

SS04 Seminar "Aktuelle Themen der Bioinformatik"
Leitung: Dirk Metzler, Lin Himmelmann

Thema:

"Linear time algorithms for finding and representing all tandem repeats in a string"
Dan Gusfield & Jens Stoye, Computer Science Department University of California, Davis
CSE - 98 - 4

Autor: Frank Abromeit (abromeit@informatik.uni-frankfurt.de)

Gliederung

| | |
|--|----|
| I. Einleitung | |
| 1.0 Einleitung | 2 |
| II. Definitionen | 2 |
| 2.0 Tandem Repeats, Tandem Array | 2 |
| 2.1 Run, Überdeckung , Überdeckende Menge von Tandem Repeats | 4 |
| 2.2 Vorwärts- und Rückwärtserweiterung | 5 |
| 2.3 Datenstruktur Suffix-Baum | 5 |
| 2.3.1 Knotenbeschriftung | 6 |
| 2.3.2 Suffix-Link | 7 |
| III. Der Algorithmus | 7 |
| 3.0 Das Ziel des Algorithmus | 7 |
| 3.1 Phase 1 - Finden der linksüberdeckenden Menge | 8 |
| 3.2 Lemma 1, Lemma 2 | 9 |
| 3.2.1 Das Leftmost-Theorem | 9 |
| 3.3 Algorithmus 1a | 11 |
| 3.4 Algorithmus 1b | 13 |
| 3.5 Phase 2 - Markieren einer Teilmenge des Vokabulars | 14 |
| 3.5.1 Bottom-up-parse | 14 |
| 3.6 Phase 3 | 17 |
| IV. Anwendungen | 18 |
| 4.0 Anwendungen | 18 |
| V. Anhang | 19 |
| 5.0 Literaturverzeichnis | 19 |

I. Einleitung

1.0 Einleitung

Tandem-Repeats (Abk. TR) sind sich wiederholende Muster von Nucleotid-Sequenzen im Genom von Organismen deren Länge von einigen bis zu 100000 Basenpaaren (Abk. bp) reichen kann. Die Untersuchung von TR ist interessant, da ihr Vorkommen, d.h. Art und Häufigkeit charakteristisch für Art und Individuum sind. Daraus ergeben sich Anwendungen wie z.B. genetische Karten (Dib et. Al (1996)) sowie Authentifizierungsverfahren wie z.B. das DNA-Fingerprintingverfahren (Jeffreys et al. 1985), das u.a. zur Verbrechsaufklärung eingesetzt wird. Interessant ist auch der Zusammenhang mit Krankheiten (Trinucleotid-Expansion-Disorders, z.B. Thao T. Tran, et. Al.), wo man eine Veränderung der Häufigkeiten einzelner TR-Typen beobachten kann. Es existieren eine Reihe von Algorithmen für das Erkennen aller TRs in einem String (Main & Lorenz 1984, Landau, Schmidt 1993, Stoye Gusfield 1998), mit jeweils Laufzeit $O(n \log n + z)$ ¹. Der vorgestellte Algorithmus von Gusfield & Stoye (Abk. G&S) findet das Vokabular, d.h. die verschiedenen Typen von TRs in einem gegebenen String S in $O(n)$. Aus dem gefundenen Vokabular lassen sich ohne Verschlechterung der asymptotischen Laufzeit alle Vorkommen von TRs und die Anzahl der TRs, die an einer bestimmten Position in S beginnen feststellen (s.a. Kosaraju 1994). Außerdem ist der Algorithmus von G&S auf die Behandlung von Tandem-Arrays erweiterbar. In der Praxis werden allerdings häufiger Alignment-Algorithmen eingesetzt, da die Eingabesequenzen Sequenzierungsfehler aufweisen, und Alignment-Algorithmen diese Fehler überlesen. Auch sind Repeatsequenzen von Natur aus, nicht perfekt, d.h. in einer langen Repeatsequenz werden vereinzelt Mutationen auftreten. Hier sind Alignment-Algorithmen String-Algorithmen wie dem von G&S überlegen.

II. Definitionen

2.0 Tandem Repeats, Tandem Array

Def-1: Tandem Repeat, Tandem Array

Sei Σ ein Alphabet,

Worte $w = \alpha^n$, ($w \in \Sigma^*$) heißen Tandem Repeats. Ist $w = (\alpha)^n$ (und $n > 1$) so heißt w Tandem Array.

¹ Wobei z die Länge der Ausgabe bezeichnet.

Def-2.: String, Teilstring über Σ^*

- Ein String S ist ein Wort über Σ^* .
- Gegeben S , ein Teilstring $S[i..j] \in S$ beginnt an der Stelle i und endet an der Stelle j in S .

(Beispiel: $S = aabbabc$, $S[3..7] = bbabc$)

Def-3 : Menge aller Tandem Repeats in S

Gegeben S , so ist die Menge aller TRs in S ist durch

$M_S := \{(i,l) \mid (i,l) \text{ ist TR in } S\}$ definiert.

wobei in (i,l) i die Anfangsposition, und l die Länge des TRs bezeichnen.

Beispiel: Sei $\Sigma = \{a,b\}$ und $S = aabaaba$.

M ergibt sich zu

$M_S = \{(1,2), (1,6), (2,6), (4,2)\}$.

Zwei TRs heißen gleich, wenn sie denselben String bezeichnen. Natürlich sind die TRs $(1,2)$ und $(4,2)$ im obigen Beispiel gleich.

