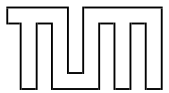
Algorithmische Bioinformatik 1

Dr. Hanjo Täubig

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

Sommersemester 2009



Übersicht

1. Algorithmen zur Textsuche
   - Boyer-Moore-Algorithmus
2. Suffix Tries

Lemma: Fall 2 (kurzer Shift)

- Definiere Intervall A:

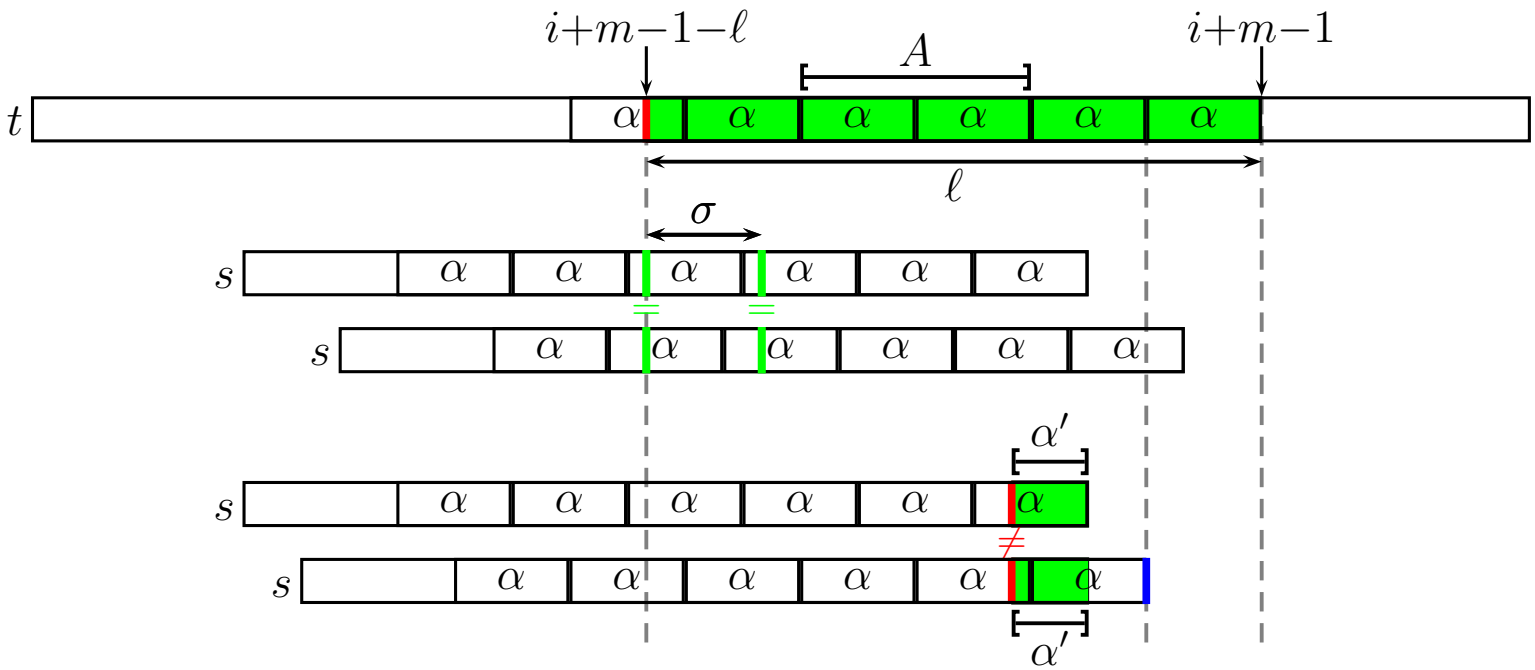
$$A := [i + m - (k-2)\sigma : i + m - 2\sigma - 1]$$

$$(k := 1 + \lfloor \ell/\sigma \rfloor \geq 5)$$

- Zu zeigen:
  Zeichen in t an den Positionen im Intervall A waren bislang noch an keinem Vergleich beteiligt
  (per Widerspruchbeweis)

- Betrachte dazu frühere Versuche, die Vergleiche im Abschnitt A hätten ausführen können.

