Suffixbäume

Definition, Konstruktion, erste Anwendungen

Stephan Beyer

Hauptseminar AFS
Institut für Theoretische Informatik
Fakultät für Informatik und Automatisierung
Technische Universität Ilmenau

14. Juni 2007

Teil I

Einleitung

Worum geht es?

Teilwörter

Sei Σ ein Alphabet konstanter Größe.

Definition

Sei $w \in \Sigma^*$. Dann ist

$$\mathcal{F}(w) := \{x \mid x \text{ ist Teilwort von } w\}$$

die **Faktormenge** (**Teilwortmenge**) von w.

Problem (Teilwortproblem)

Sei Σ ein Alphabet und $x, w \in \Sigma^*$. Ist $x \in \mathcal{F}(w)$?

Worum geht es?

Datenstrukturen

- eine spezielle Organisation der Daten, kann helfen, Laufzeiten zu verbessern
- wir verlangen von einer Datenstruktur D:
  - D hat *lineare Größe*,
  - D kann in *linearer Zeit* konstruiert werden,
  - D erlaubt (nach Vorverarbeitung von w) das *Teilwortproblem* für x in $\mathcal{O}(|x|)$ zu lösen.
- ein *Suffixbaum* (engl. „*suffix tree*“) leistet dies

Suffixbaum

Definition

Definition

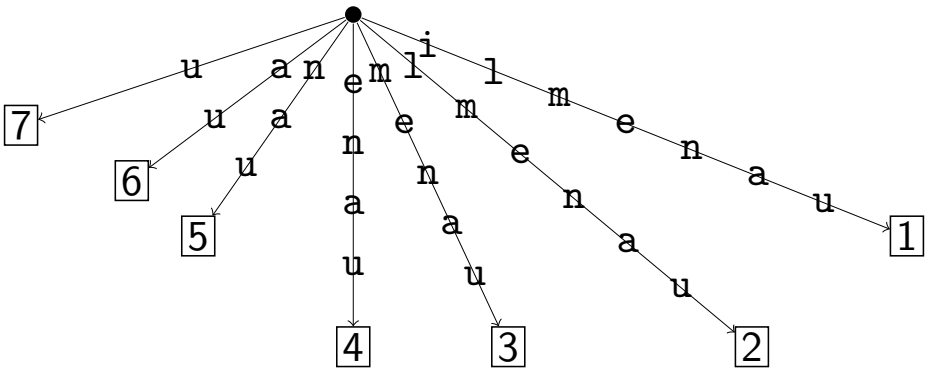
Ein (kompakter) **Suffixbaum** $\mathcal{T} := \mathcal{T}(w) = (V, E; \text{label})$ für $w = w_1\, w_2 \ldots w_m \in \Sigma^m$ ist ein gerichteter Wurzelbaum (mit Wurzel r), der den folgenden Bedingungen genügt:

- $\mathcal{T}$ hat genau m Blätter $1, \ldots, m \in V$.
- label: $E \to (\mathcal{F}(w) - \{\varepsilon\})$ ist linkstotal, d. h. jede Kante $e \in E$ ist beschriftet mit $\text{label}(e) \in \mathcal{F}(w) - \{\varepsilon\}$
- Sei π_i der Pfad von r zum Blatt i. Die (geordnete) Konkatenation der Kantenbeschriftungen auf π_i ist der an Position i startende Suffix $\text{suf}_i(w) := w_i\, w_{i+1} \ldots w_m$ von w.
- Jeder innere Knoten außer r hat mindestens zwei Kinder.
- Die Beschriftungen zweier aus einem Knoten herausführenden Kanten beginnen nicht mit gleichem Buchstaben.

Suffixbaum

Beispiele

$w = \texttt{ilmenau}$

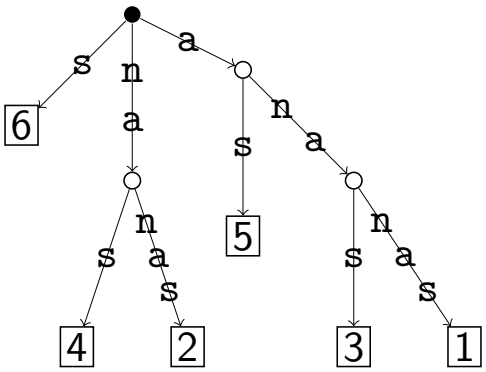


- $|w| = 7$
- 7 Knoten

Suffixbaum

Beispiele

$w = \texttt{ananas}$

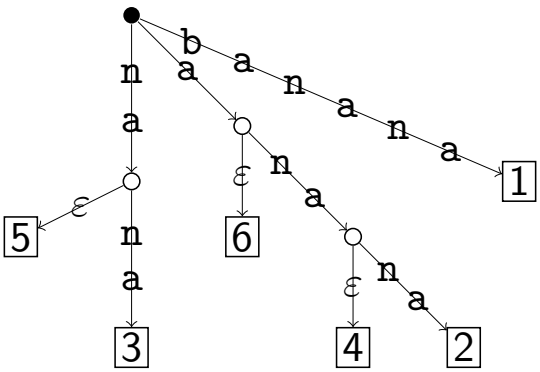


- $|w| = 6$
- 10 Knoten

Existenz

Gegenbeispiel

$w = \texttt{banana}$



Widerspruch! Definition fordert $\forall e \in E: \ \text{label}(e) \neq \varepsilon$
$\Rightarrow$ Suffixbaum für w existiert nicht.

