Algorithmentheorie

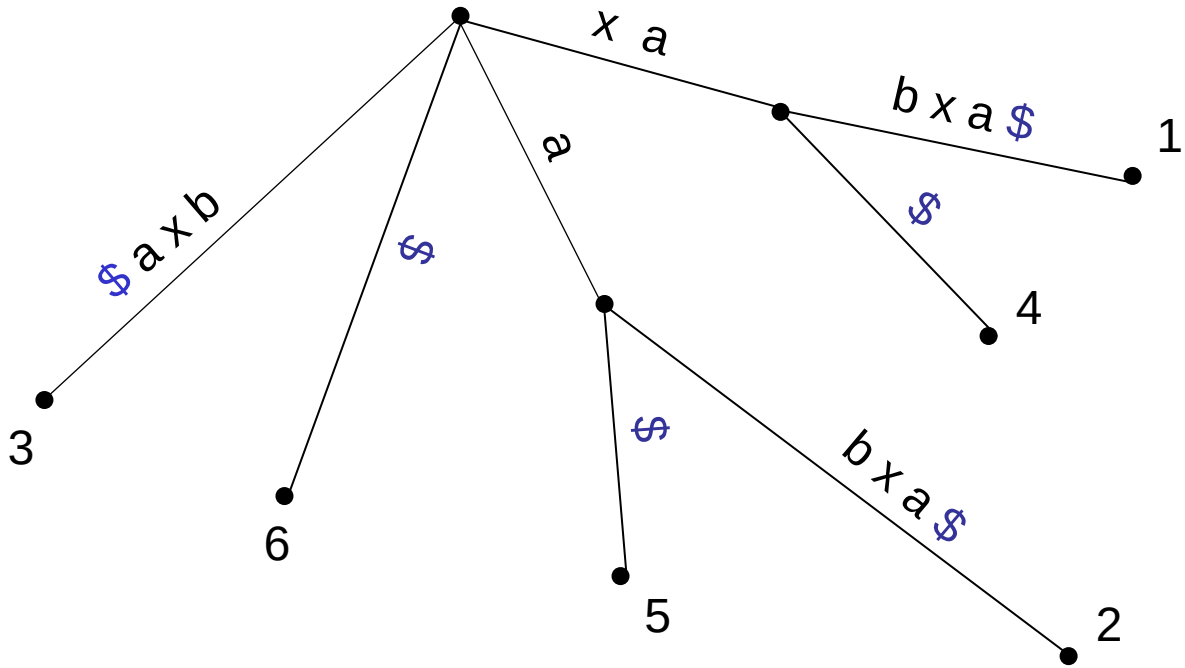
15 – Suchen in Texten (2)

Konstruktion von Suffix-Bäumen

Prof. Dr. S. Albers

Suffix-Baum

t = x a b x a $
1 2 3 4 5 6



Ukkonens Algorithmus: Impliziter Suffix-Baum

Definition: Ein *impliziter Suffix-Baum* ist der Baum, den man aus dem Suffix-Baum für *t*$ enthält, indem man

