Algorithmen II
Vorlesung am 12.12.2013

String-Matching

Institut für Theoretische Informatik · Prof. Dr. Dorothea Wagner

String-Matching – Einführung

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Definition & Motivation

Problem: String-Matching
Seien P und T Zeichenfolgen mit Zeichen aus dem Alphabet Σ, wobei $|P| < |T|$.
Finde alle Vorkommen von P (Muster, engl. Pattern) in T (Text).

Beispiel:
Das Muster P = cbc taucht im Text T = acbccabcbcbcacb an den Stellen 1, 7 und 9 auf.

T	a	c	b	c	c	a	b	c	b	c	b	c	a	c	b
	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
P		c	b	c				c	b	c					
										c	b	c			
		(1)						(7)		(9)					

Anwendungsbeispiele:

- Suche nach Textstellen in einem Textdokument.
- Suche nach einer bestimmten Sequenz von Basenpaaren in einer DNA.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Ein einfacher Algorithmus

Problem: String-Matching
Seien P und T Zeichenfolgen mit Zeichen aus dem Alphabet Σ, wobei $|P| < |T|$.
Finde alle Vorkommen von P (Muster (engl. Pattern)) in T (Text).

Definition: Vorkommen im Text
Sei $|T| = n$ und $|P| = m$. Bezeichne mit $T[i]$ für $0 \leq i < n$ ($P[j]$ für $0 \leq j < m$) das i-te (j-te) Zeichen in T (P).
Das Muster P *kommt in T an der Stelle s vor*, wenn $T[s+j] = P[j]$ für $0 \leq j < m$.

Ein einfacher Algorithmus

Problem: String-Matching
Seien P und T Zeichenfolgen mit Zeichen aus dem Alphabet Σ, wobei $|P| < |T|$.
Finde alle Vorkommen von P (Muster (engl. Pattern)) in T (Text).

Definition: Vorkommen im Text
Sei $|T| = n$ und $|P| = m$. Bezeichne mit $T[i]$ für $0 \leq i < n$ ($P[j]$ für $0 \leq j < m$) das i-te (j-te) Zeichen in T (P).
Das Muster P *kommt in* T *an der Stelle* s *vor*, wenn $T[s+j] = P[j]$ für $0 \leq j < m$.

NAIVER STRING-MATCHER(T, P) — $O((n-m) \cdot m)$

$(n, m) \leftarrow (|T|, |P|)$

for $s = 0$ **to** $n - m$ **do** — $O((n-m) \cdot m)$

  MATCH $\leftarrow$ TRUE — $O(m)$

  for $j = 0$ **to** $m - 1$ **do**

    if $T[s+j] \neq P[j]$ **then** match $\leftarrow$ FALSE

  if MATCH **then** gib aus: „Muster P taucht an Stelle s in T auf“

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Eine untere Schranke

Beobachtung: Untere Schranke
Um String-Matching zu lösen werden mindestens $\Omega(n + m)$ Schritte benötigt.

Begründung:
Ein korrekter Algorithmus muss sich wenigstens T und P komplett anschauen.

Einschränkung:
Sucht man im gleichen Text nach mehreren Mustern (oder umgekehrt), so kann man durch Vorberechnung die einzelnen Anfragen beschleunigen.

String-Matching – Rabin & Karp (1981)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rabin & Karp – Idee

Annahme:
Für das Alphabet gilt $\Sigma = \{0, 1, \ldots, 9\}$.

Bemerkung: Das ist keine echte Einschränkung, da im allgemeinen Fall jeder String als Zahl in d-ärer Darstellung mit $d = |\Sigma|$ aufgefasst werden kann.

Interpretation als Zahl:

- Bezeichne den durch P repräsentierten Zahlenwert mit p.
- Sei t_s die durch den Teilstring $T[s]\,T[s+1]\ldots T[s+m]$ repräsentierte Zahl.

Es gilt: $p = t_s$ genau dann, wenn $P[j] = T[s+j]$ für alle $0 \leq j < m$.

Rabin & Karp – Idee

Annahme:
Für das Alphabet gilt $\Sigma = \{0, 1, \ldots, 9\}$.

Bemerkung: Das ist keine echte Einschränkung, da im allgemeinen Fall jeder String als Zahl in d-ärer Darstellung mit $d = |\Sigma|$ aufgefasst werden kann.

Interpretation als Zahl:

- Bezeichne den durch P repräsentierten Zahlenwert mit p.
- Sei t_s die durch den Teilstring $T[s]\,T[s+1]\ldots T[s+m]$ repräsentierte Zahl.

Es gilt: $p = t_s$ genau dann, wenn $P[j] = T[s+j]$ für alle $0 \leq j < m$.

Der Algorithmus von Rabin & Karp:

- **Idee:** Der Vergleich von zwei Zahlen ($p = t_s$) kann in konstanter Zeit ausgeführt werden (Vergleich zweier Integers), wenn die Zahlen nicht zu groß sind.
- **Problem:** p und t_s haben $O(m)$ Bits $\rightarrow$ Vergleich braucht $O(m)$ Zeit wie beim naiven Algorithmus.
- **Trick:** Berechne p und t_s modulo einer geeigneten Zahl q und vergleiche p und t_s nur dann, wenn ihrer Reste bezüglich q gleich sind.

Rabin & Karp – Beispiel

$T = 2359023141526739921$, $P = 31415$, $q = 13$
$p = 31415 = 2416 \cdot 13 + \boxed{7}$

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$t_0 = 23590$
$= 1814 \cdot 13 + \boxed{8}$

Es gilt:

- $(t_0 \bmod q) = 8 \neq 7 = (p \bmod q)$
  $\Rightarrow t_0 \neq p$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rabin & Karp – Beispiel

$T = 2359023141526739921$, $P = 31415$, $q = 13$

$p = 31415 = 2416 \cdot 13 + \boxed{7}$

2 | 3 5 9 0 2 | 3 1 4 1 5 2 6 7 3 9 9 2 1

$t_1 = 35902$
$= 2761 \cdot 13 + \boxed{9}$

Es gilt:

- $(t_1 \bmod q) = 9 \neq 7 = (p \bmod q)$
  $\Rightarrow t_1 \neq p$

Rabin & Karp – Beispiel

$T = 2359023141526739921$, $P = 31415$, $q = 13$
$p = 31415 = 2416 \cdot 13 + \boxed{7}$

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$t_1 = 35902 = 2761 \cdot 13 + \boxed{9}$

$t_6 = 31415 = 2416 \cdot 13 + \boxed{7}$

Es gilt:

- $(t_1 \bmod q) = 9 \neq 7 = (p \bmod q)$
  $\Rightarrow t_1 \neq p$
- $(t_6 \bmod q) = 7 = (p \bmod q)$
  $\Rightarrow t_6$ und p könnten gleich sein (und sind es auch)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rabin & Karp – Beispiel

$T = 2359023141526739921$, $P = 31415$, $q = 13$
$p = 31415 = 2416 \cdot 13 + \boxed{7}$

2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1

$t_1 = 35902 = 2761 \cdot 13 + \boxed{9}$

$t_6 = 31415 = 2416 \cdot 13 + \boxed{7}$

$t_{12} = 67399 = 5184 \cdot 13 + \boxed{7}$

Es gilt:

- $(t_1 \bmod q) = 9 \neq 7 = (p \bmod q)$
  $\Rightarrow t_1 \neq p$
- $(t_6 \bmod q) = 7 = (p \bmod q)$
  $\Rightarrow t_6$ und p könnten gleich sein (und sind es auch)
- $(t_{12} \bmod q) = 7 = (p \bmod q)$
  $\Rightarrow t_{12}$ und p könnten gleich sein (sind es aber nicht)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rechnen mit dem Rest

Wie kann der Rest einer sehr großen Zahl p bezüglich q berechnet werden?
Wie können die t_s bzw. die Reste bezüglich q effizient berechnet werden?

Erinnerung: Rechenregeln

Es gelten folgende Rechenregeln:

$(a + b) \bmod q = \big((a \bmod q) + (b \bmod q)\big) \bmod q$

$(a \cdot b) \bmod q = \big((a \bmod q) \cdot (b \bmod q)\big) \bmod q$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Rechnen mit dem Rest

Wie kann der Rest einer sehr großen Zahl p bezüglich q berechnet werden?
Wie können die t_s bzw. die Reste bezüglich q effizient berechnet werden?

Erinnerung: Rechenregeln
Es gelten folgende Rechenregeln:
$(a + b) \bmod q = \big((a \bmod q) + (b \bmod q)\big) \bmod q$
$(a \cdot b) \bmod q = \big((a \bmod q) \cdot (b \bmod q)\big) \bmod q$

Damit kann $p \bmod q$ in $O(m)$ Zeit berechnet werden.

