# Konstruktion von Suffix Bäumen

Prof. Dr. S. Albers
Prof. Dr. Th. Ottmann

# Ukkonen‘s Algorithmus: Impliziter Suffix Baum

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

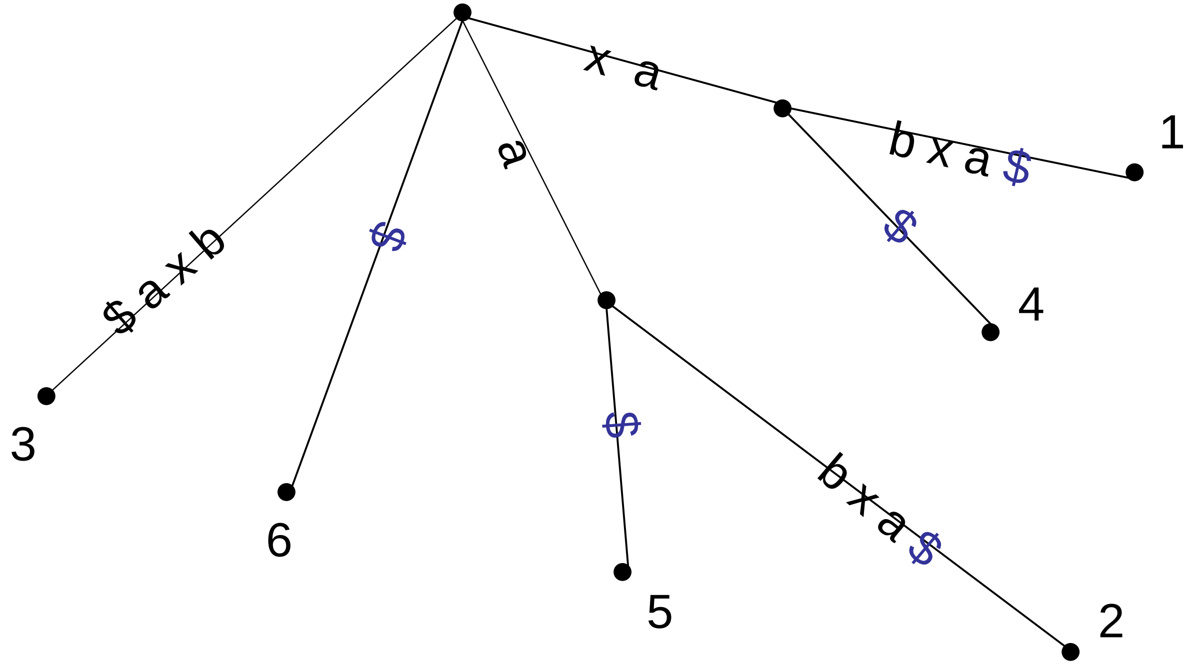
(1) $ von den Markierungen der Kanten entfernt,

(2) Kanten ohne Markierung entfernt,

(3) Knoten mit nur einem Kind entfernt.

