

Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2009



Übersicht

- 1 Suffix Trees
 - Ukkonen-Algorithmus
- 2 Paarweises Sequenzen-Alignment

Ukkonen-Algorithmus: Laufzeit

- Laufzeit wieder gemessen als Anzahl besuchter (und neu erzeugter) Knoten (inklusive Blätter)
- Alle Knoten werden insbesondere in `Canonize` besucht, neue innere Knoten werden in `TestAndSplit` erzeugt (an die dann in `Update` ein neues Blatt gehängt wird).
- Es genügt also, die betrachteten Knoten in den Prozeduren `Canonize` und `TestAndSplit` zu zählen.

Ukkonen-Algorithmus: Laufzeit

- In **Canonize** wird zunächst der aufgerufene Knoten besucht.
- Da nach jedem Aufruf von Canonize sofort ein Aufruf von TestAndSplit folgt, ist diese Zahl gleich der Anzahl Aufrufe von TestAndSplit, die wir später ermitteln.
- Mit jedem in Canonize neu aufgesuchten Knoten in der while-Schleife wird der Parameter k erhöht.
- k wird nie erniedrigt, hat zu Beginn den Wert 2 und ist nach oben durch n beschränkt.
- Also können in der while-Schleife innerhalb verschiedener Aufrufe von Canonize maximal n Knoten des Suffix-Baumes besucht werden.

Ukkonen-Algorithmus: Laufzeit

- In **TestAndSplit** unterscheiden wir zwischen erfolgreichen Aufrufen (Rückgabe-Wert ist TRUE) und erfolglosen Aufrufen (Rückgabewert ist FALSE).
- Da nach jedem erfolgreichen Aufruf die Prozedur Update verlassen wird, kann es maximal n erfolgreiche Aufrufe von TestAndSplit geben.
- In jedem erfolglosen Aufruf von TestAndSplit wird der Suffix-Baum um mindestens einen Knoten (ein Blatt und eventuell ein innerer Knoten) erweitert.
- Wie wir gesehen haben hat der Suffix-Baum für ein Wort der Länge n maximal $O(n)$ Knoten.
- Also kann es maximal $O(n)$ erfolglose Aufrufe von TestAndSplit geben.

Ukkonen-Algorithmus: Laufzeit

- Damit kann es auch insgesamt nur maximal $O(n)$ Durchläufe der while-Schleife in allen Aufrufen von Canonize geben.
- Insgesamt werden also maximal $O(n)$ Knoten im Suffix-Baum besucht (bzw. neu erzeugt).

Theorem

Ein Suffix-Baum für $t \in \Sigma^n$ lässt sich in Zeit $O(n)$ und Platz $O(n)$ mit Hilfe des Algorithmus von Ukkonen konstruieren.

Verwaltung der Kinder

- Knoten können wenige Kinder besitzen oder viele (bis zu $|\Sigma|$).
⇒ verschiedene Methoden, Bewertung nach Platz- und Zeitbedarf
- Zeitbedarf:
Wie lange dauert es, bis wir für ein Zeichen $a \in \Sigma$ das Kind gefunden haben, das über die Kante erreichbar ist, dessen Kantenlabel mit a beginnt?
- Platzbedarf:
Verwendeter Speicher für den gesamten Suffix-Baum für einen Text der Länge n

Verwaltung der Kinder

Felder: Die Kinder eines Knotens lassen sich sehr einfach mit Hilfe eines Feldes der Größe $|\Sigma|$ darstellen.

- Platz: $O(n \cdot |\Sigma|)$.

Dies folgt daraus, dass für jeden Knoten ein Feld mit Platzbedarf $O(|\Sigma|)$ benötigt wird.

- Zeit: $O(1)$.

Übliche Realisierungen von Feldern erlauben einen Zugriff in konstanter Zeit.

Der Zugriff ist also sehr schnell, wo hingegen der Platzbedarf, insbesondere bei großen Alphabeten doch sehr groß werden kann.

Verwaltung der Kinder

Listen: Kinder eines Knotens verwaltet in einer linearen Liste (sortiert oder unsortiert)

- Platz: $O(n)$.