Lemma: Fall 2 (kurzer Shift)



Lemma: Fall 2 (kurzer Shift)

Betrachte früheren Versuch, bei dem mindestens σ erfolgreiche Vergleiche ausgeführt worden sind (oberer Teil der Abbildung)

- Da mindestens σ erfolgreiche Vergleiche ausgeführt wurden, folgt aus der Periodizität von s', dass alle Vergleiche bis zur Position $i + m - \ell$ erfolgreich sein müssen.
- Der Vergleich an Position $i + m - \ell - 1$ muss hingegen erfolglos sein, da auch der jetzige Versuch, s an Position i in t zu finden, an Position $i + m - \ell - 1$ erfolglos ist.
- Behauptung: Dann hätte jeder sichere Shift gemäß der Strong-Good-Suffix-Rule die Zeichenreihe s auf eine Position größer als i verschoben.

Lemma: Fall 2 (kurzer Shift)

- Annahme: es gibt einen kürzeren sicheren Shift (wie in der Abbildung darunter dargestellt).
- Dann müsste das Zeichen an Position $i + m - \ell - 1$ in t mit einem Zeichen aus s' verglichen werden.
- Dort steht im verschobenen s' aber dasselbe Zeichen wie beim erfolglosen Vergleich des früheren Versuchs, da sich in s' die Zeichen alle σ Zeichen wiederholen.
- Damit erhalten wir einen Widerspruch zur Strong-Good-Suffix-Rule.

Lemma: Fall 2 (kurzer Shift)

Versuche, mit weniger als σ erfolgreichen Vergleichen:

- Da wir nur Versuche betrachten, die Vergleiche im Abschnitt A ausführen, muss der Versuch an einer Position im Intervall $[i-(k-2)\sigma : i-\sigma-1]$ von rechts nach links begonnen haben.
- Nach Wahl von A muss der erfolglose Vergleich auf einer Position größer als $i+m-\ell-1$ erfolgen.
- Betrachte hierzu die erste Zeile im unteren Teil der Abbildung. Sei α' das Suffix von α (und damit von s' bzw. s), in dem die erfolgreichen Vergleiche stattgefunden haben.
- Seien nun $x \neq y$ die Zeichen, die den Mismatch ausgelöst haben, wobei x unmittelbar vor dem Suffix α' in s' steht.
- Offensichtlich liegt α' völlig im Intervall $[i-m-\ell : i+m-\sigma-1]$.

Lemma: Fall 2 (kurzer Shift)

- Wir werden jetzt zeigen, dass ein Shift auf $i - \sigma$ (wie im unteren Teil der Abbildung darunter dargestellt) zulässig ist.
- Die alten erfolgreichen Vergleiche stimmen mit dem Teilwort in s' überein.
- Nach Voraussetzung steht an der Position unmittelbar vor α' in s' das Zeichen x und an der Position des Mismatches das Zeichen y.
- Damit ist ein Shift auf $i - \sigma$ zulässig.
- Da dies aber nicht notwendigerweise der Kürzeste sein muss, erfolgt ein sicherer Shift auf eine Position kleiner gleich $i - \sigma$.

Lemma: Fall 2 (kurzer Shift)

- Erfolgt ein Shift genau auf Position $i - \sigma$, dann ist der nächste Versuch bis zur Position $i + m - \ell - 1$ in t erfolgreich.
- Da wir dann also mindestens σ erfolgreiche Vergleiche ausführen, folgt, wie oben erläutert, ein Shift auf eine Position größer als i (oder der Versuch wäre erfolgreich abgeschlossen worden).
- Andernfalls haben wir einen Shift auf Position kleiner als $i - \sigma$.
- Aber auch dann gilt, dass in allen folgenden Fällen ein Shift genau auf die Position $i - \sigma$ zulässig bleibt.
- Egal, wie viele Shifts ausgeführt werden, irgendwann kommt ein Shift auf die Position $i - \sigma$.
- Also erhalten wir letztendlich immer einen Shift auf eine Position größer als i, aber nie einen Shift auf die Position i.