Def-4.: Vokabular von S

Das Vokabular der TRs in S wird durch die Menge

$V_S := \{(i,l) \mid (i,l) \text{ ist TR in } S, \text{ und } (i,l) \text{ ist ungleich jedem anderen TR in } S\}$ beschrieben.

Ein TR in V_S wird als Typ bezeichnet. Im obigen Beispiel gilt: $V_S = \{(1,2), (1,6), (2,6)\}$.

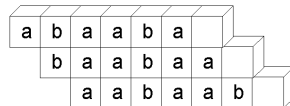
2.1 Run, Überdeckung, überdeckende Menge von Tandem Repeats

Def-5: Run von Tandem Repeats

Ein Intervall $[i..j]$ heißt Run von TRs der Länge l , falls: $(i,l), (i+1,l), \dots, (j,l)$ alle TRs sind.

Beispiel: Run der Länge 6

- Sei $S = \text{abaabaabbaaabaaba}$



Notation $[1..3]^6$

Def-6: Überdeckung von Tandem Repeats

Gegeben S , sowie $(i,l), (j,l) \in M_S$ so gilt die Bezeichnung:

Der TR (i,l) überdeckt den TR (j,l) gdw. $i < j, (i,l), (j,l) \in [h, \dots, m]_l$

,d.h. es existiert ein l -Run in S , in dem (i,l) und (j,l) beide enthalten sind.

Def-7: Überdeckende Menge (*covering set*)

Eine Teilmenge P aller TRs von S heißt überdeckende Menge (*covering set*) gdw. Es gibt keinen TR v im Vokabular von S , der nicht durch ein $p \in P$ überdeckt wird.

Beispiel: Überdeckende Menge für den String ababccababa

- a. $(7,4) = \text{abab}$ überdeckt $(8,4) = \text{baba}$
- b. $(6,2) = \text{cc}$

Durch die Menge $P = \{(7,4), (6,2)\}$ wird das Vokabular von $S = \{\text{cc}, \text{abab}, \text{baba}\}$ im obigen Sinn überdeckt, d.h. P ist überdeckende Menge für S .

Def-8: Linksüberdeckende Menge (*leftmost covering set*)

Eine Teilmenge P aller TRs von S heißt linksüberdeckende Menge (*leftmost covering set*) gdw. P ist überdeckende Menge, und alle TRs in P sind linkeste Vorkommen des jeweiligen Typs.

Beispiel: Linksüberdeckende Menge für $S = ababccababa$

Da der TR (7,4) zwar den TR (8,4) überdeckt, aber nicht das linkeste Vorkommen von abab in S ist lautet die linksüberdeckende Menge (leftmost-covering-set) für das obige Beispiel $\{(1,4),(6,2),(8,4)\}$.

2.2 Vorwärts- und Rückwärtserweiterung

Def-9: Vorwärtserweiterung, Rückwärtserweiterung

Sei wieder S ein String über Σ^* : Die längste gemeinsame Vorwärts- (Rückwärts-) erweiterung zweier Indizes x,y in S ist die Länge des längsten Teilstrings, der in S jeweils an den Positionen x und y beginnt (endet).

Beispiel:

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | a | b | b | b | a | b | a | b | b | b | a | b | a | b | b |
| | | 3 | | | | 7 | | 9 | | | | | | | | |

Für die Indizes 3 und 9 ist die längste Vorwärtserweiterung $|abbbababb| = 9$. Die längste Rückwärtserweiterung ist $|aba| = 3$. Für die Indizes 3 und 7 ist die längste Rückwärtserweiterung $|ab| = 2$, für 2 und 7 $|\epsilon| = 0$.

2.3 Datenstruktur Suffix-Baum

Def-10: Suffixbaum

Sei $S \in \Sigma^*$, der Suffixbaum von S ist der Baum $T(S)$ für den gilt:

1. Jede Kante in $T(S)$ ist mit einem String aus Σ^* beschriftet.
2. Keine zwei ausgehenden Kanten eines Knotens v sind gleich beschriftet.
3. Man kann in $T(S)$ jeden Suffix von S ablesen, wenn man einen Weg von der Wurzel zu einem Blatt abgeht.
4. $T(S)$ ist minimal bezüglich seiner Knotenanzahl.

2.3.1 Knotenbeschriftung

Def-11: Knotenbeschriftung

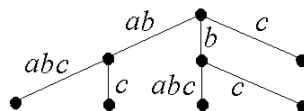
Für einen Knoten v im Suffixbaum $T(S)$ ist $L(v)$ (Knotenbeschriftung) der String, den man erhält, wenn man alle Kantenbeschriftungen von der Wurzel bis zu v konkateniert. $D(v)$ ist dann die Länge von $L(v)$, d.h. $D(v) = |L(v)|$.

Beispiel: Suffixbaum für $S = ababc$

S hat Suffixe :

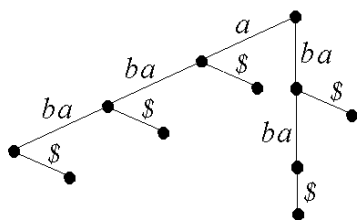
ababc
babc
abc
bc
c

Suffixbaum :



Wenn im obigen Beispiel aba ein Suffix von S wäre, hätte man folgendes Problem: Da aba ein Präfix von S ist gibt es keinen Weg in T der von der Wurzel zu einem Blatt führt, und aba beschreibt. (Durch Einfügen zusätzlicher Kanten verstößt man in jedem Fall gegen Bedingung 2 (oben). Damit dieser Fall nicht eintritt fügt man generell an des Ende von S ein Endzeichen (Stopper-Symbol) $\$$ an. D.h. aus $ababac$ wird $ababac\$$.