Für jeden Knoten ist der Platzbedarf proportional zur Anzahl seiner Kinder. Damit ist Platzbedarf insgesamt proportional zur Anzahl der Knoten des Suffix-Baumes, da jeder Knoten (mit Ausnahme der Wurzel) das Kind eines Knotens ist. Im Suffix-Baum gilt, dass jeder Knoten entweder kein oder mindestens zwei Kinder hat. Für solche Bäume ist bekannt, dass die Anzahl der inneren Knoten kleiner ist als die Anzahl der Blätter. Da ein Suffix-Baum für einen Text der Länge n maximal n Blätter besitzt, folgt daraus die Behauptung für den Platzbedarf.

Verwaltung der Kinder

- Listen:
- Zeit: $O(|\Sigma|)$.
Leider ist hier die Zugriffszeit auf ein Kind sehr groß, da im schlimmsten Fall (aber auch im Mittel) die gesamte Kinderliste eines Knotens durchlaufen werden muss und diese bis zu $|\Sigma|$ Elemente umfassen kann.

Verwaltung der Kinder

Balancierte Bäume : Die Kinder lassen sich auch mit Hilfe von balancierten Suchbäumen (AVL-, Rot-Schwarz-, B-Bäume, etc.) verwalten:

- Platz: $O(n)$
Da der Platzbedarf für einen Knoten ebenso wie bei linearen Listen proportional zur Anzahl der Kinder ist, folgt die Behauptung für den Platzbedarf unmittelbar.
- Zeit: $O(\log(|\Sigma|))$.
Da die Tiefe von balancierten Suchbäumen logarithmisch in der Anzahl der abzuspeichernden Schlüssel ist, folgt die Behauptung unmittelbar.

Verwaltung der Kinder

Hashfunktion Eine weitere Möglichkeit ist die Verwaltung der Kinder aller Knoten in einem einzigen großen Feld der Größe $O(n)$. Um nun für ein Knoten auf ein spezielles Kind zuzugreifen wird dann eine Hashfunktion verwendet:

$$h : V \times \Sigma \rightarrow \mathbb{N} : (v, a) \mapsto h(v, a)$$

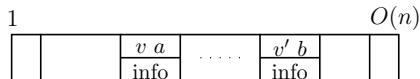
Zu jedem Knoten und dem Symbol, die das Kind identifizieren, wird ein Index des globales Feldes bestimmt, an der die gewünschte Information enthalten ist.

Verwaltung der Kinder

Hashfunktion Leider bilden Hashfunktionen ein relativ großes Universum von potentiellen Referenzen (hier Paare von Knoten und Symbolen aus Σ also $\Theta(n \cdot |\Sigma|)$) auf ein kleines Intervall ab (hier Indizes aus $[1 : \ell]$ mit $\ell = \Theta(n)$). Daher sind **Kollisionen** prinzipiell nicht auszuschließen. Ein Beispiel ist das so genannte **Geburtstagsparadoxon**. Ordnet man jeder Person in einem Raum eine Zahl aus dem Intervall $[1 : 366]$ zu (nämlich ihren Geburtstag), dann ist ab 23 Personen die Wahrscheinlichkeit größer als 50%, dass zwei Personen denselben Wert erhalten. Also muss man beim Hashing mit diesen Kollisionen leben und diese geeignet auflösen.

Verwaltung der Kinder

Hashfunktion Um solche Kollisionen überhaupt festzustellen, enthält jeder Feldeintrag i neben den normalen Informationen noch die Informationen, wessen Kind er ist und über welches Symbol er von seinem Vater erreichbar ist. Somit lassen sich Kollisionen leicht feststellen und die üblichen Operationen zur Kollisionsauflösung anwenden.



Skizze: Realisierung mittels eines Feldeintrags und Hashing

Verwaltung der Kinder

Hashfunktion

- Platz: $O(n)$

Das folgt unmittelbar aus der obigen Diskussion.

Zeit: $O(1)$

Im Wesentlichen erfolgt der Zugriff in konstanter Zeit, wenn man voraussetzt, dass sich die Hashfunktion einfach (d.h. in konstanter Zeit) berechnen lässt und dass sich Kollisionen effizient auflösen lassen.

