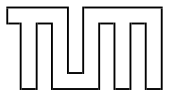
Algorithmische Bioinformatik 1

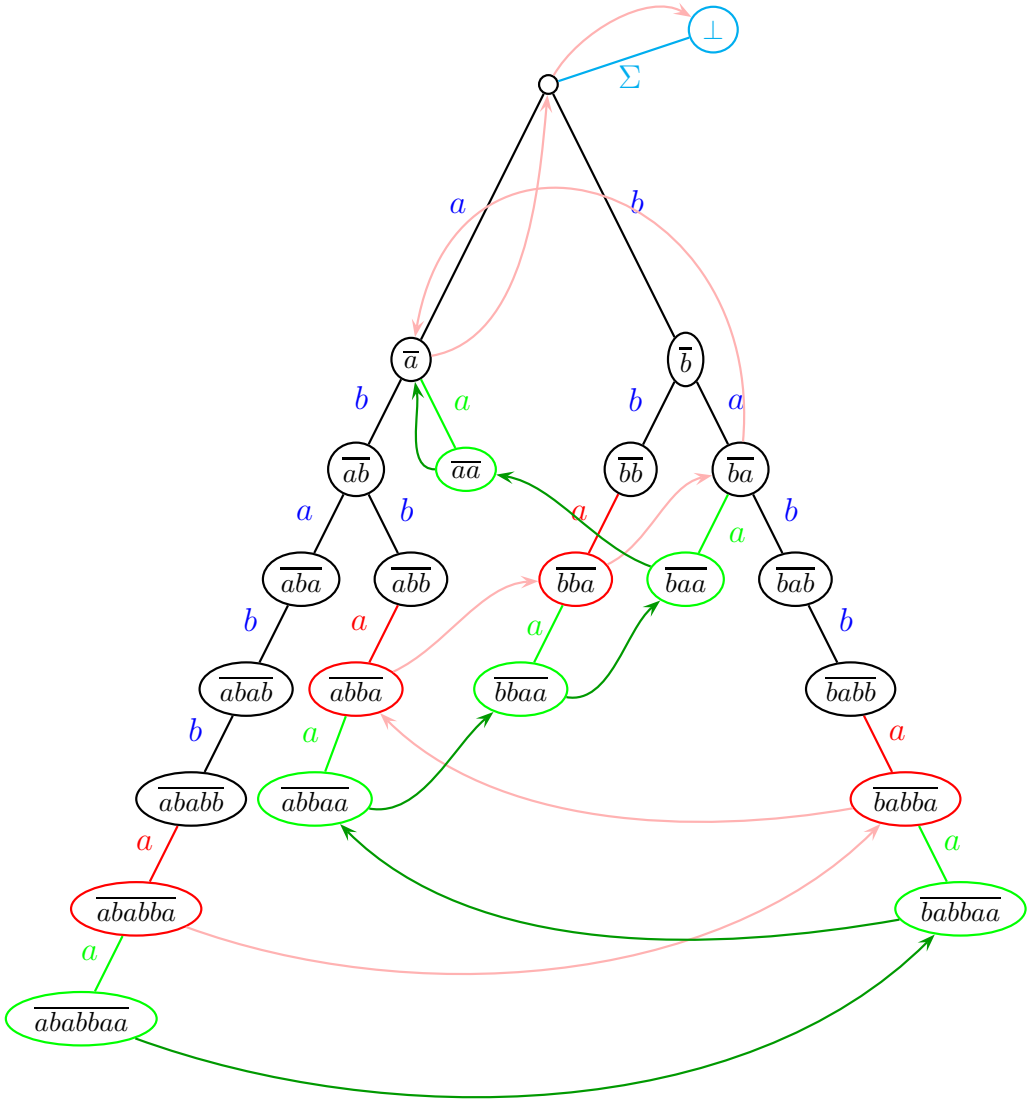
Dr. Hanjo Täubig

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

Sommersemester 2009



Übersicht



Beispiel: Suffix-Trie für *ababbaa* mit Suffix-Links

Suffix-Links

Definition ($\overline{w}$)

Sei $t \in \Sigma^*$ und sei w ein Teilwort von t.

