

# Algorithmen II

## Vorlesung am 06.12.2012

String-Matching

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER



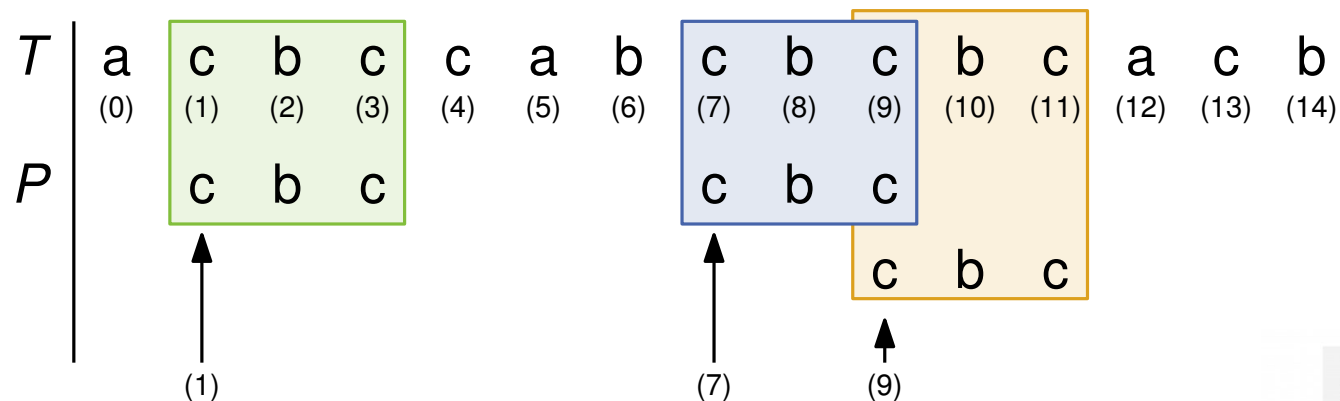
# String-Matching – Einführung

## Problem: String-Matching

Seien  $P$  und  $T$  Zeichenfolgen mit Zeichen aus dem Alphabet  $\Sigma$ , wobei  $|P| < |T|$ .  
Finde alle Vorkommen von  $P$  (Muster, engl. Pattern) in  $T$  (Text).

## Beispiel:

Das Muster  $P = cbc$  taucht im Text  $T = acbccabcbbcabc$  an den Stellen 1, 7 und 9 auf.



## Anwendungsbeispiele:

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



## Problem: String-Matching

Seien  $P$  und  $T$  Zeichenfolgen mit Zeichen aus dem Alphabet  $\Sigma$ , 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  $0 \leq j < m$ .

## Problem: String-Matching

Seien  $P$  und  $T$  Zeichenfolgen mit Zeichen aus dem Alphabet  $\Sigma$ , 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“

## **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)

# Rabin & Karp – Idee

## Annahme:

Für das Alphabet gilt  $\Sigma = \{0, 1, \dots, 9\}$ .

**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  $T[s] T[s+1] \dots T[s+m]$  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  $\Sigma = \{0, 1, \dots, 9\}$ .

**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  $T[s] T[s+1] \dots T[s+m]$  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

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

$$p = 31415 = 2416 \cdot 13 + 7$$

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

$t_0 = 23590$   
 $= 1814 \cdot 13 + 8$

**Es gilt:**

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

# 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
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

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

**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
$t_1 = 35902$						$t_6 = 31415$												
$= 2761 \cdot 13 + \boxed{9}$						$= 2416 \cdot 13 + \boxed{7}$												

## 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)

# 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
$t_1 = 35902$						$t_6 = 31415$					$t_{12} = 67399$							
$= 2761 \cdot 13 + \boxed{9}$						$= 2416 \cdot 13 + \boxed{7}$					$= 5184 \cdot 13 + \boxed{7}$							