Beispiel: Berechnung von 31415 mod 13 (31415 = 31000 + 400 + 10 + 5)

31 mod 13 = 5

Rechnen mit dem Rest

Wie kann der Rest einer sehr großen Zahl p bezüglich q berechnet werden?
Wie können die t_s bzw. die Reste bezüglich q effizient berechnet werden?

Erinnerung: Rechenregeln
Es gelten folgende Rechenregeln:
$(a + b) \bmod q = \big((a \bmod q) + (b \bmod q)\big) \bmod q$
$(a \cdot b) \bmod q = \big((a \bmod q) \cdot (b \bmod q)\big) \bmod q$

Damit kann $p \bmod q$ in $O(m)$ Zeit berechnet werden.

Beispiel: Berechnung von 31415 mod 13 $(31415 = 31000 + 400 + 10 + 5)$

$31 \bmod 13 = 5$ $\Rightarrow 310 \bmod 13 = (10 \cdot 5) \bmod 13 = 11$

$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2$

Rechnen mit dem Rest

Wie kann der Rest einer sehr großen Zahl p bezüglich q berechnet werden?
Wie können die t_s bzw. die Reste bezüglich q effizient berechnet werden?

Erinnerung: Rechenregeln
Es gelten folgende Rechenregeln:
$(a + b) \bmod q = \big((a \bmod q) + (b \bmod q)\big) \bmod q$
$(a \cdot b) \bmod q = \big((a \bmod q) \cdot (b \bmod q)\big) \bmod q$

Damit kann p mod q in $O(m)$ Zeit berechnet werden.

Beispiel: Berechnung von 31415 mod 13 (31415 = 31000 + 400 + 10 + 5)

$31 \bmod 13 = 5$
$\Rightarrow 310 \bmod 13 = (10 \cdot 5) \bmod 13 = 11$
$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2$
$\Rightarrow 3140 \bmod 13 = (10 \cdot 2) \bmod 13 = 7$
$\Rightarrow 3141 \bmod 13 = (7 + 1) \bmod 13 = 8$

Rechnen mit dem Rest

Wie kann der Rest einer sehr großen Zahl p bezüglich q berechnet werden?
Wie können die t_s bzw. die Reste bezüglich q effizient berechnet werden?

Erinnerung: Rechenregeln

Es gelten folgende Rechenregeln:

$(a + b) \bmod q = \big((a \bmod q) + (b \bmod q)\big) \bmod q$

$(a \cdot b) \bmod q = \big((a \bmod q) \cdot (b \bmod q)\big) \bmod q$

Damit kann $p \bmod q$ in $O(m)$ Zeit berechnet werden.

Beispiel: Berechnung von $31415 \bmod 13$ $(31415 = 31000 + 400 + 10 + 5)$

$31 \bmod 13 = 5$

$\Rightarrow 310 \bmod 13 = (10 \cdot 5) \bmod 13 = 11$

$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2$

$\Rightarrow 3140 \bmod 13 = (10 \cdot 2) \bmod 13 = 7$

$\Rightarrow 3141 \bmod 13 = (7 + 1) \bmod 13 = 8$

$\Rightarrow 31410 \bmod 13 = (10 \cdot 8) \bmod 13 = 2$

$\Rightarrow 31415 \bmod 13 = (2 + 5) \bmod 13 = 7$

Rechnen mit dem Rest

Wie kann der Rest einer sehr großen Zahl p bezüglich q berechnet werden?
Wie können die t_s bzw. die Reste bezüglich q effizient berechnet werden?

Erinnerung: Rechenregeln
Es gelten folgende Rechenregeln:
$(a+b) \bmod q = \big((a \bmod q) + (b \bmod q)\big) \bmod q$
$(a \cdot b) \bmod q = \big((a \bmod q) \cdot (b \bmod q)\big) \bmod q$

Damit kann $p \bmod q$ in $O(m)$ Zeit berechnet werden.

Beispiel: Berechnung von 31415 mod 13 (31415 = 31000 + 400 + 10 + 5)

$31 \bmod 13 = 5$ $\Rightarrow 310 \bmod 13 = (10 \cdot 5) \bmod 13 = 11$
$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2$ $\Rightarrow 3140 \bmod 13 = (10 \cdot 2) \bmod 13 = 7$
$\Rightarrow 3141 \bmod 13 = (7 + 1) \bmod 13 = 8$ $\Rightarrow 31410 \bmod 13 = (10 \cdot 8) \bmod 13 = 2$
$\Rightarrow 31415 \bmod 13 = (2 + 5) \bmod 13 = 7$

Außerdem gilt: $t_{s+1} = \underbrace{\left(t_s - 10^{m-1} \cdot T\left[s\right]\right)}_{\text{erste Stelle Löschen}} \underbrace{\cdot 10 + T\left[s+m\right]}_{\text{nächste Stelle anfügen}}$

$\Rightarrow t_{s+1} \bmod q$ kann aus $t_s \bmod q$ in $O(1)$ Zeit berechnet werden

(unter der Voraussetzung, dass $10^{m-1} \bmod q$ schon berechnet und q klein ist)

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der Algorithmus von Rabin & Karp

RABIN-KARP-MATCHER(T, P, q)

$(n, m) \leftarrow (|T|, |P|)$

$\widehat{p} \leftarrow p \bmod q$

$\widehat{t_0} \leftarrow t_0 \bmod q$

for $s = 0$ **to** $n - m$ **do**

  if $\widehat{t_s} = \widehat{p}$ **then**

    MATCH $\leftarrow$ TRUE

    for $j = 0$ **to** $m - 1$ **do**

      if $T[s + j] \neq P[j]$ **then** match $\leftarrow$ FALSE

    if MATCH **then** gib aus: „Muster P taucht an Stelle s in T auf“

  $\widehat{t_{s+1}} \leftarrow$ Berechne $\widehat{t_{s+1}}$ aus $\widehat{t_s}$

Der Algorithmus von Rabin & Karp

RABIN-KARP-MATCHER(T, P, q)

$(n, m) \leftarrow (|T|, |P|)$

$\widehat{p} \leftarrow p \bmod q$

$\widehat{t_0} \leftarrow t_0 \bmod q$

for $s = 0$ **to** $n - m$ **do**

if $\widehat{t_s} = \widehat{p}$ **then** $O(1)$ für Vergleich $O(m)$

MATCH $\leftarrow$ TRUE $O(m)$

for $j = 0$ **to** $m - 1$ **do**

if $T[s+j] \neq P[j]$ **then** match $\leftarrow$ FALSE

if MATCH **then** gib aus: „Muster P taucht an Stelle s in T auf“

$\widehat{t_{s+1}} \leftarrow$ Berechne $\widehat{t_{s+1}}$ aus $\widehat{t_s}$

Der Algorithmus von Rabin & Karp

RABIN-KARP-MATCHER(T, P, q)

$(n, m) \leftarrow (|T|, |P|)$

$\widehat{p} \leftarrow p \bmod q$

$\widehat{t_0} \leftarrow t_0 \bmod q$

for $s = 0$ **to** $n - m$ **do**

if $\widehat{t_s} = \widehat{p}$ **then** $O(1)$ für Vergleich $O(m)$

MATCH $\leftarrow$ TRUE $O(m)$

for $j = 0$ **to** $m - 1$ **do**

if $T[s + j] \neq P[j]$ **then** match $\leftarrow$ FALSE

if MATCH **then** gib aus: „Muster P taucht an Stelle s in T auf“

$\widehat{t_{s+1}} \leftarrow$ Berechne $\widehat{t_{s+1}}$ aus $\widehat{t_s}$ $O(1)$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der Algorithmus von Rabin & Karp

Karlsruher Institut für Technologie

RABIN-KARP-MATCHER(T, P, q) $O((n-m)\cdot m)$

$(n, m) \leftarrow (|T|, |P|)$ $O((n-m)\cdot m)$

$\widehat{p} \leftarrow p \bmod q$

$\widehat{t_0} \leftarrow t_0 \bmod q$

for $s = 0$ **to** $n - m$ **do**

  if $\widehat{t_s} = \widehat{p}$ **then** $O(1)$ für Vergleich $O(m)$

    MATCH $\leftarrow$ TRUE $O(m)$

    for $j = 0$ **to** $m - 1$ **do**

      if $T[s+j] \neq P[j]$ **then** match $\leftarrow$ FALSE

    if MATCH **then** gib aus: „Muster P taucht an Stelle s in T auf"

  $\widehat{t_{s+1}} \leftarrow$ Berechne $\widehat{t_{s+1}}$ aus $\widehat{t_s}$ $O(1)$

Worst Case: Laufzeit $O((n-m)\cdot m)$

Der Algorithmus von Rabin & Karp

Karlsruher Institut für Technologie

RABIN-KARP-MATCHER(T, P, q) $O((n-m)\cdot m)$

$(n, m) \leftarrow (|T|, |P|)$ $O((n-m)\cdot m)$

$\widehat{p} \leftarrow p \bmod q$

$\widehat{t_0} \leftarrow t_0 \bmod q$

for $s = 0$ **to** $n-m$ **do**

    if $\widehat{t_s} = \widehat{p}$ **then** $O(1)$ für Vergleich $O(m)$

        MATCH $\leftarrow$ TRUE $O(m)$

        for $j = 0$ **to** $m-1$ **do**

            if $T[s+j] \neq P[j]$ **then** match $\leftarrow$ FALSE

        if MATCH **then** gib aus: „Muster P taucht an Stelle s in T auf“

    $\widehat{t_{s+1}} \leftarrow$ Berechne $\widehat{t_{s+1}}$ aus $\widehat{t_s}$ $O(1)$

Worst Case: Laufzeit $O((n-m)\cdot m)$

Unterschied zu naivem Algorithmus: Der $O(m)$ teure Vergleich wird nur gemacht, wenn t_s und p modulo q gleich sind.