Im Suffix-Trie für t bezeichnen wir den Knoten, der über die Kantenbeschriftung w erreichbar ist, mit $\overline{w}$.

Definition (Suffix-Link)

Sei $t \in \Sigma^*$ und sei aw mit $a \in \Sigma$ ein Teilwort von t.

Dann soll der Suffix-Link von $\overline{aw}$ zum Knoten $\overline{w}$ zeigen, kurz: $suffix\_link(\overline{aw}) = \overline{w}$.

Virtuelle Wurzel

- Damit auch die Wurzel einen Suffix-Link besitzt, ergänzen wir den Suffix-Trie um eine virtuelle Wurzel $\perp$ und setzen $\mathit{suffix\_link}(\mathit{root}) = \perp$.

- Diese virtuelle Wurzel $\perp$ besitzt für jedes Zeichen $a \in \Sigma$ eine Kante zur eigentlichen Wurzel.

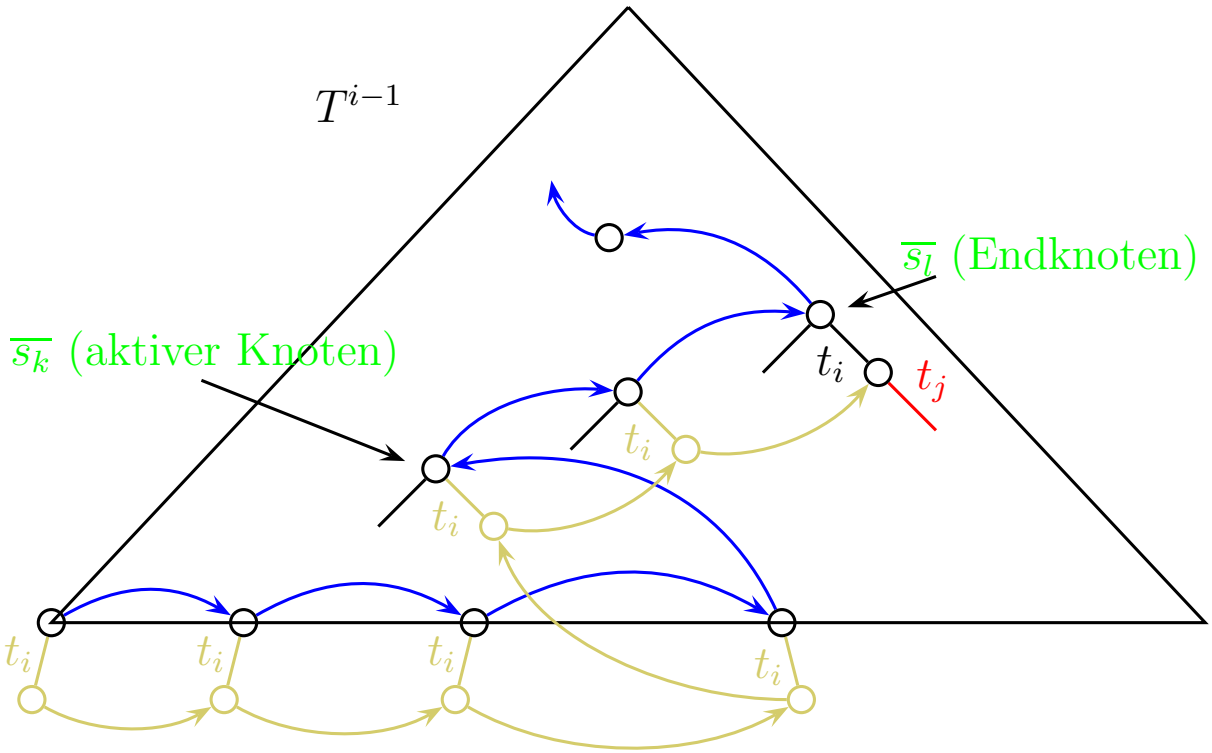
- Dies wird aus algorithmischer Sicht hilfreich sein, da für diesen Knoten dann keine Suffix-Links benötigt werden.

