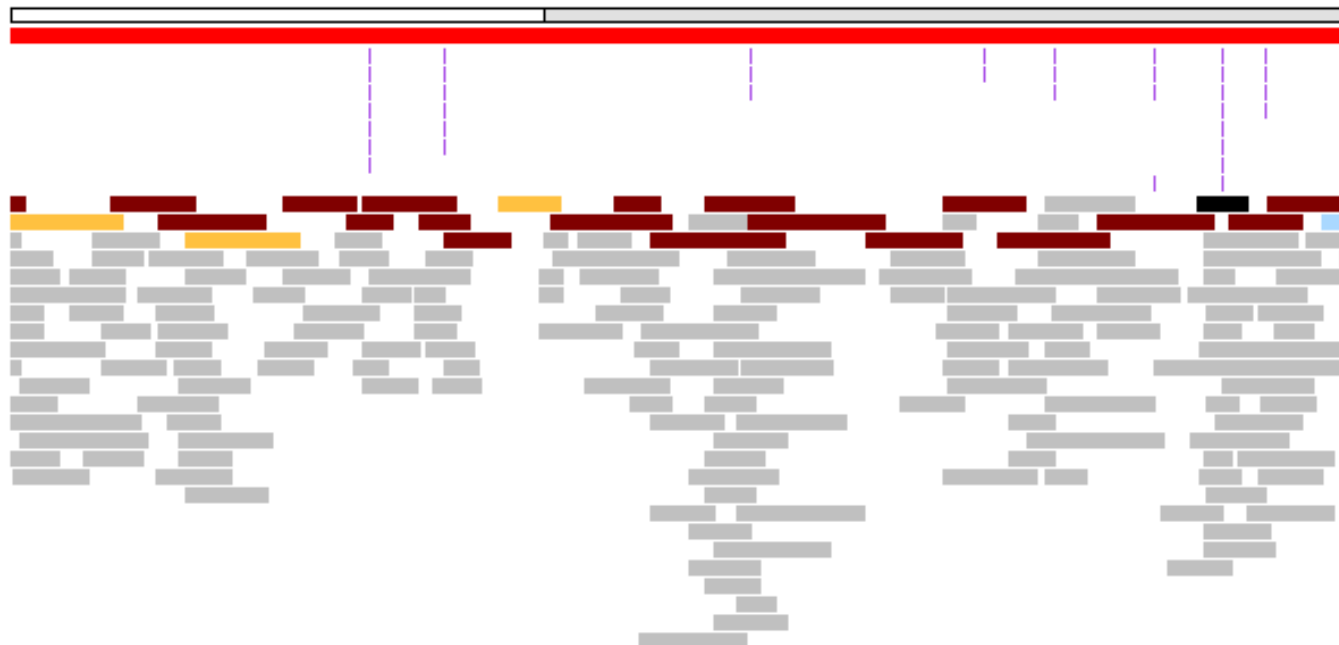
1. Das menschliche Genom

- ... AGGCTGATGGATTAGAGACC ...
- 23 Chromosomenpaare
- ~ 3.000.000.000 Buchstaben
- Chromosomen kann nicht direkt sequenzieren

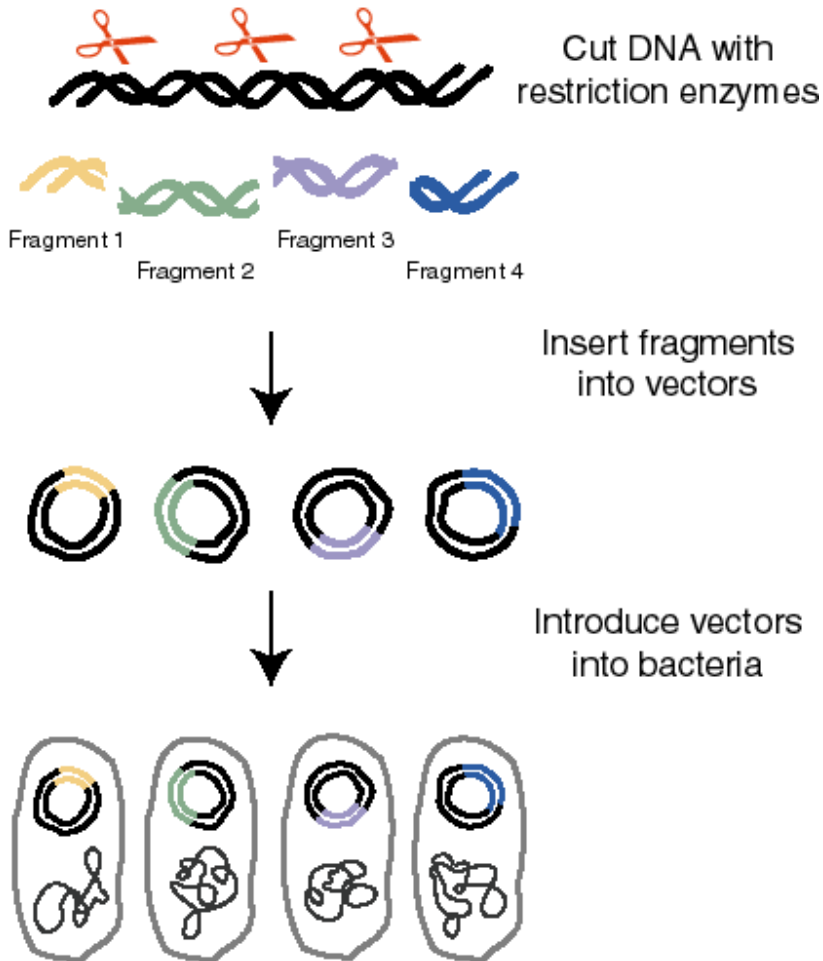


Mapping und Sequenzierung

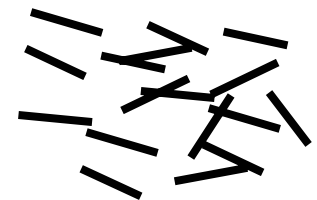
- Zerlegung in Bruchstücke (Clonierung)
- Berechnung aller Überlappungen
- Bestimmung der wahrscheinlichsten Gesamtsequenz
- Variante: Bestimmung des **Minimum Tiling Paths**