## 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)

# 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:

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

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

# 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:

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

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

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

**Beispiel:** Berechnung von  $31415 \bmod 13$

$$(31415 = 31000 + 400 + 10 + 5)$$

$$31 \bmod 13 = 5$$

# 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:

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

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

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

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

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

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



# 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:

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

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

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

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

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

$$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2 \quad \Rightarrow \quad 3140 \bmod 13 = (10 \cdot 2) \bmod 13 = 7$$

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

# 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:

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

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

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

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

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

$$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2 \quad \Rightarrow \quad 3140 \bmod 13 = (10 \cdot 2) \bmod 13 = 7$$

$$\Rightarrow 3141 \bmod 13 = (7 + 1) \bmod 13 = 8 \quad \Rightarrow \quad 31410 \bmod 13 = (10 \cdot 8) \bmod 13 = 2$$

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

# 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:

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

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

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

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

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

$$\Rightarrow 314 \bmod 13 = (11 + 4) \bmod 13 = 2 \quad \Rightarrow \quad 3140 \bmod 13 = (10 \cdot 2) \bmod 13 = 7$$

$$\Rightarrow 3141 \bmod 13 = (7 + 1) \bmod 13 = 8 \quad \Rightarrow \quad 31410 \bmod 13 = (10 \cdot 8) \bmod 13 = 2$$

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

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

$\Rightarrow t_{s+1} \bmod q$  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)

# Der Algorithmus von Rabin & Karp

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

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

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

$\hat{t}_0 \leftarrow t_0 \bmod q$

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

**if**  $\hat{t}_s = \hat{p}$  **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“

$\hat{t}_{s+1} \leftarrow$  Berechne  $\hat{t}_{s+1}$  aus  $\hat{t}_s$

# Der Algorithmus von Rabin & Karp

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

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

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

$\hat{t}_0 \leftarrow t_0 \bmod q$

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

<b>if</b> $\hat{t}_s = \hat{p}$ <b>then</b>	$O(1)$ für Vergleich	$O(m)$
MATCH $\leftarrow$ TRUE		$O(m)$
<b>for</b> $j = 0$ <b>to</b> $m - 1$ <b>do</b>		
<b>if</b> $T[s + j] \neq P[j]$ <b>then</b> match $\leftarrow$ FALSE		
<b>if</b> MATCH <b>then</b> gib aus: „Muster $P$ taucht an Stelle $s$ in $T$ auf“		

$\hat{t}_{s+1} \leftarrow$  Berechne  $\hat{t}_{s+1}$  aus  $\hat{t}_s$

# Der Algorithmus von Rabin & Karp

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

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

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

$\hat{t}_0 \leftarrow t_0 \bmod q$

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

<b>if</b> $\hat{t}_s = \hat{p}$ <b>then</b>	$O(1)$ für Vergleich	$O(m)$
MATCH $\leftarrow$ TRUE		$O(m)$
<b>for</b> $j = 0$ <b>to</b> $m - 1$ <b>do</b>		
<b>if</b> $T[s + j] \neq P[j]$ <b>then</b> match $\leftarrow$ FALSE		
<b>if</b> MATCH <b>then</b> gib aus: „Muster $P$ taucht an Stelle $s$ in $T$ auf“		
$\hat{t}_{s+1} \leftarrow$ Berechne $\hat{t}_{s+1}$ aus $\hat{t}_s$		$O(1)$