Konstruktion

- Idee der Konstruktion des Suffix-Tries T^i aus T^{i-1}:
  - alle Knoten ablaufen, die den Suffixen von $t_1 \cdots t_{i-1}$ im Suffix-Trie T^{i-1} entsprechen, und
  - an diese Knoten ein neues Kind über eine Kante mit Kantenlabel t_i anhängen, sofern dieses noch nicht existiert

- Wie können wir alle Suffixe von $t_1 \cdots t_{i-1}$ effizient ablaufen?
  - Wir beginnen hierzu beim längsten Suffix ($t_1 \cdots t_{i-1}$ selbst), und
  - erreichen das nächst kürzere Suffix jeweils über den zugehörigen Suffix-Link.

Konstruktion



Skizze: Konstruktion eines Suffix-Tries

Algorithmus

Konstruktionsalgorithmus für Suffix-Tries

- *curr_node*
  aktuell betrachteter Knoten, an den ein neues Kind über eine Kante mit Kantenlabel t_i angehängt werden soll
- *new_node*
  das jeweils neu angehängte Blatt
- *prev_node*
  das zuletzt davor angehängte Blatt (außer zu Beginn jeder Phase, wo *prev_node* nicht wirklich sinnvoll definiert ist)
- *some_node*
  (nur dann definiert, wenn die while-Schleife abbricht)
  das Kind des aktuell betrachteten Knotens *curr_node*, von dem bereits eine Kante mit Kantenlabel t_i ausgeht

Algorithmus

Algorithmus 11 : BuildSuffixTrie(char $t[]$, int n)

$T := (\{root, \bot\}, \{\bot \xrightarrow{x} root : x \in \Sigma\})$;
$suffix\_link(root) := \bot$;
$longest\_suffix := prev\_node := root$;
for $(i := 1;\ i \leq n;\ i{+}{+})$ **do**
    $curr\_node := longest\_suffix$;
    while $(curr\_node \xrightarrow{t_i} some\_node\ does\ not\ exist)$ **do**
        Add $curr\_node \xrightarrow{t_i} new\_node$ to T;
        if $(curr\_node = longest\_suffix)$ **then**
            $longest\_suffix := new\_node$;
        else
            $suffix\_link(prev\_node) := new\_node$;
        $prev\_node := new\_node$;
        $curr\_node := suffix\_link(curr\_node)$;
    $suffix\_link(prev\_node) := some\_node$;

Laufzeitanalyse für die Konstruktion von T^n

- Bei der naiven Methode (ohne Suffix-Links) muss zur Verlängerung eines Suffixes das ganze Suffix durchlaufen werden.
- Daher sind zum Anhängen von t_i an $t_j \cdots t_{i-1}$ wiederum $O(i-j+1)$ Operationen erforderlich:

$$\sum_{i=1}^{n} \underbrace{\sum_{j=1}^{i} O(i-j+1)}_{\text{Zeit f. } T^{i-1} \to T^i} = O\left(\sum_{i=1}^{n}\sum_{j=1}^{i} j\right) = O\left(\sum_{i=1}^{n} i^2\right) = O(n^3)$$

- Dies ist also sogar schlechter als unser erster Algorithmus.

Laufzeitanalyse für die Konstruktion von T^n

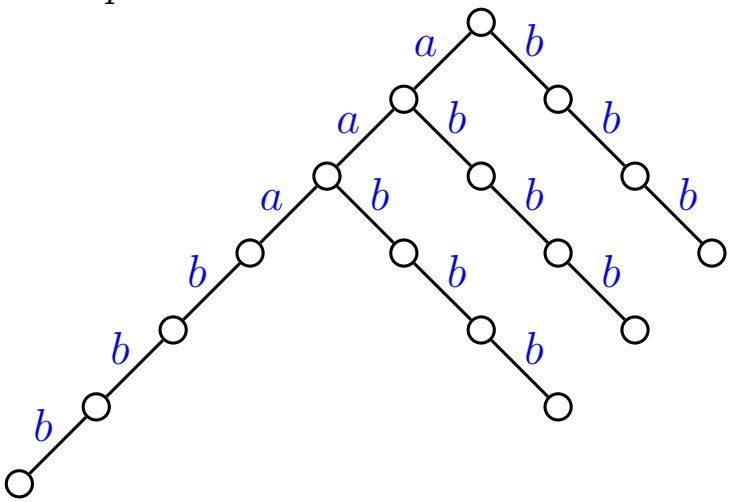
- Durch die Suffix-Links benötigt das Verlängern jedes Suffixes $t_j \cdots t_{i-1}$ um t_i nur noch $O(1)$ Operationen:

