Algorithmen und Datenstrukturen 2
Kap. 19: Suffixbäume

Robert Giegerich

Faculty of Technology
Bielefeld University

robert@TechFak.Uni-Bielefeld.DE

Vorlesung, U. Bielefeld, Sommer 2011

Bekannte Verfahren zur exakten Suche in Zeichenreihen

Exakte Suche: Finde Auftreten von Muster P in Text T.
$|P| = m, |T| = n$. $a, b, c \in A$. $w, u, v \in A^*$. $m << n$.
Was wir schon kennen:

- Naive Suche: Laufzeit $O(mn)$ im worst case
- Boyer-Moore, Boyer-Moore-Horspool: $O(m) + O(n)$ (Vorverarbeitung von P + Suche in T im average case)
- Knuth-Morris-Pratt: $O(m) + O(n)$ (Vorverarbeitung von P + Suche in T im worst case)

In den Messungen haben wir gesehen, dass praktisch der KMP nur für kleine Alphabete und Strings mit vielen Wiederholungen schneller ist als der (einfachere) BMH.

Neue Idee: Indexstrukturen

Welche Variante fehlt noch:
Vorverarbeitung von T zu "Indexstruktur", die schnelle Suche erlaubt. Schwierigkeiten:
Erstellen des Index möglichst in $O(n)$.
Platzbedarf nicht zu gross, also $O(\eta n)$ mit $\eta \in O(1)$.
Konkurrierende Ideen:

- alphabetischer Index (ggf. nur für Stichworte)
- k-Wort-Index (Positionen aller Teilworte der Länge k)
- Suffix-Baum: $O(n) + O(km)$
  (Vorverarbeitung von T und Suche bei k Auftreten von P)
- Suffix-Array: $O(n) + O(log_2 n)$

Ja ist das denn die Möglichkeit?

Was bedeutet Suche in O(m)?

Ja ist das denn die Möglichkeit?

Was bedeutet Suche in O(m)?

Nachdem der Suffixbaum einmal konstruiert ist:

- Die Suchzeit ist unabhängig von der Größe des Textes !!!
- Die Suche nach einem Auftreten ist mit $O(m)$ optimal.
- Die Suche nach allen Auftreten hängt von der Größe der Ausgabe ab.

Wichtige Algorithmen:
[Weiner 1973, McCreight 1976, Ukkonen 1994]
[Giegerich, Kurtz, Stoye 2003]

A^+-Baum

A ist Alphabet. Sei $w, u, v \in A^*$.
Ist $w = uv$, so heißt u Präfix und v Suffix von w.

Definition (1)

Ein A^+-Baum ist ein gerichteter Baum mit Wurzel, dessen Kanten mit Zeichenreihen aus A^+ markiert sind. Es gilt die Bedingung:
Ist $y \neq z$, $x \stackrel{a\ldots}{\rightarrow} y$ und $x \stackrel{b\ldots}{\rightarrow} z$, so gilt $a \neq b$.

Namen für die Knoten

Beobachtungen:
Der Pfad von der Wurzel zu einem Blatt ist mit einem String w markiert, der aus der Konkatenation der Kantenmarkierungen im Pfad besteht.
Wir nennen diesen Knoten $\overline{w}$. Das ergibt eine eindeutige Bezeichnung für jeden Knoten – warum?

Namen für die Knoten

Beobachtungen:
Der Pfad von der Wurzel zu einem Blatt ist mit einem String w markiert, der aus der Konkatenation der Kantenmarkierungen im Pfad besteht.
Wir nennen diesen Knoten $\overline{w}$. Das ergibt eine eindeutige Bezeichnung für jeden Knoten – warum?

Weil es keine zwei gleich markierten Wurzelpfade geben kann.

Namen für die Knoten

Beobachtungen:
Der Pfad von der Wurzel zu einem Blatt ist mit einem String w markiert, der aus der Konkatenation der Kantenmarkierungen im Pfad besteht.
Wir nennen diesen Knoten $\overline{w}$. Das ergibt eine eindeutige Bezeichnung für jeden Knoten – warum?