Existenz

Lösung

Beobachtung

Ein Suffixbaum für w existiert nicht, gdw. es für eine Zerlegung $w = u\,v$ ein Teilwort $z \neq \varepsilon$ in u gibt, sodass $z = v$.

Lösung:

- Erweiterung des Alphabets Σ um ein Endezeichen \$
- Erstellung des Suffixbaums für w\$ statt für w

Da \$ nicht in w vorkommt, existiert $\mathcal{T}(w\$)$.

Praktisch:

- ASCII-Zeilenbegrenzungszeichen `CR` bzw. `LF`
- String-Endezeichen in C hat ASCII-Code `0`

Eigenschaften

lineare Größe

Satz

Der Suffixbaum $\mathcal{T}$ für $w \in \Sigma^m$ hat $\mathcal{O}(m)$ viele Knoten.

Beweis.

$\mathcal{T}$ hat nach Definition m Blätter. Ein Baum mit m Blättern hat maximal $m - 1$ innere Knoten, wenn jeder innere Knoten mindestens zwei Kinder hat. Nach Definition hat jeder innere Knoten von $\mathcal{T}$, außer der Wurzel, mindestens zwei Kinder.

- Die Wurzel von $\mathcal{T}$ habe mindestens zwei Kinder. Dann ist die Knotenanzahl höchstens $2\,m - 1 \in \mathcal{O}(m)$.
- Die Wurzel habe nur ein Kind $\Rightarrow 2 \in \mathcal{O}(m)$ Knoten.
- Die Wurzel habe kein Kind $\Rightarrow 1 \in \mathcal{O}(m)$ Knoten.

□

Eigenschaften

lineare Größe (2)

Problem

Die Länge aller Kantenbeschriftungen label(e) *zusammen ist im Worst-Case* $\frac{m\cdot(m+1)}{2} \in \mathcal{O}(m^2)$.

Lösung

Statt Teilwörtern wird ein Paar von Indizes *(Start, Ende) gespeichert. D. h.* label *wird „umdefiniert“*

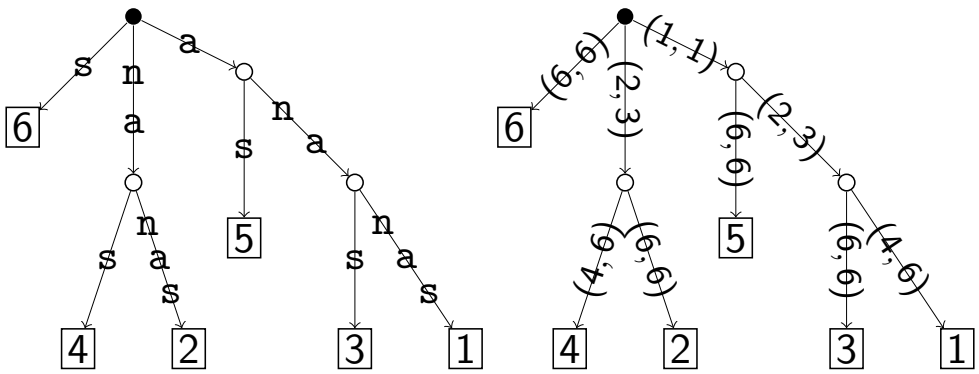
$$\text{label}: \; E \to \{(a,b) \mid a \in \{1,\ldots,m\}, b \in \{a,\ldots,m\}\}$$

und der zusätzliche Platzbedarf beschränkt sich auf $\mathcal{O}(1)$*; zusammen mit* w *auf* $\mathcal{O}(m)$.

Eigenschaften

lineare Größe – Beispiel

$w = \texttt{ananas}$



Eigenschaften

effiziente Lösung des Teilwortproblems

Satz

Mithilfe des Suffixbaums $\mathcal{T}$ für $w \in \Sigma^m$ kann das Teilwortproblem für $x \in \Sigma^n, n \leq m$ in $\mathcal{O}(n)$ gelöst werden.

Algorithmus (Beweisidee)

Sei $x = x_1 x_2 \ldots x_n$ und $\mathcal{T} = (V, E, \text{label})$. Setze $i := 1$ und u zur Wurzel von $\mathcal{T}$. Betrachte die herausführenden Kanten $e = (u, v)$. Fälle:

- Für alle e gilt $\text{label}(e)_1 \neq x_i$. Dann ist $x \notin \mathcal{F}(w)$. ($\mathcal{O}(|\Sigma|) = \mathcal{O}(1)$)
- Es gibt ein e mit $\text{label}(e) = x_i x_{i+1} \ldots x_j y_1 y_2 \ldots y_k, k \geq 0$.
  - Ist $j = n$, so ist $x \in \mathcal{F}(w)$. ($\mathcal{O}(n)$)
  - Ist $j < n$ und $k > 0$, so ist $x \notin \mathcal{F}(w)$. ($\mathcal{O}(n)$)
  - Ist $j < n$ und $k = 0$, dann setze $i := j + 1$ und betrachte nun $u := v$.

Die Korrektheit ist induktiv klar. Die Gesamtlaufzeit beträgt $\mathcal{O}(n)$.