Welche Laufzeit kann man erwarten?

Der Algorithmus von Rabin & Karp

Welche Laufzeit kann man erwarten?

Annahme: Die Berechnung $x \bmod q$ ist eine „Zufallsabbildung“ von $\Sigma^\star$ nach $\mathbb{Z}_q$.

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der Algorithmus von Rabin & Karp

Welche Laufzeit kann man erwarten?

Annahme: Die Berechnung $x \bmod q$ ist eine „Zufallsabbildung" von $\Sigma^\star$ nach $\mathbb{Z}_q$.

Unzulässige Übereinstimmungen: $t_s \neq p$ aber $\widehat{t_s} = \widehat{p}$
Die Wahrscheinlichkeit für eine unzulässige Übereinstimmung ist $\frac{1}{q}$.
$\Rightarrow$ Die erwartete Anzahl von unzulässigen Übereinstimmungen ist in $O\left(\frac{n}{q}\right)$.

Zulässige Übereinstimmungen: $t_s = p$ und damit auch $\widehat{t_s} = \widehat{p}$
Sei v die Anzahl zulässiger Übereinstimmungen.

Der Algorithmus von Rabin & Karp

Welche Laufzeit kann man erwarten?

Annahme: Die Berechnung x mod q ist eine „Zufallsabbildung“ von $\Sigma^\star$ nach $\mathbb{Z}_q$.

Unzulässige Übereinstimmungen: $t_s \neq p$ aber $\widehat{t_s} = \widehat{p}$
Die Wahrscheinlichkeit für eine unzulässige Übereinstimmung ist $\frac{1}{q}$.
$\Rightarrow$ Die erwartete Anzahl von unzulässigen Übereinstimmungen ist in $O\left(\frac{n}{q}\right)$.

Zulässige Übereinstimmungen: $t_s = p$ und damit auch $\widehat{t_s} = \widehat{p}$
Sei v die Anzahl zulässiger Übereinstimmungen.

$\Rightarrow$ Erwartete Laufzeit („average-case“): $O\Big(m + n + m \cdot \left(v + \frac{n}{q}\right)\Big)$

$\Rightarrow$ $O(n+m)$ falls v klein (konstant) und $q > m$.

Wahl von q: Möglichst große Primzahl, sodass alle Berechnungen noch im Integer Bereich möglich sind.

String-Matching mit endlichen Automaten

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Idee

1. Baue einen endlichen Automation $\mathcal{A}_P$ (bezüglich des Musters P), sodass:
   - $\mathcal{A}_P$ akzeptiert genau die Wörter, die P als Suffix haben (also auf P enden).
2. Führe $\mathcal{A}_P$ mit dem Text T als Eingabe aus.
   - Nach jedem Vorkommen von P in T ist $\mathcal{A}_P$ in einem akzeptierenden Zustand.

Definition: Endlicher Automat **(Definition 7.1)**

Ein *endlicher Automat* $\mathcal{A}$ ist ein Tupel $(Q, q_0, A, \Sigma, \delta)$, mit

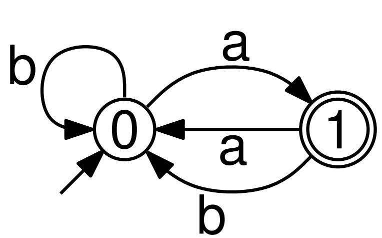
$$
\begin{aligned}
Q &\quad - \quad \text{endliche Menge von } \textit{Zuständen}\\
q_0 \in Q &\quad - \quad \textit{Startzustand}\\
A \subseteq Q &\quad - \quad \text{Menge von } \textit{akzeptierenden Zuständen}\\
\Sigma &\quad - \quad \text{endliches } \textit{Eingabealphabet}\\
\delta\colon Q \times \Sigma \longrightarrow Q &\quad - \quad \textit{Übertragungsfunktion}
\end{aligned}
$$

Beispiel:

$Q = \{0, 1\}$, $q_0 = 0$, $A = \{1\}$, $\Sigma = \{\mathrm{a}, \mathrm{b}\}$ und

$\delta(0, a) = 1$, $\delta(0, b) = 0$,

$\delta(1, a) = 0$, $\delta(1, b) = 0$



Akzeptiert Wörter, die auf ungerade Anzahl a's enden.

Grundlegende Definitionen

Definition: Zustandsfunktion
Ein endlicher Automat $\mathcal{A}$ induziert eine *Zustandsfunktion* $\mathcal{C} : \Sigma^\star \longrightarrow Q$, mit $\mathcal{C}(w)$ ist der Zustand von $\mathcal{A}$ nach Lesen des Wortes w. Rekursive Definition:

$$
\begin{aligned}
\mathcal{C}(a) &= \delta(q_0, a) \text{ für } a \in \Sigma \\
\mathcal{C}(wa) &= \delta(\mathcal{C}(w), a) \text{ für } w \in \Sigma^\star \text{ und } a \in \Sigma
\end{aligned}
$$

Beachte: w wird von $\mathcal{A}$ akzeptiert, genau dann wenn $\mathcal{C}(w) \in A$.

Grundlegende Definitionen

Definition: Zustandsfunktion
Ein endlicher Automat $\mathcal{A}$ induziert eine *Zustandsfunktion* $\mathcal{C} : \Sigma^\star \longrightarrow Q$, mit $\mathcal{C}(w)$ ist der Zustand von $\mathcal{A}$ nach Lesen des Wortes w. Rekursive Definition:

$$
\begin{aligned}
\mathcal{C}(a) &= \delta(q_0, a) \text{ für } a \in \Sigma \\
\mathcal{C}(wa) &= \delta(\mathcal{C}(w), a) \text{ für } w \in \Sigma^\star \text{ und } a \in \Sigma
\end{aligned}
$$

Beachte: w wird von $\mathcal{A}$ akzeptiert, genau dann wenn $\mathcal{C}(w) \in A$.

Definition: Präfix, Suffix
Ein Wort w ist *Präfix* eines Wortes x, falls $x = wy$ für ein Wort y.
Ein Wort w ist *Suffix* eines Wortes x, falls $x = yw$ für ein Wort y.

Beispiel: „Algo“ ist ein Präfix von „Algorithmen II“, „en II“ ist ein Suffix.

Grundlegende Definitionen

Definition: Zustandsfunktion
Ein endlicher Automat $\mathcal{A}$ induziert eine *Zustandsfunktion* $\mathcal{C} : \Sigma^\star \longrightarrow Q$, mit $\mathcal{C}(w)$ ist der Zustand von $\mathcal{A}$ nach Lesen des Wortes w. Rekursive Definition:

$$\begin{aligned} \mathcal{C}(a) &= \delta(q_0, a) \text{ für } a \in \Sigma \\ \mathcal{C}(wa) &= \delta(\mathcal{C}(w), a) \text{ für } w \in \Sigma^\star \text{ und } a \in \Sigma \end{aligned}$$

Beachte: w wird von $\mathcal{A}$ akzeptiert, genau dann wenn $\mathcal{C}(w) \in A$.

Definition: Präfix, Suffix
Ein Wort w ist *Präfix* eines Wortes x, falls $x = wy$ für ein Wort y.
Ein Wort w ist *Suffix* eines Wortes x, falls $x = yw$ für ein Wort y.

Beispiel: „Algo“ ist ein Präfix von „Algorithmen II“, „en II“ ist ein Suffix.

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\text{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\text{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

Beispiel: Für P = ab gilt: $\text{suf}_P(\text{ccaca}) = 1$, $\text{suf}_P(\text{ccab}) = 2$, $\text{suf}_P(\text{abcc}) = 0$.