Lemma: Fall 2 (kurzer Shift)

- Damit ist bewiesen, dass bei einem kurzen Shift die Zeichen im Abschnitt A von t zum ersten Mal verglichen worden sind.
- Da auch der Vergleich des letzten Zeichens von s mit einem Zeichen aus t ein initialer gewesen sein musste, folgt dass mindestens $1 + |A|$ initiale Vergleiche ausgeführt wurden.
- Da $|A| \geq \ell - 4\sigma$, erhalten wir die gewünschte Abschätzung:

$$1 + \ell = (1 + |A|) + (\ell - |A|) \leq (1 + |A|) + 4\sigma.$$

- Da wie schon weiter oben angemerkt, der Vergleich des letzten Zeichens von s immer initial sein muss, und alle Vergleiche im Bereich A initial sind, folgt damit die Behauptung des Lemmas.

Laufzeit der Hauptprozedur

Lemma

Bei einer erfolglosen oder erfolgreichen Suche nach dem ersten Vorkommen von $s \in \Sigma^m$ in $t \in \Sigma^n$ treten maximal $5n + m$ Zeichenvergleiche auf.

Laufzeit der Hauptprozedur

Beweis.

Bezeichne dazu

- $V(n)$ die Anzahl aller Vergleiche, um in einem Text der Länge n zu suchen, und
- I_i bzw. V_i für $i \in [1 : k]$ die Anzahl der initialen bzw. aller Vergleiche, die beim i-ten Versuch ausgeführt wurden,
- σ_i die Länge des Shifts, der nach dem i-ten Vergleich ausgeführt wird.
  War der i-te (und somit letzte) Vergleich erfolgreich, so sei $\sigma_i := m$ (Unter anderem deswegen gilt die Analyse nur für einen erfolglosen bzw. den ersten erfolgreichen Versuch).

Laufzeit der Hauptprozedur

Beweis.

$$
\begin{aligned}
V(n) &= \sum_{i=1}^{k} V_i \quad \text{im letzten Test maximal } m \text{ Vergleiche} \\
&\leq \sum_{i=1}^{k-1} V_i + m \qquad \text{(siehe vorheriges Lemma)} \\
&\leq \sum_{i=1}^{k-1} (l_i + 4\sigma_i) + m \quad \text{(maximal } n \text{ initiale Vergleiche)} \\
&\leq n + 4 \sum_{i} \sigma_i + m \quad \text{Summe der Shifts ist maximal } n \\
&\quad \text{(Anfang von } s \text{ bleibt innerhalb von } t\text{)} \\
&\leq n + 4n + m = 5n + m
\end{aligned}
$$

Laufzeit

Satz

Der Boyer-Moore Algorithmus benötigt für das erste Auffinden des Suchmusters $s \in \Sigma^m$ in einem Text $t \in \Sigma^n$ maximal $5n + m$ Vergleiche.

Satz

Der Boyer-Moore Algorithmus benötigt maximal $3(n + m)$ Vergleiche um zu entscheiden, ob eine Zeichenreihe der Länge m in einem Text der Länge n enthalten ist.

Erweiterung auf alle Vorkommen

- Analyse ist nur für den Fall einer erfolglosen Suche oder dem ersten Auffinden von s in t gültig.
- Das Beispiel $s = a^m$ und $t = a^n$ zeigt, dass der Boyer-Moore-Algorithmus beim Auffinden aller Vorkommen von s in t wieder quadratischen Zeitbedarf bekommen kann
- Diese worst-case Laufzeit lässt sich jedoch mit einem Trick vermeiden: Nach einem erfolgreichen Test $s = t_i \cdots t_{i+m-1}$ verschieben wir das Muster gemäß der Strong-Good-Suffix-Rule
  (Man erhält denselben Shift, wie beim KMP-Algorithmus).
- Nach diesem erfolgreichen Test werden keine Vergleiche mehr mit Zeichen in t an einer Position kleiner als $i + m$ ausgeführt.
- Da wir ja wissen, dass $s = t_i \cdots t_{i+m-1}$ gilt, werden Vergleiche im Bereich kleiner als Position $i + m$ in t jetzt dadurch ersetzt, dass wir testen, ob der Bereich links von der Position $i + m$ ein Rand von s ist.