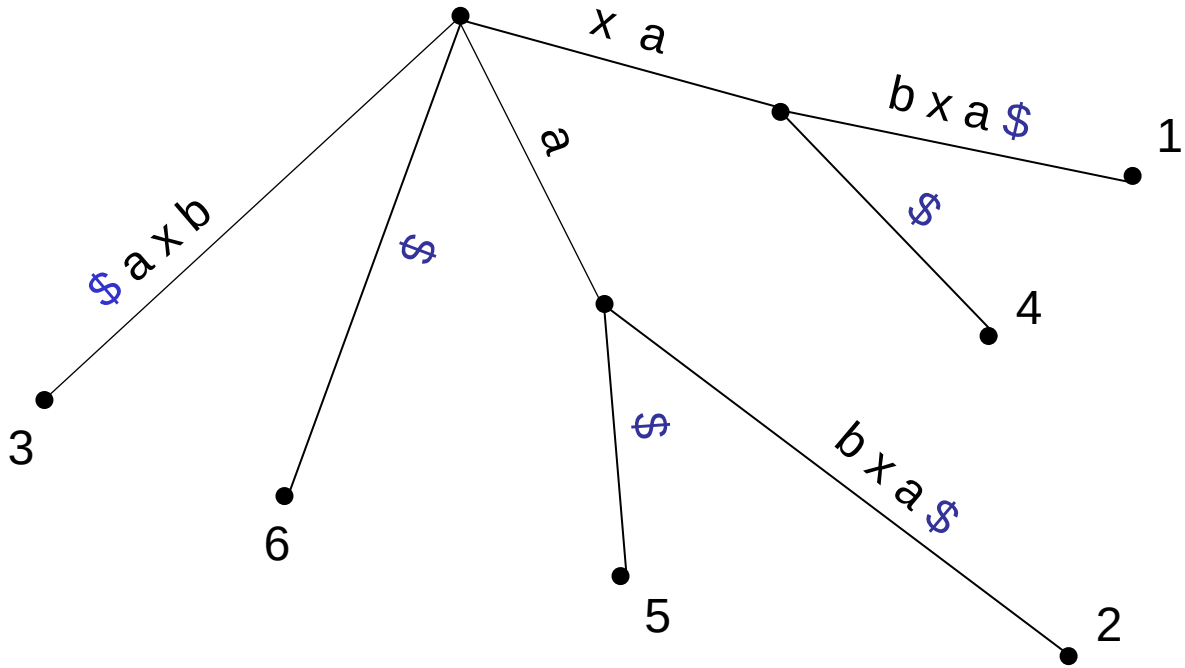
(1) $ von den Markierungen der Kanten entfernt,

(2) Kanten ohne Markierung entfernt,

(3) Knoten mit nur einem Kind entfernt.

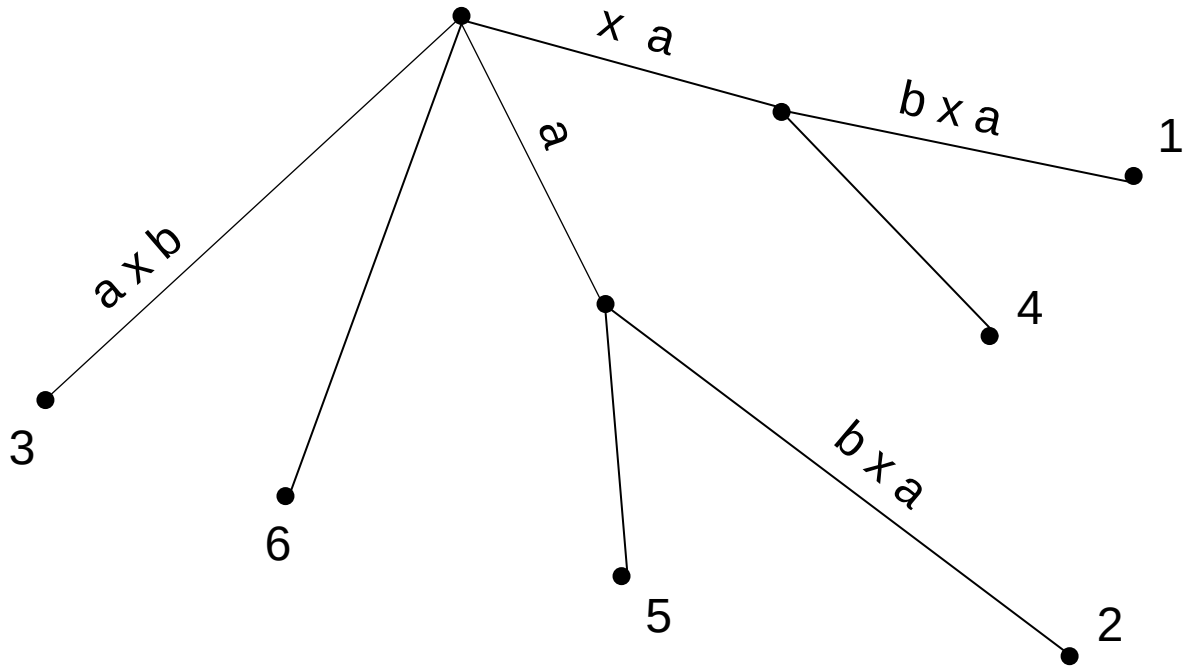
Ukkonens Algorithmus: Impliziter Suffix-Baum