$$\sum_{i=1}^{n} \sum_{j=1}^{i} O(1) = \sum_{i=1}^{n} O(i) = O(n^2)$$

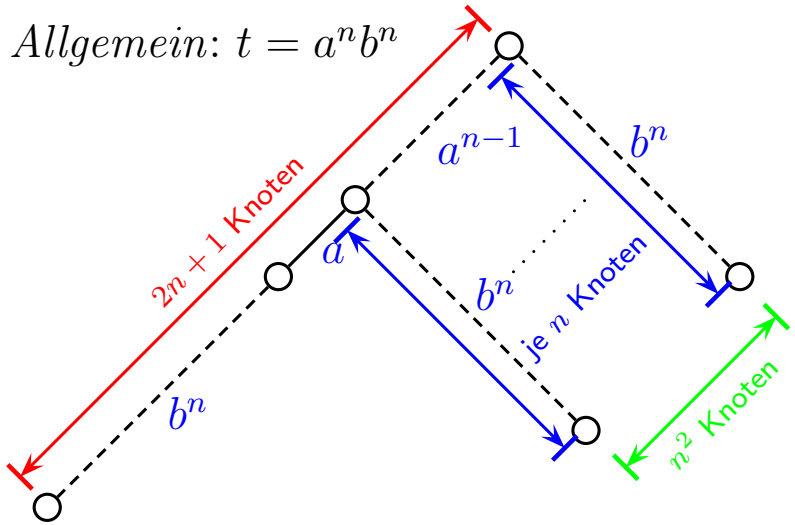
- Die Laufzeit ist noch nicht besser als beim ersten Algorithmus, aber wir werden die Idee später zur Konstruktion von Suffix-Bäumen wiederverwenden.

Größe von Suffix Tries

Beispiel: $t = aaabbb$



Allgemein: $t = a^n b^n$



Beispiel: Potentielle Größe von Suffix-Tries (für $a^n b^n$)

- Der Suffix-Trie für ein Wort t der Form $t = a^n b^n$ hat damit insgesamt

$$(2n+1) + n^2 = (n+1)^2 = \Theta(n^2)$$

viele Knoten, was nicht optimal ist.