# Der Algorithmus von Rabin & Karp

RABIN-KARP-MATCHER( $T, P, q$ )	$O((n - m) \cdot m)$
$(n, m) \leftarrow ( T ,  P )$	$O((n - m) \cdot m)$
$\hat{p} \leftarrow p \bmod q$	
$\hat{t}_0 \leftarrow t_0 \bmod q$	
<b>for</b> $s = 0$ <b>to</b> $n - m$ <b>do</b>	
<b>if</b> $\hat{t}_s = \hat{p}$ <b>then</b>	$O(1)$ für Vergleich $O(m)$
MATCH $\leftarrow$ TRUE	$O(m)$
<b>for</b> $j = 0$ <b>to</b> $m - 1$ <b>do</b>	
<b>if</b> $T[s + j] \neq P[j]$ <b>then</b> match $\leftarrow$ FALSE	
<b>if</b> MATCH <b>then</b> gib aus: „Muster $P$ taucht an Stelle $s$ in $T$ auf“	
$\hat{t}_{s+1} \leftarrow$ Berechne $\hat{t}_{s+1}$ aus $\hat{t}_s$	$O(1)$

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

# Der Algorithmus von Rabin & Karp

```
RABIN-KARP-MATCHER( $T, P, q$ )  $O((n - m) \cdot m)$   
   $(n, m) \leftarrow (|T|, |P|)$   $O((n - m) \cdot m)$   
   $\hat{p} \leftarrow p \bmod q$   
   $\hat{t}_0 \leftarrow t_0 \bmod q$   
  for  $s = 0$  to  $n - m$  do  
    if  $\hat{t}_s = \hat{p}$  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“  
     $\hat{t}_{s+1} \leftarrow$  Berechne  $\hat{t}_{s+1}$  aus  $\hat{t}_s$   $O(1)$ 
```

**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  $\Sigma^*$  nach  $\mathbb{Z}_q$ .

# Der Algorithmus von Rabin & Karp

Welche Laufzeit kann man erwarten?

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

**Unzulässige Übereinstimmungen:**  $t_s \neq p$  aber  $\hat{t}_s = \hat{p}$

Die Wahrscheinlichkeit für eine unzulässige Übereinstimmung ist  $\frac{1}{q}$ .

⇒ Die erwartete Anzahl von unzulässigen Übereinstimmungen ist in  $O\left(\frac{n}{q}\right)$ .

**Zulässige Übereinstimmungen:**  $t_s = p$  und damit auch  $\hat{t}_s = \hat{p}$

Sei  $v$  die Anzahl zulässiger Übereinstimmungen.

# Der Algorithmus von Rabin & Karp

Welche Laufzeit kann man erwarten?

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

**Unzulässige Übereinstimmungen:**  $t_s \neq p$  aber  $\hat{t}_s = \hat{p}$

Die Wahrscheinlichkeit für eine unzulässige Übereinstimmung ist  $\frac{1}{q}$ .

⇒ Die erwartete Anzahl von unzulässigen Übereinstimmungen ist in  $O\left(\frac{n}{q}\right)$ .

**Zulässige Übereinstimmungen:**  $t_s = p$  und damit auch  $\hat{t}_s = \hat{p}$

Sei  $v$  die Anzahl zulässiger Übereinstimmungen.

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

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

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

# String-Matching mit endlichen Automaten

1. Baue einen endlichen Automaten  $\mathcal{A}_P$  (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

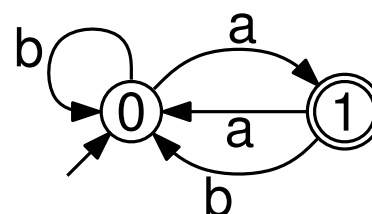
- $Q$  – endliche Menge von *Zuständen*
- $q_0 \in Q$  – *Startzustand*
- $A \subseteq Q$  – Menge von *akzeptierenden Zuständen*
- $\Sigma$  – endliches *Eingabealphabet*
- $\delta: Q \times \Sigma \rightarrow Q$  – *Übertragungsfunktion*

## Beispiel:

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

## Definition: Zustandsfunktion

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

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

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

## Definition: Zustandsfunktion

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

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

**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: Zustandsfunktion

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

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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



## Definition: Suffixfunktion

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

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

$\Rightarrow$  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  $Q = \{0, 1, \dots, m\}$ ,  $q_0 = 0$ ,  $A = \{m\}$  und  $\delta(q, a) = \text{suf}_P(P[1 \dots q]a)$ .