Clonieren



- Schneiden des Chromosoms mit Restriktionsenzymen
 - Länge hängt ab von
 - Spezifität des Restriktionsenzym
 - Länge der Behandlung (partieller Verdau)
- Bruchstücke unterschiedlicher Länge
- Auftrennen nach Länge
 - Gelelektrophorese
- Clonierung in Bakterien
 - **Vervielfältigung**
- Ergebnis:



Sequenzierung

- Gegeben: Clone unbekannter Sequenz
- Gesucht: Sequenz

- Unmöglich: Ansehen, Messen, Mikroskop, etc.
- Verfahren von Sanger, 1972:
Radioactive Dideoxy Sequencing

Heute

- Fluoreszente Markierung
- Hochdurchsatz
- Sehr billig

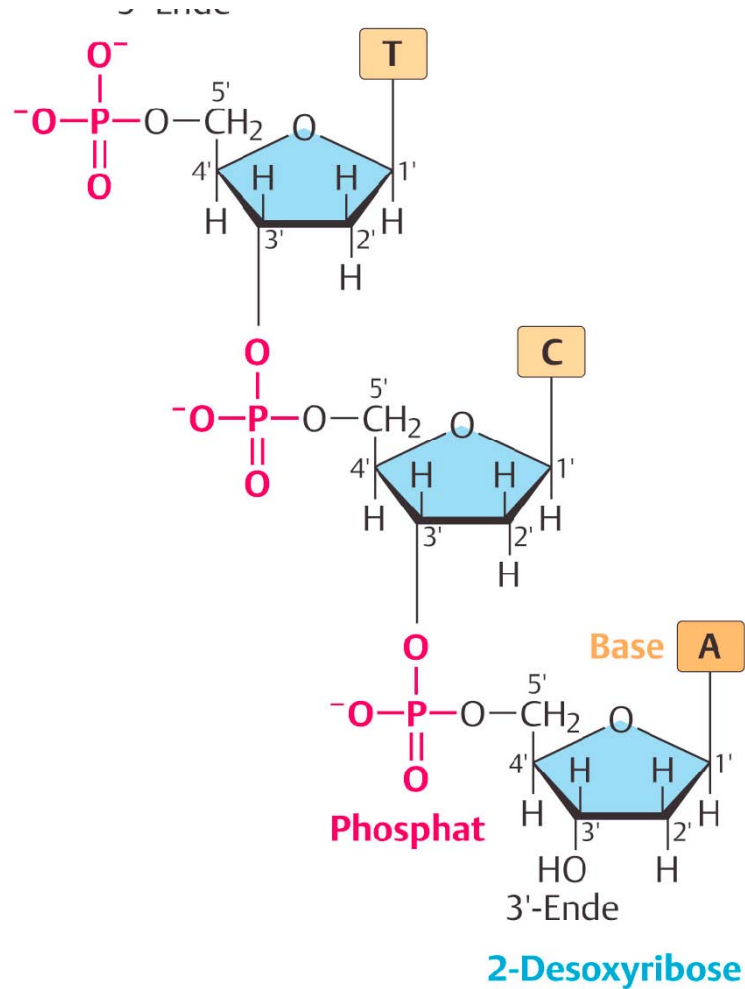


Quelle: <http://www.geneticsplace.com>

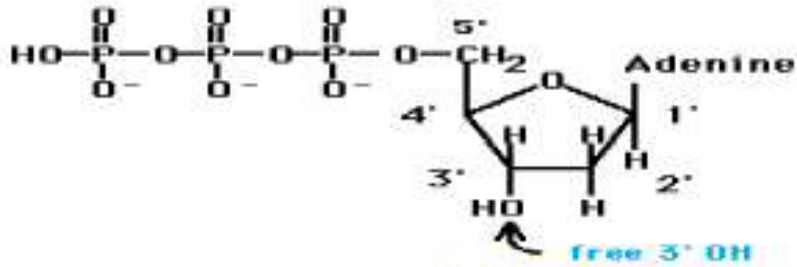
Sequenzierung nach Sanger

- Voraussetzungen
 - Sequenz hat einen definierten Anfang
 - Teil des Clonierungsvektors
 - Dient als Bindungsstelle für Primer
 - Sequenzierungsanfang
 - Polymerase
 - Bindet an doppelsträngigen Abschnitt
 - Verlängert einsträngige DNA entlang des Templates
- Deoxy versus Dideoxy Nucleotide
 - DNA besteht aus Deoxy Nucleotiden (dNTP)
 - Einbau von Dideoxy Nucleotiden (ddNTP) möglich
 - ddNTP stoppt Polymerase

