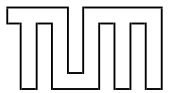
Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

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

Sommersemester 2009



Übersicht

Mehrfachtextsuche / Naiver Ansatz

Problem

Mehrfachtextsuche

Eingabe: *Ein Text* $t \in \Sigma^n$ *und eine Menge* $S = \{s^1, \ldots, s^\ell\} \subseteq \Sigma^+$ *von* ℓ *Suchwörtern mit* $m := \sum_{s \in S} |s|$.

Gesucht: *Taucht ein Suchwort* $s \in S$ *im Text* t *auf?*

Wir nehmen hier zunächst an, dass in S kein Suchwort Teilwort eines anderen Suchwortes aus S ist.

Naiver Ansatz

- KMP-Algorithmus für jedes Suchwort $s \in S$ auf t anwenden
- Kosten für das Preprocessing (Erstellen der Border-Tabellen):

$$\sum_{i=1}^{\ell}(2|s^i| - 1) \leq \sum_{i=1}^{\ell} 2|s^i| = 2m.$$

- Kosten für den eigentlichen Suchvorgang:

$$\sum_{i=1}^{\ell}(2n - |s^i| + 1) \leq 2\ell \cdot n - m + \ell.$$

- Gesamtkosten: $O(\ell n + m)$
  Ziel: Elimination des Faktors ℓ

Suchwort-Baum

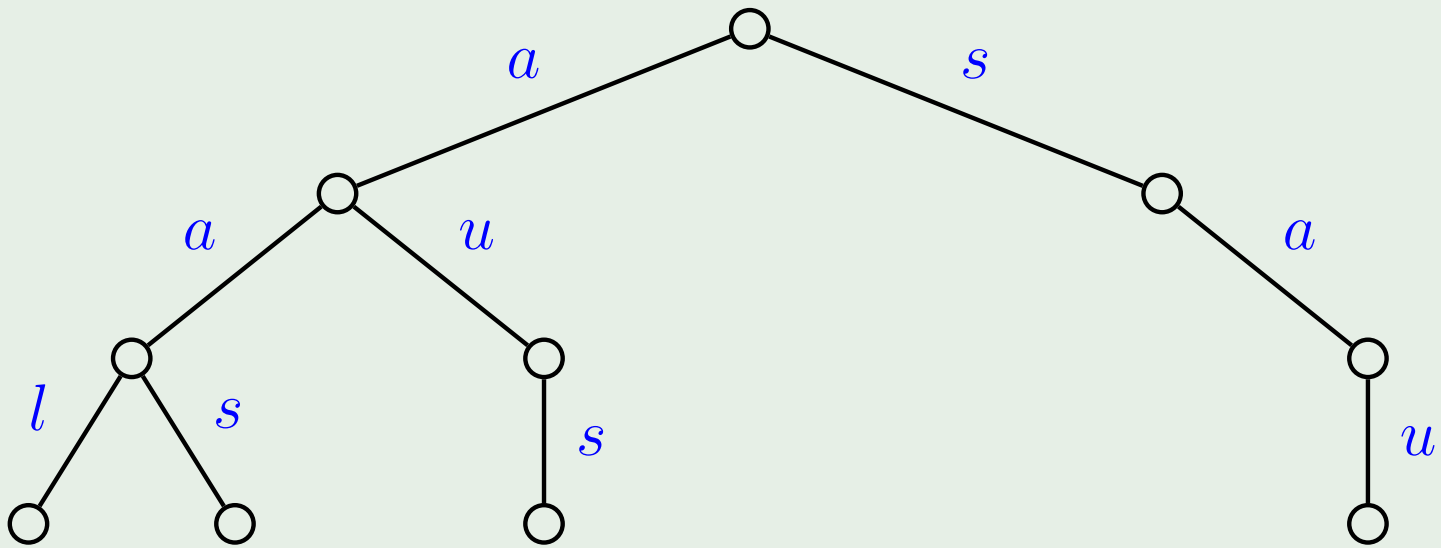
Definition

Ein Suchwort-Baum (Trie) für eine Menge S von Wörtern aus Σ^+ ist ein gewurzelter Baum mit folgenden Eigenschaften:

- Jede Kante ist mit einem Zeichen aus Σ markiert.
- Die von einem Knoten ausgehenden Kanten besitzen paarweise verschiedene Markierungen.
- Jedes Suchwort $s \in S$ wird auf einen Knoten v abgebildet, so dass s entlang des Pfades von der Wurzel zu v steht.
- Jedem Blatt ist ein Suchwort zugeordnet.