Erweiterung auf alle Vorkommen

- Mögliche Abfragen, ob $t_j \cdots t_{i+m-1}$ ein Rand ist, werden ja für steigende j gestellt.
- Somit werden die Anfragen nach Rändern nach fallender Länge gestellt. Die absteigende Aufzählung der Ränder von s lässt sich, wie schon gesehen, mit der bereits berechneten Border-Tabelle sehr leicht ausführen.
- Dabei müssen wir die Border-Tabelle im schlimmsten Falle m-Mal durchlaufen.
- Da aber jedes Durchlaufen der Border-Tabelle auch einem Shift von mindestens Eins entspricht, fällt nur ein additiver Zusatzaufwand von $O(m)$ an.
- Gesamtlaufzeit bleibt bei $O(n+m)$

Übersicht

1 Algorithmen zur Textsuche

2 Suffix Tries
- Suffix Tries
- Online-Konstruktion für Suffix-Tries

Ziel

- noch schnellere Verfahren zum Suchen in Texten
- zu durchsuchender Text soll vorverarbeitet werden (möglichst in Zeit und Platz $O(|t|)$)
- danach sehr schnelle Suche nach Wort s (wiederholbar für unterschiedliche Suchwörter und möglichst in Zeit $O(|s|)$)

Suffix Tries

Ein Trie ist eine Datenstruktur, in der eine Menge von Wörtern gespeichert werden kann.