Suffixbaum für ababa:

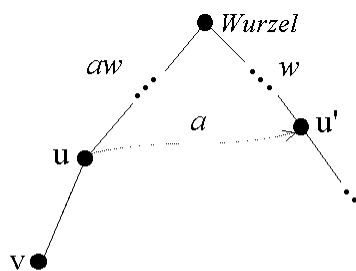


2.3.2 Suffix-Link

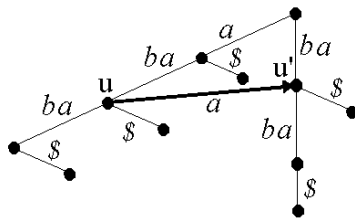
Def-11: Suffixlink

Gegeben $T(S)$,

Ein Suffixlink in $T(S)$ ist eine Querkante von einem Knoten u mit der Beschriftung $Bs(u) = aw$ ($a \in \Sigma, w \in \Sigma^*$) zu einem Knoten u' mit der Beschriftung $Bs(u') = w$. Die Querkante wird mit a beschriftet. Trivialerweise existiert für jeden Blattknoten (mit Ausnahme des kürzesten Suffix) ein Suffixlink in T .



Beispiel: Suffixlink



Ein Suffixlink ist beispielsweise gegeben durch (u, u') , da

$$Bs(u) = aba \quad Bs(u') = ba$$

Es ist die Kante (u, u') in $T(S)$ einzutragen. Die Berechnung des Suffixbaums (inklusive aller Suffixlinks) ist effizient in Zeit $O(n)$ möglich (siehe z.B. Gusfield 97).

III. Der Algorithmus

3.0 Das Ziel des Algorithmus

Das Ziel des Algorithmus ist es das Vokabular aller TRs in S zu bestimmen. Dazu wird in Phase I zunächst eine linksüberdeckende Menge (vergl. Def-7) für alle TRs in S bestimmt. In zwei verbleibenden Phasen wird zuerst eine Teilmenge der in Phase I gefundenen linksüberdeckenden Menge im Suffix-Baum $T(S)$ markiert. Schließlich wird in Phase 3 mit Hilfe von Suffix-Links und eines Suffix-Walks das gesamte Vokabular in T markiert.

3.1 Phase 1 - Finden der linksüberdeckenden Menge

In Phase I wird eine linksüberdeckende Menge (vergl. Def-7) gesucht. Die Grundlage der Analyse ist hierbei die Lempel-Ziv-Dekomposition (Lempel, Ziv 1976), die einen gegebenen String S in sog. Blöcke aufteilt. Die Lempel-Ziv-Dekomposition besteht im Prinzip aus der Bestimmung der maximalen Vorwärtserweiterung zweier indizes in S (vergl. Def. 9) und daran anschließend einer rekursiven Berechnung der Blöcke. Die LZ-Dekomposition läßt sich in O(n) berechnen (vergl. Rodeh, Pratt, Even 1981). Ein durch die Dekomposition von S erhaltener Block B mit Startindex i markiert den String s, der in S an der Position j (j<i) nochmals beginnt und maximale Länge hat (*Blockeigenschaft*).

Beispiel: LZ-Dekomposition von S = abaabaabbaaabaaba\$

Lempel-Ziv-Dekomposition I

L_i = Länge des maximalen Präfix von S[i..N]
Für den es ein j<i gibt mit S[j..n] = S[i..m]

| | | | | | | | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| S | a | b | a | a | b | a | a | b | b | a | a | a | b | a | a | b | a | S |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| L_i | 0 | 0 | 1 | 5 | 4 | 3 | 2 | 1 | 3 | 2 | 6 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Beispiel : Für i = 8 ist b maximal, denn der Teilstring bb kommt links der Position 8 nicht mehr vor.
Da die Länge von b = 1 \rightarrow $L_8 = 1$

Im ersten Schritt wird die Länge der maximalen Vorwärtserweiterung von L(i) bestimmt. Die maximale Vorwärtserweiterung (i,j) für festes i und (j<i) für alle Positionen i läßt sich in O(n) berechnen (LZ76). Die max. Vorwärtserweiterung (L(i)) kann danach für jede Position i in S in konstanter Zeit abgerufen werden.

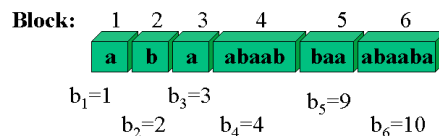
Im 2. Schritt findet die rekursive Berechnung der Blöcke statt:

Lempel-Ziv-Dekomposition II

| | | | | | | | | | | | | | | | | | | |
|-------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|
| S | a | b | a | a | b | a | a | b | b | a | a | a | b | a | a | b | a | S |
| i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| L_i | 0 | 0 | 1 | 5 | 4 | 3 | 2 | 1 | 3 | 2 | 6 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

S wird zerlegt: $b_1 := 1, b_{i+1} = b_i + \max(1, L_{b_i})$

(b_i ist die Anfangsposition eines Blocks in S)



Blöcke sind TRs verwandt, denn wenn s der String ist den ein Block u beschreibt, dann besagt die Blockeigenschaft, daß entweder $s = \gamma u$ ($\gamma \in \Sigma^*$) oder $s = s' u$ (s' ein Präfix von s, so daß $\exists i : s'[1..i] = s$) links des Blockanfangs von u in S erscheint.