Eigenschaften

Konstruktion in Linearzeit

Satz

Der Suffixbaum $\mathcal{T}$ für $w \in \Sigma^m$ kann in Linearzeit konstruiert werden.

1973 erster solcher Algorithmus von Weiner [Wei73]

1976 speicher-effizienterer Algorithmus von McCreight [McC76]

1995 Online-Konstruktion von Ukkonen [Ukk95]

Verallgemeinerte Suffixbäume

Definition (1)

Definition („reine“ Variante)

Sei eine Menge $M = \{w_1, w_2, \ldots, w_k\}$ gegeben mit $w_i = w_{i,1}\, w_{i,2} \ldots w_{i,m_i} \in \Sigma^{m_i}\ \forall i \in \{1, \ldots, k\}$. Ein (**verallgemeinerter**) **Suffixbaum** $\mathcal{T} := \mathcal{T}(M)$ ist ein Suffixbaum, der alle Wörter aus M repräsentiert. $\mathcal{T}$ hat genau $m_1 + m_2 + \cdots + m_k$ Blätter mit den Bezeichnungen $(1,1), (1,2), \ldots, (1, m_1)$, $(2,1), (2,2), \ldots, (2, m_2)$, $\ldots$, $(k,1), (k,2), \ldots, (k, m_k)$, d. h. Blatt (a, b) hat Pfadbeschriftung $\text{label}((a,b)) = w_{a,b}\, w_{a,b+1} \ldots w_{a,m_a}$.

Existenzvoraussetzung:

- $\mathcal{T}(w_i)$ existiert für alle i
- alle w_i haben paarweise unterschiedliche w_{i,m_i} (notfalls erzwungen durch einmalige Endezeichen)

Verallgemeinerte Suffixbäume

Definition (2)

Definition (Variante mit mehrdeutigen Blättern)

Sei eine Menge $M = \{w_1, w_2, \ldots, w_k\}$ gegeben mit $w_i = w_{i,1}\, w_{i,2} \ldots w_{i,m_i} \in \Sigma^{m_i}\ \forall i \in \{1, \ldots, k\}$. Ein (**verallgemeinerter**) **Suffixbaum** $\mathcal{T} := \mathcal{T}(M)$ ist ein Suffixbaum, der alle Wörter aus M repräsentiert. $\mathcal{T}$ hat höchstens $m_1 + m_2 + \cdots + m_k$ Blätter mit der Bezeichnung $\{(a,b) \mid \text{label}((a,b)) = w_{a,b}\, w_{a,b+1} \ldots w_{a,m_a}\}$.

Unterschied:

- haben zwei Wörter den gleichen Suffix, so gibt es mindestens ein mehrdeutiges Blatt

$\rightarrow$ Bezeichnung als Menge von (a, b)-Paaren.

Existenzvoraussetzung:

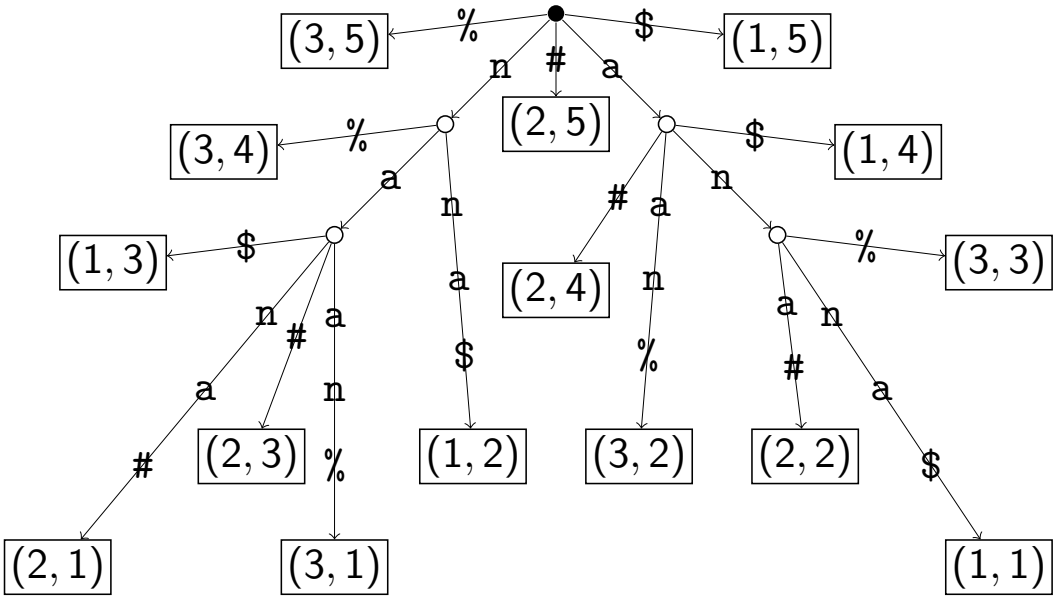
- $\mathcal{T}(w_i)$ existiere für alle i
- w_i ist kein echtes Teilwort von w_j für alle $i \neq j$

Verallgemeinerte Suffixbäume

Beispiel – reine Variante

- $M = \{\texttt{anna}, \texttt{nana}, \texttt{naan}\}$

$\longrightarrow$ $w_1 = \texttt{anna\$}$, $w_2 = \texttt{nana\#}$, $w_3 = \texttt{naan\%}$