t = x a b x a $
1 2 3 4 5 6



Ukkonens Algorithmus: Impliziter Suffix-Baum

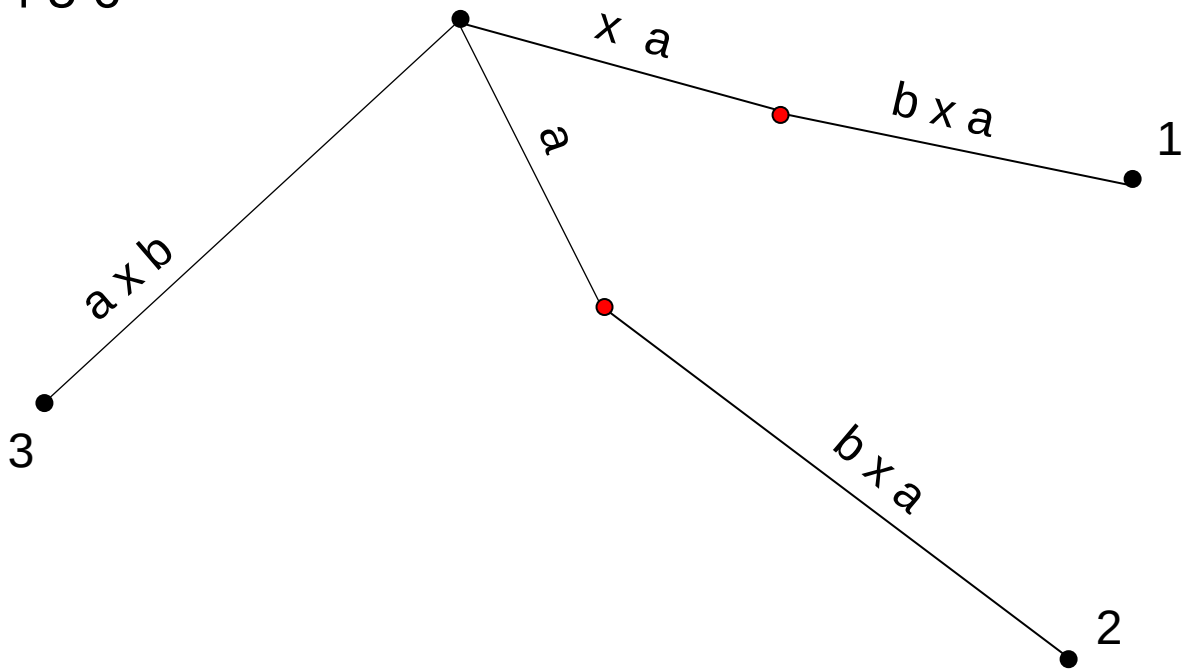
(1) Entfernen der Kantenmarkierung $ für *t* = x a b x a $