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

Beispiel: Für P = ab gilt: $\mathrm{suf}_P(\text{ccaca}) = 1$, $\mathrm{suf}_P(\text{ccab}) = 2$, $\mathrm{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\text{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\text{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

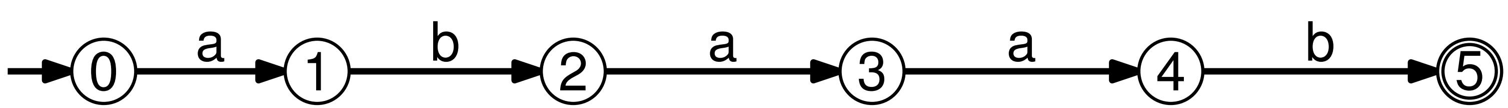
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\text{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \text{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: ... ...a ...ab ...aba ...abaa ...abaab

Der String-Matching-Automat

Definition: Suffixfunktion

Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

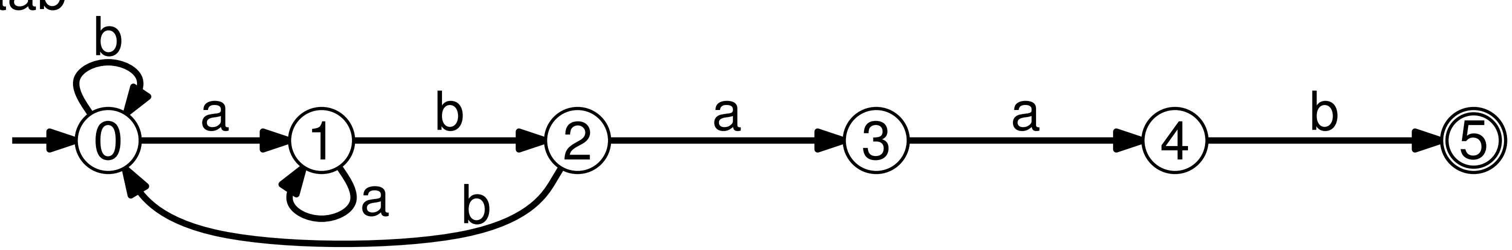
$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat

Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: ...   ...a   ...ab   ...aba   ...abaa   ...abaab

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\text{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\text{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

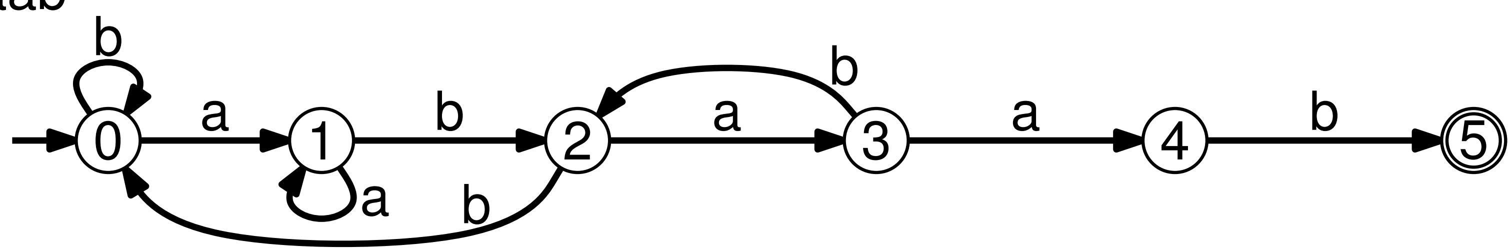
Beispiel: Für P = ab gilt: $\text{suf}_P(\text{ccaca}) = 1$, $\text{suf}_P(\text{ccab}) = 2$, $\text{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\text{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \text{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: … …a …ab …aba …abaa …abaab

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\text{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\text{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

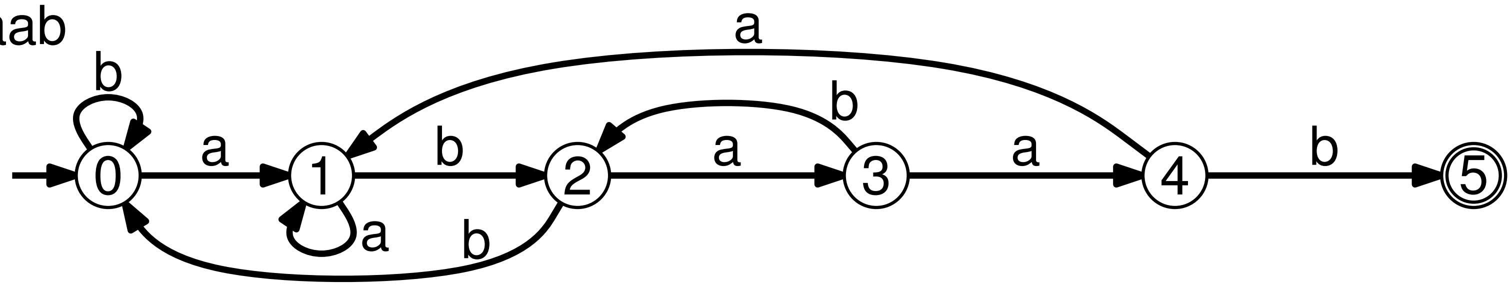
Beispiel: Für P = ab gilt: $\text{suf}_P(\text{ccaca}) = 1$, $\text{suf}_P(\text{ccab}) = 2$, $\text{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\text{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \text{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: … …a …ab …aba …abaa …abaab

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

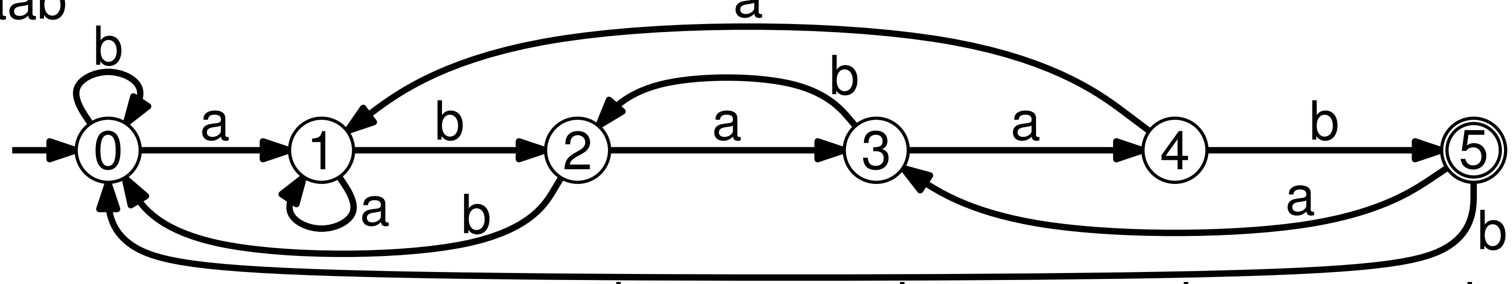
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: $\dots$ $\dots$a $\dots$ab $\dots$aba $\dots$abaa $\dots$abaab

Der String-Matching-Automat

Karlsruher Institut für Technologie

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P \colon \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

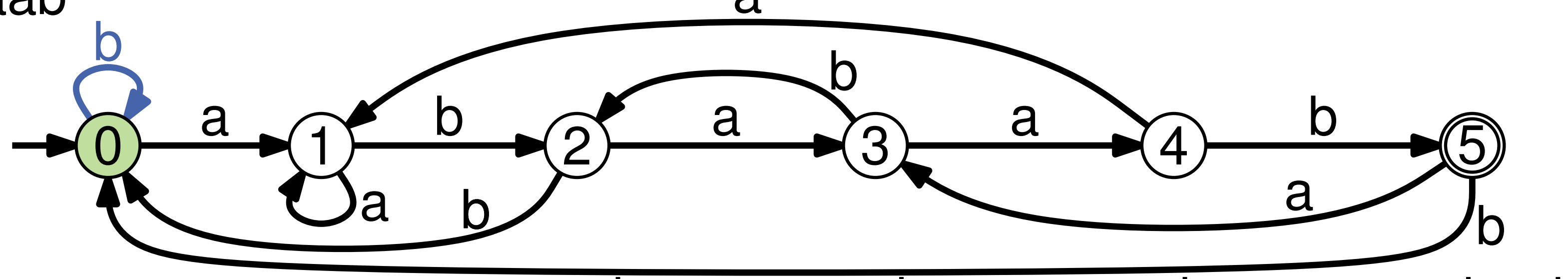
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: ... ...a ...ab ...aba ...abaa ...abaab

Ausführung für T = b a b a a b a a b b

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

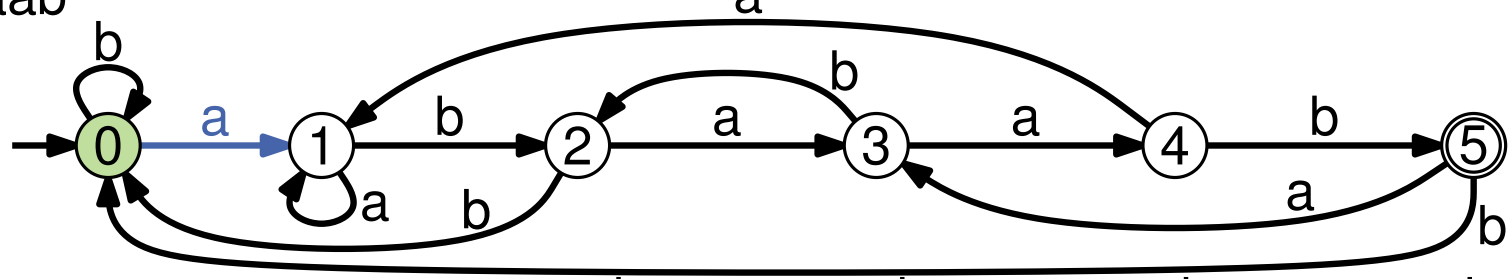
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: … …a …ab …aba …abaa …abaab

Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\text{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\text{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

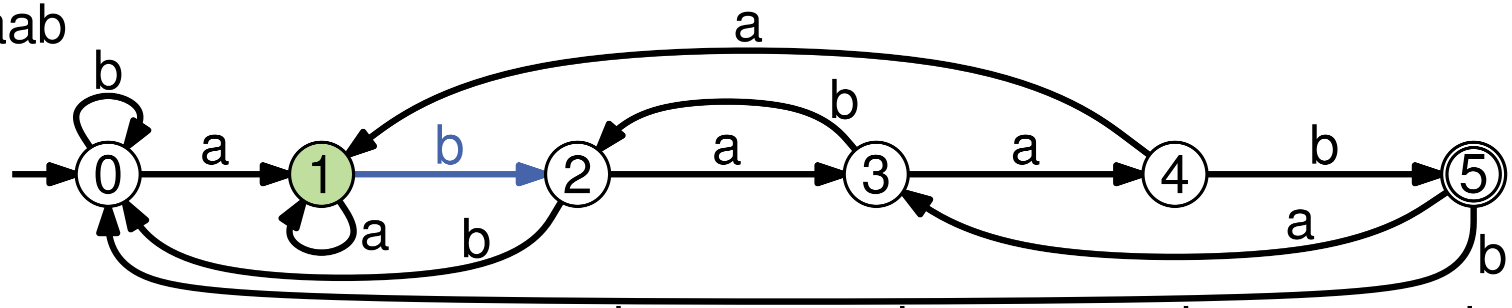
Beispiel: Für P = ab gilt: $\text{suf}_P(\text{ccaca}) = 1$, $\text{suf}_P(\text{ccab}) = 2$, $\text{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\text{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \text{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: $\ldots$ $\ldots$a $\ldots$ab $\ldots$aba $\ldots$abaa $\ldots$abaab

Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P\colon \Sigma^\star \longrightarrow \{0,\ldots,|P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

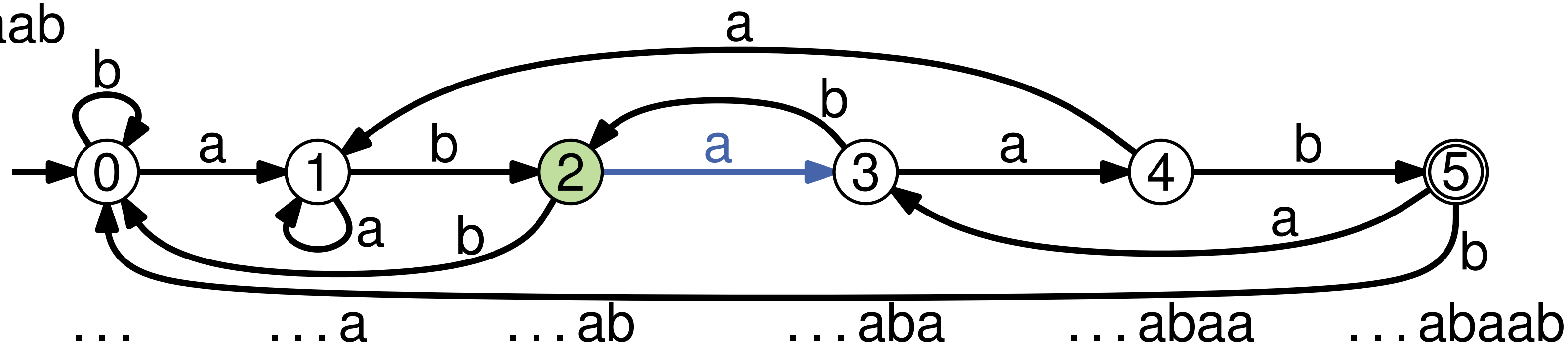
Beispiel: Für P = ab gilt: $\mathrm{suf}_P(\text{ccaca}) = 1$, $\mathrm{suf}_P(\text{ccab}) = 2$, $\mathrm{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1\ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P \colon \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

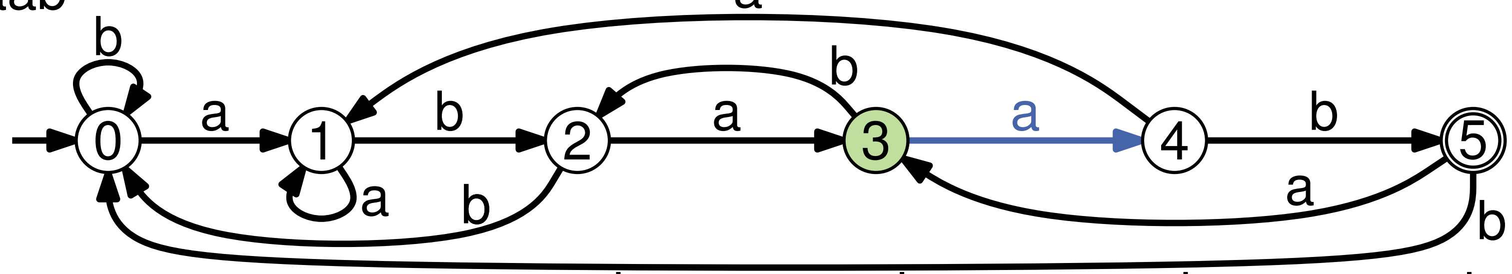
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: ...   ...a   ...ab   ...aba   ...abaa   ...abaab

Ausführung für T = b a b a a b a a b b

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\text{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\text{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

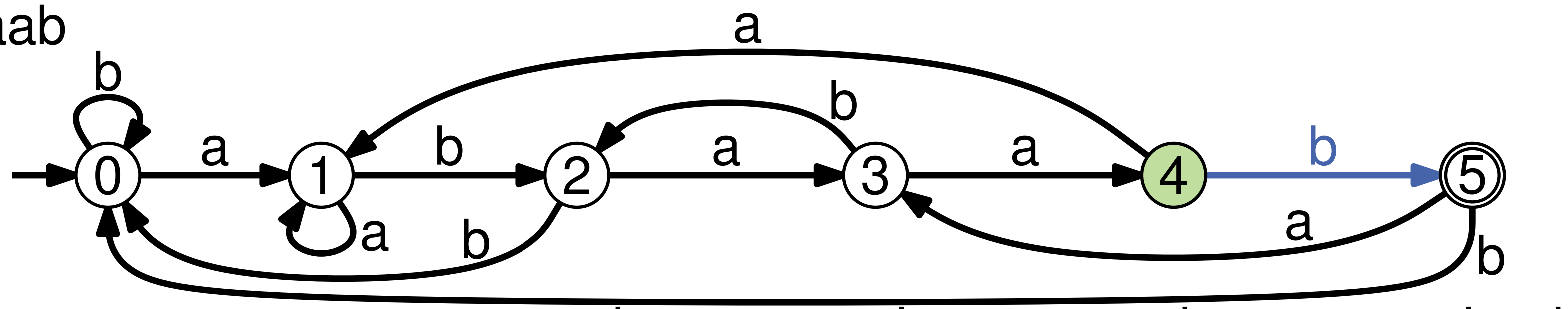
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\text{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \text{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: ... ...a ...ab ...aba ...abaa ...abaab

Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

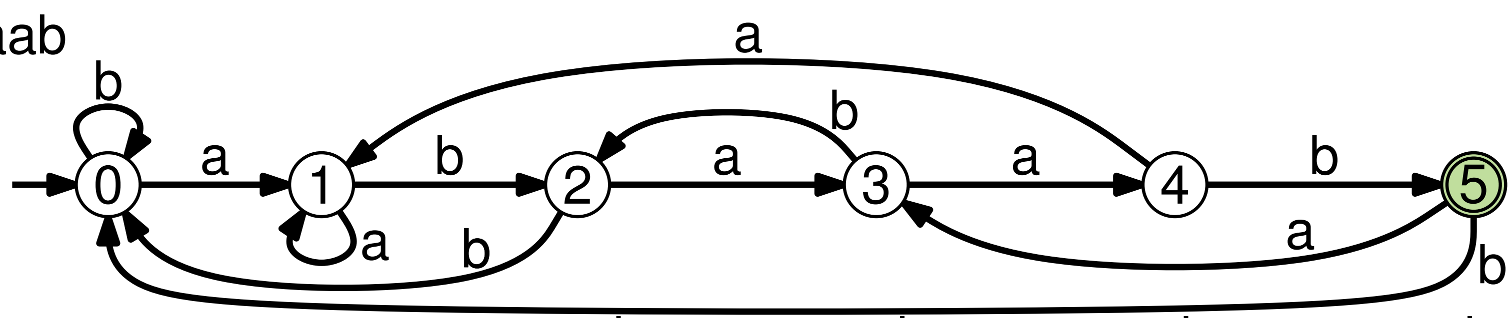
Beispiel: Für P = ab gilt: $\mathrm{suf}_P(\text{ccaca}) = 1$, $\mathrm{suf}_P(\text{ccab}) = 2$, $\mathrm{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: … | …a | …ab | …aba | …abaa | …abaab