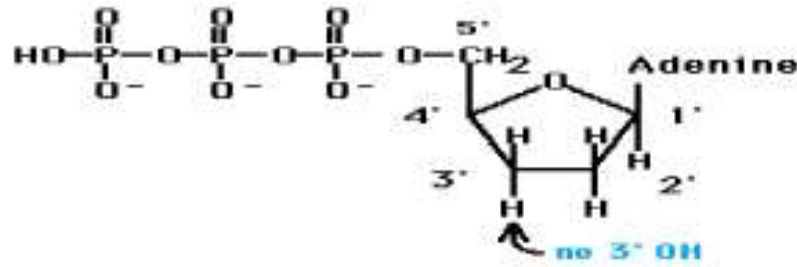
Struktur eines DNA Strangs



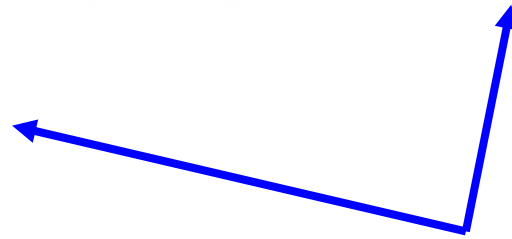
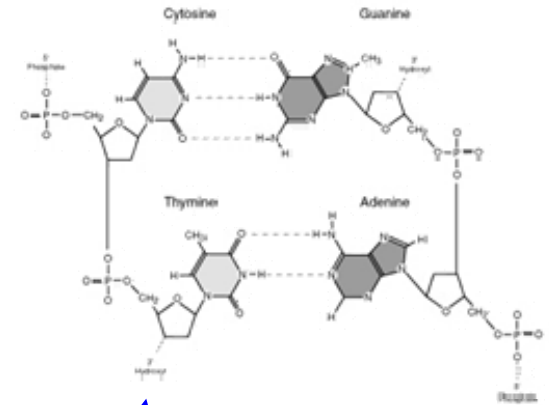
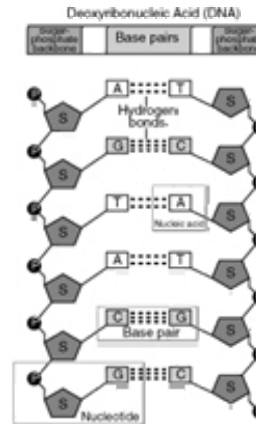
dNTP versus ddNTP



dNTP



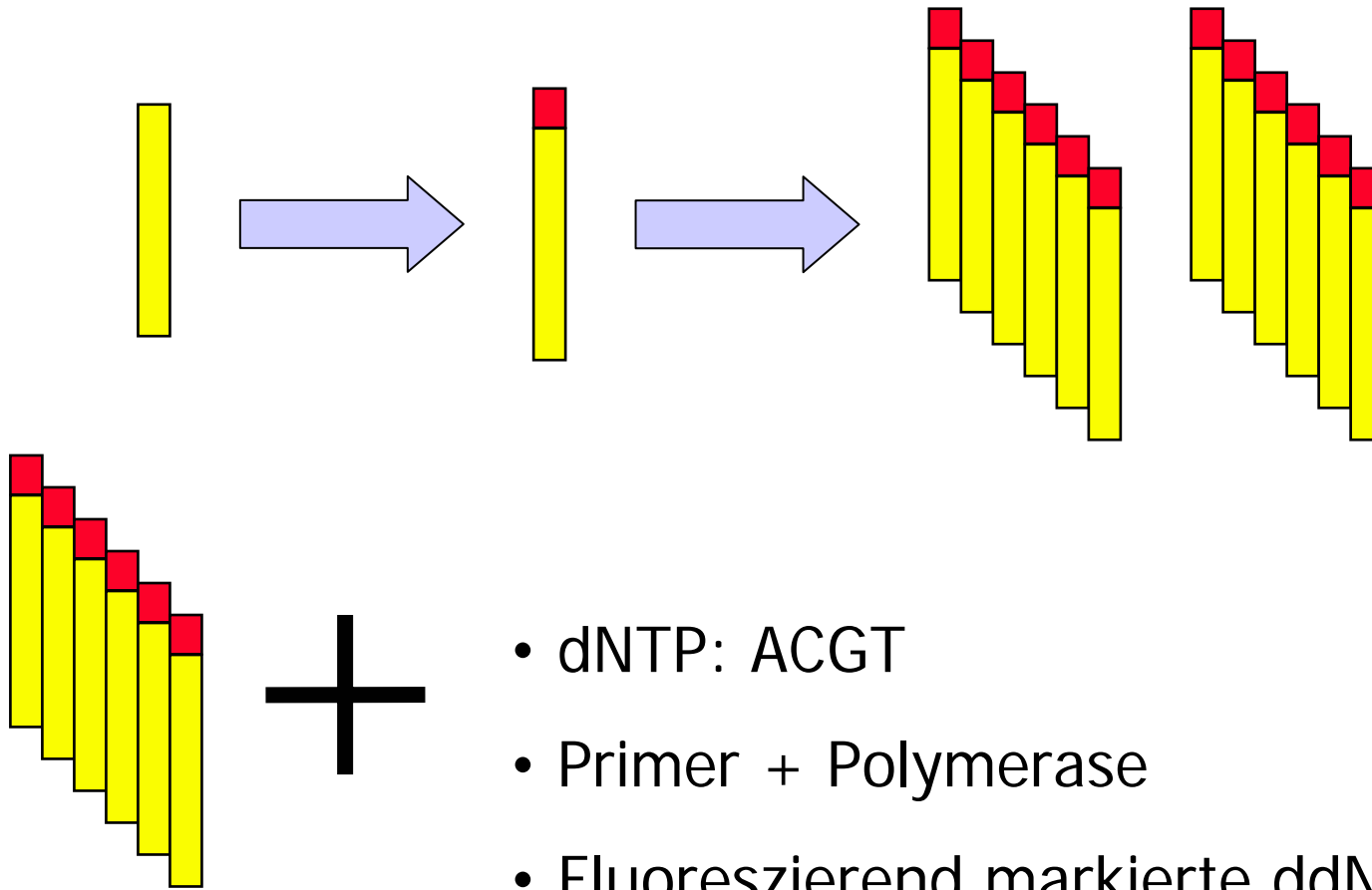
ddNTP



- Dideoxy-Base: keine freie OH Gruppe
- Dideoxy-Base wird eingebaut
- Danach können keine weiteren Basen mehr angehängt werden
- Polymerase fällt ab