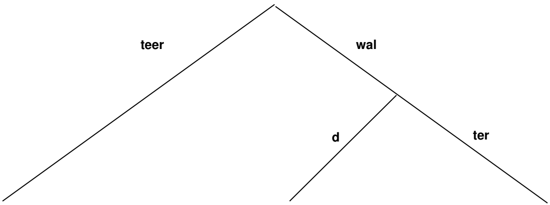
Weil es keine zwei gleich markierten Wurzelpfade geben kann.
Und wie heißt die Wurzel?

Namen für die Knoten

Beobachtungen:
Der Pfad von der Wurzel zu einem Blatt ist mit einem String w markiert, der aus der Konkatenation der Kantenmarkierungen im Pfad besteht.
Wir nennen diesen Knoten $\overline{w}$. Das ergibt eine eindeutige Bezeichnung für jeden Knoten – warum?

Weil es keine zwei gleich markierten Wurzelpfade geben kann.
Und wie heißt die Wurzel?
Natürlich $\overline{\varepsilon}$.
Man sagt, der Baum t "enthält" die Wörter $\{w | \overline{w} \text{ ist Knoten in } t\}$
.

Baum zu {wal, walter, wald, teer}



Der Baum enthält

- als Blattknoten $\overline{walter}$, $\overline{wald}$, $\overline{teer}$
- als inneren Knoten $\overline{wal}$
- als "implizite" Knoten z.B. $\overline{tee}$, $\overline{w}$, $\overline{wa}$, $\overline{walt}$

Konstruktionsaufgabe

Gegeben eine Menge von Wörtern W, konstruiere den A^+-Baum t, der W enthält, und sonst (nur) die Präfixe der $w \in W$.

- Dieser Baum ist EINDEUTIG bestimmt, abgesehen von
  – der Anordnung der Unterbäume eines Knotens,
  – Benutzung von nicht-verzweigenden Knoten
- Man sortiert z.B. die Wörter in W alphabetisch und fasst von der Wurzel her zusammen.

Atomarer Suffixbaum (AST)

Gegeben Text T.

Definition (2)

Der atomare Suffixbaum $ast(T)$ ist der A^+-Baum, dessen Kanten mit einzelnen Zeichen aus A markiert sind, und der alle Suffixe von T enthält.

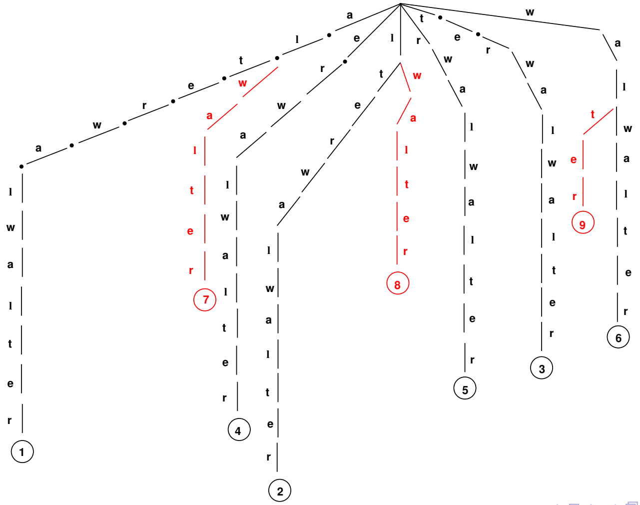
Als innere Knoten enthält er zwangsläufig auch alle Präfixe der Suffixe. Also gilt

$$\overline{w} \text{ ist Knoten in } T \text{ gdw. } w \text{ ist Teilwort in } T$$

.

Beispiel zu $ast(T)$

$T =$ "alterwalwalter"



Eigenschaften des atomaren Suffixbaums

Was leistet ein atomarer Suffixbaum?