Suchwort-Baum

Beispiel



Suchwort-Baum für $\{aal, aas, aus, sau\}$

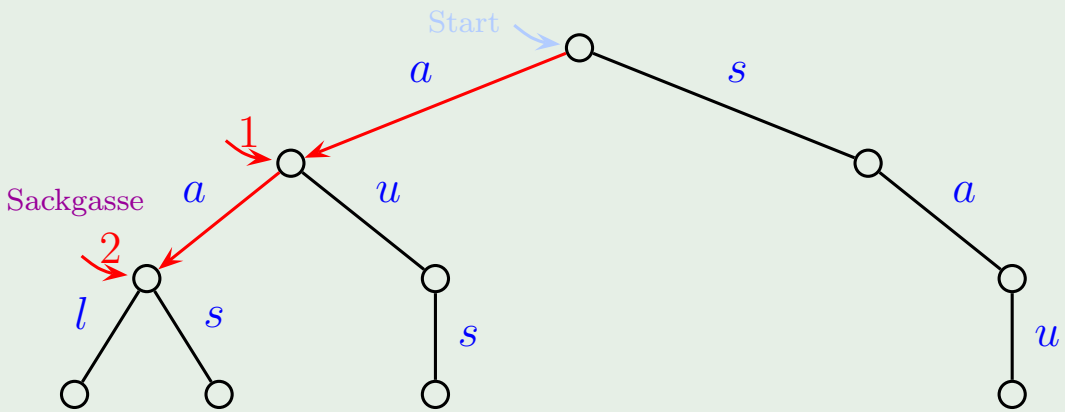
Suchen mit Hilfe des Suchwort-Baums

Wie kann man nun mit diesem Suchwort-Baum im Text t suchen?

- Die Buchstaben des Textes t werden im Suchwort-Baum abgelaufen.
- Sobald man an einem Blatt gelandet ist, hat man eines der gesuchten Wörter gefunden.
- Man kann jedoch auch in Sackgassen landen: Knoten, von denen keine Kante mit einem gesuchten Kanten-Label ausgeht.

Suchwort-Baum

Beispiel



Suche mit dem Suchwort-Baum im Text $t =$ $\underset{\downarrow}{\overset{1}{a}}$ $\underset{\downarrow}{\overset{2}{a}}$ u s

Nach der Abarbeitung des Teilwortes *aa* gerät man in eine Sackgasse, da diesen Knoten keine Kante mit Label *u* verlässt. Somit wird das Teilwort *aus*, obwohl in *t* enthalten, nicht gefunden.

Failure-Links

- Als Ausweg aus den Sackgassen werden in den Baum so genannte Failure-Links eingefügt.

Definition

Ein Failure-Link eines Suchwort-Baumes ist ein Verweis von einem Knoten v auf einen Knoten w im Baum, so dass die Kantenmarkierungen von der Wurzel zum Knoten w das längste Suffix des bereits erkannten Teilwortes bilden (= Wort zu Knoten v).

Failure-Links

- Die Failure-Links der Kinder der Wurzel werden so initialisiert, dass sie direkt zur Wurzel zeigen.
- Die Failure-Links der restlichen Knoten werden nun Level für Level von oben nach unten berechnet.
- Betrachten wir dazu den Knoten v mit Vater w, wobei das Kantenlabel von w zu v gerade a sei.
- Wir folgen dann dem bereits berechneten Failure-Link von w zu x.

