Einführung in die Bioinformatik

Prof. Dr. Caroline Friedel

Lehr- und Forschungseinheit
Bioinformatik

Was ist Bioinformatik?

Theoretische und Praktische Informatik
Statistik, Mathematik
Molekularbiologie, Biochemie, Genetik,
Evolutionsbiologie, …

„Die Bioinformatik ist eine interdisziplinäre Wissenschaft, die Probleme aus den Lebenswissenschaften mit theoretischen computergestützten Methoden löst“

Wikipedia

Informatik → Methoden:

- Algorithmen
- Theoretische Informatik
- Datenbanken
- Softwareentwicklung

Biologie → Anwendungen:

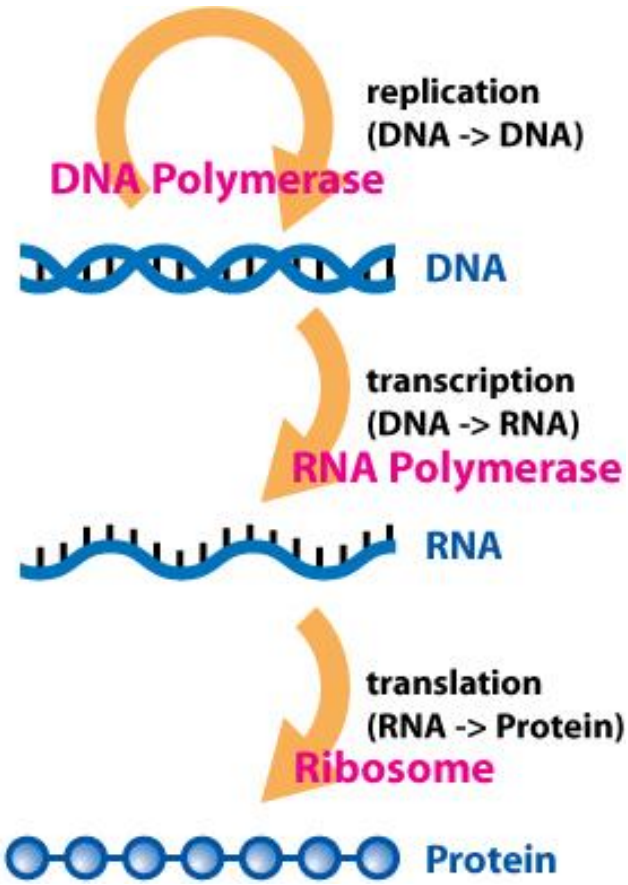
- Genomsequenzierung
- Genexpressionsanalyse
- Biologische Netzwerke
- …

Ein bisschen Biologie

- Zentrales Dogma der Biologie (Francis Crick, 1958)

Abstrakt: Sequenzen

Gene



…ACGTTCGATA…

4 Basen

Transkripte/mRNA

…ACGUUCGAUA…

20 Aminosäuren

…TFD…

Der Klassiker: Sequenzsuche

- Input:
  - Text t, z.B. Genomsequenz
  - Muster s, z.B. Poly-A Signal AATAAA
- Gesucht: Vorkommen von Muster im Text

- Naiver Ansatz:
  - Vergleiche s Buchstabe für Buchstabe mit t
  - Bei Ungleichheit wird s um eine Position nach rechts verschoben
  - Vergleich von s mit t beginnt von neuem
  - Wiederholung bis entweder s in t gefunden oder klar, dass s in t nicht enthalten

Beispiele

- t = a a a a a a a a a a

```
a a a b
  a a a b
    a a a b
      a a a b
```

(10-4+1)*4 Vergleiche

…

- t = a a b a a b a a b a a b a a b

```
a a b a a a
  a a b a a a
    a a b a a a
      a a b a a a
```