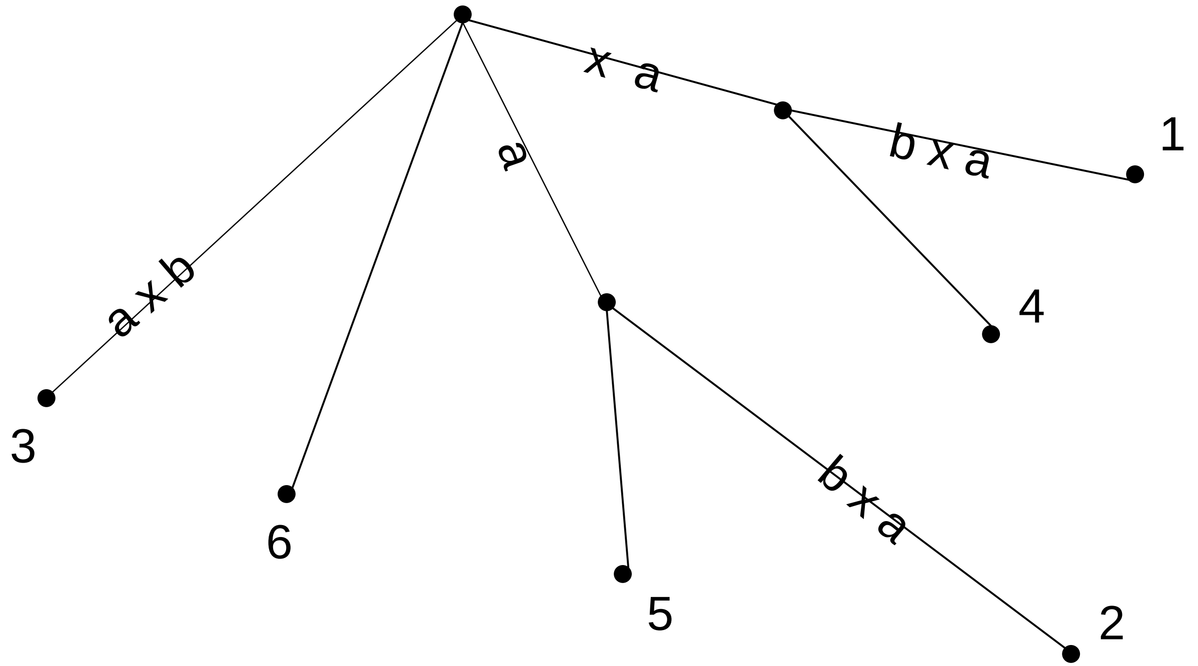
# Ukkonen‘s Algorithmus: Impliziter Suffix Baum