Verwaltung der Kinder

Hybrid Kombination

- Knoten auf niedrigem Level, d.h. nah bei der Wurzel, haben meist einen sehr großen Verzweigungsgrad.
- Dort sind also fast alle potentiellen Kinder auch wirklich vorhanden.
- Knoten auf recht großem Level haben hingegen relativ wenige Kinder.
- Hybride Implementierung:
Für Knoten auf niedrigem Level verwendet man für die Verwaltung der Kinder einfache Felder, während man bei Knoten auf großem Level auf eine der anderen Varianten umsteigt.

Übersicht

- 1 Suffix Trees
- 2 Paarweises Sequenzen-Alignment
 - Distanz- und Ähnlichkeitsmaße

Distanz und Ähnlichkeit von Sequenzen

Paarweises Sequenzenalignment

- Ähnlichkeit von 2 Sequenzen bzw.
- Wie kann die eine Sequenz aus der anderen hervorgegangen sein?

M o n **k** e y

M o n - e y

Insertion

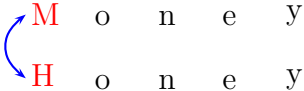
Deletion

The diagram shows two sequences: 'Mon**k**ey' and 'Mon-ey'. A blue arrow labeled 'Insertion' points from the 'k' in the second sequence to the 'k' in the first sequence. A blue arrow labeled 'Deletion' points from the '-' in the first sequence to the '-' in the second sequence.

Beispiel: Veränderung durch Insertion bzw. Deletion

Distanz und Ähnlichkeit von Sequenzen

- Neben Einfügen und Löschen ist das Ersetzen von Zeichen eine weitere Möglichkeit.

Substitution  M o n e y
H o n e y

The diagram illustrates a substitution operation. On the left, the word "Substitution" is written in blue. To its right, a blue curved arrow points from the letter 'M' in the top row to the letter 'H' in the bottom row. The top row contains the letters 'M', 'o', 'n', 'e', 'y' and the bottom row contains 'H', 'o', 'n', 'e', 'y'. The letters 'o', 'n', 'e', and 'y' are aligned vertically between the two rows.

Beispiel: Veränderung durch Substitution

Edit-Distanz

Definition

Sei Σ ein Alphabet und sei $-$ ein neues Zeichen, d.h. $- \notin \Sigma$.
Dann bezeichne $\bar{\Sigma} := \Sigma \cup \{-\}$ das um $-$ erweiterte Alphabet und
 $\bar{\Sigma}_0^2 := \bar{\Sigma} \times \bar{\Sigma} \setminus \{(-, -)\}$.

Das Zeichen $-$ werden wir auch als *Leerzeichen* bezeichnen.

Edit-Operationen

Definition

Eine **Edit-Operation** ist ein Paar $(x, y) \in \overline{\Sigma}_0^2$ und (x, y) heißt

- *Match*, wenn $x = y \in \Sigma$;
- *Substitution*, wenn $x \neq y$ mit $x, y \in \Sigma$;
- *Insertion*, wenn $x = -$, $y \in \Sigma$;
- *Deletion*, wenn $x \in \Sigma$, $y = -$.

Als *Indel-Operation* bezeichnet man eine Edit-Operation, die entweder eine Insertion oder Deletion ist.

Bemerkung: Eine neutrale (NoOp)-Operation $(x, x) \in \Sigma \times \Sigma$ ist hier als Edit-Operation zugelassen. Manchmal wird dies auch nicht erlaubt.

Edit-Operationen

Definition

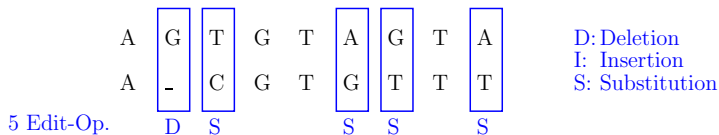
Ist (x, y) eine Edit-Operation und sind $a, b \in \Sigma^*$, dann gilt $a \xrightarrow{(x,y)} b$ (d.h. a kann mit Hilfe der Edit-Operation (x, y) in b umgeformt werden), wenn

- $x, y \in \Sigma \wedge \exists i \in [1 : |a|] : (a_i = x) \wedge (b = a_1 \cdots a_{i-1} \cdot y \cdot a_{i+1} \cdots a_{|a|})$ (Substitution oder Match)
- $x \in \Sigma \wedge y = - \wedge \exists i \in [1 : |a|] : (a_i = x) \wedge (b = a_1 \cdots a_{i-1} \cdot a_{i+1} \cdots a_{|a|})$ (Deletion)
- $x = - \wedge y \in \Sigma \wedge \exists i \in [1 : |a| + 1] : (b = a_1 \cdots a_{i-1} \cdot y \cdot a_i \cdots a_{|a|})$ (Insertion)