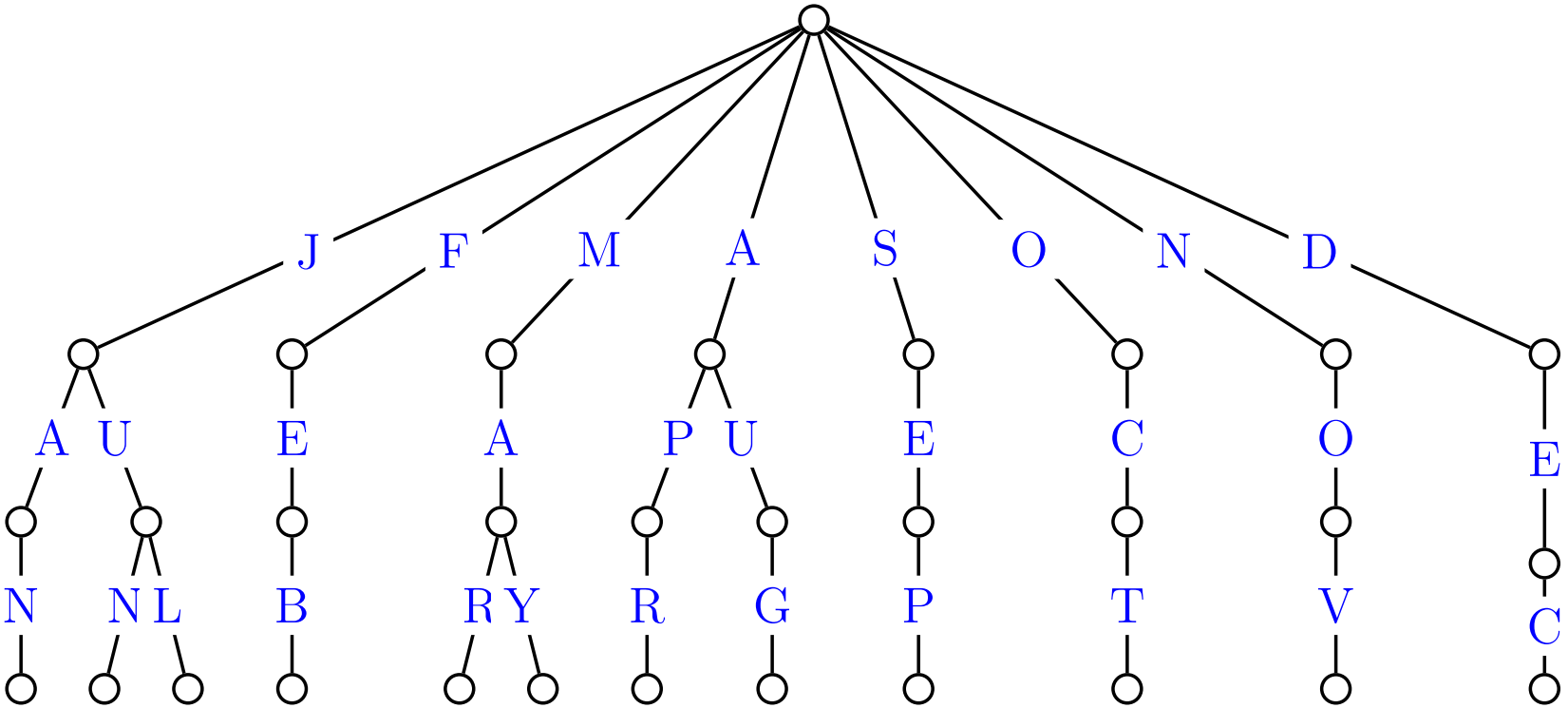
Definition

Ein Trie für eine Menge $S \subseteq \Sigma^+$ 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 Wort $s \in S$ wird auf einen Knoten v abgebildet, so dass s entlang des Pfades von der Wurzel zu v steht;
- Jedem Blatt ist ein Wort aus S zugeordnet.

Beispiel

Trie für die Abkürzungen der Monatsnamen



Vorteil

- Der Suchwort-Baum des Aho-Corasick-Algorithmus ohne die Failure-Links ist nichts anderes als ein Trie für die Menge der Suchwörter.

- Vorteil eines Tries: man kann darin nach einem Wort der Länge m in Zeit $O(m)$ suchen.