Schritt 1 und 2



- dNTP: ACGT
- Primer + Polymerase
- Fluoreszierend markierte ddNTP: ACGT

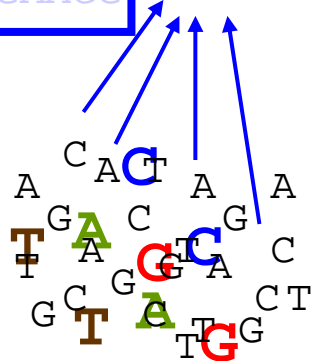
Schritt 3

Primer

Template

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

Polymerase



ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCGAGTT**A**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCGA**G**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCGAGTTAGT**T**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

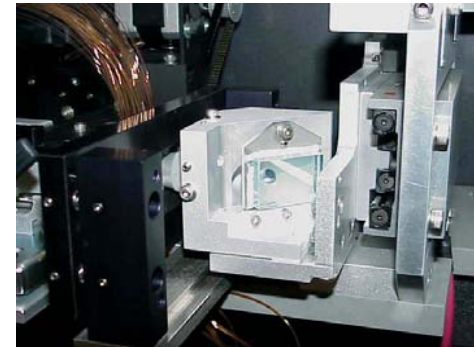
ACGAACGCGAGTTAGTTAG**T**

ACGAACGCGAGTTAGTTAGACCAGTAGCCA...

ACGAACGCG**A**

Schritt 4

Laser &
Detektoren



ACGAACGCGAGTT**A**

ACGAACGCGA**G**

ACGAACGCGAGTTAGT**T**

ACGAACGCGAGTTAGTTAG**T**

ACGAACGCG**A**

Gel / Kapillar
Elektrophorese

ACGAACG**C**

ACGAACG**C**G

ACGAACGCG**A**

ACGAACGCGA**G**

ACGAACGCGAG**T**

ACGAACGCGAGT**T**

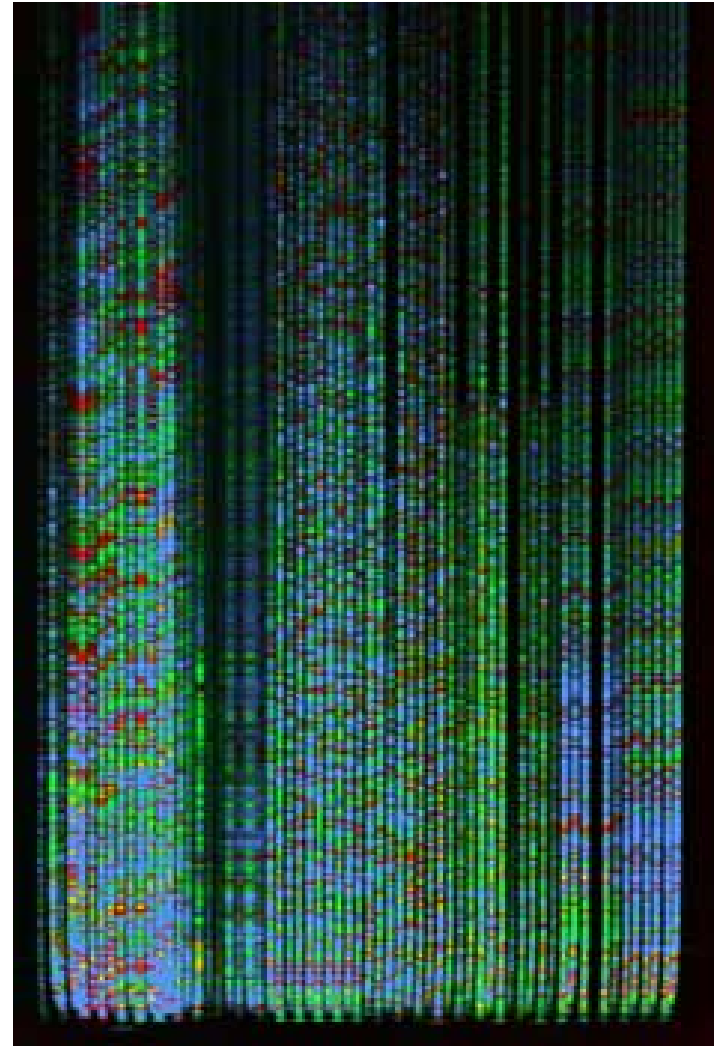
ACGAACGCGAGTT**A**

ACGAACGCGAGGTTA**G**



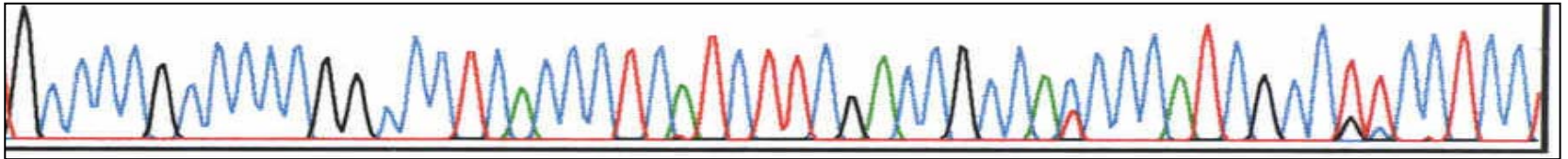
Ergebnis (roh)

- Heutige Geräte
 - >36 Läufe parallel
 - Kapillarelektrophorese (statt Gelen)
 - Direktes Laden von 96 Quellen
- Sanger
 - Radioaktive Markierung
 - 4 Mischungen (A,G,T,P)
 - 4 Gele (Linien)



Ergebnis (Zwischenprodukt)

- Signalverarbeitung (Rauschen, ...)