# Der String-Matching-Automat

## Definition: Suffixfunktion

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

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

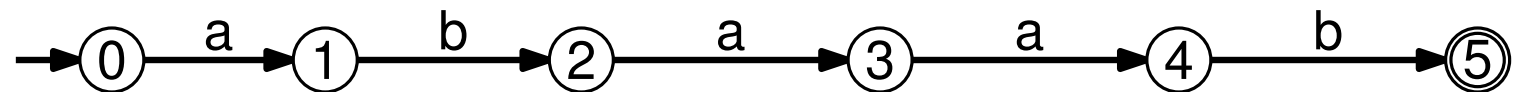
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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

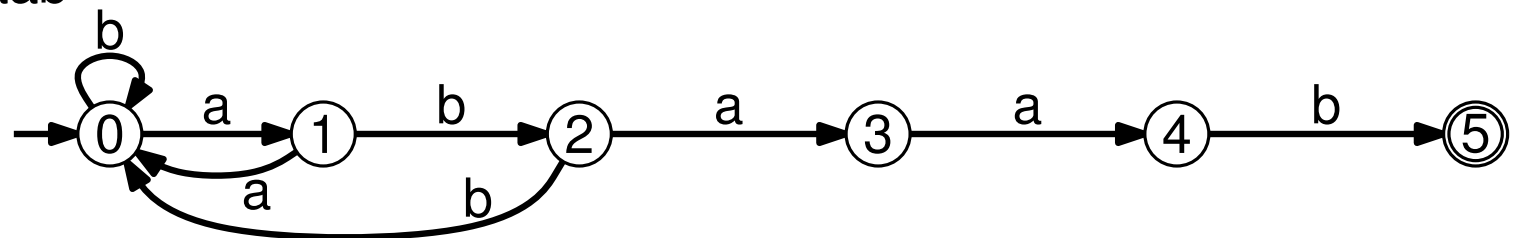
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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

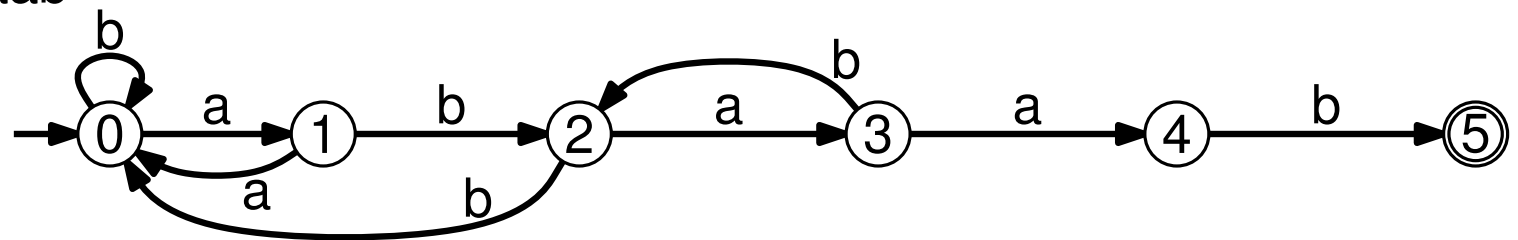
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



gelesenes Wort: ...

...