= 8*4= 32 Vergleiche

< (15-6+1)*6 Vergleiche

Obere Schranke für die Anzahl der Vergleiche

…

Worst case-Analyse

Pseudocode

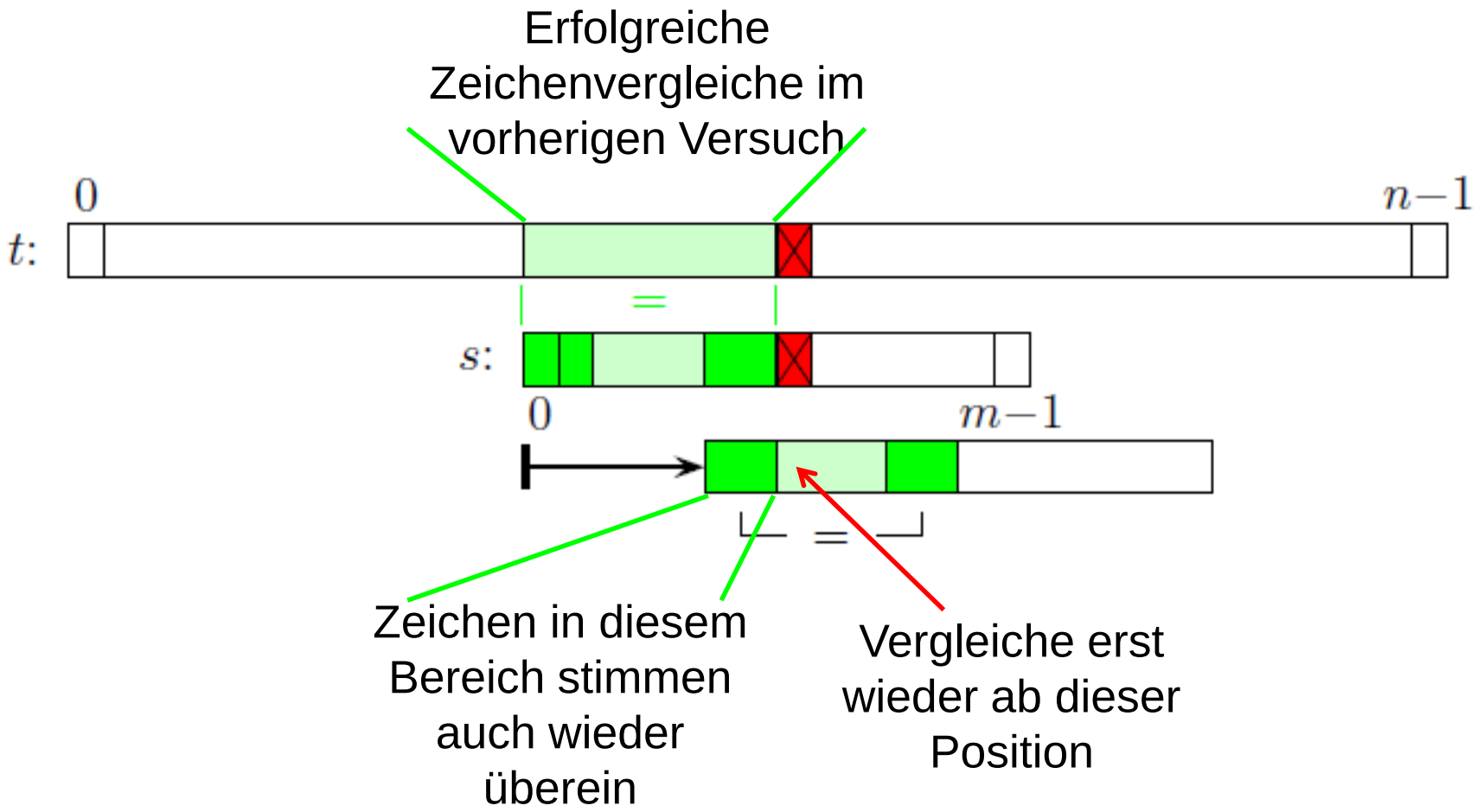
Länge von $t = n$
Länge von $s = m$

```
bool Naiv (char t[], int n, char s[], int m)
begin
    int i := 0, j := 0;
    while (i ≤ n − m) do
        while (t[i + j] = s[j]) do        Match
            j++;
            if (j = m) then return TRUE;
        i++;        Mismatch
        j := 0;
    return FALSE;
end
```

- Laufzeit:
  - Äußere Schleife *(n-m+1)* mal durchlaufen
  - Innere maximal m mal
  - Vergleich auf Zeichen *((n-m+1)m)*-mal
  - $\leq c \cdot n \cdot m$
  - $O(nm)$
- Für Genom und AATAAA:
- $\approx 3 * 10^9 * 6$ Vergleiche

Idee für Verbesserung

- Nutze erfolgreiche Vergleiche aus



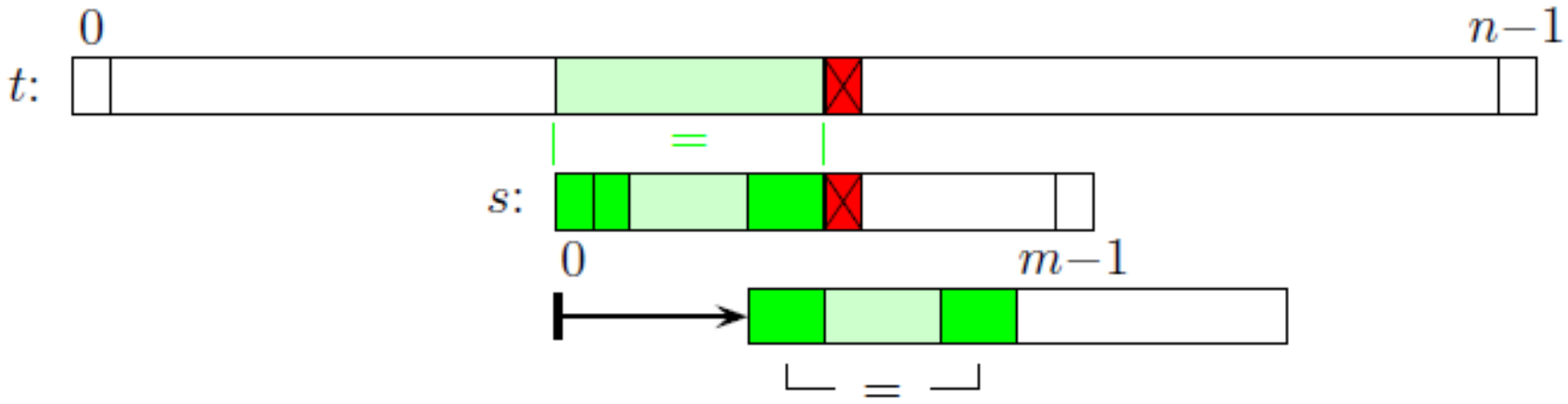
Beispiel

- t = a a b a a b a a b a a b a a b

  a a b a a a

  a a b a a a

Vergleiche erst wieder ab hier



Beispiel

- t = a a b a a b a a b a a b a a b

```
a a b a a a
        a a b a a a
            a a b a a a
                a a b a a a
```

- Anzahl Vergleiche?
- = 14 erfolgreiche Vergleiche
- + 4 nicht erfolgreiche
- =18
- <32

Präprozessierung

- Berechne nicht bei jedem Mismatch neu, wie weit man verschieben kann
- Vorberechnung einer “Border”-Tabelle
- border[i]= Länge des längsten Teil-Wortes, dass an Position $i-1$ endet und am Start von s auch vorkommt
- s= aabaaa=$s_0 \dots s_5$
- border[1] = 0
- border[2] = 1
- border[3] = 0
- border[4] = 1
- border[5] = 2
- border[6] = 2