- Übersetzung in **Traces**
 - 4 Arrays, jedes für eine Farbe
 - Intensitätswerte in regelmäßigen Zeitabschnitten
- Theoretisch
 - Peaks entdecken
 - Immer nur eine Farbe
 - Sequenz zuordnen

Vom Tracefile zur Sequenz

- Tracefiles sind Rohdaten der Sequenzierung
- Verschiedene Verfahren / Tools, um aus Tracefiles Sequenzen zu berechnen
- Komplexe Probleme
 - Base Calling
 - [Assembly](#)
 - Finishing

Assembly

- Finden von Überlappungen von Sequenzen
- Redundanz ist
 - Notwendig: Verbindung von Teilstücken nur durch Überlappungen
 - Konflikträchtig: Widersprüche, Mehrdeutigkeit
- Beachtung von Sequenzierfehlern
- Auswahl der plausibelsten Anordnungen

Fehler ?

```
tggacaagcaaagattga
      acatttttgaac
gcaaagattgctg
      tggacaagcaaagattga
acatttttgaac
      gcaaagattgctg
```

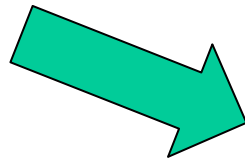
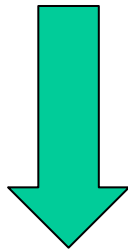
Greedy Algorithmus?

accgtaaagcaaagatta

aaagcaaagattattg

aagattattgaaccggt

attattgccagta

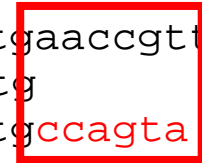


accgtaaagcaaagatta

aagattattgaaccggt

aaagcaaagattattg

attattgccagta



accgtaaagcaaagatta
aagattattgaaccggt

aaagcaaagattattg

attattgccagta

Assembly – Abstrakte Formulierung

- **SUPERSTRING**

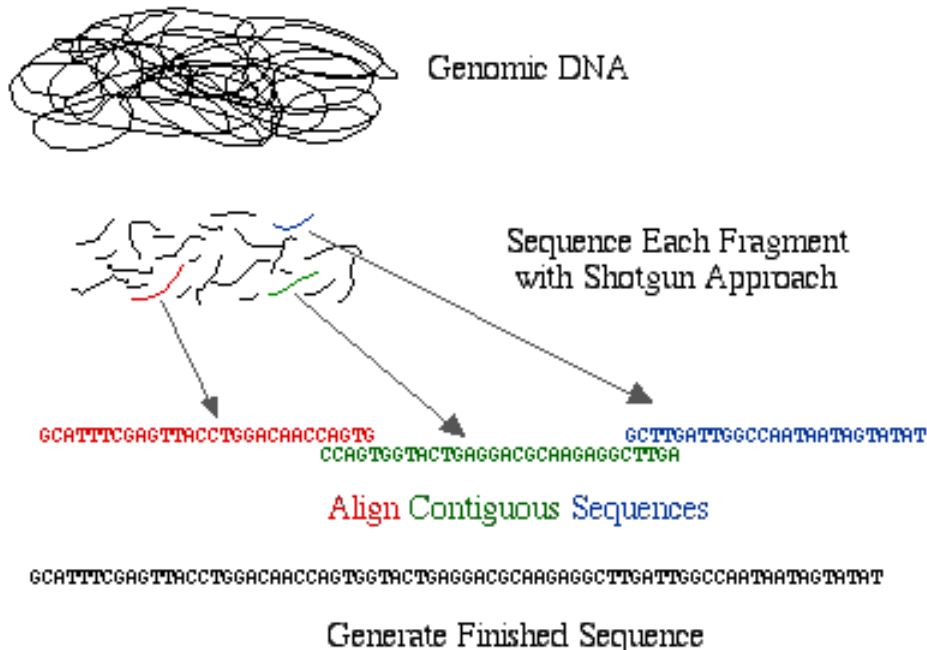
- Geg.: Menge S von Strings
- Ges.: String T so, dass
 - (a) $\forall s \in S: s \in T$ (s Substring von T)
 - (b) $\forall T'$, für die (a) gilt, gilt: $|T| \leq |T'|$ (T ist minimal)
- NP-vollständig

- **Assembly: Verschärfungen von SUPERSTRING**

- Fehler in Sequenzen (s „ungefähr Substring“ von T)
- Zwei Orientierungen von s möglich