Ukkonens Algorithmus: Impliziter Suffix-Baum

(2) nicht markierten Kanten entfernen

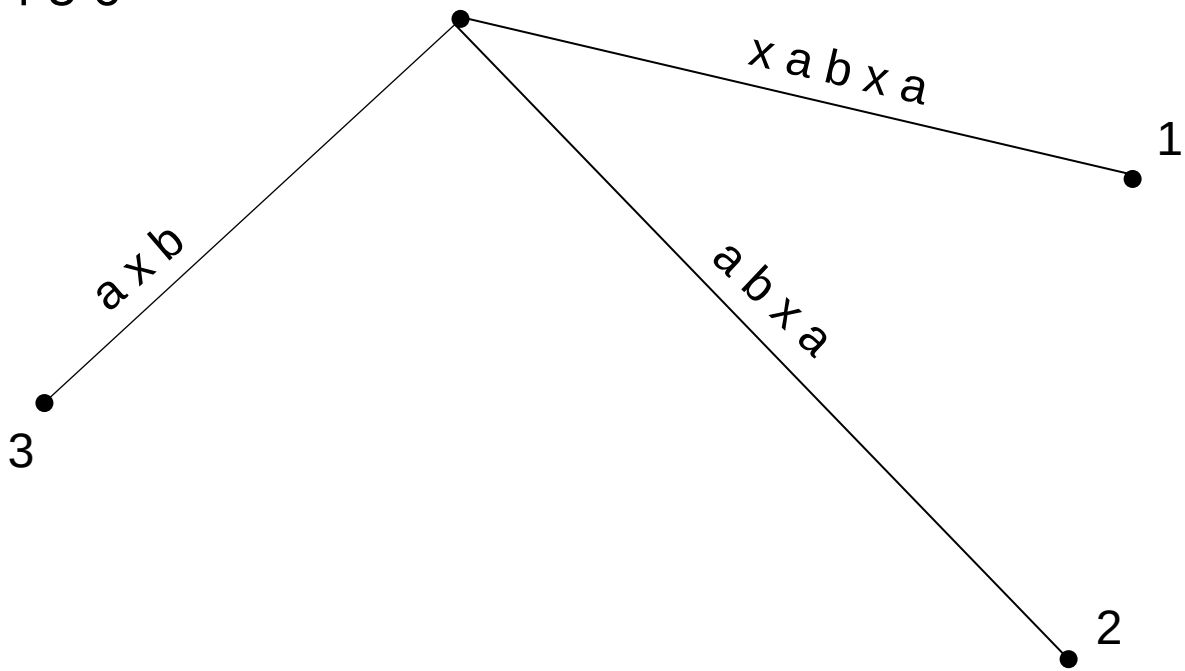
t = x a b x a $
1 2 3 4 5 6



Ukkonens Algorithmus: Impliziter Suffix-Baum

(3) Knoten mit nur einem Kind entfernen

t = x a b x a $
1 2 3 4 5 6



Ukkonens Algorithmus

Sei $t = t_1 t_2 t_3 \ldots t_m$

Ukk arbeitet online: Der Suffix-Baum $ST(t)$ wird schrittweise durch Konstruktion einer Reihe von impliziten Suffix-Bäumen für alle Präfixe von t konstruiert:

$$ST(\varepsilon), ST(t_1), ST(t_1 t_2), \ldots, ST(t_1 t_2 \ldots t_m)$$

$ST(\varepsilon)$ ist der leere implizite Suffix-Baum.

Er besteht nur aus der Wurzel.

Ukkonens Algorithmus

Die Methode wird *online* genannt, weil in jedem Schritt der implizite Suffix-Baum für ein Anfangsstück von *t* konstruiert wird, ohne den Rest des Inputstrings zu kennen.

Der Algorithmus arbeitet also inkrementell, da er den Inputstring zeichenweise von links nach rechts liest.

Ukkonens Algorithmus

Inkrementelle Konstruktion des impliziten Suffixbaums:

Induktionsanfang: $ST(\varepsilon)$ besteht nur aus der Wurzel.

Induktionsschritt: Aus $ST(t_1 \ldots t_i)$ wird $ST(t_1 \ldots t_i t_{i+1})$, für alle $i < m$

Sei T_i der implizite Suffix-Baum für $t[1...i]$

- Zuerst konstruiert man T_1 : Der Baum hat nur eine Kante, die mit dem Zeichen t_1 markiert ist.
- Die in Phase $i+1$ zu lösende Aufgabe ist, T_{i+1} aus T_i zu konstruieren.
- Iteriere für Phasen $i=1$ bis $m-1$

Ukkonens Algorithmus

Pseudo-Code Formulierung von Ukk:

Konstruiere Baum T_1

for *i* = 1 **to** *m*–1 **do**

begin {Phase *i*+1}

  for *j* = 1 **to** *i* +1 **do**

    begin {Erweiterung *j*}

      Finde im aktuellen Baum das Ende des Pfades von der Wurzel, der mit *t*[*j* ... *i*] markiert ist.