Sei $s = ((x_1, y_1), \dots, (x_m, y_m))$ eine Folge von Edit-Operationen mit $a_{i-1} \xrightarrow{(x_i, y_i)} a_i$, wobei $a_i \in \Sigma^*$ für $i \in [0 : m]$ und $a := a_0$ und $b := a_m$, dann schreibt man auch kurz $a \xrightarrow{s} b$.

Sequenz-Transformationen

$AGTGTAGTA \xrightarrow{s} ACGTGTTT$ mit $s = ((G, -), (T, C), (A, G), (G, T), (A, T))$
 oder mit $s = ((G, T), (A, G), (G, -), (A, T), (T, C))$



Beispiel: Transformation mit Edit-Operationen

Anmerkung: Es kommt nicht unbedingt auf die Reihenfolge der Edit-Operationen an.

Sequenz-Transformationen

$AGTGTAGTA \xrightarrow{s'} ACGTGTTT$ mit $s' = ((-, C), (A, -), (G, -), (A, T))$

A	-	G	T	G	T	A	G	T	A	
A	C	G	T	G	T	-	-	T	T	
	I					D	D		S	

4 Edit-Op. D: Deletion
I: Insertion
S: Substitution

Beispiel: Transformation mit anderen Edit-Operationen

Dieselben Sequenzen können auch mit Hilfe anderer Edit-Operationen ineinander transformiert werden:

Kosten

- Kosten einer Transformationsfolge setzen sich zusammen aus den Kosten der einzelnen Edit-Operationen.
- Man verwendet eine **Kostenfunktion** $w : \bar{\Sigma} \times \bar{\Sigma} \rightarrow \mathbb{R}_+$
- Bsp.: alle Kosten bis auf die Matches auf 1 setzen.
- Für Matches sind Kosten größer als Null in der Regel nicht sinnvoll.
- In der Biologie (wo die Sequenzen Basen oder insbesondere Aminosäuren repräsentieren) wird man jedoch intelligentere Kostenfunktionen wählen.

Kostenfunktion, Edit-Distanz

Definition

Sei $w : \overline{\Sigma}_0^2 \rightarrow \mathbb{R}_+$ eine Kostenfunktion. Seien $a, b \in \Sigma^*$ und sei $s = (s_1, \dots, s_\ell)$ eine Folge von Edit-Operationen mit $a \xrightarrow{s} b$. Dann sind die **Kosten der Edit-Operationen s** definiert als

$$w(s) := \sum_{j=1}^{\ell} w(s_j).$$

Die **Edit-Distanz** von $a, b \in \Sigma^*$ ist definiert als

$$d_w(a, b) := \min_s \left\{ w(s) : a \xrightarrow{s} b \right\}.$$

Dreiecksungleichung

- Folgende Beziehung soll gelten:

$$\forall x, y, z \in \Sigma : w(x, y) + w(y, z) \geq w(x, z)$$

- Betrachte zum Beispiel eine Mutation (x, z) , die als direkte Mutation relativ selten (also teuer) ist, sich jedoch sehr leicht (d.h. billig) durch zwei Mutationen (x, y) und (y, z) ersetzen lässt.
- Dann sollten die Kosten für diese Mutation durch die beiden billigen beschrieben werden, da man in der Regel nicht feststellen kann, ob eine beobachtete Mutation direkt oder über einen Umweg erzielt worden ist.
- Diese Bedingung ist beispielsweise erfüllt, wenn w eine Metrik ist.

Metrik

Definition

Sei M eine beliebige Menge.

Eine Funktion $w : M \times M \rightarrow \mathbb{R}_+$ heißt **Metrik** auf M , wenn die folgenden Bedingungen erfüllt sind:

$$(M1) \quad \forall x, y \in M : w(x, y) = 0 \Leftrightarrow x = y \quad (\text{Definitheit})$$

$$(M2) \quad \forall x, y \in M : w(x, y) = w(y, x) \quad (\text{Symmetrie})$$

$$(M3) \quad \forall x, y, z \in M : w(x, z) \leq w(x, y) + w(y, z) \quad (\text{Dreiecksungleichung})$$

Dann heißt (M, w) auch **metrischer Raum**.