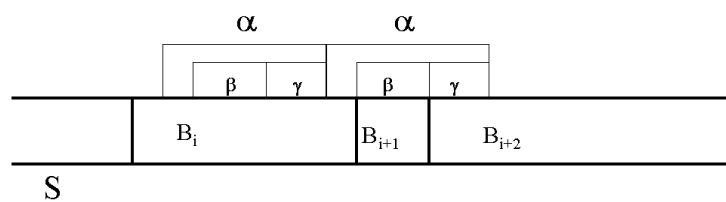
3.2 Lemma 1 & Lemma 2

Zwei Lemmata (siehe auch Crochemore 83 ,Crochemore 86 & Crochemore 94) stellen eine direkte Verbindung zwischen Blöcken und TRs her.

Lemma 1:

"Die rechte Hälfte eines TR $\alpha\alpha$ erstreckt sich maximal über 2 Blöcke der LZ-Dekomposition."

Man betrachte ein Gegenbeispiel:



Wie man in der Abbildung erkennt ist der Teilstring $\beta\gamma$ zweimal in S enthalten. Der mittlere Block B_{i+1} der genau β umfasst ist jedoch nicht maximal nach der Def. eines Lempel-Ziv Blocks, da $\beta\gamma$ im Block B_i vorkommt.

Lemma 2:

"Das linkeste Vorkommen jedes TR erstreckt sich über mindestens 2 Blöcke."

Hiermit ist gemeint, daß ein linkerster (leftmost) TR nicht komplett in einem Block liegen kann. Durch die Blockeigenschaft wird dies ausgeschlossen, denn wenn $\alpha\alpha$ nur in einem Block liegt, dann kommt $\alpha\alpha$ nach Def. nochmals links des aktuellen Blocks in S vor, und kann also nicht das linkeste Vorkommen von $\alpha\alpha$ sein. Aus den beiden Lemmata folgt das nun folgende Theorem.

3.2.1 Das Leftmost-Theorem

Leftmost-Theorem:

Es habe das linkeste Vorkommen des TR $\alpha\alpha$ seinen Mittelpunkt ² im Block B , dann gilt entweder:

LM1 $\alpha\alpha$ hat sein linkes Ende im Block B und sein rechtes Ende im Block $B+1$.

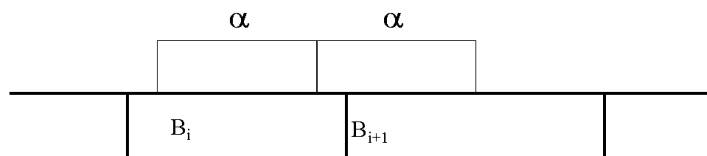
oder:

LM2 Die linke Hälfte von $\alpha\alpha$ berührt den Block $B-1$, und möglicherweise auch Blöcke weiter links. (Das rechte Ende kann innerhalb von Block B oder im Block $B+1$ liegen.)

² Der Mittelpunkt eines TR ist durch das letzte Zeichen der linken Hälfte von $\alpha\alpha$ bestimmt.

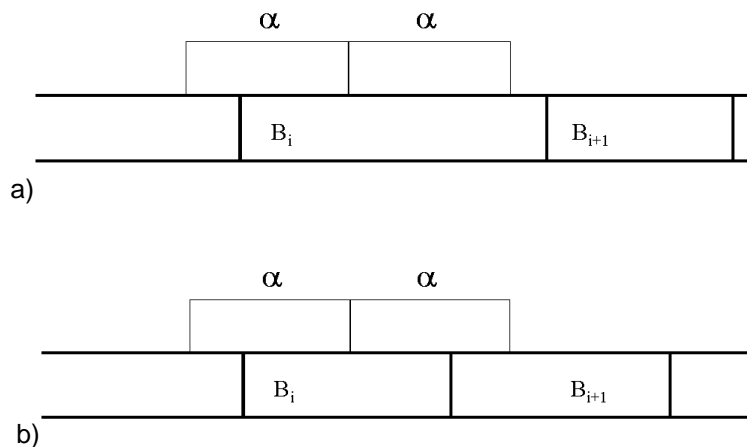
Man kann die gefundenen Regeln für leftmost TR graphisch darstellen. In jedem der 2 Fälle berührt die rechte Hälfte eines TR (wegen Lemma1) entweder ein oder zwei Blöcke. Wieviele Blöcke die linke Hälfte eines TR berührt ist durch das Lemma jedoch nicht bestimmt. Ein leftmost TR berührt wegen Lemma 2 mindestens 2 Blöcke.

LM1



Für LM1 ist die linke Hälfte des TR (nach Voraussetzung) in einem einzigen Block enthalten. Die rechte Hälfte muß dann wegen Lemma 2 automatisch 2 Blöcke berühren. Für LM2 können 2 Fälle eintreten:

LM2



Für LM2 ist die linke Hälfte des TR über 2 oder mehr Blöcke verteilt. Daraus resultiert dann die Möglichkeit, daß die rechte Hälfte einen Block, wie in a) gezeigt, oder auch zwei Blöcke, wie in b) gezeigt berühren kann.