- $\overline{w}$ Blatt in $ast(T) \quad \Rightarrow \quad w$ ist Suffix von T.
- Umkehrung gilt nicht!
- Sei \$ ein Zeichen, das nicht in T vorkommt. In $ast(T\$)$ entspricht jeder Suffix einem Blatt. Man kann die Startposition jeden Suffixes s am Blatt $\overline{s}$ anhängen.
- Suche nach Auftreten von P in T in $O(m)$ möglich – verfolge P als Pfad in $ast(T)$.
- Alle Auftreten von P sind in einem Unterbaum zusammengefasst.

Problem: Die Größe von $ast(T)$ ist $O(n^2)$!!

(Kompakter) Suffixbaum

Nicht verzweigende Pfade im Baum liefern keine Information und werden zu einer Kante zusammengezogen:

Definition (3)

Der (kompakte) Suffixbaum $st(T)$ ist ein A^+-Baum, der die Suffixe von T enthält und in dem alle inneren Knoten verzweigen.

(Kompakter) Suffixbaum

Nicht verzweigende Pfade im Baum liefern keine Information und werden zu einer Kante zusammengezogen:

Definition (3)

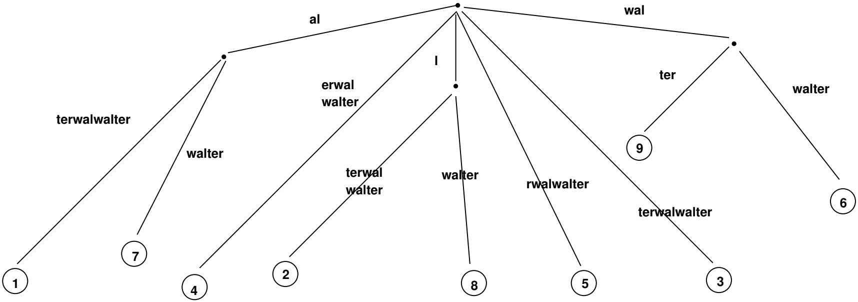
Der (kompakte) Suffixbaum $st(T)$ ist ein A^+-Baum, der die Suffixe von T enthält und in dem alle inneren Knoten verzweigen.

Jetzt gibt es statt $O(n^2)$ Knoten immer noch $O(n^2)$ Zeichen an den Kanten – wo ist der Platzgewinn?

- Wieviele Knoten hat er kompakte Suffixbaum? Weniger auf jeden Fall – aber sind es $O(n)$?
- Wieviele Platz brauchen die Kantenmarkierungen? Immer noch $O(n^2)$?

Beispiel zu $st(T)$

$T =$ "alterwalwalter"



Linearer Platzbedarf des Suffixbaums

Wie groß ist der Platzbedarf für den Index, zusätzlich zum Text?

- Zahl der Knoten: $O(n)$. Beweis: Maximal n Blätter; jeder innere Knoten verzweigt; Binärbaum mit n Blättern hat maximal $2n$ Knoten; der Suffixbaum also maximal $2n \in O(n)$ Knoten.
- Jede Kantenmarkierung ist ein Teilwort T(i,j) und kann durch (i, j) dargestellt werden. Platzbedarf pro Wort $O(1)$, eine Kante pro Knoten (ausgenommen die Wurzel), also $O(n)$ Kanten und damit auch $O(n)$ Platzbedarf für die Markierungen.

Der Suffixbaum erhöht den Platzbedarf für den Text um einen konstanten Faktor. Sorgfältige Implementierungen erreichen einen Faktor 5 - 9, was in der Praxis immer noch ein Problem sein kann.

Weitere Eigenschaften des Suffixbaums

Suffixbaum ist vielseitig einsetzbar

- P in T? ist in $O(m)$ Schritten entscheidbar.
- Repeat-Analyse: Alle wiederholten Auftreten des gleichen Musters sind in einem Unterbaum zusammengefasst und lassen sich effizient ablesen.
- Palindrome: Palindrome findet man leicht in $st(T\$T^{-1})$
- Statistik: Statistik über Auftreten aller k-Worte läßt sich im oberen Teil des Baumes ablesen. Man propagiert die Anzahl der Blätter aller Unterbäume in Richtung Wurzel.