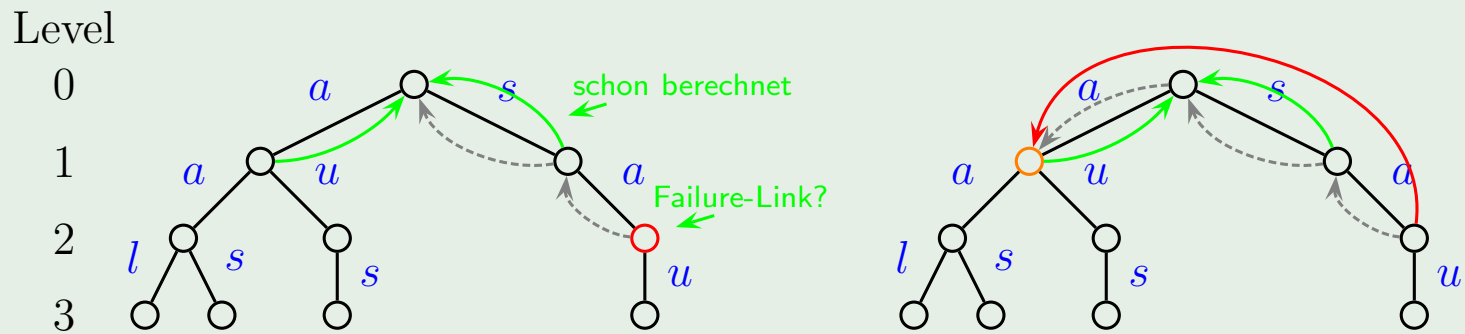
Failure-Links

- Hat der Knoten x eine ausgehende Kanten zum Knoten y mit Markierung a, so setzen wir den Failure-Link vom Knoten v auf y.
- Andernfalls folgen wir wiederum dem Failure-Link von x.

- Dieses Verfahren endet,
  - wenn ein Knoten über Failure-Links erreicht wird, von dem eine ausgehende Kante mit Label a zu einem Knoten z erreicht wird
  - oder aber die Wurzel erreicht wird, von der keine ausgehende Kante die Markierung a trägt.
- Im ersten Fall wird der Failure-Link von v auf den Knoten z gesetzt, im zweiten Fall setzt man den Failure-Link von v auf die Wurzel des Baumes.