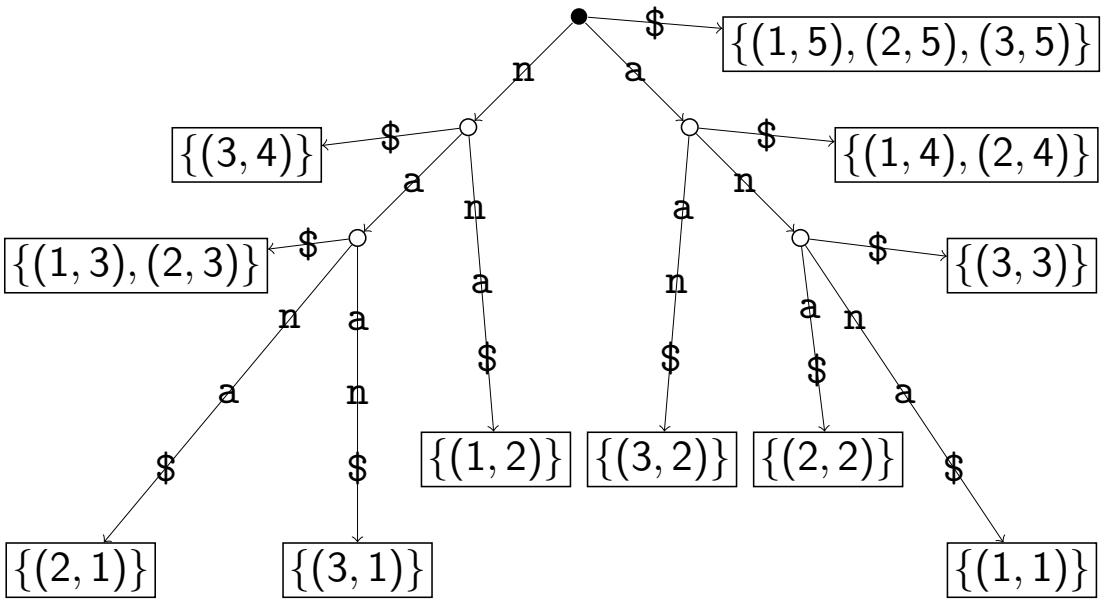


Verallgemeinerte Suffixbäume

Beispiel – mehrdeutige Blätter

- $M = \{\texttt{anna}, \texttt{nana}, \texttt{naan}\}$

$\longrightarrow$ $w_1 = \texttt{anna\$}$, $w_2 = \texttt{nana\$}$, $w_3 = \texttt{naan\$}$



Implementierungsdetails

Speicherung der ausgehenden Kanten für jeden Knoten anhand erstem Buchstaben der Kantenbeschriftung:

- Array mit Dimension $s := |\Sigma|$
  - Zugriff in $\mathcal{O}(1)$
  - Platz pro Knoten: $\mathcal{O}(s)$
  - Platzverschwendung, wenn spärlich besetzt
- sortierte, einfach verkettete Liste
  - Zugriff in $\mathcal{O}(s)$
  - Sortierung: kein zusätzlicher Aufwand, da Einfügungen immer erst nach Zugriffsversuch $\rightarrow$ in $\mathcal{O}(1)$; dafür kein voller Durchlauf bei Nichtvorhandensein einer Kante
  - nicht mehr Platzverbrauch als notwendig (evtl.: Pointer)
- balancierter Baum $\rightarrow \mathcal{O}(\log s)$
- Hashing $\rightarrow$ ideal: $\mathcal{O}(1)$
- Mischungen, denn obere Knoten haben höheren Ausgangsgrad als untere

Teil II

Konstruktion von Suffixbäumen

Überblick

Im Folgenden werden Algorithmen diskutiert mit...

- Eingabe: $w = w_1 \dots w_m \in \Sigma^m$ mit $m \geq 1$
- Suffixbaum für w existiere.
- Das Alphabet Σ habe *konstante* Größe.
- Ausgabe: Suffixbaum $\mathcal{T}(w)$.

Intuitiver Algorithmus

Algorithmus

1. Setze $\mathcal{T} :=$ 

2. **für** $i = 2, \dots, m$ **tue**
   1. Vergleiche $w_i\, w_{i+1} \dots w_m$ sukzessive mit den Zeichen $z_1, z_2, \dots$ der Beschriftungen entlang des (eindeutigen) Weges von der Wurzel. Stoppe bei $w_j \neq z_k$ mit $i \leq j < m, 1 \leq k \leq m - i$
   2. Wenn gestoppt innerhalb einer Kante (u, v), dann
      - Füge neuen Knoten x direkt zwischen Beschriftungszeichen z_{k-1} und z_k ein.

      sonst wurde bei einem Knoten x gestoppt.
   3. Füge neue Kante (x, i) mit $\text{label}((x, i)) = w_j\, w_{j+1} \dots w_m$ ein.