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



# Ukkonen‘s Algorithmus: Impliziter Suffix Baum

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

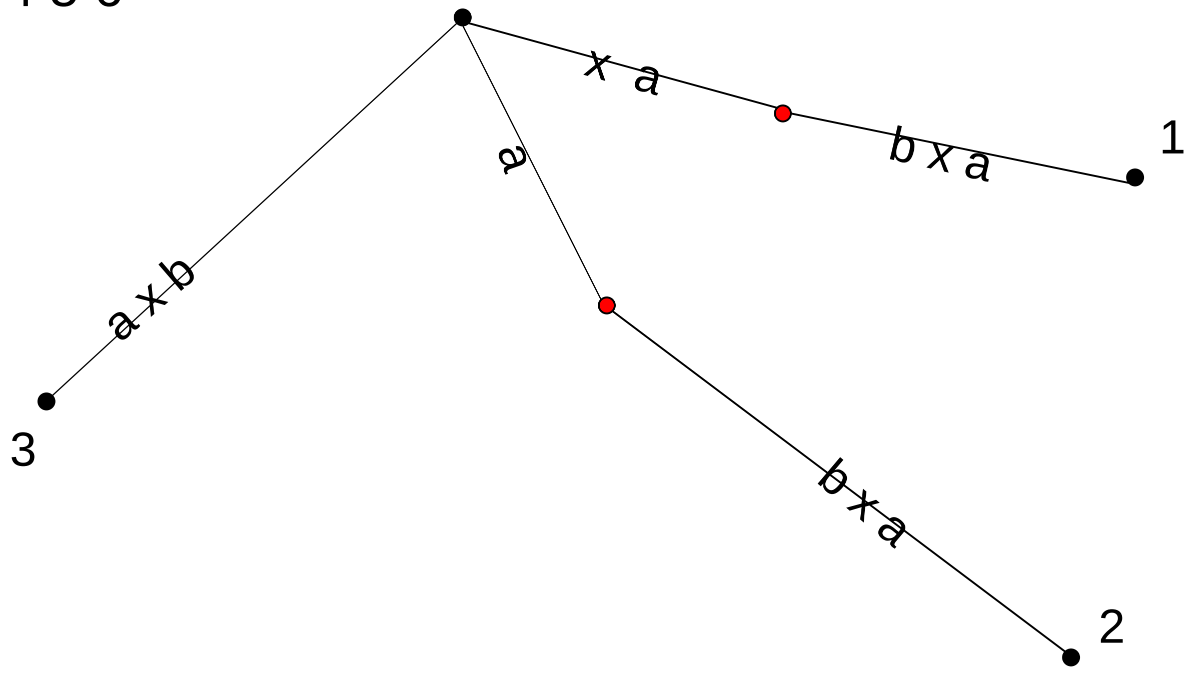


# Ukkonen‘s Algorithmus: Impliziter Suffix Baum

(2) nicht markierten Kanten entfernen

$t$ = x a b x a $

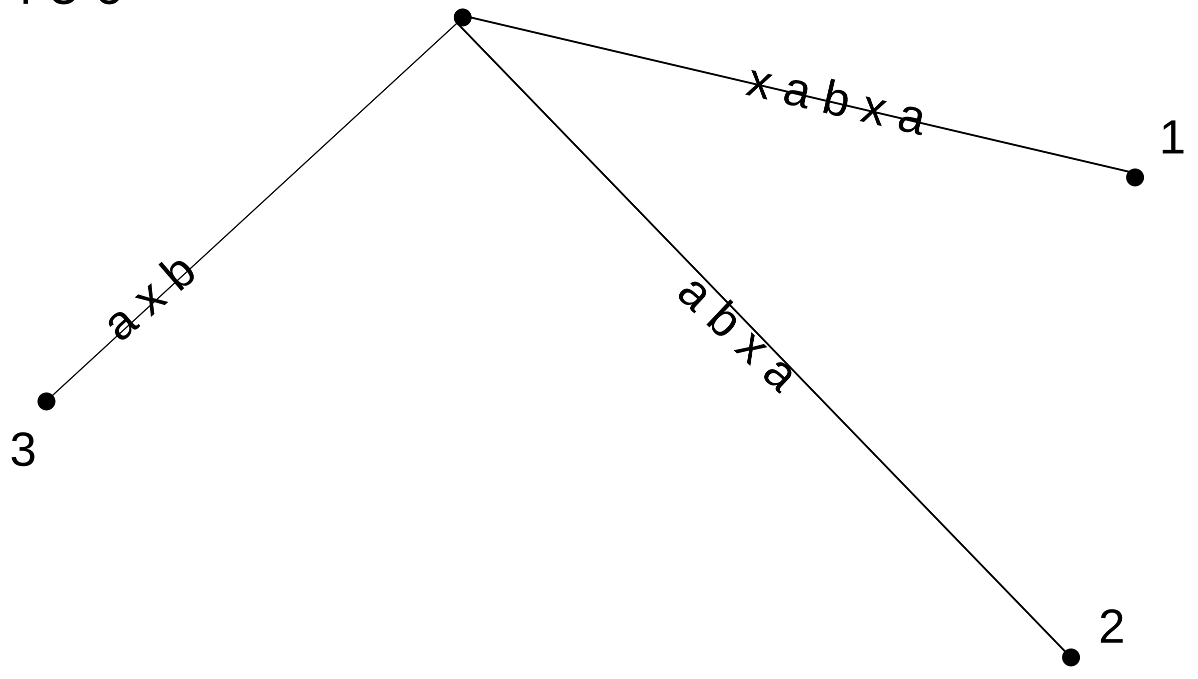
1 2 3 4 5 6



# Ukkonen‘s Algorithmus: Impliziter Suffix Baum

(3) Knoten mit nur einem Kind entfernen

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



# Ukkonen‘s 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

# Ukkonen‘s 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 Input String zeichenweise von links nach rechts liest.

# Ukkonen‘s Algorithmus

**Inkrementelle Konstruktion des impliziten Suffixbaumes:**

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

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

- Sei $I_i$ der implizite Suffix Baum für $t[1...i]$
- Zuerst konstruiert man $I_1$ : Der Baum hat nur eine Kante, die mit dem Zeichen $t_1$ markiert ist.
- Die in Phase i+1 zu lösende Aufgabe ist, $I_{i+1}$ aus $I_i$ für i = 1 bis m – 1 zu konstruieren.