$s_0 \dots s_{i-1}$

- a
- aa
- aab
- aaba
- aabaa
- aabaaa

Laufzeit

- Algorithmus von Knuth, Morris und Pratt:
- Maximal $2n + m$ Vergleiche, um alle Vorkommen von Muster s der Länge m in einem Text t der Länge n zu finden
- Für Genom und AATAAA:
- Im worst case: $2 * 3 * 10^9 + 6$ Vergleiche

- Für Muster der Länge 1000: $2 * 3 * 10^9 + 1000$ Vergleiche
- $\ll 1000 * 2 * 10^9$ Vergleiche für naiven Ansatz

- Auch machbar für sehr lange Muster

Suche nach mehreren Suchwörtern

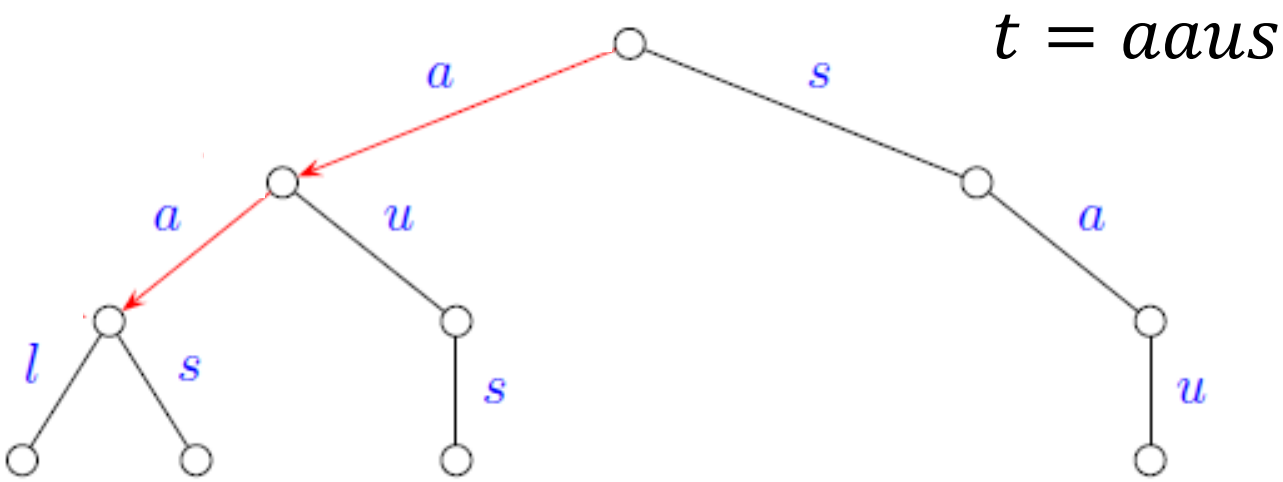
- Mehrfachtextsuche
  - Eingabe: Ein Text t der Länge n und eine Menge S von l Suchwörtern mit $m :=$ Gesamtlänge der Suchwörter
  - Gesucht: Taucht ein Suchwort $s \in S$ im Text t auf?

- Naiver Ansatz: KMP für jedes Suchwort einzeln
- $2ln + m$ Vergleiche

- Für menschliches Genom und 100 Suchwörter à 10 Zeichen:
- $2 * 100 * 3 * 10^9 + 1000$

Aho-Corasick-Algorithmus

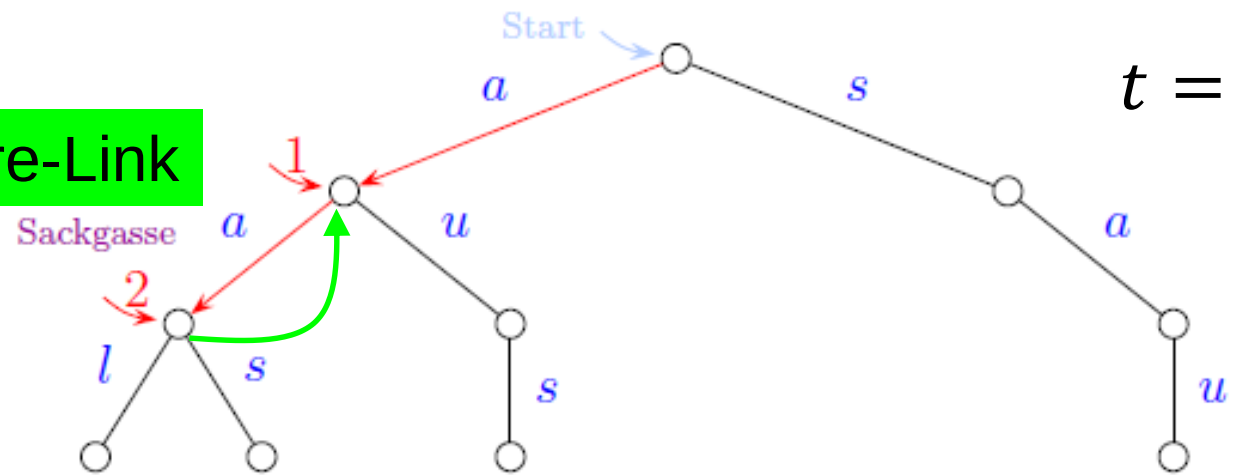
- Idee: Bilde einen Suchwort-Baum
- Suchwort-Baum für {*aal, aas, aus, sau*}



- Suche im Prinzip ablaufen des Textes *t* im Baum
- Wenn Blatt erreicht, Suchwort gefunden
- Problem: Sackgassen