3.3 Algorithmus 1a

Die Suche nach der linksüberdeckenden Menge, also nach linkesten TR beschränkt sich auf die durch das leftmost-Theorem gefundenen TR Typen. Der Algorithmus 1a findet ausschließlich den TR-Typ LM1, und der Algorithmus 1b die TR aus LM2. Zusammen finden sie alle leftmost TRs in S.

Algorithmus 1a:

Für k von 1 bis $|B|$

Let $q = h_1 - k$

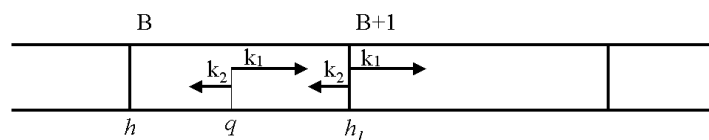
Berechne k_1 als die längste Vorwärtserweiterung für die Positionen h_1 und q .

Berechne k_2 als die längste Rückwärtserweiterung für die Positionen $h_1 - 1$ und $q - 1$

Falls $k_1 + k_2 \geq k$ und $k_1 > 0$:

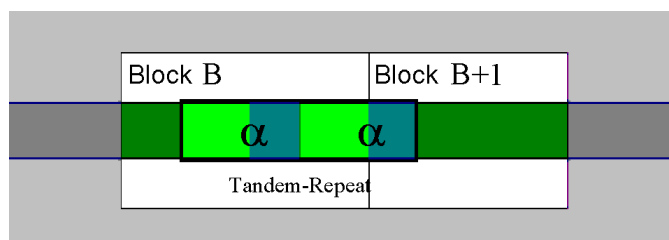
Gib den TR $(\text{maximum}(q - k_2, q - k + 1), 2k)$ aus.

Im Bild (unten) ist die Funktionsweise des Algorithmus illustriert.



In der Zeichnung bezeichnen die Variablen h, q und h_1 Positionen in S, d.h. $S[h]$ ist das h -te, $S[q]$ das q -te und $S[h_1]$ ist das h_1 -te Zeichen in S. Mit den Pfeilen werden die längste Vorwärtserweiterung (Rückwärtserweiterung) der Positionen q und h_1 dargestellt, und k_1 und k_2 bezeichnen die Länge dieser Erweiterungen. In der Schleife des Algorithmus wird q beginnend bei der Position $h_1 - 1$ nach links bis zur Position h bewegt, d.h. die Variable q nimmt nacheinander die Werte $h_1 - 1, h_1 - 2, h_1 - 3, \dots, h + 1, h$ an. In jedem Schleifendurchgang werden für q und h_1 die längste Vorwärtserweiterung (Rückwärtserweiterung) berechnet. Aus k_1 und k_2 kann bestimmt werden, ob ein TR vom Typ LM1 vorliegt. Wie schon in der Abbildung (oben) gezeigt wurde hat dieser TR-Typ die Eigenschaft, daß seine rechte Hälfte im Block B beginnt und im Block B+1 aufhört. Die linke Hälfte ist vollständig in B enthalten. Als erste Bedingung für den TR muß dann gelten, daß $k_1 > 0$ ist, denn nur dann endet seine rechte Hälfte im Block B+1. Andererseits kann k_1 aber auch nicht größer als $|B+1|$ werden, da sonst der Block B+1 nicht maximal wäre. Die zweite Bedingung ($k_1 + k_2 \geq k$) besagt, daß k_1 und k_2 eine bestimmte Länge haben müssen, damit ein TR vorliegt. Betrachtet man den Spezialfall, daß $k_1 + k_2 = k$ ($\Leftrightarrow k_1 + k_2 = h_1 - q$), so gewinnt man eine Vorstellung von der Lage eines TR in S.

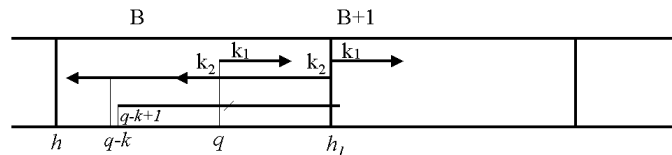
Abbildung: ($k_1 + k_2 = k$)



Bei diesem Spezialfall beginnt der TR genau an der Position $q-k_2$, und hat die Länge $2k$. Wenn jedoch $k_1 + k_2 < k$ ($\Leftrightarrow k_1 + k_2 < h_1 - q$) liegt kein TR vor, denn die linke und die rechte Hälfte, die durch die Teilstrings $S[q-k_2 \dots q+k_1]$ und $S[h_1-k_2 \dots h_1+k_1]$ gebildet werden stoßen nicht aneinander. Wenn $k_1 + k_2 > k$ gilt gibt es zwei feinere Unterscheidungen, nämlich 1. ($k_1 > k$) und 2. ($k_2 > k$).