... a

... ab

... aba

... abaa

... abaab

## Definition: Suffixfunktion

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

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

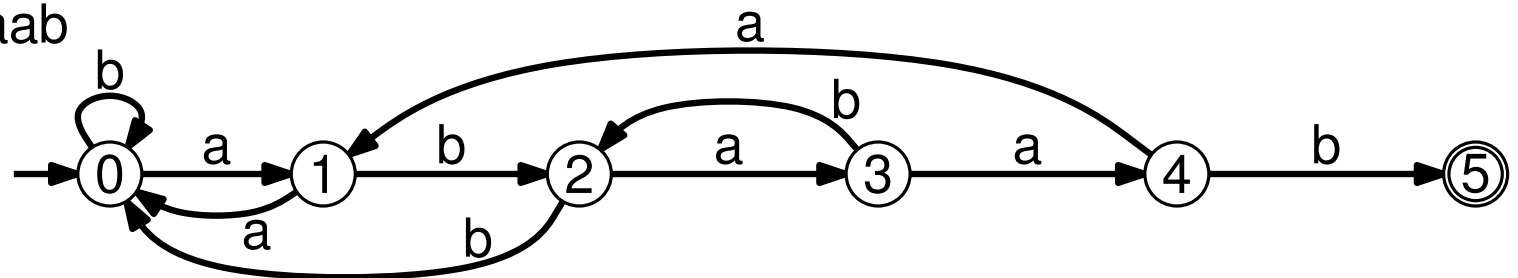
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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

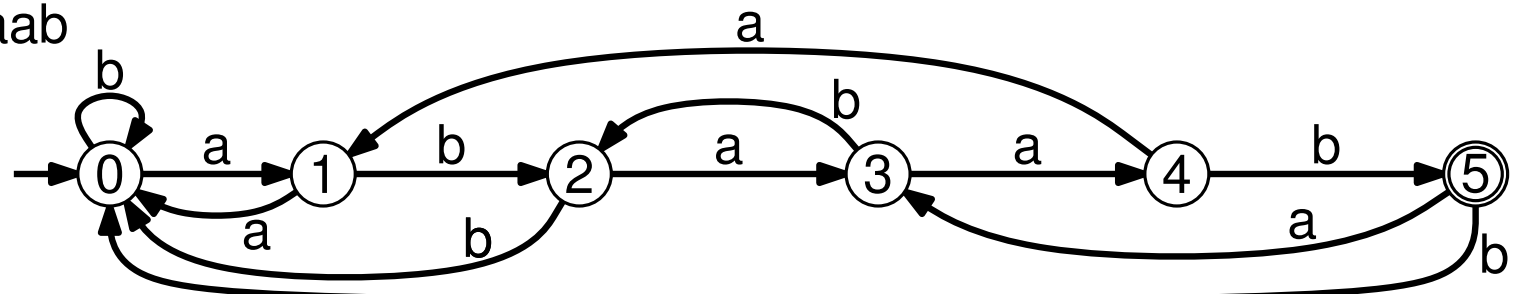
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

## Definition: Suffixfunktion

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

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

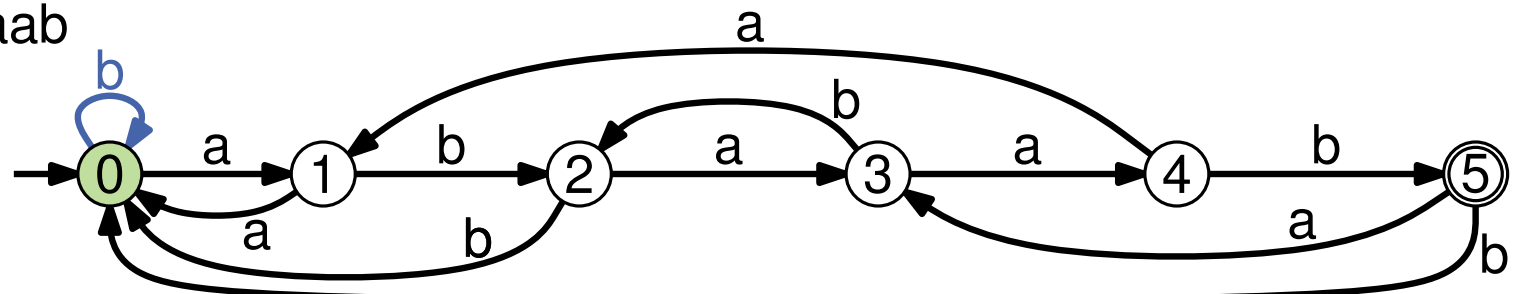
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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