Suffix-Trie

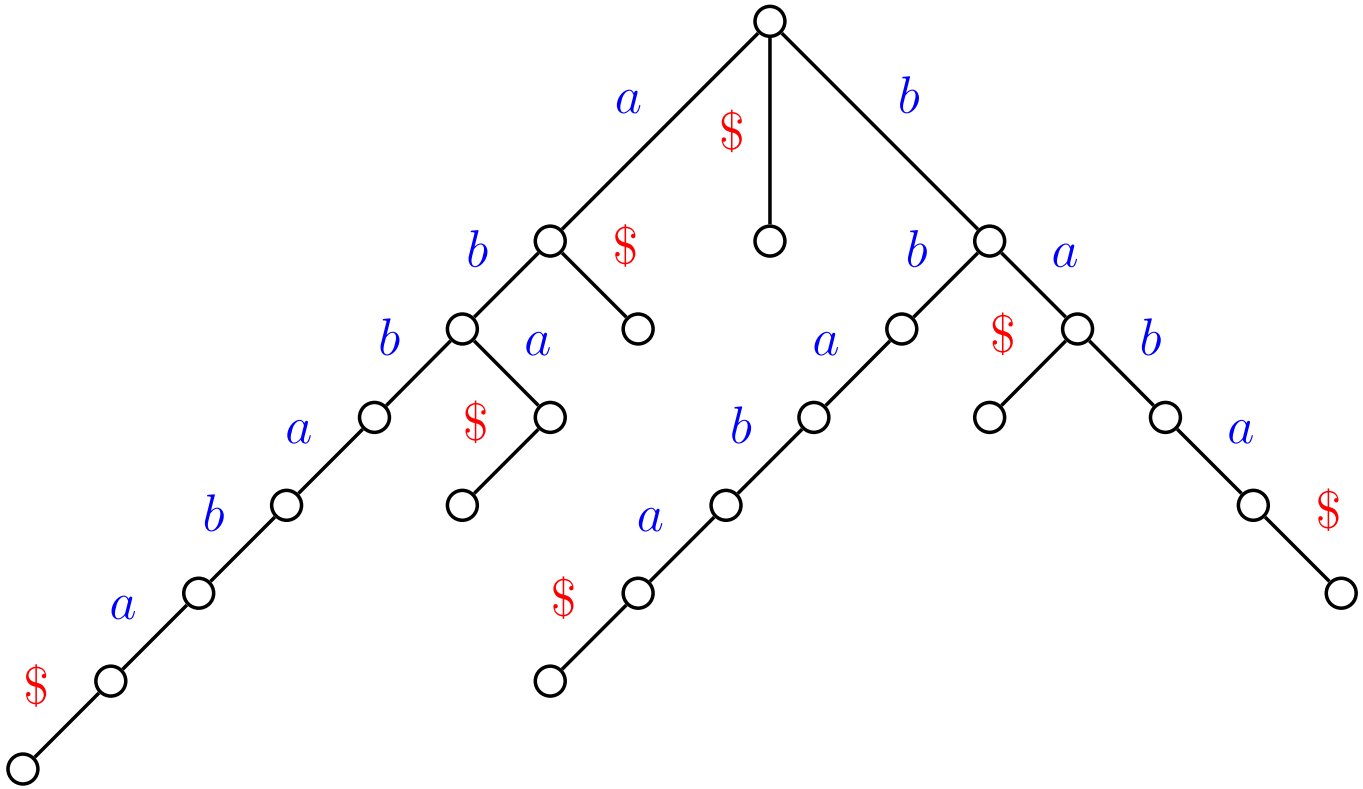
Definition

Ein Suffix-Trie für ein Wort $t \in \Sigma^*$ ist ein Trie für alle Suffixe von $t = t_1 \cdots t_n$, d.h. $S = \{t_i \cdots t_n : i \in [1 : n+1]\}$, wobei $t_{n+1} \cdots t_n = \epsilon$.

In der Literatur werden Suffix-Tries oft auch als atomare Suffix-Bäume bezeichnet.

Suffix Trie: Beispiel mit Sentinel

Suffix-Trie für das Wort $t = abbaba\$$



Damit kein Suffix von t ein Präfix eines anderen Suffixes ist und somit jedes Suffix von t in einem Blatt endet, hängt man an das Ende von t oft ein Sonderzeichen $ (Sentinel, $\$ \notin \Sigma$) an.

Suffix-Trie: Aufbau

Der Suffix-Trie kann so aufgebaut werden, dass nach und nach die einzelnen Suffixe von t in den Trie eingefügt werden, beginnend mit dem längsten Suffix.

Algorithmus 10 : BuildSuffixTrie(char $t[]$, int n)

tree $T = empty()$;
for $(i := 1; i \leq n; i{+}{+})$ **do**
  insert(T, $t_i \cdots t_n$);

Suffix-Trie: Laufzeitanalyse

- charakteristische Operationen: besuchte und modifizierte Knoten der zugrunde liegenden Bäume
- Einfügen des Suffixes $t_i \cdots t_n$ benötigt genau $O(|t_i \cdots t_n|) = O(n - i + 1)$ Operationen
- Laufzeit:

$$\sum_{i=1}^{n}(n - i + 1) \;=\; \sum_{i=1}^{n} i \;=\; O(n^2)$$

Fakt

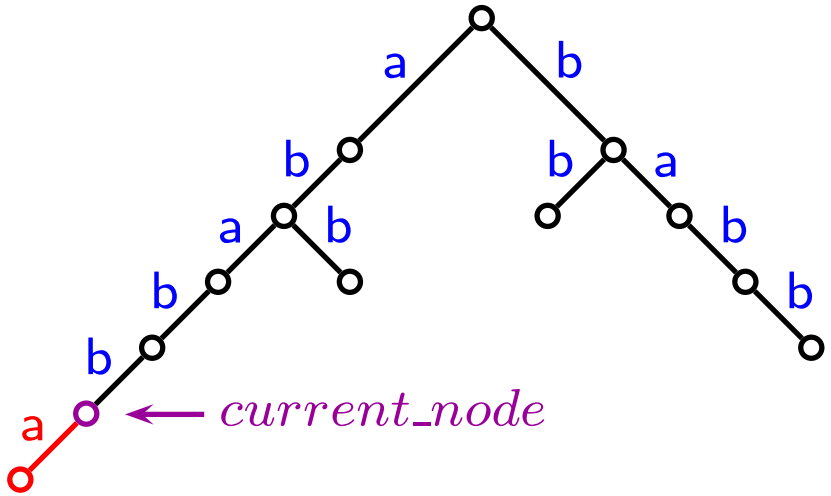
Sei $t \in \Sigma^*$, dann ist $w \in \Sigma^*$ genau dann ein Teilwort von t, wenn w ein Präfix eines Suffixes von t ist.