In diversen Anwendungen werden zusätzliche Informationen mit den Knoten von $st(T)$ assoziiert.

Suffix Links

Definition (4)

Falls in T die Wörter wa und wb enthält, mit $a \neq b$, so heißt w (rechts-)verzweigend in T.

Beobachtungen über Teilwort w aus T:

- w ist verzweigend gdw. $\overline{w}$ Knoten im Suffixbaum ist
- Wenn w kein Blatt ist und nicht verzweigt, heißt $\overline{w}$ impliziter Knoten
- Ist aw verzweigend in T, dann auch w

Definition (5)

Zusätzliche Kanten $\overline{aw} \rightarrow \overline{w}$ nennt man Suffix-Links. Sie helfen bei der Suffixbaum-Konstruktion.

Die WOTD Konstruktion (1)

Write only, top down Suffixbaumkonstruktion

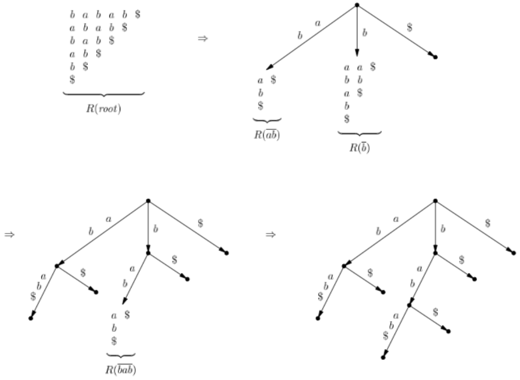


Figure 1. The write-only top-down construction of $ST(babab)$.

Die WOTD Konstruktion (2)

- Repräsentation der Suffixe durch Startpositionen in T in $O(n)$.
- Rekursive top-down-Konstruktion, beginnend an der Wurzel mit allen Suffixen
- Gruppierung der Suffixe für den Unterbaum nach Anfangsbuchstabe $a \in A$
- Bestimmung des längsten gemeinsamen Präfix für jede Gruppe: $lcp(\{aw_1, ...aw_k\}) = ap$ so dass $w_i = pv_i$ und $|p|$ maximal
- ap markiert die Kante zum Unterbaum für $\{v_1, ..., v_k\}$
- Konstruktion terminiert, wenn k $= 1$

Eigenschaften von WOTD

- top-down: WOTD eignet sich für *lazy* Suffixbäume
- top-down: Jeder Unterbaum kann getrennt von den anderen konstruiert werden
- write-only: Knoten im Baum werden nur konstruiert, danach nicht mehr gelesen
- write-only: Gute Lokalität in Bezug auf Prozessor-Cache

Komplexität von WOTD

Aufwand der Schritte:

- Sortiere k Suffixe nach erstem Zeichen: $O(k)$; $\sum k \leq n$
  Wie geht das? Vgl. A&D 1.
- Bestimme gemeinsamen Präfix: $O(|p| \cdot k)$ mit $k \in O(n)$
- Aufbau der Baumstruktur: $O(1)$ pro Knoten

Insgesamt

- Worst case: $|k| \in O(n)$, Laufzeit $O(n^2)$!
- Average case: $|p| = log\ n$, Laufzeit $O(n \cdot log\ n)$
- Kein Faktor $|A|$.

Welcher String produziert den worst case?