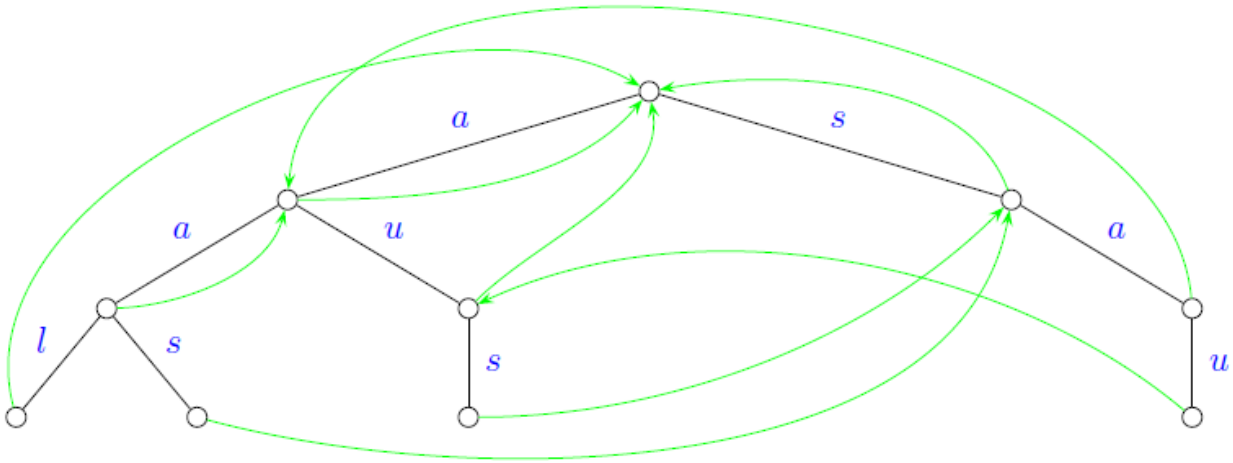
Lösung: Failure-Links

Failure-Link



$t = aaus$

- Suchwort-Baum mit allen Failure-Links



Laufzeit

- Konstruktion des Suchwort-Baumes mit Failure-Links: $2m = O(m)$
- Suche im Suchwort-Baum: $2n = O(n)$

- Insgesamt: $O(n + m)$

- Für menschliches Genom und 100 Suchwörter à 10 Zeichen:
- $2 * 1000 + 2 * 3 * 10^9$

- Aho-Corasick entwickelt 1975
- Implementiert z.B. in Unix *fgrep*

Suffix-Bäume

- Aho-Corasick: Präprozessierung der Suchwörter
- Alternative: Präprozessierung des Textes
- Sinnvoll, wenn oft der gleiche Text durchsucht wird
- Anwendung in der Bioinformatik: Sequenzierung

Sequenzierung

- Bestimmung der Basen-Sequenz z.B. eines Genoms
- Nur in kleinen Stücken möglich
- Reads
- Whole-genome shotgun
- Millionen von kurzen Reads
- Assembly oder Resequencing
- Resequencing = Vergleich gegen bekanntes Genom

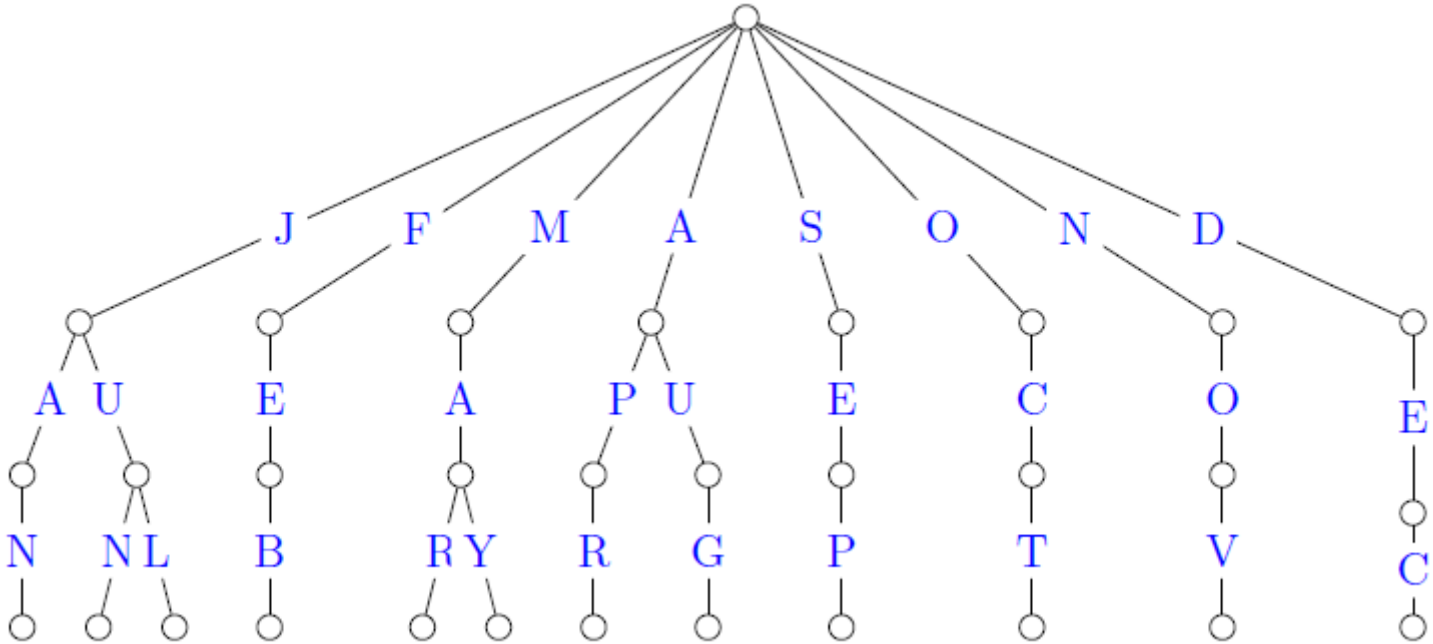
Sequenzierung

2.5-5 Milliarden Reads * 2 * 150 Basen pro Read
> 750 Milliarden Basen
> 250 * Länge des menschlichen Genoms