Suffix-Trie: Suche

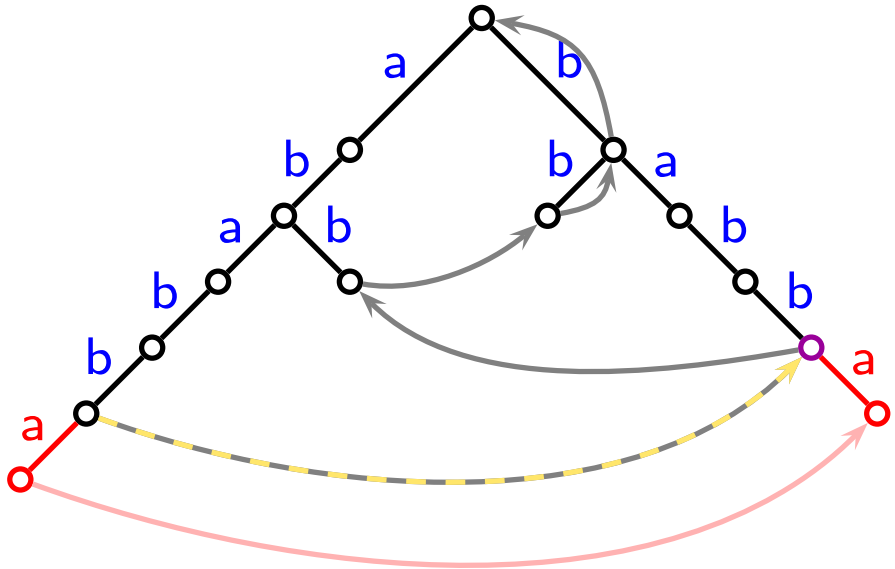
- Suche in Suffix-Tries nach einem Teilwort w von t
- Ablaufen der Buchstabenfolge $(w_1, \ldots, w_{|w|})$ im Suffix Trie
- Sobald man auf einen Knoten trifft, von dem man nicht mehr weiter suchen kann, weiß man, dass w nicht in t enthalten sein kann.
- Andernfalls ist w ein Teilwort von t.

Online-Konstruktion

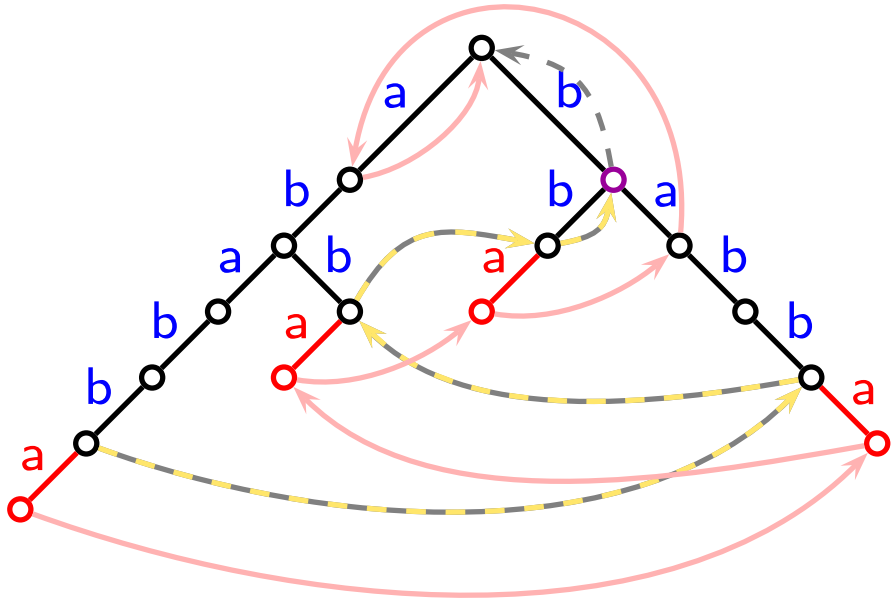
- Sei $t \in \Sigma^n$ und sei T^i der Suffix-Trie für $t_1..t_i$.
- Ziel ist es nun, den Suffix-Trie T^i aus aus dem Trie T^{i-1} zu konstruieren.
- T^0 ist einfach zu konstruieren, da $t_1..t_0 = \epsilon$ ist und somit T^0 der Baum ist, der aus nur einem Knoten (der Wurzel) besteht.
- Im Folgenden ist der sukzessive Aufbau eines Suffix-Tries für *ababba* aus dem Suffix-Trie für *ababb* ausführlich beschrieben.
- Dabei wird auch von so genannten Suffix-Links Gebrauch gemacht, die gleich noch formal eingeführt werden.
- Beispiel: $t = ababbaa$; Konstruktion von T^6 aus T^5