Ausführung für T = b $\underbrace{\text{a b a a b}}$ a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Karlsruher Institut für Technologie

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

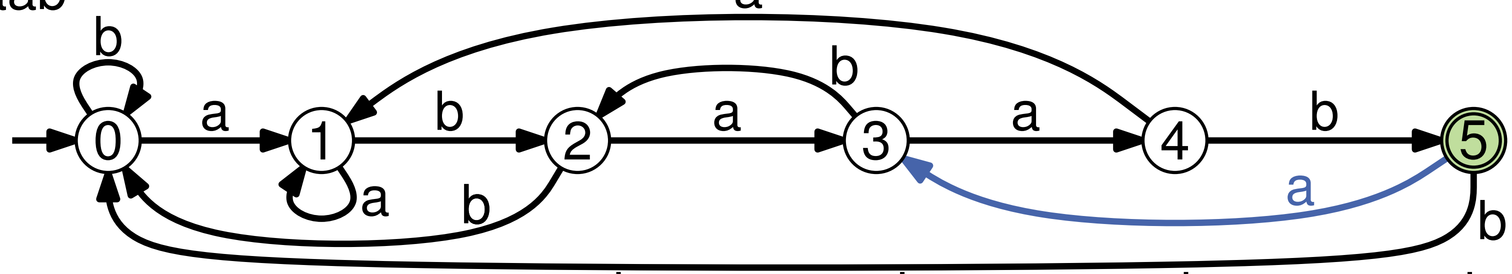
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: … …a …ab …aba …abaa …abaab

Ausführung für T = b $\underbrace{\text{a b a a b}}$ a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

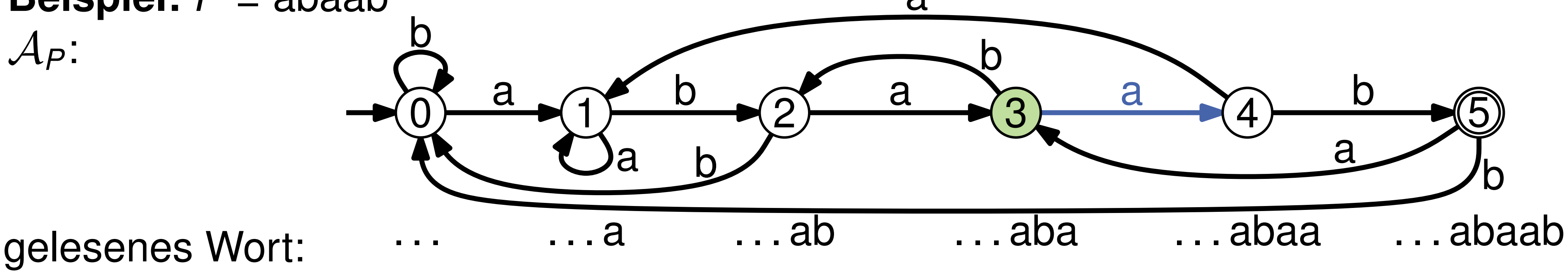
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

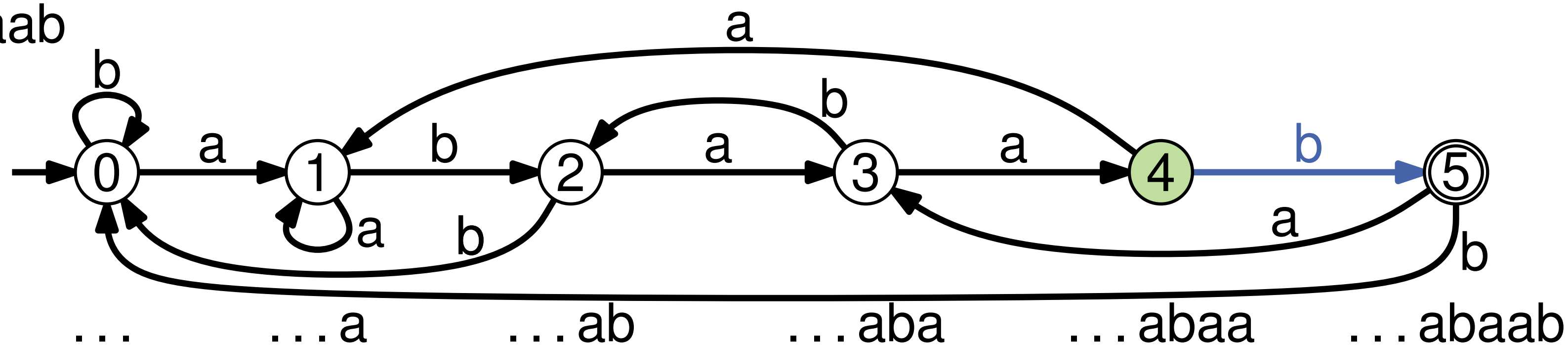
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P \colon \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

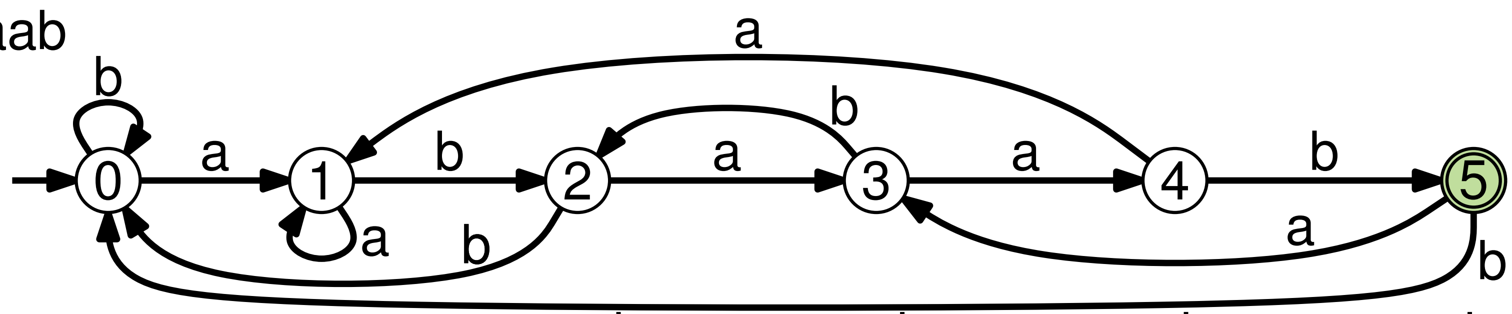
Beispiel: Für P = ab gilt: $\mathrm{suf}_P(\text{ccaca}) = 1$, $\mathrm{suf}_P(\text{ccab}) = 2$, $\mathrm{suf}_P(\text{abcc}) = 0$.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: $\ldots$ $\quad$ $\ldots$a $\quad$ $\ldots$ab $\quad$ $\ldots$aba $\quad$ $\ldots$abaa $\quad$ $\ldots$abaab

Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \dots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

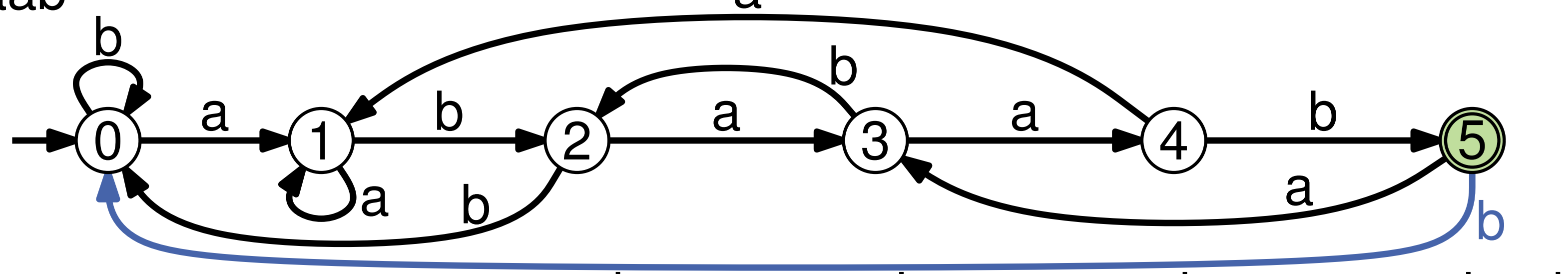
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \dots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \dots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: … …a …ab …aba …abaa …abaab

Ausführung für T = b a b a a b a a b b

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

Der String-Matching-Automat

Definition: Suffixfunktion
Zum Muster P definiere die *Suffixfunktion* $\mathrm{suf}_P : \Sigma^\star \longrightarrow \{0, \ldots, |P| = m\}$ wie folgt: $\mathrm{suf}_P(w)$ ist die Länge des maximalen Präfixes von P, das Suffix von w ist.