Problemdimension: Whole Genome Shotgun

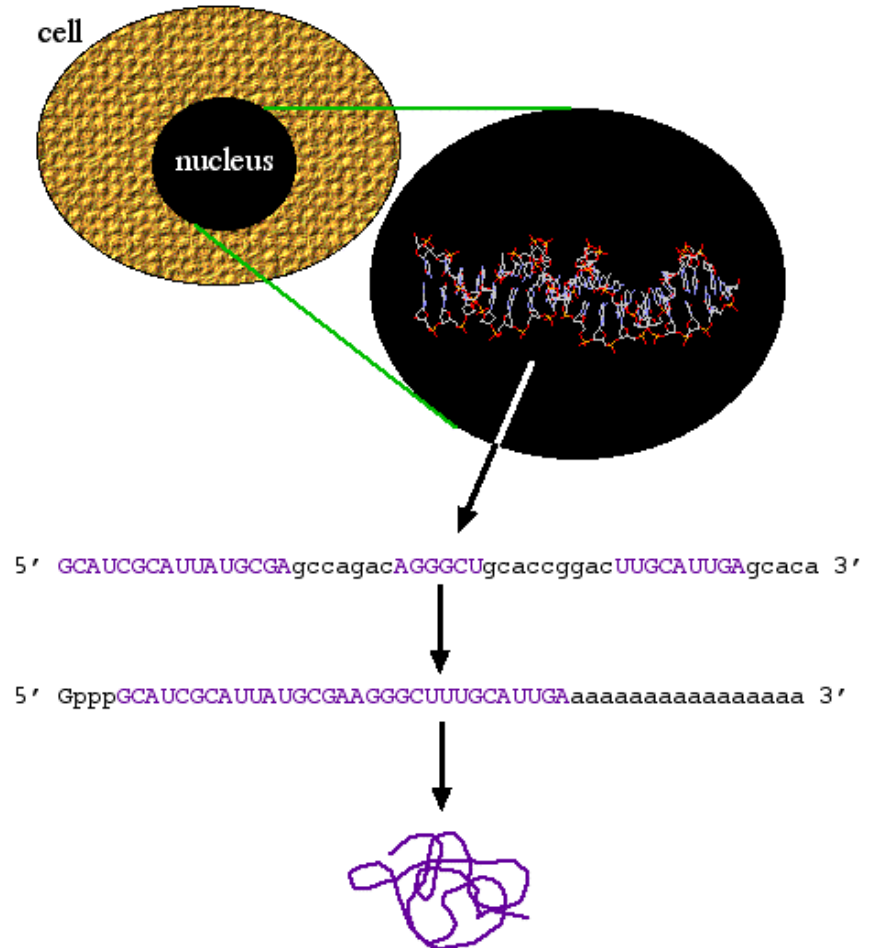
Whole Genome Shotgun Sequencing Method



- Zerbrechen von **kompletten Genomen** in Stücke 1KB-100KB
 - Alle Stücke (an-)sequenzieren
 - Celera:
 - Homo sap.: Genom: 3 GB, **28.000.000 Reads**
 - Drosophila: Genom: 120 MB, 3.200.000 Reads
- Schnelle Algorithmen notwendig

2. Stringvergleiche beim EST Clustering

- Wie kann man Gene finden?
- Eine Möglichkeit: mRNA einer Zelle untersuchen
 - Genexpression -> mRNA
 - mRNA extrahieren
 - mRNA in DNA rückübersetzen (cDNA)
 - cDNAs sequenzieren
 - Sequenz (Exons? Introns?) im Genom finden



cDNA/EST Bibliotheken

- cDNAs: **Gene / kodierende Regionen**
 - Viel interessanter als genomische DNA
 - cDNA enthält (meist) PolyA und 3' Ende
 - cDNA enthält (oft) nicht 5' Ende
 - Weil Reverse Transcriptase vorher abfällt
- Inhalt einer cDNA Bibliothek abhängig von
 - Gewebe
 - Entwicklungsstadium (Embryo - Erwachsenen)
 - **Organismusstatus (Krank – Gesund)**
- **EST** = "Expressed Sequence Tags"
 - "Ansequenzierte" cDNAs

Stringvergleiche beim EST Clustering

- Problem
 - 6.000.000 EST Sequenzen in EMBL (Stand 2004)
 - **Wie viele Gene?**
- Nicht 6.000.000, denn
 - Viele Laboren erzeugen EST Sequenzen
 - Gene werden in Kopien exprimiert – gleiche cDNA
 - Differentielles Splicing
 - „Ansequenzierung“ – Links? Rechts? Mitte?
 - Sehr gering/selten exprimierte Gene

EST Clustering

- Berechnung der Anzahl Gene durch Clustering
- Typisches Verfahren
 - Schritt 1: Berechnung der Überlappung / Ähnlichkeit aller Sequenzpaare
 - Schritt 2: Berechnung von Sequenzclustern durch Bildung der transitiven Hülle
 - Schritt 3: Zuordnung isolierter Sequenzen zu Clustern mit geringerer Ähnlichkeitsanforderung

Problemdimension: UniGene

- Clustern aller cDNA und EST in Genbank
 - Größenordnung
 - 6.000.000 Sequenzen
 - Länge 300-4.000 Basenpaare
 - Anzahl Sequenzvergleiche: $O(n^2)$
 - Schnelle Stringvergleichsalgorithmen notwendig
- Wöchentliche Aktualisierung
- Ergebnis 11/2005
 - 54.576 Cluster
 - Sequenzen pro Cluster: Von 2 (viele) bis 10.800 (wenige)