Berechnung der Failure-Links

Beispiel

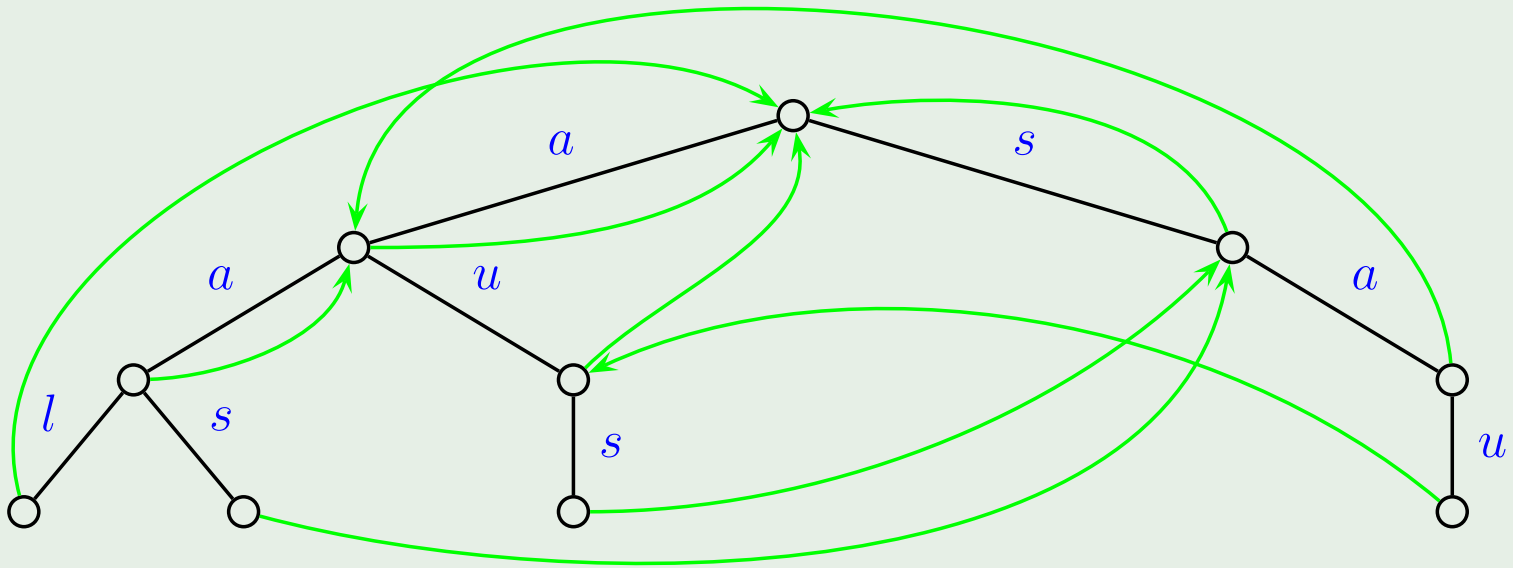


Berechnung des Failure-Links des zu sa gehörigen Knotens v:

- Inspektion des Vaters w, also des zu s gehörigen Knotens,
- Verfolgen des Failure-Links von w zu x, hier die Wurzel (ε),
- x hat eine ausgehende Kante mit Label a zum Knoten y (a),
- Failure-Link von v (sa) wird auf y (a) gesetzt.

Berechnung der Failure-Links

Beispiel



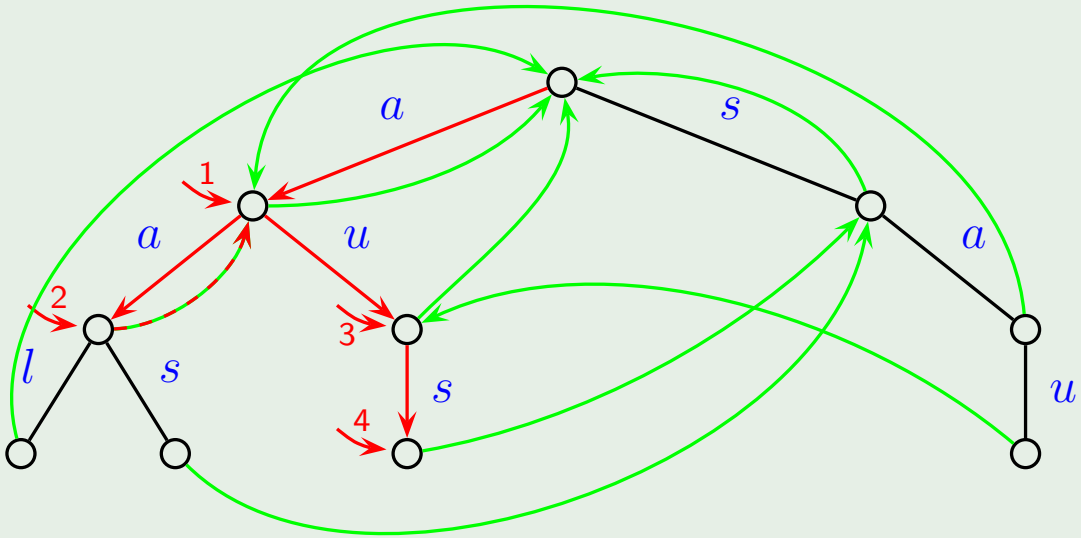
Kompletter Baum mit allen Failure-Links

Suche mit Failure-Links

- Ein Suchwort $s \in S$ ist genau dann im Text t enthalten, wenn man beim Durchlaufen der Buchstaben von t im Suchwort-Baum in einem Blatt ankommt.
- Sind wir in einer Sackgasse gelandet, d.h. es gibt keine ausgehende Kante mit dem gewünschten Label, so folgen wir dem Failure-Link und suchen von dem so aufgefundenen Knoten aus weiter.
- Da wir angenommen haben, dass kein Suchwort Teilwort eines anderen Suchwortes ist, können wir beim Folgen von Failure-Links nie zu einem Blatt gelangen.