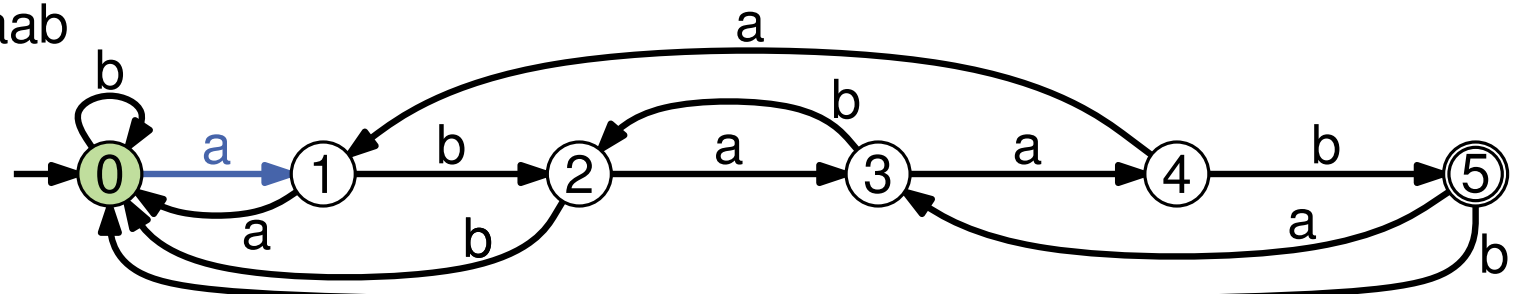
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \mathbf{b} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{b}$



# Der String-Matching-Automat

## Definition: Suffixfunktion

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

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

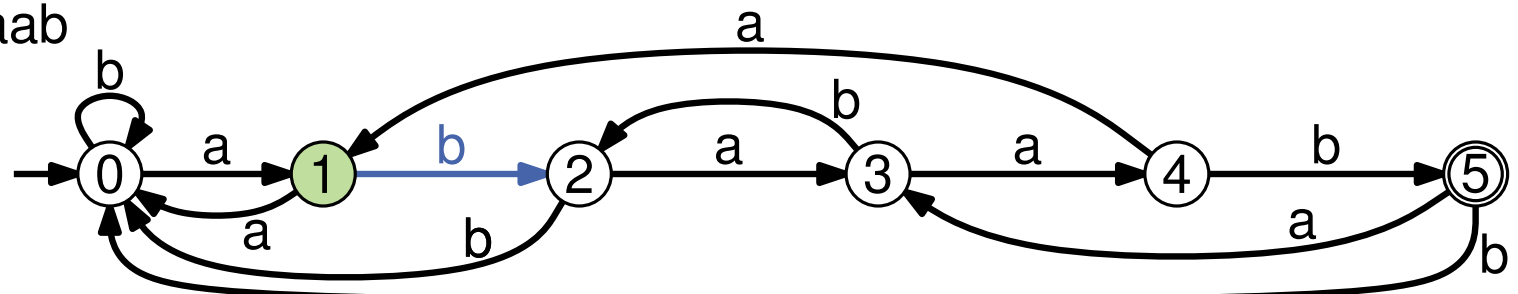
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \mathbf{b} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{b}$