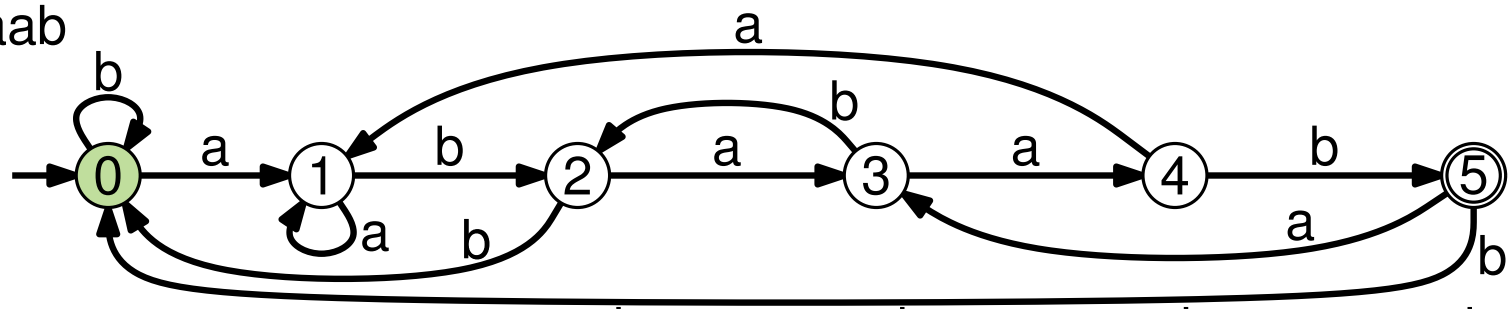
Beispiel: Für P = ab gilt: suf_P(ccaca) = 1, suf_P(ccab) = 2, suf_P(abcc) = 0.

$\Rightarrow$ Das Wort w hat P genau dann als Suffix, wenn $\mathrm{suf}_P(w) = m$.

Definition: String-Matching-Automat
Der *String-Matching-Automat* $\mathcal{A}_P$ zu P ist definiert durch $Q = \{0, 1, \ldots, m\}$, $q_0 = 0$, $A = \{m\}$ und $\delta(q, a) = \mathrm{suf}_P(P[1 \ldots q]a)$.

Beispiel: P = abaab

$\mathcal{A}_P$:



gelesenes Wort: . . .   . . . a   . . . ab   . . . aba   . . . abaa   . . . abaab

Ausführung für T = b a b a a b a a b b

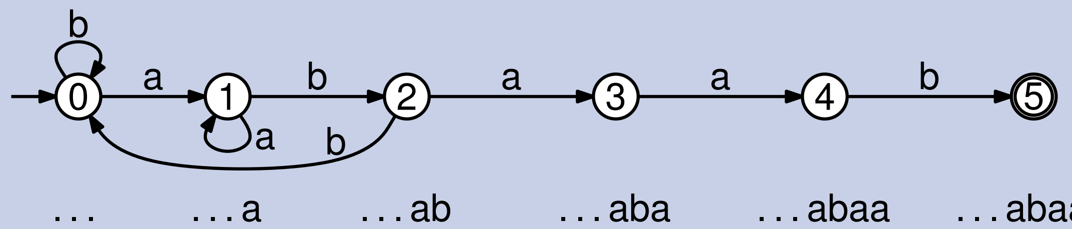
Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat

ÜBERGANGSFUNKTION(P, Σ)

$m \leftarrow |P|$
for q = 0 **to** m **do**
  for $a \in \Sigma$ **do**
    $k \leftarrow \min(m, q + 1)$
    while $P[1 \ldots k]$ *kein Suffix von* $P[1 \ldots q]a$ **do**
      $k \leftarrow k - 1$
    $\delta(q, a) \leftarrow k$

Beispiel: P = abaab



ENDLICHER-AUTOMAT-MATCHER(T, δ, m)

$n \leftarrow |T|$
$q \leftarrow q_0$
for i = 0 **to** $n - 1$ **do**
  $q \leftarrow \delta(q, T[i])$
  if $q = m$ **then** gib aus: „Muster P taucht an Stelle $i - m + 1$ in T auf“

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

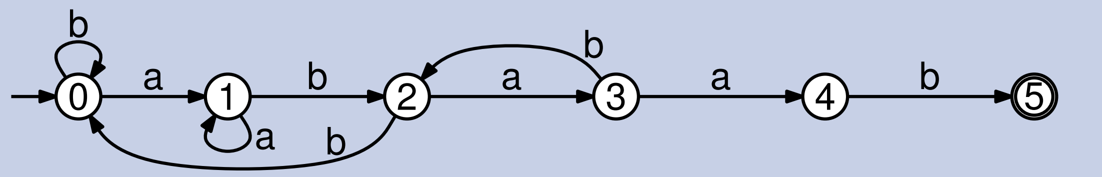
String-Matching-Automat

Karlsruher Institut für Technologie

ÜBERGANGSFUNKTION(P, Σ)

$m \leftarrow |P|$
for $q = 0$ **to** m **do**
  for $a \in \Sigma$ **do**
    $k \leftarrow \min(m, q+1)$
    while $P[1 \ldots k]$ *kein Suffix von* $P[1 \ldots q]a$ **do**
      $k \leftarrow k - 1$
    $\delta(q, a) \leftarrow k$

Beispiel: P = abaab



ENDLICHER-AUTOMAT-MATCHER(T, δ, m)

$n \leftarrow |T|$
$q \leftarrow q_0$
for $i = 0$ **to** $n - 1$ **do**
  $q \leftarrow \delta(q, T[i])$
  if $q = m$ **then** gib aus: „Muster P taucht an Stelle $i - m + 1$ in T auf“

String-Matching-Automat

ÜBERGANGSFUNKTION(P, Σ) $O(m \cdot |\Sigma| \cdot m^2)$

```
m ← |P|
for q = 0 to m do                                          O(|Σ| · m²)
    for a ∈ Σ do
        k ← min(m, q + 1)                                  O(m²)
        while P[1 ... k] kein Suffix von P[1 ... q]a do
            k ← k − 1
        δ(q, a) ← k
```

Laufzeit: $O(|\Sigma| \cdot m^3)$ (geht auch in $O(|\Sigma| \cdot m)$).

ENDLICHER-AUTOMAT-MATCHER(T, δ, m) $O(n)$

```
n ← |T|
q ← q₀
for i = 0 to n − 1 do
    q ← δ(q, T[i])
    if q = m then gib aus: „Muster P taucht an Stelle i − m + 1 in T auf“
```

Laufzeit: $O(n)$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1\ldots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Beweis:

ÜBERGANGSFUNKTION(P, Σ)

$\ldots$
$k \leftarrow \min(m, q+1)$
while $P[1\ldots k]$ *kein Suffix von* $P[1\ldots q]a$ **do**
  $k \leftarrow k - 1$
$\delta(q, a) \leftarrow k$

Warum muss man nicht wa betrachten?

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Beweis:

w_1	$\dots$	$w_{\ell-q}$	$w_{\ell-q+1}$	$w_{\ell-q+2}$	$\dots$	$w_{\ell-1}$	w_ℓ	a	(wa)
			$=$	$=$	$=$	$=$	$=$	$=$	
			p_1	p_2	$\dots$	p_{q-1}	p_q	a	$(P[1 \dots q]a)$

Gleichheit, da $\mathrm{suf}_P(w) = q$

ÜBERGANGSFUNKTION(P, Σ)

$\dots$

$k \leftarrow \min(m, q+1)$

while $P[1 \dots k]$ *kein Suffix von* $P[1 \dots q]a$ **do**

$\quad k \leftarrow k - 1$

$\delta(q, a) \leftarrow k$

Warum muss man nicht wa betrachten?