Laufzeit: $\mathcal{O}(m^2)$

Algorithmus nach Ukkonen

grob

Algorithmus (intuitiver Online-Algorithmus)

1. Erzeuge $\mathcal{T} = \bullet \xrightarrow{w_1} \boxed{1}$ {Phase 1}
2. **für** $j = 2, \ldots, m$ **tue** {Phase j}
   - **für** $i = 1, \ldots, j$ **tue** {Erweiterung i}
     - Finde Ende des Pfades von der Wurzel zum Suffix $w_i\, w_{i+1} \ldots w_{j-1}$ bzw. ε bei $i = j$
     - Füge dort ggf. w_j ein. Fälle:
       - Ende ist ein Blatt $\Rightarrow$ füge w_j an die Beschriftung
       - Es gibt kein folgendes w_j $\Rightarrow$ füge neue Blattkante mit Beschriftung w_j ein

Laufzeit, naiv implementiert: $\mathcal{O}(m^3)$

Verbesserung der Laufzeit

Lineare Laufzeit[1] wird erreicht durch

- Überspringen uninteressanter Beschriftungen;
- Folgen von sog. *Suffix-Links* zum Finden des Ende eines Pfades;
- Stoppen, sobald keine Veränderungen mehr passieren können;
- $\mathcal{O}(1)$-Behandlung von Kanten, die zu Blättern führen.

Diese Tricks werden im Folgenden kurz erläutert.

[1] konstante Laufzeit in jeder Phase

Trick 1: Überspringen

- Gesucht: Ende eines Pfades von $w_a \dots w_b, a < b$
- Naiv: alle Buchstaben einer Kante betrachten
- Besser: ersten Buchstaben und Länge einer Kante betrachten
  - Setze zu Beginn $g := b - a + 1$ (Gesamt-/Restlänge).
  1. Wähle Kante e mit $\text{label}(e) = x_1 \dots x_l$ anhand x_1.
  2. Wenn $l < g$, dann
     - überspringe Kante
     - setze $g := g - l$
     - gehe zu 1.

     sonst
     - gesuchtes Ende ist auf $\text{label}(e)_g$
- Laufzeit: Anzahl *Knoten* auf Pfad (naiv: Anzahl *Buchstaben* auf Pfad)

Trick 2: Suffix-Links

Definition

Definition

Sei $v \in V$ beliebiger Knoten im Suffixbaum $\mathcal{T}$. Dann ist die **Wegbeschriftung** $\text{label}(v)$ **des Knotens** v die Konkatenation der Kantenbeschriftungen auf dem Weg von der Wurzel zum Knoten v.

Definition

Sei v ein innerer Knoten mit $\text{label}(v) = x\,\alpha, x \in \Sigma, \alpha \in \Sigma^*$. $(v, s(v))$ heißt **Suffix-Link**, gdw. $s(v) \in V$ mit $\text{label}(s(v)) = \alpha$ existiert.

Für $\alpha = \varepsilon$ ist $s(v)$ die Wurzel.

Trick 2: Suffix-Links

Existenz

Bemerkung

Die Teil-Suffixbäume nach Phase j heißen auch **implizite Suffixbäume für den Präfix** $w_i \dots w_j$.

Fakt (Beweis in [Gus97], Lemma 6.1.1, Folgerung 6.1.1 und 6.1.2)

Existiert in einem (impliziten) Suffixbaum ein Knoten v mit $\text{label}(v) = x\,\alpha$, so existiert auch der Knoten $s(v)$ mit $\text{label}(s(v)) = \alpha$.

Trick 2: Suffix-Links

Verwendung im Algorithmus

- In Erweiterung $i = 1$ einer Phase j: gehe zu Blatt 1 (z. B. mit eigenen Pointer) und füge w_j an.
- Eine Erweiterung $i - 1$ in Phase j findet das Ende E des Pfades mit Beschriftung $w_{i-1}\, w_i \dots w_{j-1}$, d. h. an Knoten oder auf Kante.
- In Phase j und Erweiterung $i \geq 2$ tue:
  1. Liegt E auf innerem Knoten, dann sei dieser Knoten v, sonst gehe zum ersten Knoten v über E.
  2. Wenn v nicht Wurzel, dann folge Suffix-Link $s(v)$.
  3. Folge Pfad von $w_i \dots w_{j-1}$ (siehe Trick 1).
  4. Füge w_j ein (siehe groben Algorithmus).
  5. Wenn in Erweiterung $i - 1$ ein Knoten u erstellt wurde, dann erstelle Suffix-Link $(u, s(u))$, wobei $s(u)$ der Knoten ist, an dem $w_i \dots w_{j-1}$ endet.

Trick 3: vorzeitiges Stoppen

- Beobachtung 1: In Erweiterung i von Phase j passiert äußerlich nichts, wenn w_j direkt nach gefundenem Ende des Pfades kommt.
- Beobachtung 2: Danach passiert in der gesamten Phase äußerlich nichts mehr.
- Konsequenz: Beende Phase j (und merke i).