Suche mit Failure-Links

Beispiel



Beispiel: $t = \overset{1}{\downarrow} a \overset{2}{\downarrow} a \overset{3}{\downarrow} u \overset{4}{\downarrow} s$

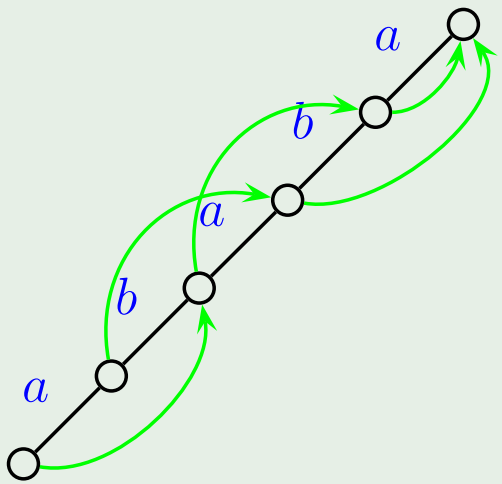
Suche in $t = aaus$ mit Failure-Links

Einelementige Suchwortmenge / border-Tabelle

- Enthält die Menge S nur ein einziges Suchwort, so erhalten wir als Spezialfall ein Äquivalent der *border*-Tabelle des KMP-Algorithmus.
- Im Suchwort-Baum wird dabei auf das entsprechend längste (echte oder leere) Suffix verwiesen, das gleichzeitig Präfix ist, also auf den eigentlichen Rand.
- In der Tabelle *border* ist allerdings nur die Länge dieses Suffixes bzw. Präfixes gespeichert.
- Wenn S einelementig ist, liefern beide Methoden dieselben Informationen.

Einelementige Suchwortmenge / border

Beispiel



Level 0, $border[0] = -1$

Level 1, $border[1] = 0$

Level 2, $border[2] = 0$

Level 3, $border[3] = 1$

Level 4, $border[4] = 2$

Level 5, $border[5] = 3$

Einelementige Suchwortmenge $S = \{ababa\}$

Berechnung der Failure-Links

Algorithmus 4 : compute_failure_links(tree T)

```
// T = (V, E) ist Suchwort-Baum
forall the (v ∈ V) do
    // Levelweises Durchlaufen der Knoten
    Sei v' der Elternknoten von v mit (v' --x--> v) ∈ E;
    w := Failure_Link(v');
    while ((∀y ∈ V : (w --x--> y) ∉ E) && (w ≠ root)) do
        w := Failure_Link(w);
    if (w --x--> w') then
        Failure_Link(v) := w';
    else
        Failure_Link(v) := root;
```

Laufzeit der Failure-Link-Berechnung

- Die Laufzeit zur Berechnung der Failure-Links beträgt $O(m)$.
- Um dies zu zeigen, betrachten wir ein festes Suchwort $s \in S$.
- Wir zeigen zunächst nur, dass für die Berechnung der Failure-Links der Knoten auf dem Pfad von s im Suchwort-Baum $O(|s|)$ Vergleiche ausgeführt werden.
- Durch Summation über alle Suchwörter $s \in S$ werden wir den Gesamtaufwand etwas zu hoch abschätzen (aber immer noch linear in m).
- Wie bei der Analyse der Berechnung der Tabelle *border* des KMP-Algorithmus unterscheiden wir erfolgreiche und erfolglose Vergleiche.

Laufzeit der Failure-Link-Berechnung

- Für jeden Knoten kann es bei der Failure-Link-Berechnung nur einen erfolgreichen Vergleich geben, denn danach ist die Berechnung für diesen Knoten fertig.
- Es kann also für alle Failure-Link-Berechnungen für die Knoten auf einem Pfad entsprechend s maximal $|s|$ erfolgreiche Vergleiche (d.h., $\exists$ Kante $w \xrightarrow{x}$) geben, da man dann zum nächsttieferen Knoten auf dem Pfad von s wechselt (und der tiefste Knoten ist auf Level $|s|$).