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \ldots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Beweis:

w_1	$\ldots$	$w_{\ell-q}$	$w_{\ell-q+1}$	$w_{\ell-q+2}$	$\ldots$	$w_{\ell-1}$	w_ℓ	a	(wa)
			$=$	$=$	$=$	$=$	$=$	$=$	
			p_1	p_2	$\ldots$	p_{q-1}	p_q	a	$(P[1 \ldots q]a)$
					$=?$	$=?$	$=?$	$=?$	
					p_1	$\ldots$	p_{k-1}	p_k	$(P[1 \ldots k])$

Gleichheit, da $\mathrm{suf}_P(w) = q$

ÜBERGANGSFUNKTION(P, Σ)

$\ldots$
$k \leftarrow \min(m, q+1)$
while $P[1 \ldots k]$ *kein Suffix von* $P[1 \ldots q]a$ **do**
  $k \leftarrow k - 1$
$\delta(q, a) \leftarrow k$

Warum muss man nicht wa betrachten?

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Beweis:

w_1	$\dots$	$w_{\ell-q}$	$w_{\ell-q+1}$	$w_{\ell-q+2}$	$\dots$	$w_{\ell-1}$	w_ℓ	a	(wa)
			$=$	$=$	$=$	$=$	$=$	$=$	
			p_1	p_2	$\dots$	p_{q-1}	p_q	a	$(P[1 \dots q]a)$
					$\overset{?}{=}$	$\overset{?}{=}$	$\overset{?}{=}$	$\overset{?}{=}$	
					p_1	$\dots$	p_{k-1}	p_k	$(P[1 \dots k])$

Gleichheit, da $\mathrm{suf}_P(w) = q$

Aus $k \leq q + 1$ folgt:

- $P[1 \dots k]$ Suffix von wa $\Leftrightarrow$ $P[1 \dots k]$ Suffix von $P[1 \dots q]a$
- $\Rightarrow \mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$

ÜBERGANGSFUNKTION(P, Σ)

$\dots$

$k \leftarrow \min(m, q + 1)$

while $P[1 \dots k]$ *kein Suffix von* $P[1 \dots q]a$ **do**

  $k \leftarrow k - 1$

$\delta(q, a) \leftarrow k$

Warum muss man nicht wa betrachten?

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \dots, n$ gilt $\mathcal{C}(T[1 \dots i]) = \mathrm{suf}_P(T[1 \dots i])$.

Beweis: Induktion über i

```
ÜBERGANGSFUNKTION(P, Σ)
  m ← |P|
  for q = 0 to m do
    for a ∈ Σ do
      k ← min(m, q + 1)
      while P[1 ... k] kein Suffix von P[1 ... q]a do
        k ← k − 1
      δ(q, a) ← k
```


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\text{suf}_P(wa) = \text{suf}_P(P[1 \dots q]a)$, wobei $q = \text{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \dots, n$ gilt $\mathcal{C}(T[1 \dots i]) = \text{suf}_P(T[1 \dots i])$.

Beweis: Induktion über i

I.A.: $i = 1$

- Es gilt $\mathcal{C}(T[1]) = \delta(0, T[1]) = \text{suf}_P(T[1])$.

```
ÜBERGANGSFUNKTION(P, Σ)
  m ← |P|
  for q = 0 to m do
    for a ∈ Σ do
      k ← min(m, q + 1)
      while P[1 ... k] kein Suffix von P[1 ... q]a do
        k ← k − 1
      δ(q, a) ← k
```

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\text{suf}_P(wa) = \text{suf}_P(P[1 \ldots q]a)$, wobei $q = \text{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \ldots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \ldots, n$ gilt $\mathcal{C}(T[1 \ldots i]) = \text{suf}_P(T[1 \ldots i])$.

Beweis: Induktion über i

I.S.:

$$\mathcal{C}(T[1 \ldots i]) \overset{\text{lies letztes Zeichen}}{=} \delta(q, T[i]) \qquad \text{mit } q = \mathcal{C}(T[1 \ldots i-1]) \overset{\text{Induktion}}{=} \text{suf}_P(T[1 \ldots i-1])$$

```
ÜBERGANGSFUNKTION(P, Σ)
  m ← |P|
  for q = 0 to m do
    for a ∈ Σ do
      k ← min(m, q + 1)
      while P[1 ... k] kein Suffix von P[1 ... q]a do
        k ← k − 1
      δ(q, a) ← k
```


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \dots, n$ gilt $\mathcal{C}(T[1 \dots i]) = \mathrm{suf}_P(T[1 \dots i])$.

Beweis: Induktion über i

I.S.:

$$\mathcal{C}(T[1 \dots i]) \overset{\text{lies letztes Zeichen}}{=} \delta(q, T[i]) \qquad \text{mit } q = \mathcal{C}(T[1 \dots i-1]) \overset{\text{Induktion}}{=} \mathrm{suf}_P(T[1 \dots i-1])$$

$$\overset{\text{Definition von } \delta}{=} \mathrm{suf}_P(P[1 \dots q]T[i])$$

ÜBERGANGSFUNKTION(P, Σ)
- $m \leftarrow |P|$
- **for** $q = 0$ **to** m **do**
  - **for** $a \in \Sigma$ **do**
    - $k \leftarrow \min(m, q+1)$
    - **while** $P[1 \dots k]$ *kein Suffix von* $P[1 \dots q]a$ **do**
      - $k \leftarrow k - 1$
    - $\delta(q, a) \leftarrow k$

Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \dots, n$ gilt $\mathcal{C}(T[1 \dots i]) = \mathrm{suf}_P(T[1 \dots i])$.

Beweis: Induktion über i

I.S.:

lies letztes Zeichen

Induktion

$$\mathcal{C}(T[1 \dots i]) = \delta(q, T[i]) \quad \text{mit } q = \mathcal{C}(T[1 \dots i-1]) = \mathrm{suf}_P(T[1 \dots i-1])$$

Definition von δ → $= \mathrm{suf}_P(P[1 \dots q]T[i])$

Lemma → $= \mathrm{suf}_P(T[1 \dots i-1]T[i])$

ÜBERGANGSFUNKTION(P, Σ)

```
m ← |P|
for q = 0 to m do
    for a ∈ Σ do
        k ← min(m, q + 1)
        while P[1 ... k] kein Suffix von P[1 ... q]a do
            k ← k − 1
        δ(q, a) ← k
```


Institut für Theoretische Informatik
Prof. Dr. Dorothea Wagner

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1\dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1\dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1,\dots,n$ gilt $\mathcal{C}(T[1\dots i]) = \mathrm{suf}_P(T[1\dots i])$.

Beweis: Induktion über i

I.S.:

lies letztes Zeichen

Induktion

$$\mathcal{C}(T[1\dots i]) = \delta(q, T[i]) \quad \text{mit } q = \mathcal{C}(T[1\dots i-1]) = \mathrm{suf}_P(T[1\dots i-1])$$

Definition von δ: $= \mathrm{suf}_P(P[1\dots q]T[i])$

Lemma: $= \mathrm{suf}_P(T[1\dots i-1]T[i])$

klar: $= \mathrm{suf}_P(T[1\dots i])$

```
ÜBERGANGSFUNKTION(P, Σ)
  m ← |P|
  for q = 0 to m do
    for a ∈ Σ do
      k ← min(m, q + 1)
      while P[1...k] kein Suffix von P[1...q]a do
        k ← k − 1
      δ(q, a) ← k
```

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \dots, n$ gilt $\mathcal{C}(T[1 \dots i]) = \mathrm{suf}_P(T[1 \dots i])$.

Folgerung:
Falls $\mathcal{A}_P$ bei Abarbeitung von $T[1 \dots i]$ im Zustand m endet, dann ist P Suffix von $T[1 \dots i]$, taucht also an Stelle $i - m + 1$ in T auf.

String-Matching-Automat – Korrektheit

Lemma: Anhängen eines Zeichens
Für jedes $w \in \Sigma^\star$ und $a \in \Sigma$ ist $\mathrm{suf}_P(wa) = \mathrm{suf}_P(P[1 \dots q]a)$, wobei $q = \mathrm{suf}_P(w)$.

Satz: Korrektheit **(Satz 7.1)**
Sei $\mathcal{C}$ die Zustandsfunktion zu $\mathcal{A}_P$ und sei $T[1 \dots n]$ die Eingabe in $\mathcal{A}_P$. Für alle $i = 1, \dots, n$ gilt $\mathcal{C}(T[1 \dots i]) = \mathrm{suf}_P(T[1 \dots i])$.

Folgerung:
Falls $\mathcal{A}_P$ bei Abarbeitung von $T[1 \dots i]$ im Zustand m endet, dann ist P Suffix von $T[1 \dots i]$, taucht also an Stelle $i - m + 1$ in T auf.

Anmerkungen:

- Kann zu einem Algorithmus mit optimaler Laufzeit $O(n + m)$ weiterentwickelt werden (Knuth-Morris-Pratt). $\mathcal{A}_P$ wird dabei nicht explizit berechnet.
- Nach einer Vorberechnungszeit von $O(|\Sigma| \cdot m)$ können Texte in $O(n)$ Zeit durchsucht werden. $\rightarrow$ Gut für Suche eines Musters in mehreren Texten.
- Nächste Vorlesung: Suche in einem Text nach verschiedenen Mustern.