Trick 4: einmal Blatt, immer Blatt

Beobachtung:

- ist ein Blatt einmal erstellt, so bleibt es erhalten
- nur die Beschriftung kann noch um Zeichen erweitert werden

Daher:

- nutze als Ende-Index für die Kantenbeschriftung einen *globalen Platzhalter* p
- erhöhe p in jeder Phase um 1
- dann wird an allen Blattkanten die Beschriftung um ein Zeichen erweitert; in $\mathcal{O}(1)$

mögliche effiziente Realisierung in Praxis:

- spezieller Typ „Blattkante“, die nur Startindex speichert, oder
- p ist Pointer und zeigt auf Phasenwert j

Algorithmus nach Ukkonen

das Resultat

Algorithmus (nach Esko Ukkonen, 1995)

1. Erzeuge $\bullet \xrightarrow{(1,p)} \boxed{1}$ mit Platzhalter $p := 1$ und setze $i_{\text{alt}} := 1$.
2. **für** $j = 2, 3, \ldots, m$ **tue** $\{\text{Phase } j \geq 2\}$
   1. Setze $p := j$. $\{\text{Trick 4}\}$
   2. **für** $i = i_{\text{alt}}, i_{\text{alt}} + 1, \ldots, j$ **tue** $\{\text{Erweiterung } i\}$
      1. Finde Ende des Pfades zum Suffix $w_i\, w_{i+1} \ldots w_{j-1}$ (ε bei $i = j$). $\{\text{Trick 2 \& 1}\}$
      2. **wenn** w_j nicht auf Ende folgt, **dann**
         - füge neue Blattkante mit Beschriftung w_j (also Paar (j, p)) ein. Erstelle ggf. Suffix-Link.

         sonst breche i-Zählschleife ab. $\{\text{Trick 3}\}$
   3. Setze $i_{\text{alt}} := i$.

Algorithmus nach Ukkonen

Korrektheit und Laufzeit

Satz

Der Algorithmus nach Esko Ukkonen *konstruiert einen Suffixbaum $\mathcal{T}$ für w in linearer Zeit.*

Der Beweis findet sich in [Ukk95]. Eine Herleitung in [Gus97].

Grobes zur Laufzeit:

- In einer Erweiterung wird (über alle Phasen hinweg) nur konstant oft der Baum traversiert. Wird nicht traversiert, dann finden die übrigen Erweiterungen einer Phase implizit statt (Platzhalter, vorzeitiges Stoppen).
- Die Traversierung erfolgt mittels Suffix-Links und Überspringen.

Algorithmus nach Weiner

grob

Algorithmus (nach Peter Weiner, 1973)

1. Erzeuge $\mathcal{T} = \bullet \xrightarrow{w_m} \boxed{\text{m}}$
2. **für** $j = m-1, \ldots, 1$ **tue**
   1. Finde Ende des Pfades des längsten Präfixes $\alpha \in \Sigma^*$ von $\text{suf}_j(w) = \alpha\,\beta$, der einem Teilwort von $\text{suf}_{j+1}(w)$ gleicht.
   2. Wenn Ende nicht auf einem Knoten v, dann erzeuge Knoten v.
   3. Erzeuge Blatt j und Kante (v, j) mit Beschriftung β.

Laufzeit, naiv implementiert: $\mathcal{O}(m^2)$

Verbesserung der Laufzeit

Lineare Laufzeit wird erreicht durch zwei Vektoren der Dimension $|\Sigma|$ an jedem inneren Knoten v:

- **Indikatorvektor**: Bitvektor, der für Buchstabe x genau dann 1 ist, wenn die Beschriftung x label(v) im Baum, beginnend von der Wurzel, vorkommt.
- **Linkvektor**: Ist für Buchstabe x ein Pointer zum Knoten u mit label$(u) = x$ label(v), falls dieser existiert.

Nicht redundant: Links fordern Knoten, Indikator nicht.

In jeder Runde j:

- Traversierung in jeder Runde von Blatt $j+1$ zu Blatt j mithilfe dieser Vektoren.
- Aktualisierung der Vektoren.

Algorithmus nach McCreight

- platzsparende Variante von Weiner: führt Suffix-Links ein, statt Indikator- und Linkvektoren
- lineare Laufzeit mit Trick „Überspringen“
- Initialisierung mit $\mathrm{suf}_1(w)$-Kante $e = (r, 1)$, d. h.
  $\mathcal{T} :=$ 

- danach realisiert Phase $i = 2, \ldots, m$ jeweils die $\mathrm{suf}_i(w)$

Weitere Bemerkungen

Konstruktion verallgemeinerter Suffixbäume

eine Möglichkeit:

- Wörter konkatenieren
- modifizierten Ukkonen/Weiner/McCreight-Algorithmus darauf aufrufen, der bei einem Endezeichen die entsprechend Kante nicht weiter bearbeitet und Blätter richtig benennt.
- Laufzeit: $\mathcal{O}(m_1 + m_2 + \cdots + m_k)$

besser:

- $w_1, w_2, \ldots$ nacheinander mit Ukkonen, d. h. $\mathcal{T}(w_1)$ normal konstruieren und dann weitere w_i einfügen. (Details ausgeblendet.)

Weitere Bemerkungen

Parallele Konstruktion

- Ein Suffixbaum für ein Wort der Länge m kann in $\mathcal{O}(\log^2 m)$ auf einer PRAM mit m Prozessoren und exklusivem Schreiben (CREW) konstruiert werden.
  Ein Beweis dazu findet sich in [CR94] und [AIL+88].
- Mit Schreibkonflikten (CRCW) kann dies auch in $\mathcal{O}(\log m)$ geschehen.