Weiner's online-Konstruktion

"Algorithm of the year" in 1973

- Liest den Text von rechts nach links
- Konstruiert zuerst die Blätter für die kürzesten Suffixe
- Zu jedem "Zeitpunkt" $T = uv$ existiert $st(v)$
- Braucht umfangreiche Hilfsstrukturen für Navigation im Baum, um Aufwand $O(n)$ zu erreichen
- Sehr kompliziert, selten implementiert

McCreight's Vorwärts-Konstruktion

Einfacherer Algorithmus von 1976

- Kein online-Verfahrem
- Trägt die Suffixe vom längsten zum kürzesten hin ein
- Zum "Zeitkpunkt" k ist der Baum ein kompakter A^+-Baum für $s_1, \ldots, s_k$, aber kein Suffixbaum
- Verwendet Suffix-Links zur Navigation im Baum, um Aufwand $O(n)$ zu erreichen.

Wie ähnlich McCeight's Verfahren einer online-Konstruktion ist, wurde viele Jahre nicht erkannt

McCreight's Konstruktion: Beispiel für "adadc"

From Ukkonen to McCreight and Weiner 339

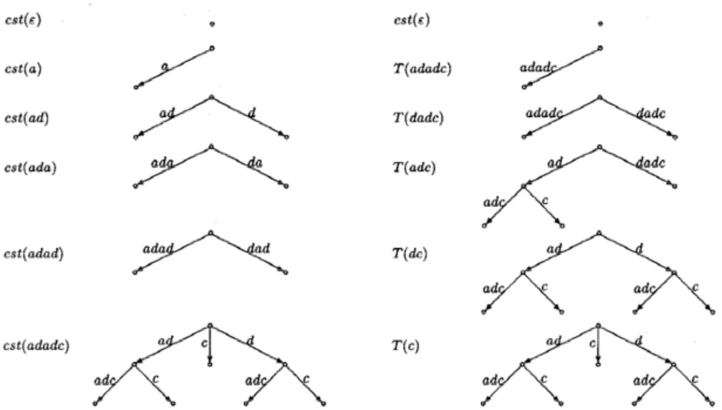


Fig. 5. Sequence of trees constructed by *ukk* and *mcc*.

Suffixbäume als Geheimwaffe der Algorithmen-Theoretiker

1976 – 1994 gabe es kaum Implementierungen von Suffix-Bäumen

- Kompliziert und mit hohem konstanten Faktor beim Speicherbedarf.
- Viel benutzt in der Algorithmen-Theorie, um das Teilproblem des exakten String Matching effizient zu lösen.
- Die grossen Sequenzdatenmengen der Genomforschung und die Textmengen des WWW führten zu einer Renaissance der Suffixbäume, beginnend etwa 1994.

Ukkonen's online-Konstruktion (1)

Grundidee für die Konstruktion in $O(n)$:

- Der Baum $st(T)$ wird "online" konstruiert, d.h. T wird zeichenweise gelesen und es existiert zu jedem "Zeitpunkt" $T = uv$ der Baum $st(u)$.
- Das Verlängern der Blattkanten wird durch geschickte Repräsentation, sog. "open edges" vermieden.
- Der jeweils "aktive Suffix" zeigt an, an welchem Punkt im Baum geändert werden muss.
- insgesamt wird n mal geändert – das bedeutet, pro Änderung dürfen (amortisiert) nur $O(1)$ Rechenschritte anfallen!!

Ukkonen's online Konstruktion (2): Beispiel für "adadc"

From Ukkonen to McCreight and Weiner 339

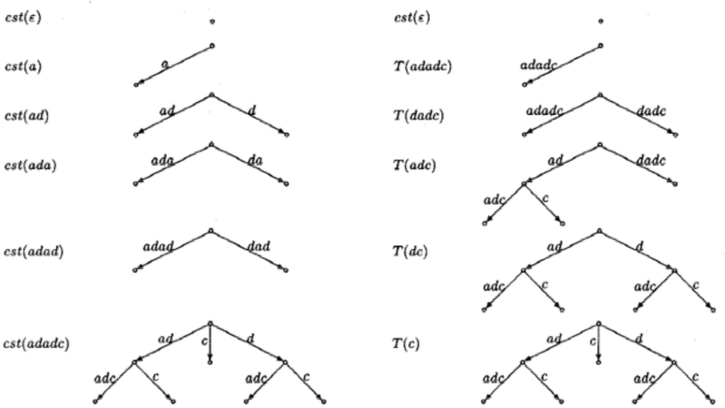


Fig. 5. Sequence of trees constructed by *ukk* and *mcc*.