	NextSeq System		HiSeq Series	
				4000*
Output Range				500 Gb
Run Time				5 days
Reads per Run	130–400 million	300 million–4 billion	2.5 billion	2.5–5 billion
Max Read Length	2 × 150 bp	2 × 250 bp (Rapid run) 2 × 125 bp (High output)	2 × 150 bp	2 × 150 bp

Tries

- Trie für 3-Buchstaben Abkürzungen der Monatsnamen



- Vorteil von Tries: Suche nach Wort der Länge m in Trie geht mit m Vergleichen

Suffix-Tries

- Suffix von Text $t = \mathrm{t}_1 \ldots \mathrm{t}_n$:
  - Teilwort, dass an einer Position i in t startet und an Position n endet

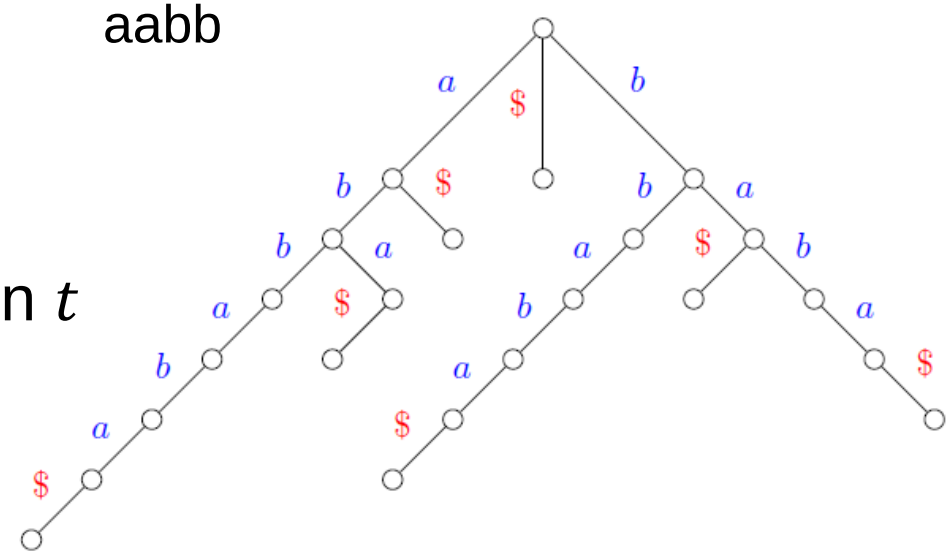
aabb

Suffixe:

ε
b
bb
abb
aabb

- Suffix-Trie für Text t:
  - Trie für alle Suffixe von t

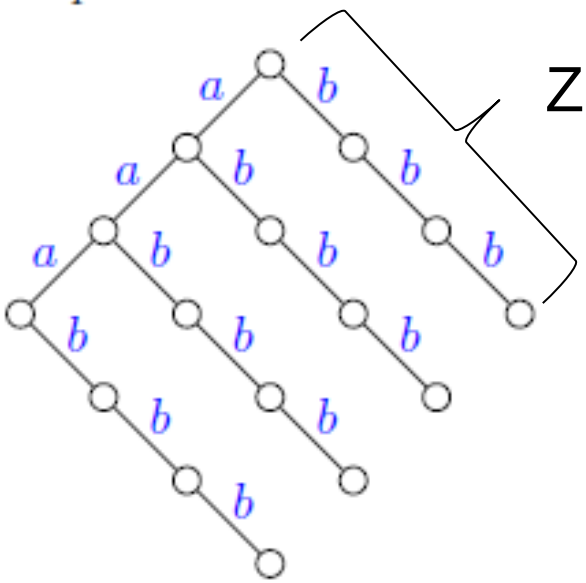
Beispiel: t = abbaba$



Suffix-Baum

- Kompaktifizierter Trie = Patricia-Trie
- **P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric (1968)