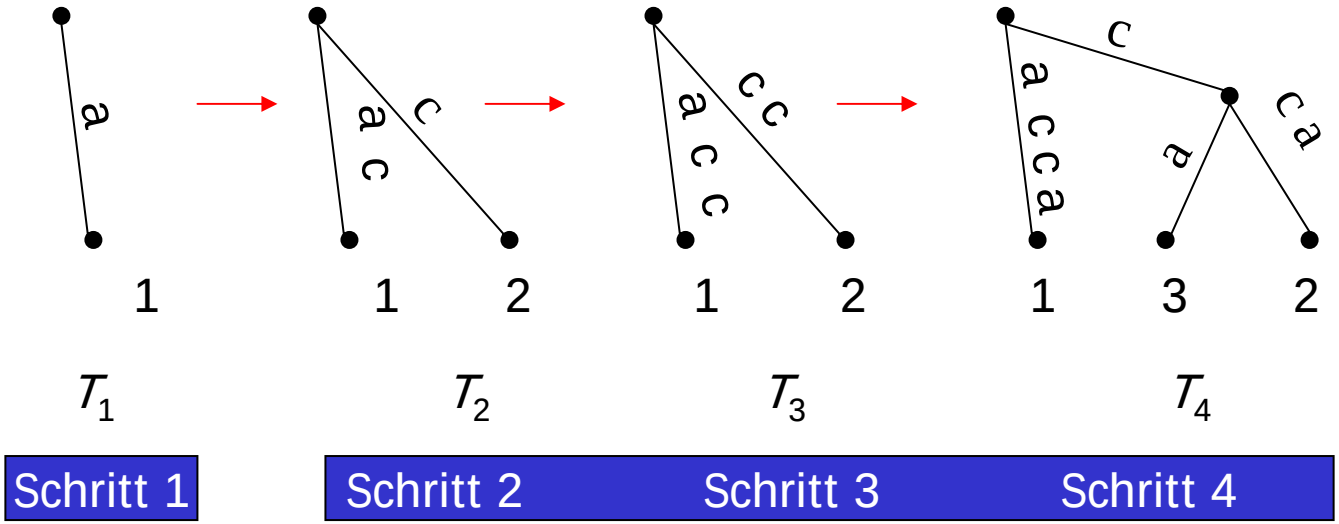
      Falls erforderlich, erweitere ihn durch Hinzufügen des Zeichens *t*[*i*+1], damit gewährleistet ist, dass der String *t*[*j*...*i*+1] im Baum ist.

      end;

end;

Ukkonens Algorithmus

t = a c c a $



Ukkonens Algorithmus

- Jede Erweiterung *j* wird so durchgeführt, dass man das Ende des mit *t*[*j*...*i*] gekennzeichneten Pfades von der Wurzel findet und den Pfad durch das Zeichen *t*[*i*+1] erweitert.

- In Phase *i*+1 wird zuerst der String *t*[1...*i*+1] in den Baum eingefügt, gefolgt von den Strings *t*[2...*i*+1] , *t*[3...*i*+1] ,.... (entsprechend den Erweiterungen 1,2,3,.... in Phase *i*+1).

- Erweiterung *i*+1 in Phase *i*+1 fügt das einzelne Zeichen *t*[*i*+1] in den Baum ein (es sei denn es ist schon vorhanden).

Ukk: Suffix-Erweiterungsregeln

Der Erweiterungsschritt *j* (in Phase *i*+1) wird nach einer der folgenden Regeln durchgeführt:

Regel 1: Wenn *t* [*j*...*i*] in einem Blatt endet, wird *t* [*i*+1] an die Markierung der zum Blatt führenden Kante angehängt.

Regel 2: Falls kein Pfad vom Ende von *t* [*j*...*i*] mit dem Zeichen *t* [*i*+1] anfängt, wird eine neue Kante zu einem neuen Blatt erzeugt, die mit dem Zeichen *t* [*i*+1] markiert wird.

(Das ist die einzige Erweiterung, die die Anzahl der Blätter im Baum erhöht! Das Blatt repräsentiert bei Position *j* beginnende Suffixe.)