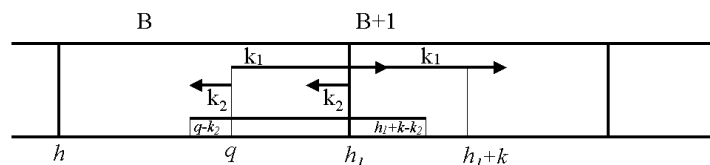
2.Fall :

In der Zeichnung (unten) überlagert der k_2 Pfeil, der bei h_1 beginnt, den k_2 Pfeil, der an der Position q beginnt. Es läßt sich also ein TR lokalisieren, dessen rechte Hälfte durch den Teilstring $S[q..h_1]$, und dessen linke Hälfte durch den Teilstring $S[q-k..q-1]$ gebildet wird. Jedoch entspricht dieser TR nicht dem Typ LM1, denn er endet nicht in $B+1$. Es ist jedoch kein Problem den linken Anfangspunkt des TR, um eine Position nach rechts zu verschieben (da $k_1 > 0$, nach Voraussetzung), so daß der TR nun bei $q-k+1$ beginnt und Typ LM1 ist. Weiter sind alle Teilstrings $S[q-k+1+i..q+k+1+i]$ ($1 \leq i \leq k_1$) LM1 TRs, die von $(q-k+1, 2k)$ überdeckt werden. Hierin hat das maximum in der Zeile $\max((q-k_2, q-k+1), 2k)$ seine Bewandnis. Für $k_2 > k$ wird $q-k+1$ als Anfangspunkt des leftmost TR ausgegeben.



1.Fall:

Wie in Fall 2 überlagert der k_1 Pfeil, der bei q beginnt, den k_1 Pfeil beginnend an Position h_1 . Auch hier läßt sich unmittelbar ein TR ausmachen, der bei der Position q beginnt, bei Position h_1+k endet und die Länge $2k$ hat und der vom Typ LM1 ist. Jedoch ist auch $S[q-k_2 \dots h_1+k-k_2]$ ein TR vom Typ LM1 (da $k_1 > 0$) der Länge $2k$, und dieser TR überdeckt den TR, der an der Position q beginnt, denn $S[q-k_2+i..h_1+k-k_2+i]$ ($1 \leq i \leq k_2$) sind alle TRs. Schließlich muß in diesem Fall also der TR $(q-k_2, 2k)$ als leftmost TR ausgegeben werden. Das linke Ende wird auch vom Algorithmus zu $\max(q-k_2, q-k+1) = q-k_2$ berechnet, da nur $k_2 < k$ ist.



3.4 Algorithmus 1b

Algorithmus 1b:

Für k von 1 bis $|B| + |B+1|$

Let $q = h + k$

Berechne k_1 als die längste Vorwärtserweiterung für die Positionen h und q

Berechne k_2 als die längste Rückwärtserweiterung für die Positionen $h-1$ und $q-1$.

Falls $k_1 + k_2 \geq k$ und $k_1 > 0$, $k_2 > 0$ und $\max(h-k_2, h-k+1) + k < h_1$:

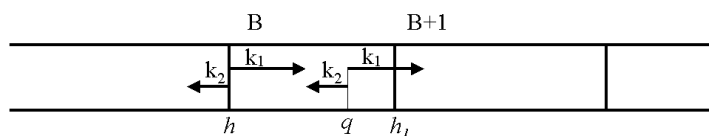
Gib den TR $(\max(h-k_2, h-k+1), 2k)$ aus.

Der Zeiger q wandert hier von der Position h ausgehend nach rechts bis zur Position $h + |B| + |B+1|$, und es wird wieder für jede Position die längste Vorwärts- bzw. Rückwärtserweiterung der Positionen h und q berechnet. Algorithmus 1b geht ganz ähnlich für TRs vom Typ LM2 vor, indem das Erkennen eines TR durch die Bedingung $k_1 + k_2 \geq k$ erfolgt. Die Bedingung für einen TR vom Typ LM2 ist (vergl. 3.2.1), daß seine linke Hälfte in irgendeinem Block $< B$ anfängt, und seine rechte Hälfte im Block B oder im Block $B+1$ endet. Durch die Bedingung $k_2 > 0$ wird sichergestellt, daß der TR im Block $B-1$ anfängt. Die Zeile $\max(h-k_2, h-k+1)$ bestimmt ähnlich wie in 1a den Anfang des TR. Durch die Bedingung $\max(h-k_2, h-k+1) + k < h_1$ wird erzwungen, daß seine rechte Hälfte wie in LM2 gefordert im Block B beginnt.

Fallunterscheidung:

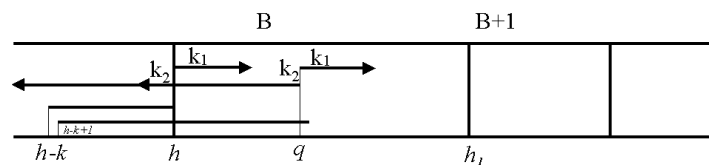
Fall 1: $(k_1 + k_2 = k)$

Der Anfang des TR ist $h-k_2$, wie man aus dem Bild (unten) ablesen kann. Das wird auch durch den Algorithmus berechnet, da $\max(h-k_2, h-k+1) = h-k_2$, denn $k > k_2$



Fall 2: $(k_1 + k_2 \geq k \wedge k_2 \geq k)$

Dieser Fall ist identisch zum Fall 2, der für 1a betrachtet wurde (vergl. die Abbildung von Fall 2 für Algorithmus 1a).



Zwar ist schon $(h-k, 2k)$ ein TR, der aber die Bedingung $k_1 > 0$ nicht erfüllt. Der TR $(h-k+1, 2k)$ überdeckt alle TRs $(h-k+1+i, 2k)$ ($1 \leq i \leq k_1$) und ist deshalb leftmost-TR.