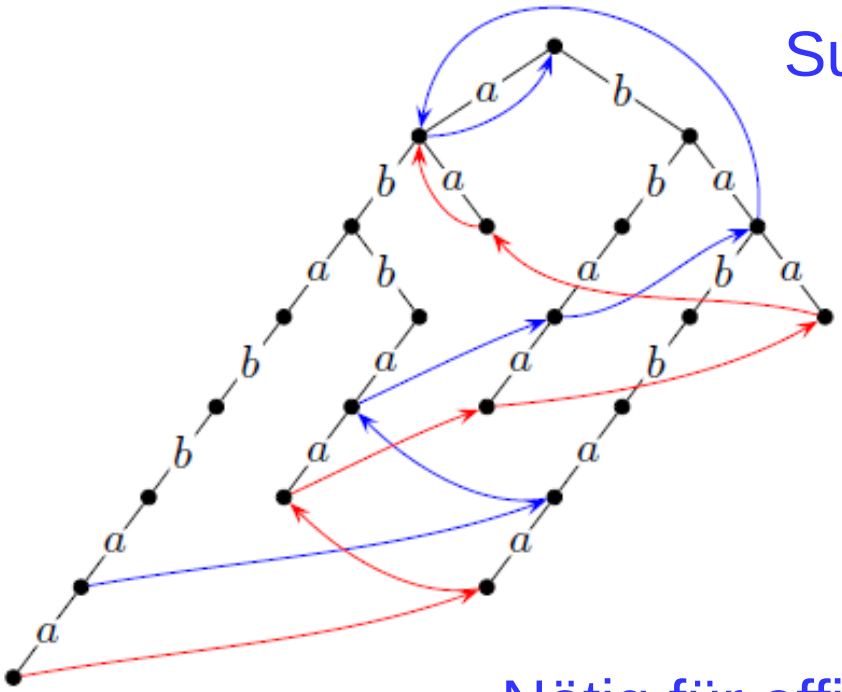
Beispiel: $t = aaabbb$



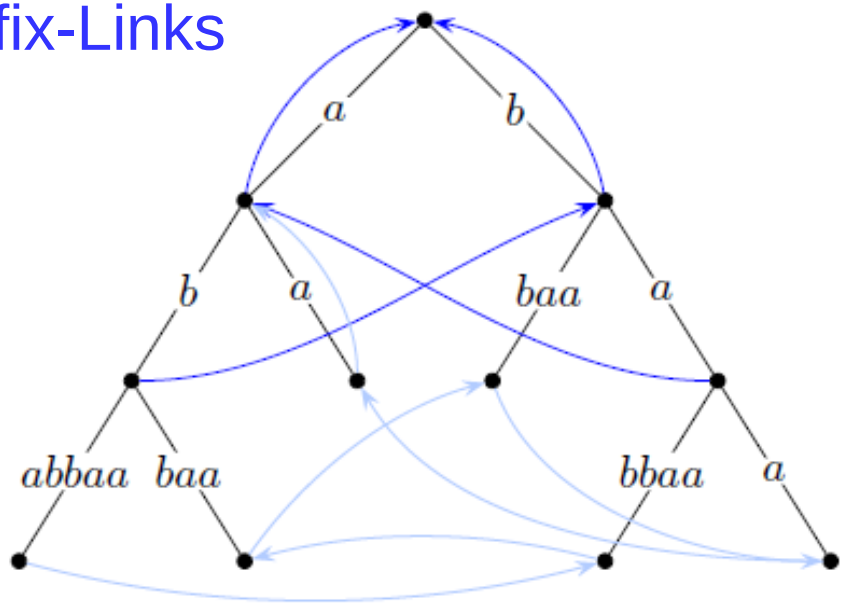
Zusammenfassen

- Worst-case Größe des Tries = $O(n * n)$

Suffix-Baum



Suffix-Links

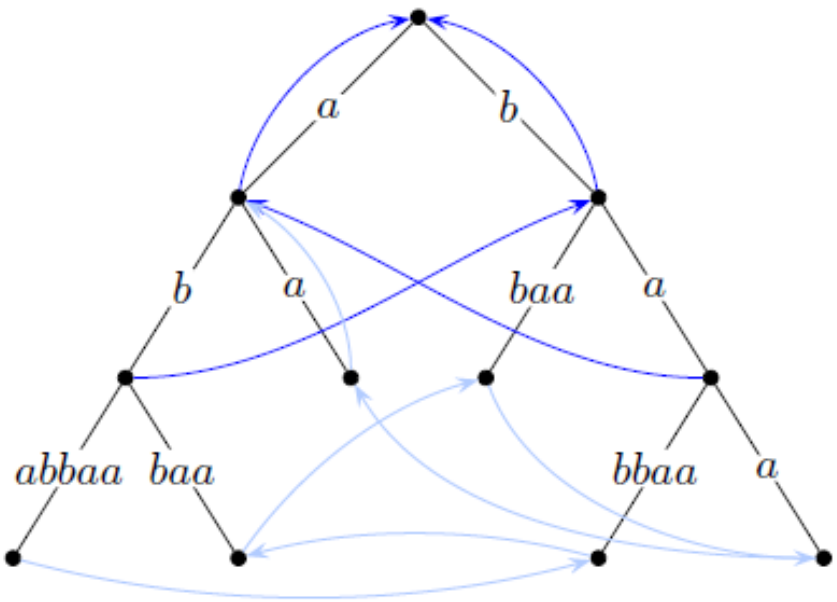


Nötig für effiziente Konstruktion des Baums, nicht Suche

- Vorverarbeitung des Textes: $O(n \cdot |\Sigma|)$ mit $|\Sigma|$ Größe des Alphabets
- Für DNA $\Sigma = \{ACGT\}$

Suche im Suffix-Baum

- $t = ababbaa$
- Gesucht $s = ab$
- Falls vorhanden ⇒ es gibt mindestens ein Suffix von t, das mit s beginnt
- ⇒ Es gibt Pfad startend an der Wurzel, der mit s beginnt
- ⇒ Wenn s gefunden, sammele Blätter in Teilbaum unter Treffer auf
- ⇒ $|s|$ Vergleiche + O(k) bei k Vorkommen von s



- Erlaubt auch Suche mit Fehlern

Verbesserungen

- Trotz linearen Platzbedarf, Suffix-Bäume in der Praxis meist zu speicherintensiv
- $\approx 6n\, log(n) + n \log(|\Sigma|)$ Bits
- Für menschl. Genom: ≈ 46 GB

- Alternative Suffix-Array
- Suffix-Array enthält für ein Wort t seine lexikographisch sortierten Suffixe repräsentiert durch ihre Anfangspositionen in t

Beispiel

- t = ababbaa
- S[1] = 7 ≙ a
- S[2] = 6 ≙ aa
- S[3] = 1 ≙ ababbaa
- S[4] = 3 ≙ abbaa
- S[5] = 5 ≙ baa
- S[6] = 2 ≙ babbaa
- S[7] = 4 ≙ bbaa

- Für effiziente Suche zusätzliche Datenstrukturen nötig
- $n\log(n) + n\log(|\Sigma|) + 4n + o(n)$ Bits
- <10 GB für menschl. Genom

Bestimmung von Sequenzähnlichkeiten

- Bisher exakte Suche, maximal einzelne Mismatches
- Biologische Sequenzen durch Evolution entstanden
- ⇒ Mutationen in der Genom-Sequenz
- Nicht nur “single nucleotide polymorphisms” (SNPs)
- Auch Deletionen und Insertionen

- Beispiel:
  - Sequenz 1: KKCGTTTWCGT
  - Sequenz 2: KKTTCGTCCTTWCCT
- Gesucht: Alignment
  - `KK--CGT--TTWCGT`
  - `||  |||  |||| |`
  - `KKTTCGTCCTTWCCT`