Metrik-Bedingungen im biologischen Kontext

- M1: Zeichen sollten nur mit sich selbst identisch sein und somit auch nur mit sich selbst den Abstand 0 haben.
- M2 ist nicht ganz so klar, da Mutationen in die eine Richtung durchaus wahrscheinlicher sein können als in die umgekehrte Richtung.
- Da wir allerdings immer nur die Sequenzen sehen und oft nicht wissen, welche Sequenz aus welcher Sequenz durch Mutation entstanden ist, ist die Annahme der Symmetrie bei Kostenfunktionen sinnvoll.
- Des Weiteren kommt es bei Vergleichen von Sequenzen oft vor, dass beide von einer dritten Sequenz abstammen und somit aus dieser durch Mutationen entstanden sind. Damit ist die Richtung von Mutationen in den beobachteten Sequenzen völlig unklar und die Annahme der Symmetrie der einzig gangbare Ausweg.

Metrik für Sequenzen

Lemma

Ist $w : \bar{\Sigma} \times \bar{\Sigma} \rightarrow \mathbb{R}_+$ eine Metrik, dann ist auch $d_w : \bar{\Sigma}^* \times \bar{\Sigma}^* \rightarrow \mathbb{R}_+$ eine Metrik.

Metrik für Sequenzen

Beweis.

Seien $a, b \in \Sigma^*$. Dann gilt:

$$\begin{aligned} 0 &= d_w(a, b) \\ &= \min \left\{ w(s) : a \xrightarrow{s} b \right\} \\ &= \min \left\{ \sum_{i=1}^r w(s_i) : a \xrightarrow{s} b \text{ mit } s = (s_1, \dots, s_r) \right\}. \end{aligned}$$

Da w immer nichtnegativ ist, muss $w(s_i) = 0$ für alle $i \in [1 : r]$ sein (bzw. $r = 0$). Somit sind alle ausgeführten Edit-Operationen Matches, d.h. $s_i = (x_i, x_i)$ für ein $x_i \in \Sigma$. Damit gilt $a = b$ und somit M1 für d_w .

Metrik für Sequenzen

Beweis.

Seien $a, b \in \Sigma^*$. Dann gilt:

$$\begin{aligned}
 d_w(a, b) &= \min \left\{ w(s) : a \xrightarrow{s} b \text{ mit } s = (s_1, \dots, s_r) \right\} \\
 &= \min \left\{ \sum_{i=1}^r w(s_i) : a = a_0 \xrightarrow{s_1} a_1 \cdots a_{r-1} \xrightarrow{s_r} a_r = b \right\} \\
 &= \min \left\{ \sum_{i=1}^r w(s_i) : b = a_r \xrightarrow{\tilde{s}_r} a_{r-1} \cdots a_1 \xrightarrow{\tilde{s}_1} a_0 = a \right\} \\
 &= \min \left\{ w(\tilde{s}^R) : b \xrightarrow{\tilde{s}^R} a \text{ mit } \tilde{s}^R = (\tilde{s}_1, \dots, \tilde{s}_r) \right\} \\
 &= d_w(b, a).
 \end{aligned}$$

Hierbei bezeichnet \tilde{s}_i für eine Edit-Operation $s_i = (x, y)$ mit $x, y \in \Sigma$ die inverse Edit-Operation $\tilde{s}_i = (y, x)$. Also gilt M2.

Metrik für Sequenzen

Beweis.

- Seien $a, b, c \in \Sigma^*$.
- Seien s und t zwei Folgen von Edit-Operationen, so dass $a \xrightarrow{s} b$ und $w(s) = d_w(a, b)$ sowie $b \xrightarrow{t} c$ und $w(t) = d_w(b, c)$.
- Dann ist die Konkatenation $s \cdot t$ eine Folge von Edit-Operationen mit $a \xrightarrow{s \cdot t} c$.
- Dann gilt weiter

$$d_w(a, c) \leq w(s \cdot t) = w(s) + w(t) = d_w(a, b) + d_w(b, c).$$

- Also gilt auch M3.