Ukkonen's online Konstruktion (3): Algorithmus

- Darstellung der internen Kanten als $T(i,j)$ wie oben, der Blattkanten als "open edges" $T(i,J)$, worin die Variable J das Ende der jeweils gelesenen Eingabe enthält.
- Beginne mit aktivem Knoten $\overline{\varepsilon}$, $J = 0$, und Suffix Link $\overline{\varepsilon} \to \overline{\varepsilon}$
- Für $T(0,J)$ is der aktive Suffix der längste embedded suffix, $s_{akt} = T(i,J), s \in T(1, J-1), i$ minimal. (s_{akt} ist der längste Suffix von $T(1,J)$, der nicht durch ein Blatt repräsentiert ist.)
- s_{akt} ist repräsentiert durch den aktiven Knoten von $st(T(1,J)$. Er kann explizit oder implizit sein.
- Beim Schritt $J \to J+1$ wird der Baum am aktiven Knoten geändert.

Ukkonen's online Konstruktion (4): Algorithmus

Erweiterungs-Schritt $J := J + 1$:
Sei s_{akt} repräsentiert durch $(\overline{x}, u)$ mit $\overline{x} \overset{uav}{\to} \overline{y}$.
Sei $T(J) = b$.

- Falls $a = b$, setze $s_{akt} = s_{akt}a$, den aktiven Knoten auf $(\overline{x}, ua)$. Keine neuen Knoten werden erzeugt.
- Falls $a \neq b$, erzeuge $\overline{xu}$ falls $u \neq \varepsilon$ (neuer expliziter Knoten), erzeuge $\overline{xu} \overset{av}{\to} y$,(Spalten der Kante), erzeuge $\overline{xu} \overset{b}{\to} z$ (neues open edge).
- Folge dem Suffix Link von $\overline{x} = \overline{x_1 \ldots x_k}$ zu $\overline{x_2 \ldots x_k}$ und wiederhole den Erweiterungsschritt, bis Fall $a = b$ eintritt.
- Bei jeder Wiederholung muss der Suffix Link von $\overline{xu}$ zum nächsten neuen Knoten eingetragen werden.

Ukkonen's online Konstruktion (5): Komplexität

- Es gibt genau n Erweiterungsschritte, mit jedem Schritt wächst J um 1
- In jeden Schritt werden 0 bis $O(n)$ neue Knoten eingefügt und Kanten gespalten. Pro Knoten/Kante in $O(1)$ Operationen.
- Sei $T(i, J)$ jeweils der aktive Suffix.
  Wird kein Knoten eingefügt, bleibt i stehen.
  Mit jedem Verfolgen eines Suffix Link rückt i um 1 vor.
  Das ist nur n mal möglich.

Insgesamt werden n mal Knoten in $O(1)$ eingebaut:
Gesamtaufwand ist $O(n)$.

From Ukkonen to McCreight and Weiner

Zusammenhang der Suffixbaum-Konstruktionen:

- Ukk und McC führen gleiche Serie von Knoten-Einfügungen und Kanten-Splits aus, nur ihre Interpretation ist anders
- Ukk und McC haben praktisch gleiche Effizienz
- Die Suffix Links in $ast(T)$ sind der reverse Präfixbaum
- In $st(T)$ sind sie ein Teil reversen Präfixbaums
- Macht man die expliziten Knoten des reversen Präfixbaums im Suffixbaum explizit und trägt man alle Suffix Links ein, entsteht der "Affixbaum", in dem man in $O(n)$ bidirektional suchen kann
- Weiner's Algorithmus ist Ukk rückwärts

[Giegerich, Kurtz: *From Ukkonen to McCreight and Weiner: A unifying view of linear time suffix tree construction.*

Algorithmica (1997) 19: 331-353]