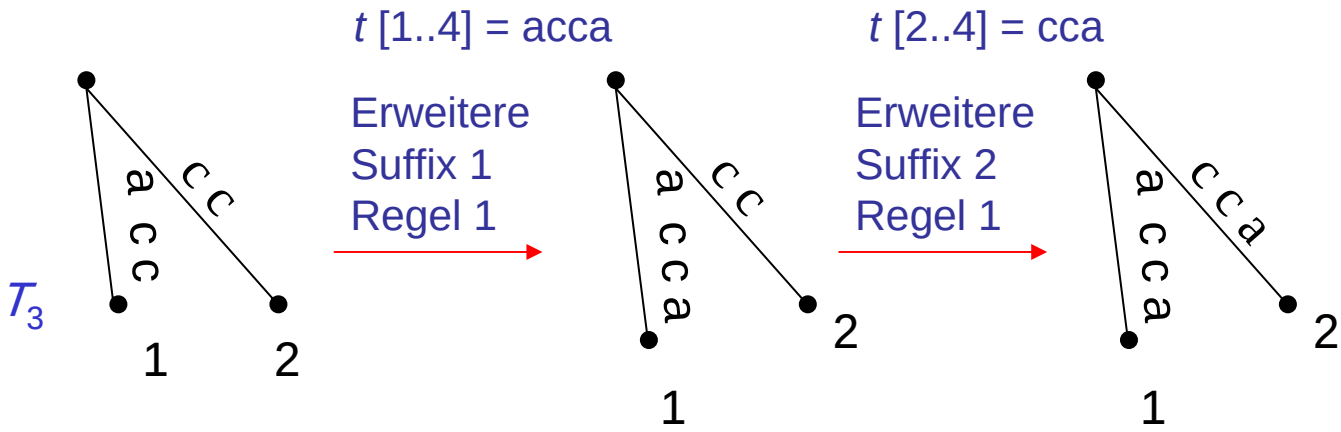
Regel 3: Falls ein mit dem Zeichen *t* [*i*+1] markierter Pfad am Ende von *t* [*j* ...*i*] beginnt, macht man nichts (denn dann tritt *t* [*j*…*i* +1] bereits im Baum auf).

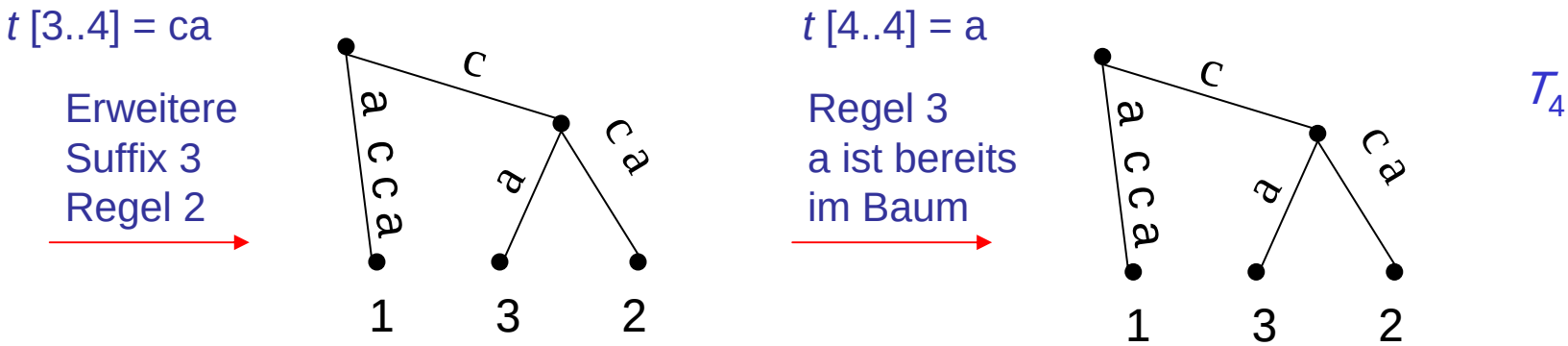
Ukkonens Algorithmus

t = a c c a $

t [1...3] = acc

t [1...4] = acca





Ukkonens Algorithmus

Bei Durchführung von Phase i+1 (der Vorgang, durch den T_{i+1} aus T_i konstruiert wird) kann man beobachten:

(1) Sobald in Erweiterung j erstmals Regel 3 angewandt wird, muss der Pfad, der mit t [j...i] in T_i markiert ist, eine Fortsetzung mit Zeichen t [i+1] haben. Dann muss auch für jedes $j' \geq j$ der Pfad, der mit t [j' ... i] markiert ist, ebenfalls eine Fortsetzung mit Zeichen t [i+1] haben.