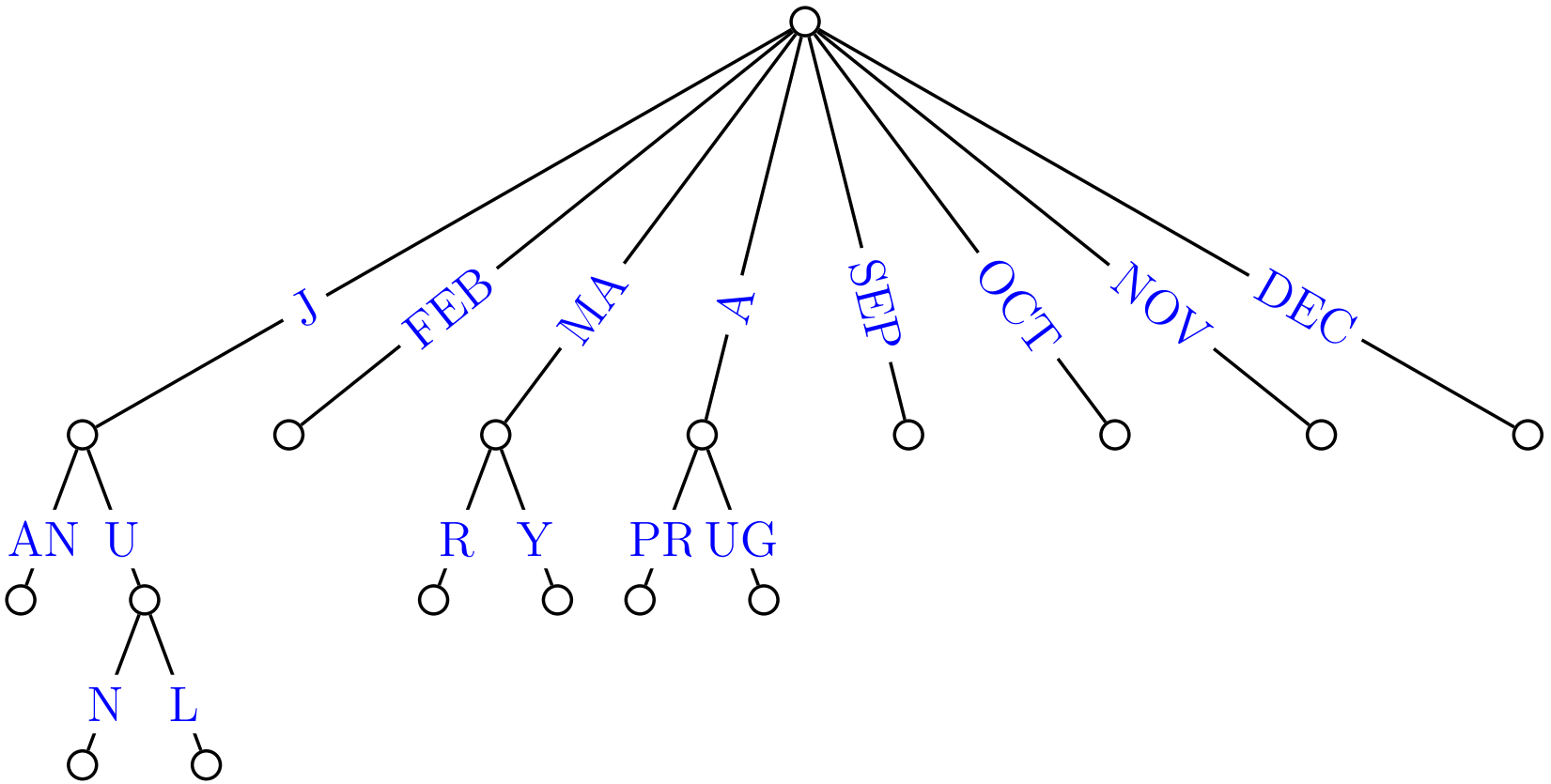
Übersicht

PATRICIA-Tries

- Idee: möglichst viele Kanten sinnvoll zusammenfassen, um den Trie zu kompaktifizieren
- Alle Pfade, die bis auf den Endknoten nur aus Knoten mit jeweils einem Kind bestehen, können zu einer Kante zusammengefasst werden.
- Diese kompaktifizierten Tries werden PATRICIA-Tries genannt (für *Practical Algorithm To Retrieve Information Coded In Alphanumeric*).

PATRICIA-Tries

PATRICIA-Trie für die Abkürzungen der Monatsnamen



Suffix-Bäume

- So kompaktifizierte Suffix-Tries werden Suffix-Bäume (engl. suffix trees) genannt.

Definition

Sei $t = t_1 \cdots t_n \in \Sigma^*$.

Ein Suffix-Baum ist ein Patricia-Trie für alle Suffixe von t, also für $S = \{t_i \cdots t_n \ : \ i \in [1 : n+1]\}$.

In der Literatur wird der hier definierte Suffix-Baum oft auch als kompakter Suffix-Baum bezeichnet.