# Ukkonen‘s Algorithmus

Pseudo-code Formulierung von ukk:

Konstruiere Baum $I_1$

**for** i = 1 **to** m – 1 **do**

**begin** {Phase i+1}

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

        **begin** {Erweiterung j}

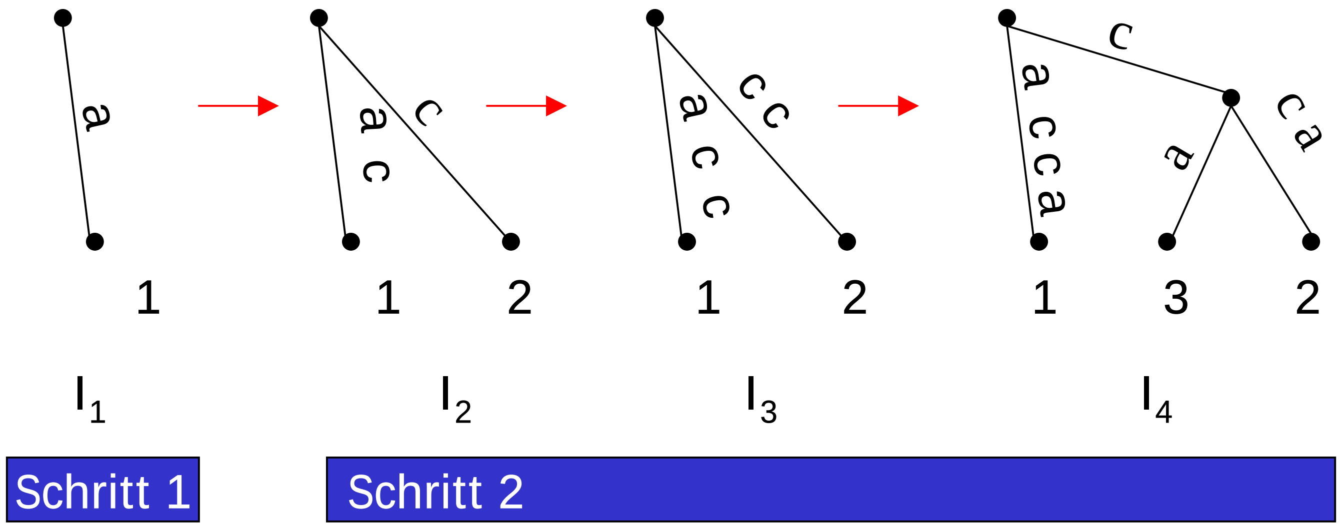
        Finde das Ende des Pfades von der Wurzel, der mit t[j ... i] markiert ist, im aktuellen Baum. Falls erforderlich, erweitere ihn durch Hinzufügen des Zeichens t[i+1], damit gewährleistet ist, dass der String t[j...i+1] im Baum ist.

        **end**;

**end**;

# Ukkonen‘s Algorithmus

t = a c c a $



# Ukkonen‘s Algorithmus

- Jede Erweiterung j wird so durchgeführt, dass man das Ende des Pfades von der Wurzel, gekennzeichnet mit t[j...i], 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 Erweiterungs-Regeln

Der Erweiterungs-Schritt 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, tut man nichts. (Denn dann tritt t[j…i+1] tritt bereits im Baum auf.)