Teil III

Erste Anwendungen

Finden von Teilwörtern

Algorithmus

Eingabe: $x = x_1\, x_2 \ldots x_n \in \Sigma^n$, $w = w_1\, w_2 \ldots w_m \in \Sigma^m$
Ausgabe: Menge der Startindizes aller x-Vorkommen in w

1. Erzeuge $\mathcal{T}(w)$ $\quad (\mathcal{O}(m))$
2. Setze $i := 1$ und u zur Wurzel von $\mathcal{T}$.
3. Betrachte die herausführenden Kanten $e = (u, v)$ in $\mathcal{O}(n)$. Fälle:
   - Für alle e gilt $\text{label}(e)_1 \neq x_1$. Dann ist $x \notin \mathcal{F}(w)$. Gib $\emptyset$ aus.
   - Es gibt ein e mit $\text{label}(e) = x_i\, x_{i+1} \ldots x_j\, y_1\, y_2 \ldots y_k$, $k \geq 0$.
     - Ist $j = n$, dann finde alle Blätter unter u und gib Menge der Blattnamen aus. $\quad (\mathcal{O}(m))$
     - Ist $j < n$ und $k > 0$, dann gib $\emptyset$ aus.
     - Ist $j < n$ und $k = 0$, dann setze $i := j + 1$, $u := v$ und gehe zu Schritt 3.

Laufzeit: $\mathcal{O}(n + m)$

Finden von Teilwörtern

Problem

Gegeben $M = \{x_1, x_2, \ldots, x_k\}$, $w \in \Sigma^*$, finde alle Vorkommen der $x_i \in \Sigma^*$ für alle $i = 1, \ldots, k$ in w.

- Sei n Gesamtlänge der Wörter aus M und $|w| = m$
- mit Suffixbäumen einfach in $\mathcal{O}(n + m)$ lösbar
- im Vergleich zu Aho-Corasick: $\mathcal{O}(n + m)$, aber
  - Aho-Corasick: $\mathcal{O}(n)$-Baum und Suche in $\mathcal{O}(m)$
  - Suffixbaum: $\mathcal{O}(m)$-Baum und Suche in $\mathcal{O}(n)$
  - Platzbedarf vs. Suchzeit
- $\mathcal{O}(n)$-Baum und $\mathcal{O}(m)$-Suche auch bei Suffixbäumen möglich

Längstes gemeinsames Teilwort

. . . zweier Wörter

Problem (2-2-LCF – „Longest Common Factor“)

Gegeben $M = \{w_1, w_2\}$ mit $w_1, w_2 \in \Sigma^*$, finde das längste gemeinsame Teilwort x von w_1 und w_2. In Zeichen: Finde $x \in \mathcal{F}(w_1) \cap \mathcal{F}(w_2)$, sodass $\forall y \in \mathcal{F}(w_1) \cap \mathcal{F}(w_2) : \ |x| \geq |y|$.

Intuitiv:

1. Erzeuge verallgemeinerten Suffixbaum $\mathcal{T}(M)$.
2. Finde Knoten v mit $|\,\text{label}(v)|$ maximal und von v muss es Weg zu w_i-Blatt und Weg zu w_j-Blatt geben, $i \neq j$. $(i = 1, j = 2)$
3. Gib label(v) aus.

Algorithmus findet auch das längste gemeinsame Teilwort *zweier* Wörter, wenn M gegeben mit $k := \text{card}(M) > 2$. („2-k-LCF“)

Längstes gemeinsames Teilwort

. . . beliebig vieler Wörter

Problem (j-k-LCF)

Gegeben $M = \{w_1, \ldots, w_k\}$ mit $w_i \in \Sigma^{m_i}\ \forall i \in \{1, \ldots, k\}$, finde das längste gemeinsame Teilwort x von j Wörtern aus M. In Zeichen: Finde $x \in \bigcap_{i \in J} \mathcal{F}(w_i)$ für ein $J \subseteq M, \text{card}(J) = j$, sodass $\forall J \subseteq M, \text{card}(J) = j\ \forall y \in \bigcap_{i \in J} \mathcal{F}(w_i) : \ |x| \geq |y|$.

Intuitiv:

1. Erzeuge verallgemeinerten Suffixbaum $\mathcal{T}(M)$.
2. Finde Knoten v mit $|\,\text{label}(v)|$ maximal und von v muss es Wege zu Blättern für j verschiedene Wörter geben.
3. Gib label(v) aus.

Längstes gemeinsames Teilwort

Realisierung und Laufzeit

Realisierung Schritt 2:

- Traversiere *bottom-up*.
- Jeder Knoten v speichert Array der Größe k. Die Komponente i des Arrays signalisiert (BOOLEsch), ob sich unter dem Knoten ein Blatt für Wort w_i befindet.
- Ein Knoten vereinigt die Arrays seiner Kindknoten mittels Disjunktion (logischem Oder).
- Sind j Werte `true`, so ist label(v) gemeinsames Teilwort.

Laufzeit: $\mathcal{O}(k \cdot (m_1 + m_2 + \cdots + m_k))$

Suffixbäume erlauben auch eine Lösung in $\mathcal{O}(m_1 + m_2 + \cdots + m_k)$. (Hier nicht besprochen.)

Anwendungen in der Bioinformatik

Bioinformatik?