Suffix-Bäume

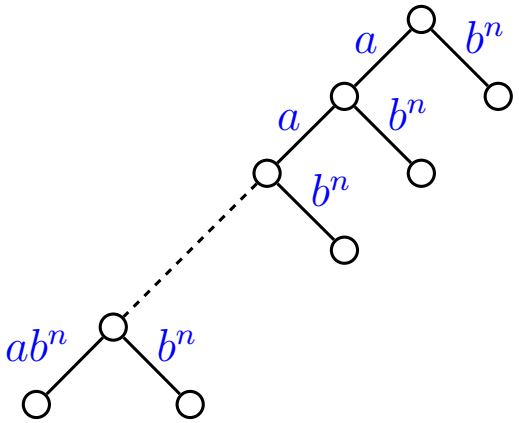
Das folgende Lemma ist aus der Informatik bzw. Graphentheorie wohlbekannt.

Lemma

Ein gewurzelter Baum, in dem jeder innere Knoten mindestens zwei Kinder hat, besitzt mehr Blätter als innere Knoten.

Da ein Suffix-Baum für $t \in \Sigma^n$ höchstens so viele Blätter wie Suffixe ungleich ϵ haben kann, also maximal n, besteht ein Suffix-Baum für t aus weniger als $2n$ Knoten.

Suffix-Bäume

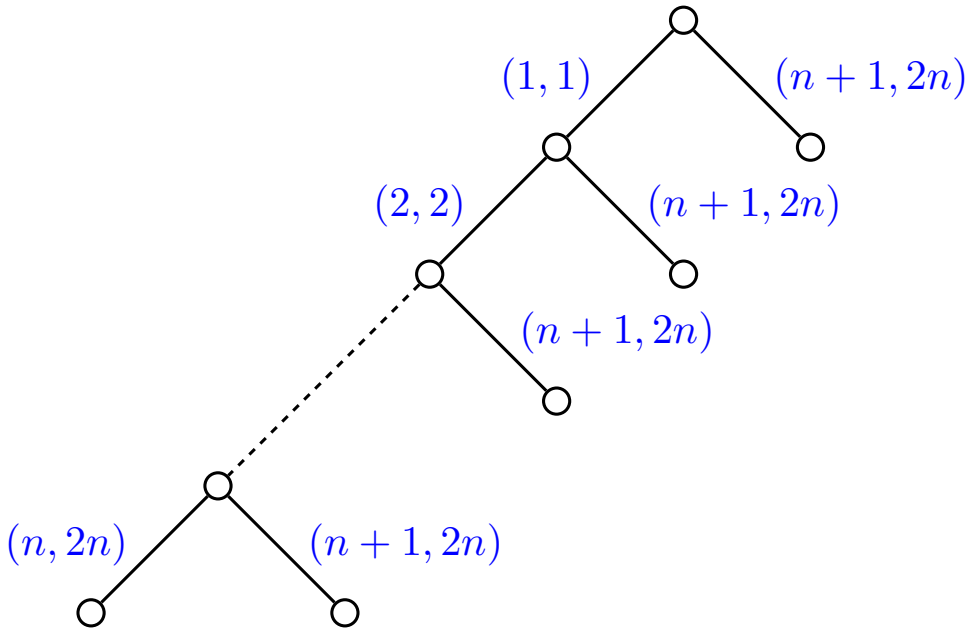


Beispiel: Kompaktifizierter Trie für $t = a^n b^n$

Suffix-Bäume

- Durch das Zusammenfassen mehrerer Kanten wurde die Anzahl der Knoten reduziert.
- Aber: dafür sind aber die Kantenlabels länger geworden.
- Deswegen werden als Labels der zusammengefassten Kanten nicht die entsprechenden Teilwörter verwendet, sondern die (Anfangs- und End-) Position, an der das Teilwort im Gesamtwort auftritt.
- Für das Teilwort $t_i \cdots t_j$ wird die Referenz (i, j) verwendet.

Suffix-Bäume



Beispiel: Echter Suffix-Baum für $t = a^n b^n$

Suffix-Bäume

- Sei $t = t_1 \cdots t_n$ und $(\overline{t_1 \cdots t_j}, \overline{t_1 \cdots t_i})$ eine Kante des Baumes, dann verwenden wir als Label $(j+1, i)$ statt $t_{j+1} \cdots t_i$.