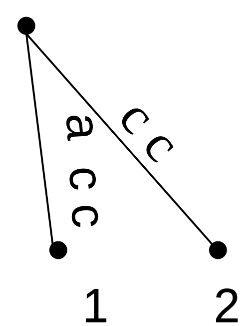
# Ukkonen‘s Algorithmus

t = a c c a $

t[1...3] = acc

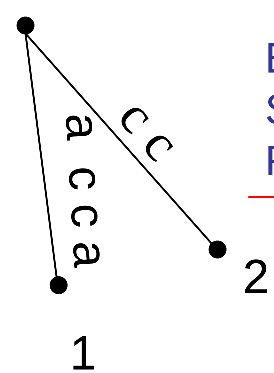
t[1...4] = acca

$I_3$



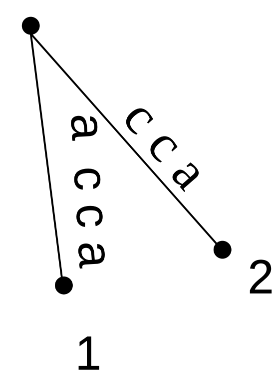

t[1..4] = acca

Erweitere
Suffix 1
Regel 1



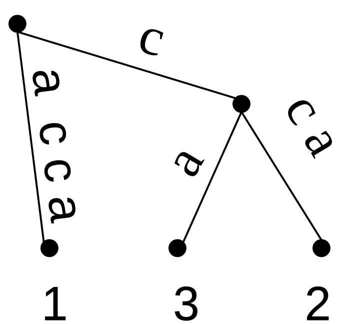

t[2..4] = cca

Erweitere
Suffix 2
Regel 1



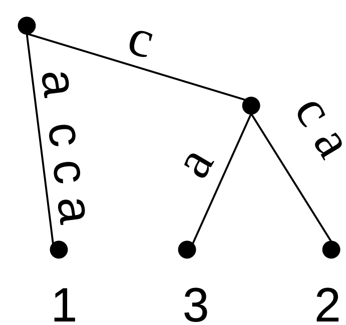

t[3..4] = ca

Erweitere
Suffix 3
Regel 2




t[4..4] = a

Regel 3
a ist bereits
im Baum




$I_4$

# Ukkonen‘s Algorithmus

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

(1) Sobald in Erweiterung j erstmals Regel 3 angewandt wurde, muss der Pfad, der mit t[j...i] im aktuellen Baum $I_i$ 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.

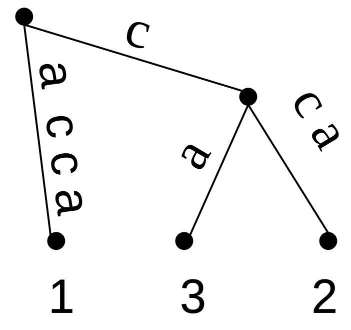
Daher wird Regel 3 auch für Erweiterung j´ für j´= j+1... i+1 angewandt.

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

# Ukkonen‘s Algorithmus

(2) Sobald man ein in $I_i$ ein Blatt erzeugt, bleibt dies ein Blatt in allen Bäumen $I_{i'}$ 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 ....




$I_4$

# Ukkonen‘s Algorithmus

Folgerung:

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

- Sei $j_i$ die letzte Erweiterung in Phase i, die nach Regel 1 oder 2 erfolgt;

  d.h. jede Erweiterung j´, mit $j' > j_i$ , erfolgt nach Regel 3.

  Da jede Anwendung von Regel 2 ein neues Blatt erstellt, gilt nach Beobachtung 2: $j_i \leq j_{i+1}$

Nicht alle Erweiterungen müssen explizit durchgeführt werden!

# Ukkonen's Algorithmus

**Beispiel:**

Phase 1: berechne Erweiterungen $1 \ldots j_1$

Phase 2: berechne Erweiterungen $j_1 + 1 \ldots j_2$

Phase 3: berechne Erweiterungen $j_2 + 1 \ldots j_3$

....