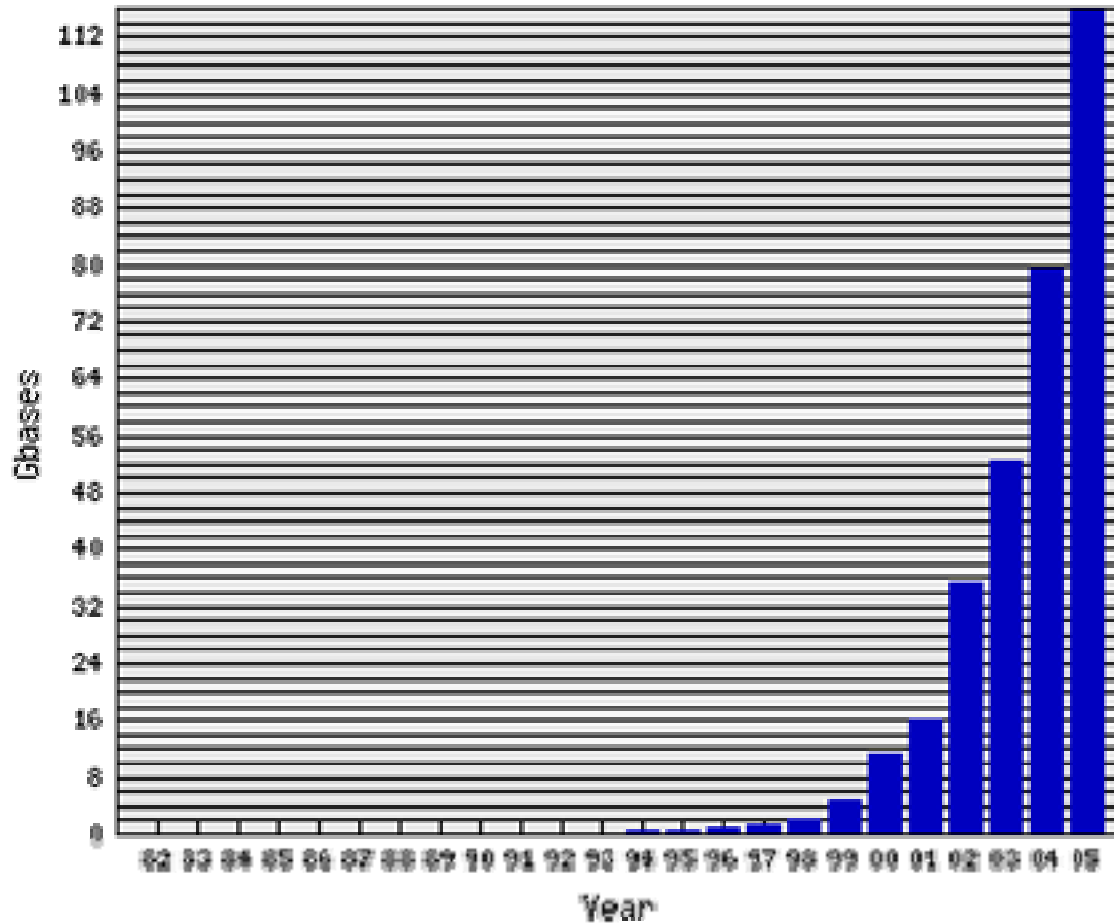
3. Funktionale Annotation

- Sequenzen bestimmen Funktionen
 - Gensequenzen => Proteinsequenz
 - Proteinsequenzen => Struktur
 - Struktur => Funktion
- Grundannahme der Bioinformatik
 - Gleiche Sequenzen – gleiche Funktion
 - Sehr ähnliche Sequenzen – sehr ähnliche Funktion
 - Etwas ähnliche Sequenzen – verwandte Funktion?
- Stimmt nicht immer
- Insbesondere zwischen verschiedenen Spezies

Standardvorgehen

- Gegeben: Eine neue DNA Sequenz
- **Annotationspipeline**
 - Suche nach ähnlichen Gensequenzen
 - Suche nach ähnlichen Promotersequenzen
 - Suche nach ähnlichen Proteinen (Übersetzung- Rückübersetzung)
 - Vorhersage neuer Genen durch Programme (trainiert auf bekannten Gensequenzen)
 - Suche nach ähnlichen Proteindomänen durch Programme (trainiert auf bekannten Proteindomänen)
 - ...
- **Alternative: Experimentelle Überprüfung**
 - Teuer, auch nicht fehlerfrei
 - Ethische / technische Machbarkeit

Problemdimension



Quelle: EMBL, Genome Monitoring Tables, Stand 9.11.2005

-
- Strings und Matching - Problemklassen

Problemklassen

- **Exaktes Matching**
 - Gegeben: Strings P, T
 - Gesucht: Alle Auftreten von P in T
 - Variante: Gegeben P_1, \dots, P_n, T : Vorkommen aller P_i in T ?
- **Approximatives Matchen**
 - Gegeben: Strings S, T
 - Gesucht: Wie ähnlich sind sich S und T ?
 - Variante: Ist P in T mit höchstens k Fehlern enthalten?
 - Variante: Vorkommen von P in T mit Wildcards?
- **Suche in Datenbanken**
 - Gegeben: Datenbank D von Sequenzen, String P
 - Gesucht: Die Top- k zu P ähnlichsten Sequenzen in D

Exaktes Matching

- Gegeben: P (Pattern) und T (Text)
 - Trivialerweise verlangen wir $|P| \leq |T|$
 - In der Regel nehmen wir an $|P| \ll |T|$
- Gesucht: Sämtliche Vorkommen von P in T
- Beispiel
 - Auffinden der Erkennungssequenzen von **Restriktionsenzymen**