Laufzeit der Failure-Link-Berechnung

- Für erfolglose Vergleiche gilt, dass Failure-Links immer nur zu Knoten auf einem niedrigeren Level verweisen.
- Bei jedem erfolglosen Vergleich springen wir also zu einem Knoten auf einem niedrigeren Level.
- Da wir nur bei einem erfolgreichen Vergleich zu einem höheren Level springen können und da der Level nie negativ sein darf, kann es nur so viele erfolglose wie erfolgreiche Vergleiche geben.
- Somit ist die Anzahl Vergleiche für jedes Wort $s \in S$ durch $O(|s|)$ beschränkt.
- Damit ergibt sich insgesamt für die Anzahl der Vergleiche

$$\leq \sum_{s \in S} O(|s|) = O(m).$$

Die Methode von Aho und Corasick

Algorithmus 5 : bool Aho-Corasick(char $t[]$, int n, char $S[]$, int m)

int $i := 0$;
tree $T(S)$; // Suchwort-Baum aus den Wörtern in S
node $v := root$;
while $(i < n)$ **do**
  while $((v \xrightarrow{t_{i+level(v)}} v')$ *in* $T)$ **do**
    $v := v'$;
    if $(v'$ *ist Blatt*$)$ **then return** TRUE;
  if $(v \neq root)$ **then**
    $i := i + level(v) - level(\text{Failure\_Link}(v))$;
    $v := \text{Failure\_Link}(v)$;
  else
    i++;

Laufzeit des Aho-Corasick-Algorithmus

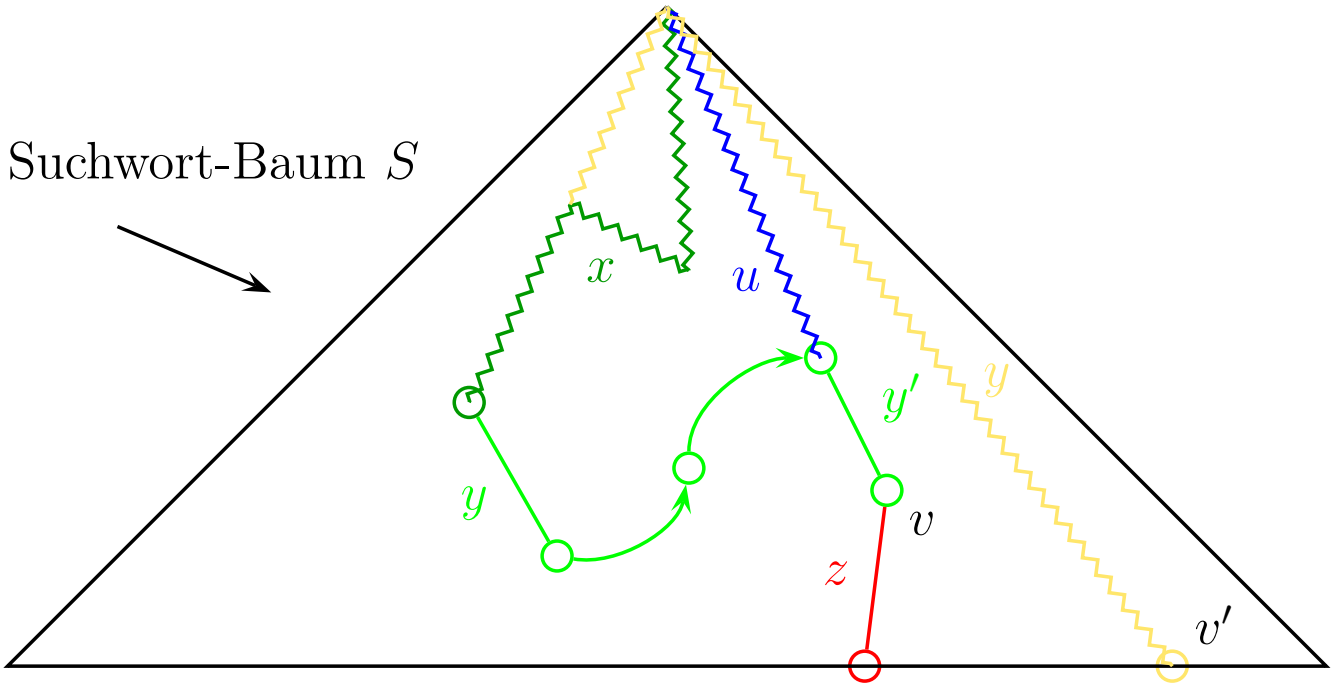
- Auch hier: Laufzeitanalyse ähnlich wie beim KMP-Algorithmus
- Da $\mathit{level}(v) - \mathit{level}(\mathrm{Failure\_Link}(v)) > 0$ ist (analog zu $j - \mathit{border}[j] > 0$ bei KMP), wird i nach jedem erfolglosen Vergleich um mindestens 1 erhöht.
- Also gibt es maximal $n - m + 1$ erfolglose Vergleiche.
- $i + \mathit{level}(v)$ erniedrigt sich nach einem erfolglosen Vergleich nie und erhöht sich nach jedem erfolgreichen Vergleich um 1 (da sich $\mathit{level}(v)$ um 1 erhöht).
- Da $i + j \in [0 : n - 1]$ ist, können maximal n erfolgreiche Vergleiche ausgeführt werden.
- Somit gibt es maximal $2n - m + 1$ Vergleiche.