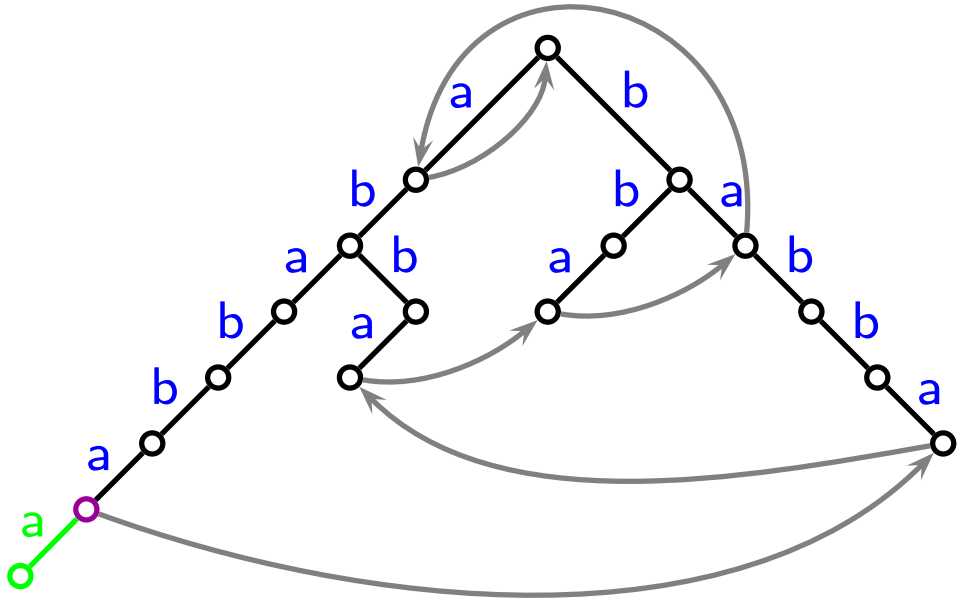
Der Trie für $t = ababb$ ist bereits konstruiert. An den Knoten, der am Ende des Pfades steht, dessen Labels das bisher längste Suffix ergibt, muss nun zuerst das neue Zeichen a angehängt werden. Anschließend muss an jedes kürzere Präfix ein a angehängt werden.



Um schnell den Knoten zu finden, der das nächst kürzere Suffix repräsentiert, sind im Trie so genannte Suffix-Links vorhanden. Diese zeigen immer auf den Knoten, der das nächst kürzere Suffix darstellt. Folge nun dem Suffix-Link des aktuellen Knotens und füge an dessen Endknoten wiederum die neue a-Kante ein und aktualisiere den Suffix-Link.



Der zuvor beschriebene Vorgang wird nun so lange wiederholt, bis man auf einen Knoten trifft, der bereits eine a-Kante besitzt. Nun muss natürlich kein neuer Knoten mehr eingefügt werden, da dieser ja bereits existiert. Allerdings müssen noch die restlichen Suffix-Links aktualisiert werden.



Nachdem T^6 vollständig aufgebaut ist, beginnt das ganze Spiel von vorne für den nächsten Buchstaben, im Beispiel a.