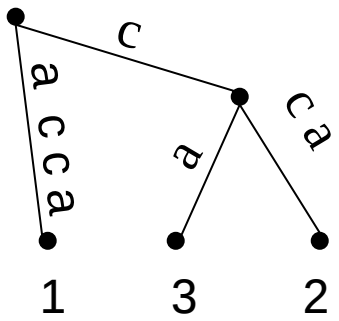
Daher wird Regel 3 auch für Erweiterungen $j' = j$+1,..., i+1 angewandt.

Wenn man also einmal Regel 3 für eine Erweiterung j in Phase i+1 angewandt hat, kann man diese Phase beenden.

Ukkonens Algorithmus

(2) Sobald man in T_i ein Blatt erzeugt, bleibt dies ein Blatt in allen Bäumen $T_{i'}$ für $i' > i$ (einmal ein Blatt, immer ein Blatt!), denn es gibt keine Regel für das Entfernen von Blättern.

t = a c c a b a a c b a ....



T_4

Ukkonens Algorithmus

Folgerung:

- Blatt 1 wird in Phase 1 erstellt. In jeder Phase *i*+1 gibt es eine Anfangssequenz von aufeinanderfolgenden Erweiterungen (angefangen bei Erweiterung 1), bei denen Regel 1 oder Regel 2 angewandt werden.

- Sei j_i die letzte Erweiterung in Phase *i*, die nach Regel 1 oder 2 erfolgt. Dann gilt: $j_i \leq j_{i+1}$

Ukkonens Algorithmus

Erweiterungen der Regel 1 können implizit durchgeführt werden!

Ukkonens Algorithmus

Der Algorithmus kann nun verbessert werden:

Die in Phase i+1 auszuführenden Erweiterungen j für j $\in$ [1, j_i] basieren alle auf Regel 1 und es wird nur konstante Zeit benötigt, um all diese Erweiterungen implizit durchzuführen.

Falls $j \in [j_i + 1, i+1]$ ist, sucht man das Ende des mit $t[j...i]$ markierten Pfades und erweitert ihn mit dem Zeichen t[i+1] indem man Regel 2 oder 3 anwendet.

Falls dabei Regel 3 angewandt wird, setzt man nur $j_{i+1} = j - 1$ und beendet Phase i+1.

Ukkonens Algorithmus

Beispiel:

Phase 1:	berechne Erweiterungen	1 ... j_1
Phase 2:	berechne Erweiterungen	j_1 +1 ... j_2
Phase 3:	berechne Erweiterungen	j_2 +1 ... j_3
....		
Phase *i*-1:	berechne Erweiterungen	j_{i-2} +1 ... j_{i-1}
Phase *i*:	berechne Erweiterungen	j_{i-1} +1 ... j_i

Ukkonens Algorithmus

- Solange der Algorithmus explizite Erweiterungen ausführt, merkt man sich im Index *j** den Index der gerade aktuell ausgeführten expliziten Erweiterung.

- Im Verlaufe der Ausführung des Algorithmus nimmt *j** niemals ab.

- Da es nur *m* Phasen gibt (*m* = |*t*|), und *j** durch *m* begrenzt ist, führt der Algorithmus daher nur *m* explizite Erweiterungen aus.