1 Mismatch, 4 Insertionen

Edit-Distanz: 5

Optimales Sequenzalignment

- Input: $s = s_1 \dots s_n$, $t = t_1 \dots t_m$
- Gesucht: Sequenz-Alignment mit minimaler Edit-Distanz
- Bestimme zuerst minimale Edit-Distanz von s und t

- Drei Möglichkeiten, wie Alignment enden kann

$\bar{s}$		x
$\bar{t}$		y

Match/Mismatch

$\bar{s}$		x
$\bar{t}$		$-$

Insertion in s

$\bar{s}$		$-$
$\bar{t}$		y

Deletion in s

Rekursive Definition

- $D(i,j)$ = Edit-Distanz für $s_1 \dots s_i$ und $t_1 \dots \mathrm{t_j}$

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

- $D(n,m)$ = Edit-Distanz von s und t

- Problem: Viele Werte unnötig oft berechnet
- z.B. $D(i-1,j-1)$ auch wieder benötigt bei Aufruf von $D(i-1,j)$ und $D(i,j-1)$

- Exponentielle Laufzeit

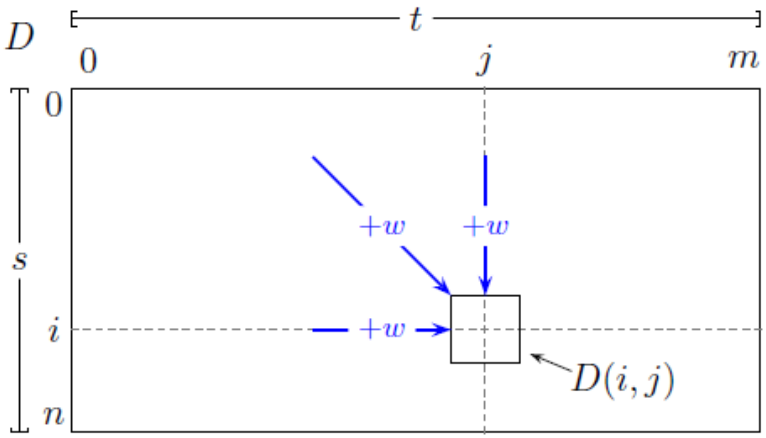
Dynamische Programmierung

- Einfacher Trick: Werte, die schon berechnet wurden, werden in Tabelle gespeichert
- Richard Bellman, 1940s

Laufzeit $O(nm)$

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

Werte bereits berechnet
Muss nur in Tabelle nachschauen



Datenbanksuche

- $O(nm)$ immer noch zu ineffizient für Suche in großen Protein- oder DNA-Datenbanken

- z.B. UniProt: >60 Millionen Protein-Sequenzen
- > 20 Milliarden Aminosäuren

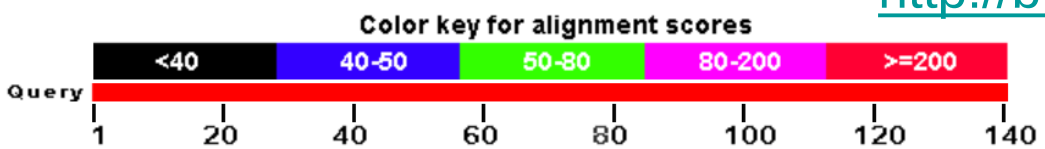
- Selbst wenn nur 1 ms für Alignment zu einem Protein
- $\Rightarrow 60 * 10^3 s > 16$ Stunden um alle ähnlichen Proteine für ein Such-Protein zu finden

- $\Rightarrow$ Schnellere approximative Ansätze
- BLAST, Fasta

BLAST= Basic Local Alignment Search Tool

CATTATCTGTTTTTGCTATTAATGTCATACCTGCTATTCAGACAAATCAATTTGCTCTCTTAATAAAGGATG
AGCTTCCTGTAGCTTTTTGTAGCTGGGCCAGTTTAGATCTGGAATGTGAGGTAAAATATATAAATGAT

http://blast.ncbi.nlm.nih.gov/Blast.cgi



Description	Max score	Total score	Query cover	E value	Ident
hemolysin D [Escherichia coli O103:H2 str. 2010C-4433]	98.6	98.6	98%	2e-24	100%
hypothetical protein CFSAN002237_12915 [Escherichia coli O104:H21 str. CFSAN002237]	98.2	98.2	98%	4e-24	100%
hemolysin-activating lysine-acyltransferase HlyC [Escherichia coli 96.0497]	98.2	98.2	98%	4e-24	100%
hypothetical protein [Escherichia coli]	97.8	97.8	98%	4e-24	100%
hemolysin D [Escherichia coli]					
hemolysin D [Escherichia coli]					
hypothetical protein [Escherichi					
hypothetical protein [Escherichi					
hemolysin C [Escherichia coli C					
hemolysin D [Escherichia coli]					
hypothetical protein [Escherichi					
hemolysin D [Escherichia coli]	97.8	97.8	98%	5e-24	100%

Download GenPept Graphics

hemolysin D, partial [Escherichia coli O103:H2 str. 2010C-4433]

Sequence ID: gb|EYZ14753.1| Length: 145 Number of Matches: 1

Range 1: 1 to 46 GenPept Graphics ▼ Next Match ▲ Previous Match

Score	Expect	Method	Identities	Positives	Gaps	Frame
98.6 bits(244)	2e-24	Composition-based stats.	46/46(100%)	46/46(100%)	0/46(0%)	+3

```
Query  3    LSVFAINVIPAIQTNQFALLIKDELPVAFCSWASLDLECEVKYIND  140
            LSVFAINVIPAIQTNQFALLIKDELPVAFCSWASLDLECEVKYIND
Sbjct  1    LSVFAINVIPAIQTNQFALLIKDELPVAFCSWASLDLECEVKYIND  46
```

J. Mol. Biol. (1990) 215, 403–410

Basic Local Alignment Search Tool

Stephen F. Altschul[1], Warren Gish[1], Webb Miller[2]
Eugene W. Myers[3] and David J. Lipman[1]

[1]*National Center for Biotechnology Information*
National Library of Medicine, National Institutes of Health
Bethesda, MD 20894, U.S.A.