Phase i-1: berechne Erweiterungen $j_{i-2} + 1 \ldots j_{i-1}$

Phase i: berechne Erweiterungen $j_{i-1} + 1 \ldots j_i$

In jeden zwei aufeinanderfolgenden Phasen gibt es höchstens einen Index, der in beiden Phasen betrachtet werden muss, um die erforderlichen Erweiterungen durchzuführen.

# Ukkonen‘s 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[i...j] 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 $j_i = j–1$ und beendet Phase i+1.

# Ukkonen`s Algorithmus

t = pucupcupu

| i: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| ε | ***p*** | pu | puc | pucu | pucup | pucupcu | 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.

# Ukkonen‘s Algorithmus

- Solange der Algorithmus explizite Erweiterungen ausführt, merkt man sich im Index j* den Index der gerade aktuell ausgeführten explizíten Erweiterung.
- Im Verlaufe der Ausführung des Algorithmus nimmt j* niemals ab, sondern kann zwischen zwei aufeinanderfolgenden Phasen höchstens gleich bleiben.
- Da es nur m Phasen gibt (m = |t|), und j* durch m begrenzt ist, führt der Algorithmus daher nur 2m explizite Erweiterungen aus.

# Ukkonen‘s Algorithmus

Revidierte Pseudocode Formulierung von uuk:

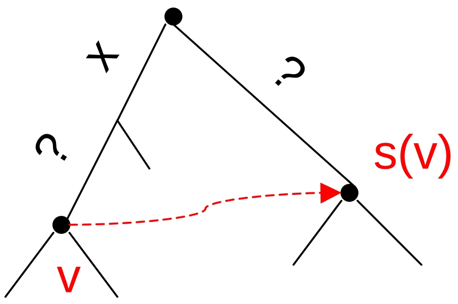
Konstruiere Baum $I_1$; $j_1 = 1$;
**for** i = 1 **to** m – 1 **do**
**begin** {Phase i+1}
  **for** j = $j_i$ +1 **to** i +1 **do**
    **begin** {Erweiterung j}
      Finde das Ende des Pfades von der Wurzel
      mit Markierung t[j ... i] im aktuellen Baum.
      Falls erforderlich, erweitere ihn durch Hinzufügung des Zeichens t[i+1], damit gewährleistet wird, dass der String t[j...i+1] im Baum ist.
      $j_{i+1}$ := j;
      **if** Regel 3 wurde angewandt im Erweiterungsschritt j
      **then** $j_{i+1}$ := j – 1 und beende Phase i+1;
      **end**;
**end**;

# Ukkonen‘s Alorithmus

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

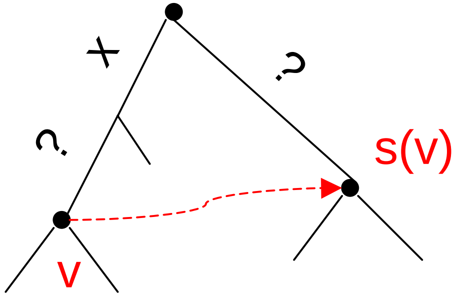
**Definition:** Sei x? ein beliebiger String, wobei x ein einzelnes Zeichen darstellt und ? ein (möglicherweise leerer) Teilstring.