Ukkonens Algorithmus

Revidierte Pseudo-Code Formulierung von Ukk:

Konstruiere Baum T_1; $j_1 = 1$;
for $i = 1$ **to** $m - 1$ **do**
begin {Phase i+1}
    Führe implizite Erweiterungen durch.
    for $j = j_i + 1$ **to** $i + 1$ **do**
        begin {Erweiterung j}
        Finde das Ende des Pfades von der Wurzel mit Markierung $t[j \ldots i]$ im aktuellen Baum. Falls erforderlich, erweitere ihn durch Hinzufügung des Zeichens $t[i+1]$, damit gewährleistet wird, dass der String $t[j \ldots i+1]$ im Baum ist.
        $j_{i+1} := j$;
        if Regel 3 angewandt **then** $j_{i+1} := j - 1$ und beende Phase i+1;
        end;
end;

Ukkonens Algorithmus

t = pucupcupu

i:	0	1	2	3	4	5	6	7	8	9
	ε	***p**	pu	puc	pucu	pucup	pucupc	pucupcu	pucupcup	pucupcupu
			***u**	uc	ucu	ucup	ucupc	ucupcu	ucupcup	ucupcupu
				***c**	cu	cup	cupc	cupcu	cupcup	cupcupu
					u	***up**	upc	upcu	upcup	upcupu
						p	***pc**	pcu	pcup	pcupu
							c	**cu**	**cup**	***cupu**
								u	up	***upu**
									p	**pu**
										u

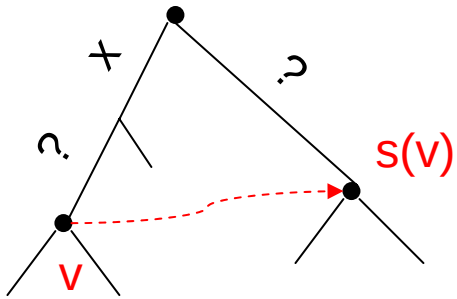
- Suffixe, die zu einer Erweiterung nach Regel 2 führen, sind mit * markiert.
- Unterstrichene Suffixe markieren die letzte Erweiterung nach Regel 2.
- Suffixe, die eine Phase beenden (erstmalige Anwendung von Regel 3) sind blau markiert.

Ukkonens Algorithmus

Die Laufzeit kann noch weiter verbessert werden, wenn man Hilfszeiger (sog. Suffix-Links) ausnutzt.

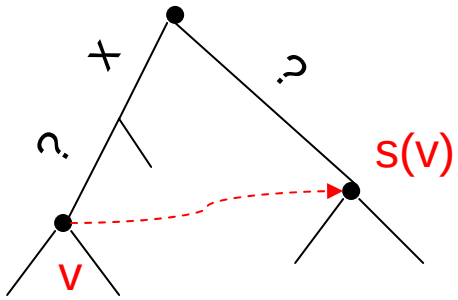
Definition: Sei *x*? ein beliebiger String, wobei *x* ein einzelnes Zeichen darstellt und ? einen (möglicherweise leeren) Teilstring.

Für jeden inneren Knoten *v* mit Kennzeichnung *x*? gilt: Falls es einen weiteren Knoten *s*(*v*) mit Pfad-Markierung ? gibt, so gibt es einen Zeiger von *v* auf *s*(*v*), der als Suffix-Link bezeichnet wird.



Ukkonens Algorithmus

Die Idee ist, Nutzen aus den Suffix-Links zu ziehen, um die Erweiterungspunkte effizienter, d.h. in amortisiert konstanter Zeit, zu finden, ohne bei jeder expliziten Erweiterung an der Wurzel beginnen zu müssen.



Ukkonens Algorithmus

- Mit der Hilfe von Suffix-Links werden die Erweiterungen basierend auf Regel 2 oder Regel 3 vereinfacht.

- Jede explizite Erweiterung kann in amortisierter Zeit O(1) ausgeführt werden (hier nicht gezeigt).

- Weil nur *m* explizite Erweiterungen durchgeführt werden, beträgt die Gesamtlaufzeit von Ukkonens Algorithmus O(*m*) (*m* = |*t*|).

Ukkonens Algorithmus

Der echte Suffix-Baum:

Der endgültige implizite Suffix-Baum T_m kann in einen wirklichen Suffix-Baum in Zeit O(m) umgewandelt werden.

(1) Füge ein Terminalsymbol $ an das Ende vom String t ein.

(2) Lasse Ukkonens Algorithmus mit diesem Zeichen weiterlaufen.

Das Resultat ist der echte Suffix-Baum in dem kein Suffix ein Präfix eines anderen Suffix ist und jedes Suffix in einem Blatt endet.