1. Schritt

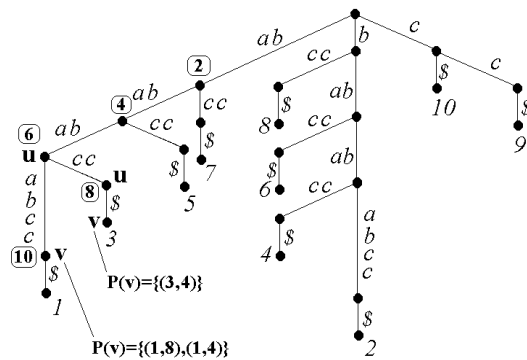
Berechnung (Suffix₁):

1. $P(v) = \{(1,8), (1,4)\} \neq \{\}$
2. $(i,l) = (1,8), D(u) = 10, l=8 \Rightarrow D(u) > l$

Berechnung (Suffix₃):

1. $P(v) = \{(3,4)\} \neq \{\}$
2. $(i,l) = (3,4), D(u) = 8, l=4 \Rightarrow D(u) > l$

2. Schritt



Berechnung (Suffix₁):

1. $P(v) = \{(1,8), (1,4)\} \neq \{\}$
2. $(i,l) = (1,8), D(u) = 6, l=8 \Rightarrow l > D(u)$
3. $P(v) := \{(1,4)\}$

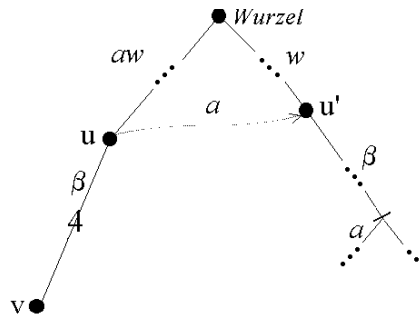
Berechnung (Suffix₃):

1. $P(v) = \{(3,4)\} \neq \{\}$
2. $(i,l) = (3,4), D(u) = 8, l=4 \Rightarrow D(u) > l$

3. Schritt

Um die Laufzeitschranke von $O(n)$ einzuhalten muß der bottom-up-parse erst alle Kinderknoten v eines Knotens u abarbeiten bevor er beim Knoten u weitermachen kann. Sonst würden viele Knoten mehrfach besucht werden, und die Laufzeit wäre im worst-case für einen vollständigen binären Baum $2^{(\log n + 1)} - 1 * \log(n+1) = O(n \log n)$. Es tritt jetzt im bottom-up Algorithmus das Problem auf, daß die P_i -Listen der Kinderknoten zu einer einzigen Liste zusammengefasst, und deswegen sortiert werden müssen. Das ist jedoch mit einer Laufzeit $\Omega(n^2)$ für alle Listen im Suffixbaum verbunden. Um dies zu vermeiden wird stattdessen bevor

Suffix-walk:



In der Zeichnung ist im linken und im rechten Pfad nichts anderes zu sehen als das, was als Rechtsrotation vorgestellt wurde. Auf dem linken Pfad ist ein leftmost-TR, der in Phase 2 schon markiert wurde mit der Beschriftung $aw\beta$ eingetragen. Dieser TR endet in diesem Beispiel 4 Zeichen vom Knoten u entfernt. In Phase 3 ist dieser TR der Startpunkt für den Suffix-Walk. Der Suffix-Walk beginnt an der Wurzel, und nimmt sobald er den Knoten u erreicht den Suffix-Link zum Knoten u' . Dieser Knoten hat nach Def. die Eigenschaft, daß seine Knotenmarkierung gerade w ist. Von u' aus läuft der Suffix-Walk weiter nach unten im Baum und geht den Pfad, der gerade β entspricht. (Nach der Def. des Suffixbaums gibt es immer einen solchen Pfad!). Dann muß jedoch das Zeichen a gefunden werden, damit die Rechtsrotation komplett ist. Dazu müssen alle Pfade, die hinter $w\beta$ beginnen getestet werden. Bei Erfolg wird der neu gefundene TR in $T(S)$ markiert, und es wird weiter rechtsrotiert, jetzt allerdings ausgehend von der Wurzel über den Pfad $w\beta a$, d.h. dem neu gefundenen TR. Die Rotationen werden solange wiederholt bis eine Rotation erfolglos endet. Dann wird der nächste TR aus der Menge Q' abgearbeitet.

IV. Anwendungen

4.0 Anwendungen

Nachdem der Suffixbaum eines Strings mit dem Vokabular der Tandem-Repeats markiert ist, lassen sich verschiedene Fragen über TR in S in linearer Laufzeit beantworten. So z.B. die Frage nach dem kürzesten bzw. längsten TR, der an Position i in S beginnt. Die Frage nach allen Vorkommen von TRs in S läßt sich leicht beantworten, wenn man im markierten Suffixbaum $T(S)$ jeweils von der Position an der ein TR-Typ endet zu allen Blättern in diesem Teilbaum herabsteigt. Jede Nummer an einem Blatt gibt die Position an, an der dieser Typ nochmals vorkommt. Ein DFS ähnlicher Algorithmus benötigt dafür $O(n)$ Laufzeit. Der Algorithmus läßt sich außerdem gut auf die Behandlung von Tandem Arrays erweitern.