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



# Ukkonen‘s Algorithmus

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



# Ukkonen`s Algorithmus

**Beispiel :** Nehme an, man führt Erweiterung 3 in Phase 6 für String t = „acacag“ aus

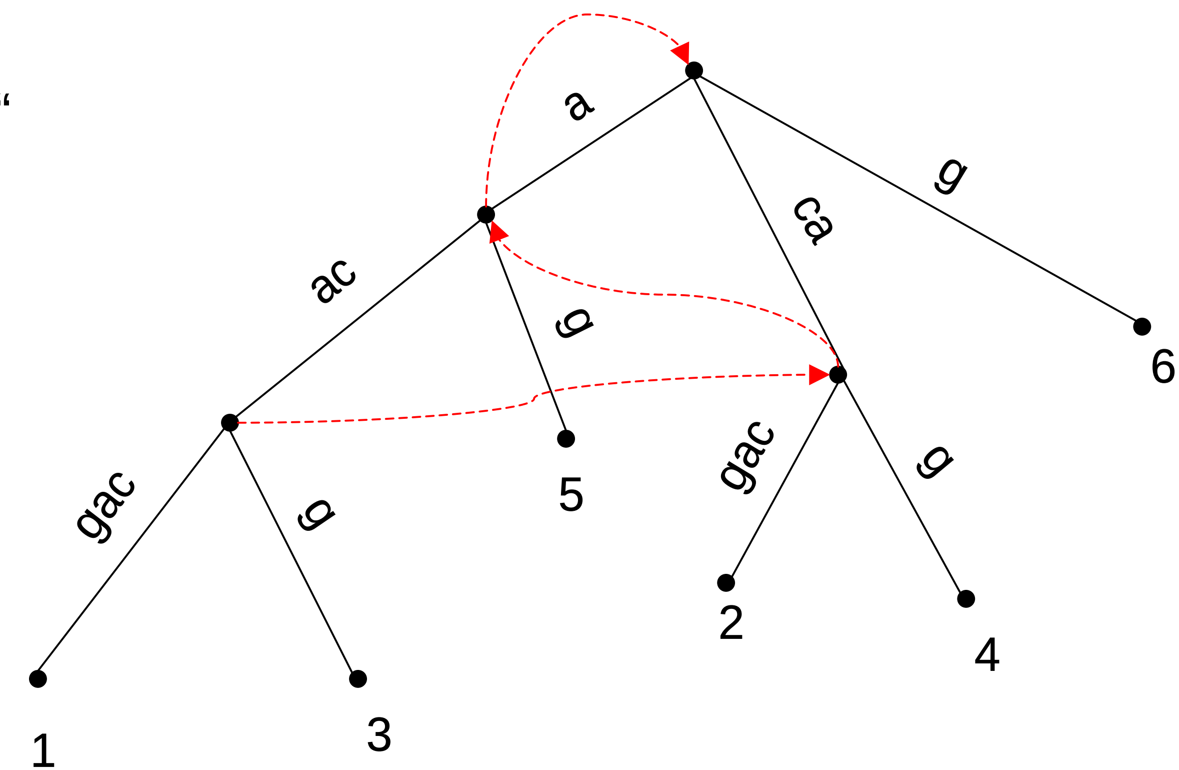
Man muss also t[3...5] = „aca“ finden, indem man von der Wurzel runterläuft und überprüft, ob „ acag“ eingefügt werden soll oder ob es bereits im Baum vorhanden ist. Dann muss man wieder von der Wurzel runterlaufen für die Suffixe „cag“, „ag“ und „g“ und Regel 2 auf sie anwenden.

# Ukkonen`s Algorithmus

**Beispiel :** Nehme an, man führt Erweiterung 3 in Phase 6 für String t = „acacag“ aus

finde t[3 ...5] = „aca“
und erweitere:

„aca g“
„ca g“
„a g“
„g“



# Ukkonen´s Algorithmus

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

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

- Weil nur 2m explizite Erweiterungen durchgeführt werden, beträgt die Gesamtlaufzeit von Ukkonen´s Algorithmus O(m) (m = |t| ).

# Ukkonen´s Algorithmus

**Der echte Suffix-Baum:**

Der endgültige implizite Suffix Baum $I_m$ kann in einen wirklichen Suffix Baum in Zeit O(m) umgewandelt werden:

(1) Füge ein Terminalsymbol $ ans Ende vom String t ein

(2) Lasse Ukkonen´s Algorithmus mit diesem Zeichen weiterlaufen

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