- Molekularbiologie, Genetik, Phylogenetik
- Analyse von Sequenzen, vorallem:
  - Wörter über $\Sigma_{\text{DNA}} = \{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$ bzw. $\Sigma_{\text{RNA}} = \{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{U}\}$.
    - DNA und RNA bestehen aus sog. *Nukleotiden*
    - ein Nukleotid hat eine Nukleotid-Basis, nämlich *Adenin*, *Guanin*, *Cytosin*, *Thymin* (nur DNA) oder *Uracil* (nur RNA).
  - Wörter über $\Sigma_{\text{Amino}} = \{\mathtt{A}, \mathtt{C}, \mathtt{D}, \mathtt{E}, \mathtt{F}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{K}, \mathtt{L}, \mathtt{M}, \mathtt{N}, \mathtt{P}, \mathtt{Q}, \mathtt{R}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{W}, \mathtt{Y}\}$
    - drei Basen kodieren Aminosäure und es gibt 20 *kanonische Aminosäuren*
    - Aminosäuresequenzen kodieren Proteine
- öffentliche Sequenz-Datenbanken (z. B. NCBI GENBANK, EMBL, DDBJ, SWISSPROT)

Szenario 1

mehrfache Suche nach Teilwörtern

Folgende Situation ist häufig anzutreffen:

- Sequenz w ist bekannt und fest
- es kommen nach und nach Anfragen, ob x_i ein Teilwort in w ist, $i = 1, 2, \ldots$. Gib alle Vorkommen zurück.

Laufzeit:

- *Vorverarbeitung*: einmalig $\mathcal{O}(|w|)$ zum Erstellen von $\mathcal{T}(w)$
- *Anfrageverarbeitung*: jeweils $\mathcal{O}(|x_i| + g)$, g ist Anzahl Vorkommen von x_i in w

Szenario 2

Einfügen in Datenbank

Ähnlich ist folgende Situation:

- Datenbank enthält festes $M = \{w_1, \ldots, w_k\}$
- neue DNA-Sequenz x soll in die Datenbank eingefügt werden

Vorgehen bei Vorliegen von x und $\mathcal{T}(M)$:

- überprüfe, ob x in $\mathcal{T}(M)$ ist
- gib alle Wörter $w \in M$ mit $x \in \mathcal{F}(w)$ zurück
- gibt es so ein w nicht, dann füge w in $\mathcal{T}(M)$ ein

Laufzeit:

- *Vorverarbeitung:* einmalig $\mathcal{O}(|w_1| + |w_2| + \cdots + |w_k|)$
- *Überprüfen:* jeweils $\mathcal{O}(|x|)$
- *Rückgabe:* jeweils $\mathcal{O}(\text{Anzahl der Vorkommen von } x)$
- *Einfügen:* jeweils $\mathcal{O}(|x|)$

DNA-Verschmutzung

- falsche DNA kann sich mit betrachteter DNA vermischen
- z. B. bei Klonierung oder anderen Verfahren im Labor
- Forschung auf verschmutzten Sequenzen nutzlos

→ Verschmutzung erkennen!

- Gegeben: DNA-Sequenz w und Menge $M = \{x_1, x_2, \ldots, x_k\}$ von DNA-Sequenzen, die w verschmutzt haben könnten.
- Gesucht: Finde alle gemeinsamen Teilwörter von w und einem x_i, $i \in \{1, \ldots, k\}$ der Länge $\geq l$.

→ Wende modifiziertes 2-k-LCF an, das nach Knoten v mit $|\,\text{label}(v)| \geq l$ sucht, statt nach maximalem $|\,\text{label}(v)|$.

Artmerkmale

- Vergleich von DNA-Sequenzen des Erbguts verschiedener Individuen einer Spezies oder phylogenetisch verwandter Spezien
- lange, gleiche Teilwörter geben Hinweis auf typische Merkmale der Spezies (auf Ebene der DNA-Basensequenzen)
- ändernde Teilwörter ändern vermutlich weniger signifikante Merkmale
- typische Anwendung für j-k-LCF

Schlusswort

- Suffixbäume bieten allgemeinen Ansatz für effiziente Behandlung vieler Wort- bzw. String-Probleme (→ Probleme der Bioinformatik; siehe Folgeseminare zu Suffixbäumen)
- weitere Anwendungen sind bspw.
  - längste Wiederholungen in Sequenzen finden
  - Suffix-Präfix-Matching
  - LEMPEL-ZIV-Datenkompression
  - „Longest Common Ancestor"-Probleme
  - ...

Referenzen

Verwendete Literatur

[Gus97] Dan Gusfield.
Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.
Cambridge University Press, New York, NY, USA, 1997.

[CR94] Maxime Crochemore and Wojciech Rytter.
Text Algorithms.
Oxford University Press, Oxford-New York, 1994.

[CR02] Maxime Crochemore and Wojciech Rytter.
Jewels of Stringology.
World Scientific Publishing, Singapore, 2002.

Referenzen

Originalpapiere

[AIL+88] Apostolico, Iliopoulos, Landau, Schieber, and Vishkin.
Parallel construction of a suffix tree with applications.
Algorithmica, 3, 1988.

[McC76] Edward M. McCreight.
A space-economical suffix tree construction algorithm.
Journal of the ACM, 23(2):262–272, April 1976.

[Ukk95] Esko Ukkonen.
On-line construction of suffix trees.
Algorithmica, 14(3):249–260, September 1995.

[Wei73] Peter Weiner.
Linear pattern matching algorithms.
In *Conference Record, IEEE 14th Annual Symposium on Switching and Automata Theory*, pages 1–11, 1973.