[2]*Department of Computer Science*
The Pennsylvania State University, University Park, PA 16802, U.S.A.

[3]*Department of Computer Science*
University of Arizona, Tucson, AZ 85721, U.S.A.

(Received 26 February 1990; accepted 15 May 1990)

A new approach to rapid sequence comparison, basic local alignment search tool (BLAST), directly approximates alignments that optimize a measure of local similarity, the maximal segment pair (MSP) score. Recent mathematical results on the stochastic properties of MSP scores allow an analysis of the performance of this method as well as the statistical significance of alignments it generates. The basic algorithm is simple and robust; it can be implemented in a number of ways and applied in a variety of contexts including straightforward DNA and protein sequence database searches, motif searches, gene identification searches, and in the analysis of multiple regions of similarity in long DNA sequences. In addition to its flexibility and tractability to mathematical analysis, BLAST is an order of magnitude faster than existing sequence comparison tools of comparable sensitivity.

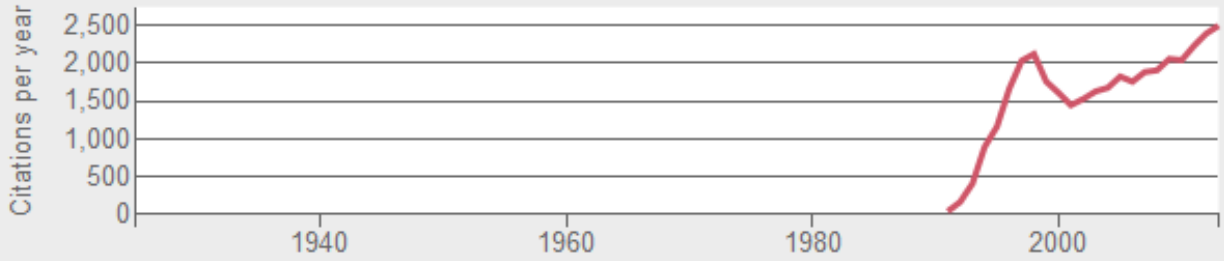
BLAST

Rank: **12** Citations: **38,380**

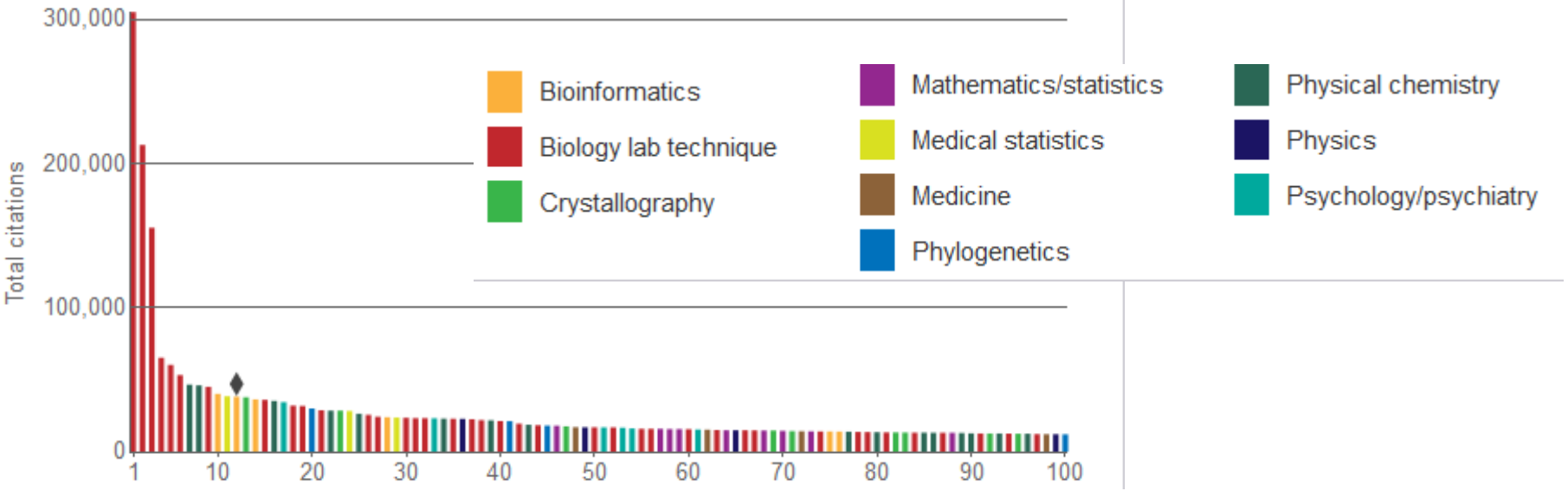
Basic local alignment search tool.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W. & Lipman, D. J.

J. Mol. Biol. **215**, 403–410 (1990).



- The most-cited research of all time
- Nature 514, 550–553 (2014)



Zusammenfassung

- Bioinformatik verwendet viele klassische Algorithmen und klassische Methoden des Algorithmen-Entwurfs
- z.B. Suche in Texten: Knuth-Morris-Pratt, Boyer-Moore, Aho-Corasick
- Dynamische Programmierung: Sequenz-Alignment
- Neue Herausforderungen durch Größe der Datensätze
  - Ähnlichkeitssuche in Sequenz-Datenbanken
  - Next-Generation-Sequencing

On the shoulders of giants

Codebreakers' Wall, Bletchely Park

Fragen?