# Der String-Matching-Automat

## Definition: Suffixfunktion

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

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

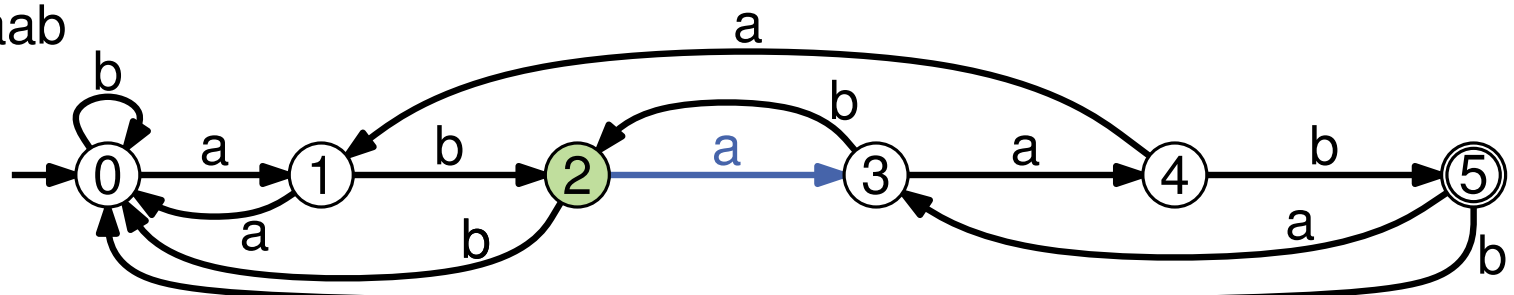
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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

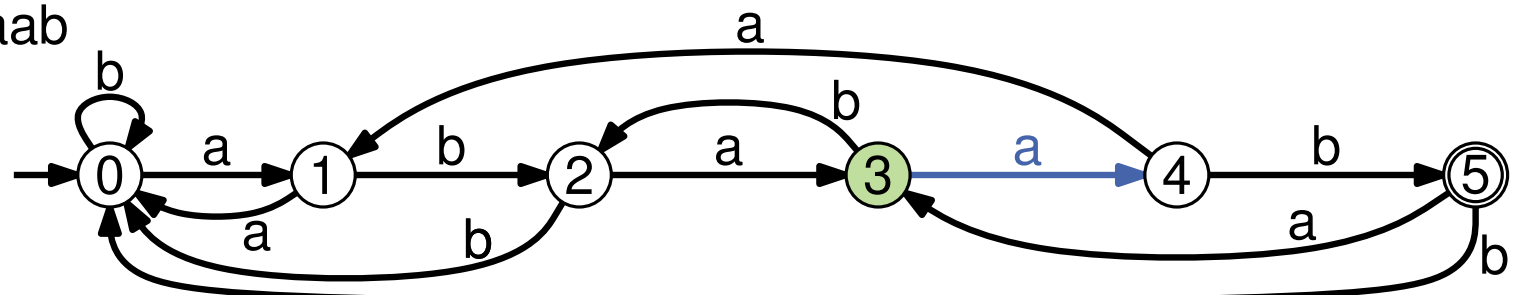
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**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*  $\text{suf}_P: \Sigma^* \rightarrow \{0, \dots, |P| = m\}$  wie folgt:  $\text{suf}_P(w)$  ist die Länge des maximalen Präfixes von  $P$ , das Suffix von  $w$  ist.

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

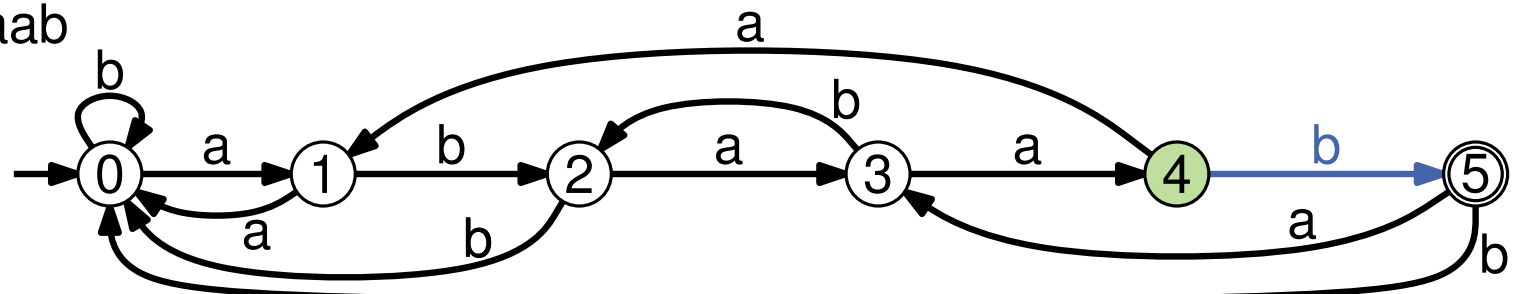
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