Eco RV - GATATC

```
tcagcttactaattaaaaattctttctagtaagtgctaagatcaagaaaaataaattaaaaataatggaacatggcacattttcctaaactcttcacagattgctaatga
ttattaattaaagaataaatggttataattttttatggtaacggaatttcctaaaatattaattcaagccatggaatgcaataagaaggactctgttaattgggtact
atccaactcaatgcaagtggaactaagttgggtattaatactctttttacatatatatgtagttattttaggaagcgaaggacaatttcactctgctaataaagggattac
atatttatttttgtgaatataaaaaatagaaagtatggtatcagattaaacttttgagaaaggtaagatgaagtaaagctgtatactccagcaataagttcaaataggc
gaaaaactttttaataacaaaagttaaataatcattttgggaattgaaatgtcaaagataattacttcacgataagtagttgaagatagtttaaattttctttttgtatt
acttcaatgaaggtaacgcaacaagattagagtatatatggccaataagggttgctgtaggaaaattattctaaggagatacgcgagaggggcttctcaaatttattcaga
gatggatggttttagatgggtgggttaagaaaaagcagatttaaattccagcaaaaactagaccttaggtttattaagcgaggcaataagttaattggaattgtaaaagatat
ctaattcttcttcatttggtaggggaaaactagtttaacttcttaccocatgcagggccataggggtcgaatacgcctgtcactaagcaaaaggaaaaatgtgagtgtagact
ttaaaccatttttattaatgactttagagaatcatgcatttgatgttactttcttaacaatgtgaacatatttatgcgattaagatgagttatgaaaaaggcgaatata
tattcagttacatagagattatagctgggtctattcttagttataggacttttgacaagatagcttagaaaaataagattatagagcttaataaaagagaacttcttggaa
tagctgcctttgggtgcagctgtaattggctattgggtatggctccagcttactgggttaggttttaatagaaaaattccocatgattgctaattatatctatcctattgagaa
caacgtgcgaagatgagtggaatgggttcatatttaactgctgggtgctatagtagttatccttagaaagatatataatctgataaagcaaaaatcctggggaaaatat
tgctaactgggtgctggtaggggtttgggggattggattatctcctctacaagaaatttgggtggttactgatataataatagagaaaaaatttaataaagatgat
```

Zeichenketten

- Definition

*Ein **String** S ist eine von links nach rechts angeordnete Liste von Zeichen eines Alphabets Σ*

- $|S|$ ist die Länge des Strings
- Positionen in S sind $1, \dots, |S|$
 - Wir zählen ab 1
- $S(i)$ (oder $S[i]$) beschreibt das Zeichen an der Position i im String S
- $S[i..j]$ ist der Substring, welcher an Position i beginnt und an Position j endet
- $S[i..j]$ ist ein leerer String, falls $i > j$
- $S[1..i]$ heißt **Präfix** von S bis zur Position i
- $S[i..]$ ist das **Suffix** von S , welches an Position i beginnt
- **Echte Präfixe und echte Suffixe** umfassen nicht den gesamten String S und sind nicht leer

Für die nächsten Wochen

Exaktes Matching

Notation

- Wir suchen im Folgenden immer P in T
- $|T| = m \neq 0$
- $|P| = n \neq 0$
- $m \gg n$
- Alphabet Σ endlich
- P, T sind Strings über Σ
- Kosten für Vergleich zweier Zeichen aus Σ : 1

Übersicht

- Naiver Algorithmus
 - $O(n \cdot m)$
- Z Algorithmus
 - $O(m+n)$
- Boyer-Moore
 - Worst Case $O(n \cdot m)$, aber Average Case sublinear
 - Erweiterung zu linearem Worst-Case
- Knuth-Morris-Pratt
 - $O(m+n)$
- Suffixbäume (viele T) – nur Suche
 - $O(n+k)$ (nach Preprocessing: $O(m)$)

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. Solange Startposition $\leq |T| - |P|$, gehe zu 2

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

Naiver Ansatz (cont.)

```
for i = 1 to |T| - |P| + 1
  match := true;
  j := 1;
  while ((match) and (j <= |P|))
    if (T(i + j - 1) <> P(j)) then
      match := false;
    else
      j := j + 1;
  end while;
  if (match) then
    -> OUTPUT i
end for;
```

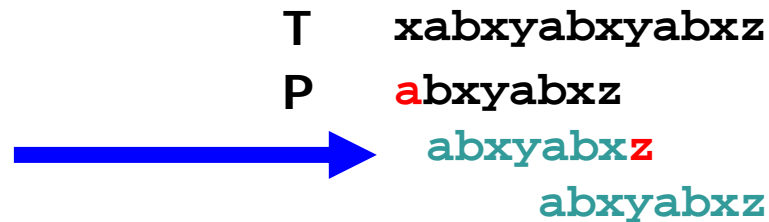
Worst-case

T	aaaaaaaaaaaaaaaa
P	aaaaat
	aaaaat
	aaaaat
	aaaaat
	...

Vergleiche : $n * (m-n+1) \Rightarrow O(m*n)$

Optimierungsidee 1

- Anzahl der Vergleiche reduzieren
 - P um mehr als ein Zeichen verschieben
 - Aber nie soweit, dass ein Vorkommen von P in T nicht erkannt wird
- Beobachtung: Zeichen



- Substring in T muss mit a beginnen
- Nächstes a in T erst an Position 6 – springe 4 Positionen
- Vorkommen von Buchstaben in T kann während des Vergleichs ab Position 2 gelernt werden

Optimierungsidee 2

2. Beobachtung: Substrings

T **x**abxyabxyab**xz**
P **a**bxyab**xz**

abxyabxz

- ABX ist doppelt in P - interne Struktur von P erkennen
 - $P[1..3] = P[5..7]$
 - Kein Vorkommen dazwischen
- Vergleich findet: $P[1..7] = T[2..8]$
- Daher
 - $P[1..3] = T[6..8]$; zwischen 2 und 6 kann in T kein Treffer liegen
 - 4 Zeichen schieben, erst ab Position 4 in P weiter vergleichen

Fazit

- Stringalgorithmen an vielen Stellen der Bioinformatik essentiell
- Sowohl exakte als auch approximative Suche notwendig
 - „Ähnlichkeit“ ist ein fundamentales Konzept der Bioinformatik
- Wegen großen Datenmengen ist hohe Performance wichtig
- Naiver Algorithmus für exaktes Matching braucht quadratische Laufzeit
 - Das ist zu langsam
 - Aber es scheint schneller zu gehen