Korrektheit des Aho-Corasick-Algorithmus

- Wenn kein Muster in t auftritt ist klar, dass der Algorithmus nicht behauptet, dass ein Suchwort auftritt.

- Wir beschränken uns also auf den Fall, dass eines der Suchwörter aus S in t auftritt.

Korrektheit des Aho-Corasick-Algorithmus



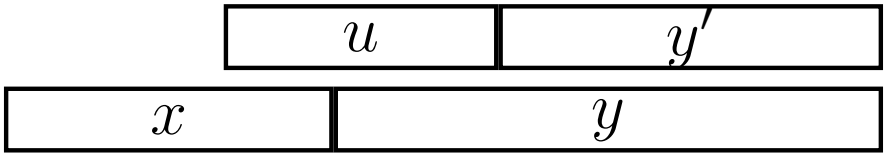
Korrektheit des Aho-Corasick-Algorithmus

- Was ist, wenn $y \in S$ als ein Teilwort von t auftritt und sich der Algorithmus unmittelbar nach dem Lesen von y in einem internen Knoten v befindet?
- Sei uy' das Wort, über das man den Knoten v auf einem einfachen Pfad von der Wurzel aus erreicht, wobei y' das Teilwort des Pfades ist, das während des Algorithmus auf diesem Pfad abgelaufen wurde.
- y' ist ein Suffix von y.
  (Wäre andernfalls y ein Suffix von y', wäre y in einem anderen Suchwort enthalten, was nach Voraussetzung ausgeschlossen ist.)

Korrektheit des Aho-Corasick-Algorithmus

- Wir behaupten, dass y ein Suffix von uy' ist.
- Da $y \in S$, gibt es im Suchwortbaum einen Pfad von der Wurzel zu einem Blatt v', der mit y markiert ist.
- Somit kann man nach der Verarbeitung von y als Teilwort von t nur an einem Knoten landen, dessen Level mindestens $|y|$ ist (ansonsten müssten wir bei v' gelandet sein).
- Somit ist $\mathit{level}(v) \geq |y|$.
- Nach Definition der Failure-Links muss dann die Beschriftung uy' des Pfades von der Wurzel zu v mindestens mit y enden.

Korrektheit des Aho-Corasick-Algorithmus



Skizze: y muss Suffix von uy' sein

Korrektheit des Aho-Corasick-Algorithmus

- Ist z die Beschriftung eines Pfades von der Wurzel zu einem Blatt, das von v aus über normale Baumkanten erreicht werden kann, dann muss y ein echtes Teilwort von $uy'z \in S$ sein.
- Dies widerspricht aber der Annahme, dass kein Suchwort aus S ein echtes Teilwort eines anderen Suchwortes in S sein kann.
- Damit befindet sich der Algorithmus von Aho-Corasick nach dem Auffinden eines Suchwortes $s \in S$ in einem Blatt des Baumes.

Ergebnis

Theorem

Sei $S \subseteq \Sigma^*$ eine Menge von Suchwörtern, so dass kein Wort $s \in S$ ein echtes Teilwort von $s' \in S$ (mit $s \neq s'$) ist.

Dann findet der Algorithmus von Aho-Corasick einen Match von $s \in S$ in $t \in \Sigma^*$ in der Zeit $O(n + m)$, wobei

- $n = |t|$ *und*
- $m = \Sigma_{s \in S} |s|$.