V. Anhang

5.0 Literaturverzeichnis

- 1) "Auswirkungen geringer Strahlendosen - Spuren sowjetischer Atomtests"
Neue Züricher Zeitung 13.5. 2004, Auszug aus Science 295, 946/1037 (2002)
- 2) Crochemore, M.
"Recherche lineaire d'un carre dans un mot"
Acad. Sci., Paris, 296:781-784 (1983)
- 3) Crochemore, M.
"An optimal algorithm for computing the repetitions in a word"
Inf. Process. Lett., 12 (5): 244-250 (1981)
- 4) Crochemore, M. & Rytter W.
"Text algorithms"
Oxford Universty Press, New York, NY, (1994)
- 5) Csink, Amy K. & Henikoff Steven
"Something from nothing: the evolution and utility of satellite repeats"
TIG May (1998) Vol.14 No.5 p. 200-204 (c) Elsevier Science Ltd.
- 6) Dib, C. et. Al.
"A comprehensive map of the human genome based on 5264 microsatellites" (1996)
- 7) Farach, M.
"Optimal suffix tree construction with large alphabets"
in Proc. 38th Annu. Symp. Found. Comput. Sci., FOCS 97, pages 137-143
New York, NY , IEEE Press (1997)
- 8) Fraenkel, A.S. & Simpson J.
"How many squares can a string contain ?"
J. Comb. Theory Ser. A, 82:112-120 (1998)
- 9) Griffiths, A.J.F. ,Gelbart, W.M. ,Lewontin R.C. ,Miller J.H.
"Modern Genetic Analysis", W.H. Freeman and Company (2002) S.284 ff.
- 10) Gusfield, Dan
"Algorithms on strings, trees and sequences"
Cambridge University Press, New York (1997)

- 11) Jeffreys, A.J.; Wilson, V. & Thein, S.L.
"Hypervariable minisatellite regions in human DNA"
Nature 314:67-73
- 12) Kolpakov, R. & Kucherov, G.
"Maximal repetitions in words or how to find all squares in linear time"
Technical Report 98-R-227, LORIA, (1998)
- 13) mreps
<http://bioweb.pasteur.fr/seqanal/interfaces/mreps.html>
- 14) Kosaraju, S.R.
"Computation of squares in a string"
in M. Crochemore & D. Gusfield (editors) Combinatorial pattern matching: 5th Annual Symposium, CPM 94. Asilomar, California, June 1994, Proceedings, number 807 in Lecture Notes in Computer Science, pages 146-150, Springer Verlag Berlin (1994)
- 15) Lakowicz, J.R.
"Topics in Fluorescence Spectroscopy, Vol.7 DNA Technology"
Kluwer Academic/Plenum Publishers New York (2003)
- 16) Landau, G. M. & Schmidt, J.P
"An algorithm for approximate tandem repeats"
in A. Apostolico, M. Crochemore, Z. Galil & U. Manber (editors),
Combinatorial pattern matching: 4th Annual Symposium, CPM 93.
Padova, Italy, June 1993, Proceedings, number 684 in Lecture Notes in Computer Science, pages 120-133, Springer Verlag Berlin (1993)
- 17) Lempel, Abraham & Ziv, Jacob
"On the complexity of finite sequences"
IEEE Transactions on Information Theory, vol. IT-22, No.1, january (1976)
- 18) Macas, Jiri ; Meszaros, Tibor & Nouzova, Marcela
"PlantSat:a specialized database for plant satellite repeats"
Bioinformatics, vol.18 no. 1 (2002) p. 28-35
- 19) Main, M.G.
"Detecting leftmost maximal periodicities"
Discrete applied mathematics, 25:145-153, 1989
- 20) Main, M.G. & Lorentz, J.R.
"Linear time recognition of squarefree strings"
in A. Apostolico and Z. Galil (editors), Combinatorial Algorithms on Words, volume F12 of NATO ASI Series, pages 271-278, Springer Verlag Berlin (1985)

- 21) Pupko, Tal & Graur, Dan
"Evolution of microsatellites in the yeast *saccharomyces cerevisiae*: Role of length and number of repeated units"
Journal of Molecular Evolution 48:313-316 Springer Verlag New York (1999)

- 22) "Reputer"
<http://www.genomes.de>
<http://bibiserv.techfak.uni-bielefeld.de/library/reputer/manual>

- 23) Rodeh, Michael; Pratt, R. Vaughan & Even Shimon
"Linear algorithm for data compression via string matching"
Journal of the association for computing machinery, vol. 28 No.1, january (1981), pp.16-24

- 24) Stoye, Jens & Gusfield, Dan
"Simple and flexible detection of contiguous repeats using a suffix tree"
in M. Farach, editor, combinatorial pattern matching: 9th Annual Symposium, CPM 98, Piscataway, New Jersey, USA, July 1998, Proceedings, number 1448 in Lecture Notes in Computer Science, pages 140-152, Berlin, 1998. Springer Verlag.

- 25) "Tandem-Tools"
<http://tandem.bu.edu>

- 26) "The earliest recorded plant virus disease"
Nature, vol.422 24 APRIL 2003 www.nature.com/nature

- 27) Tran, Thao T., Emanuele II, Vincent A. & Zhou, Tong G.
"Techniques for detecting approximate tandem repeats in DNA"
School of Electrical and Computer Engineering, Georgia Institute of Technology
Atlanta, USA