- Damit hat der Suffix-Baum nicht nur $O(|t|)$ viele Knoten, sondern er benötigt auch für die Verwaltung der Kantenlabels nur linear viel Platz (anstatt $O(n^2)$).

Ukkonens Algorithmus

- $\hat{T}^i$: Suffix-Baum für $t_1 \cdots t_i$
- Wir wollen nun den Suffix-Baum $\hat{T}^i$ aus dem Suffix-Baum $\hat{T}^{i-1}$ aufbauen.
- Diese Konstruktion kann man aus der Konstruktion des Suffix-Tries T^i aus dem Suffix-Trie T^{i-1} erhalten.

Ukkonens Algorithmus

Definition

Sei

- $t = t_1 \cdots t_n \in \Sigma^*$
- $i \in [1 : n]$
- $s_j = t_j \cdots t_{i-1}$ für $j \in [1 : i]$
- $\overline{s}_j$ der zugehörige Knoten im Suffix-Trie T^{i-1} für $t_1 \cdots t_{i-1}$
- $\overline{s}_i = \overline{\epsilon} = root$, und $\overline{s}_{i+1} = \perp$, wobei $\perp$ die virtuelle Wurzel ist, von der für jedes Zeichen $a \in \Sigma$ eine Kante zur eigentlichen Wurzel führt.

Ukkonens Algorithmus

Definition

Sei ℓ der größte Index, so dass für alle $j < \ell$ im Suffix-Trie T^{i-1} eine neue Kante für t_i an den Knoten $\overline{s}_j$ angehängt werden muss. Der Knoten $\overline{s}_\ell$ im Suffix-Trie T^{i-1} heißt Endknoten.

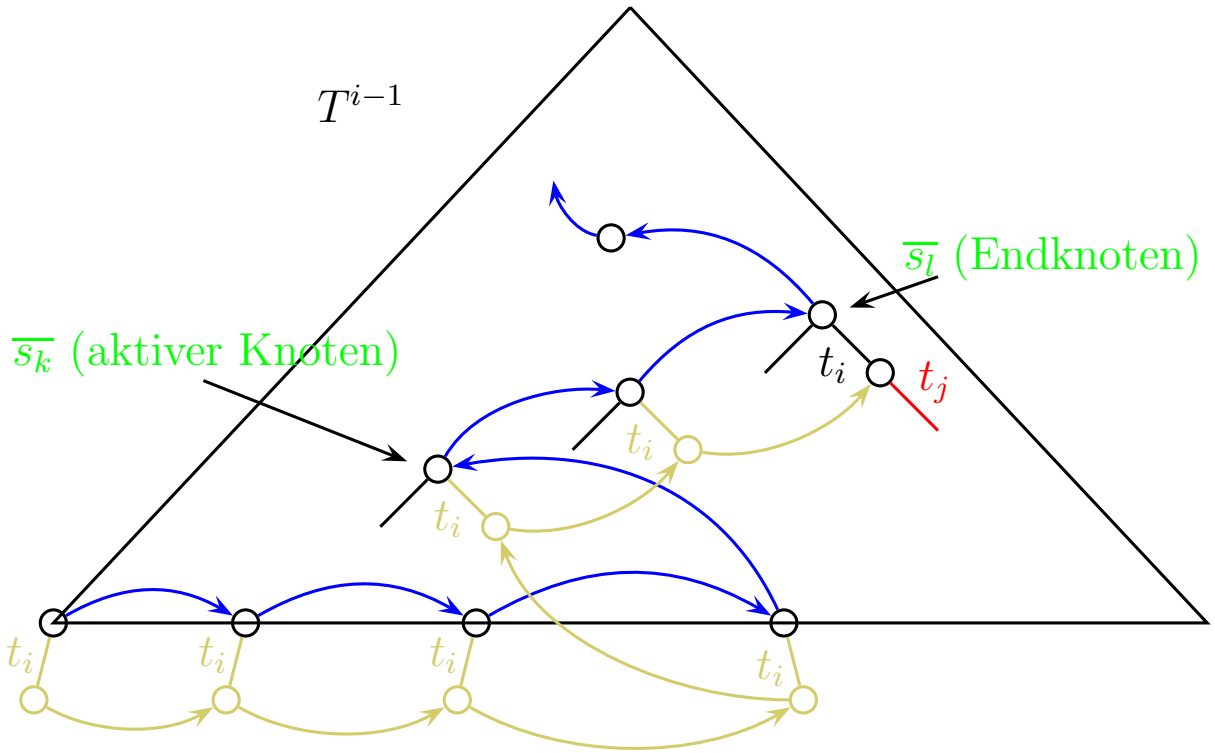
(Anders formuliert: alle längeren Suffixe $s_j = t_j \cdots t_{i-1}$ müssen verlängert werden, aber für $\overline{s}_\ell$ existiert die gesuchte Erweiterung um t_i schon.)

Sei k der größte Index, so dass für alle $j < k$ $\overline{s}_j$ ein Blatt ist. Der Knoten $\overline{s}_k$ heißt aktiver Knoten.

(Anders formuliert: alle längeren Suffixe entsprechen Blättern im aktuellen Baum und s_k ist das längste Suffix, dass einem inneren Knoten entspricht.)

Ukkonens Algorithmus

- Da $\overline{s}_1$ immer ein Blatt ist, gilt $k > 1$,
- und da $\overline{s}_{i+1} = \perp$ immer eine Kante mit dem Label t_i besitzt, gilt $\ell \leq i + 1$.
- Weiterhin sieht man, dass nach Definition $k \leq \ell$ gilt.



Explizite und implizite Knoten

Definition

Sei

- $t = t_1 \cdots t_n \in \Sigma^*$,
- $t' = t_1 \cdots t_i$ für ein $i \in [1 : n]$,
- w ein Teilwort von t'.

Dann heißt ein Knoten $\overline{w}$ des Suffix-Tries T^i für t' explizit, wenn es auch im Suffix-Baum $\hat{T}^i$ für t' einen Knoten gibt, der über die Zeichenreihe w erreichbar ist. Andernfalls heißt er implizit.

Offene Referenzen

- Die eigentliche Arbeit beim Aufbau des Suffix-Tries ist zwischen dem aktiven Knoten und dem Endknoten zu verrichten.
- Bei allen Knoten „vor“ dem aktiven Knoten, also bei allen bisherigen Blättern, muss einfach ein Kind mit dem Kantenlabel t_i eingefügt werden.
- Im Suffix-Baum bedeutet dies, dass an das betreffende Kantenlabel einfach das Zeichen t_i angehängt wird.
- Um sich nun diese Arbeit beim Aufbau des Suffix-Baumes zu sparen, werden statt der Teilwörter als Kantenlabels so genannte (offene) Referenzen benutzt.

Referenz

Definition

Sei

- $t = t_1 \cdots t_n \in \Sigma^*$,
- $i \in [1 : n]$,
- w ein Teilwort von $t_1 \cdots t_i$ mit $w = uv$, wobei $u = t_j \cdots t_{k-1}$ und $v = t_k \cdots t_p$.

Ist $s = \overline{u}$ im Suffix-Baum $\hat{T}^i$ realisiert (also explizit in T^i), dann heißt $(s, (k, p))$ eine Referenz für $\overline{w}$ (im Suffix-Trie T^i).

Wir nehmen an, dass $t_j \cdots t_p = w$ das erste Vorkommen von w in t ist (w könnte ja mehrmals als Teilwort von t auftreten).

Beispiele, Kanonische Referenzen

Beispiele für Referenzen in $t = ababbaa$:

1. $w = \overset{babb}{2345} \hat{=} (\overline{b}, (3,5)) \hat{=} (\overline{ba}, (4,5)) \hat{=} (\overline{\epsilon}, (2,5))$.
2. $w = \overset{bbaa}{4567} \hat{=} (\overline{b}, (5,7)) \hat{=} (\overline{bbaa}, (8,7))$.
3. $w = ba \hat{=} (\overline{b}, (3,3))[\hat{=} (\overline{b}, (6,6))]$.

Definition

Eine Referenz $(s, (k,p))$ heißt kanonisch, wenn $p - k$ minimal ist.