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

## Definition: Suffixfunktion

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

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

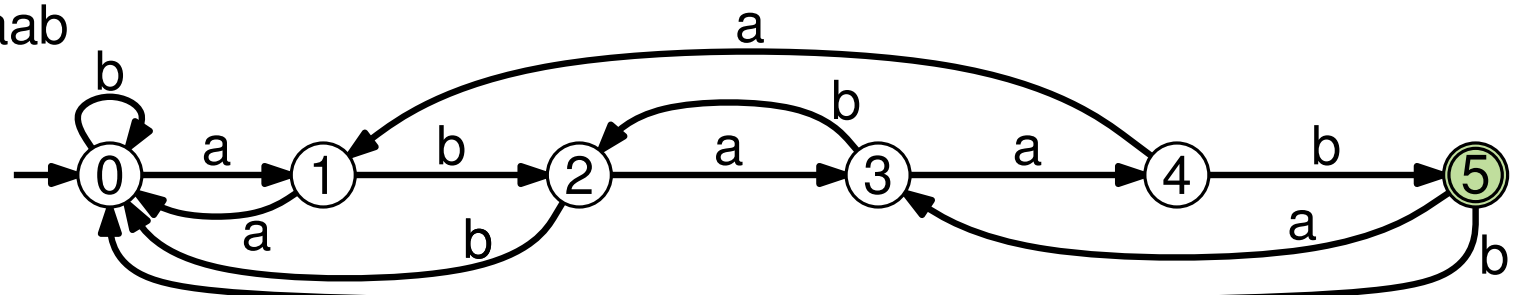
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \underline{b a b a a b} a a b b$

## Definition: Suffixfunktion

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

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

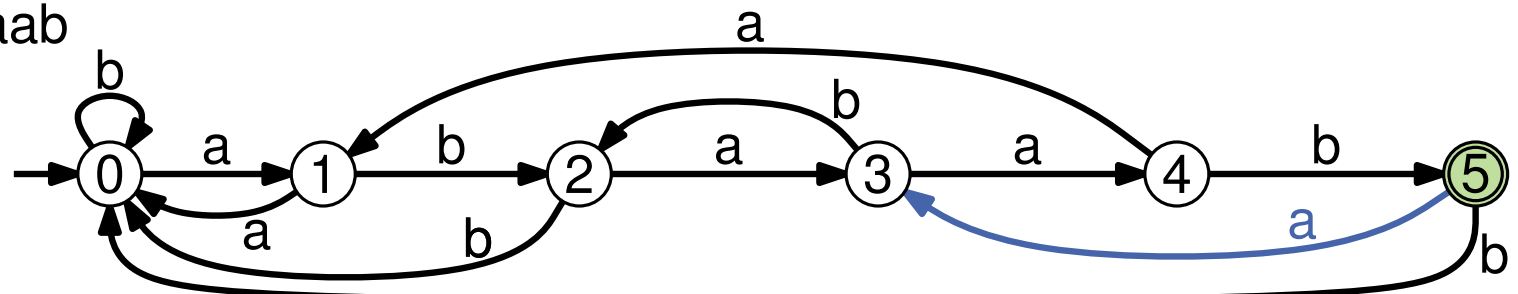
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



gelesenes Wort: ... a ab aba abaa abaab

Ausführung für  $T = \underline{babaab}aabb$

## Definition: Suffixfunktion

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

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

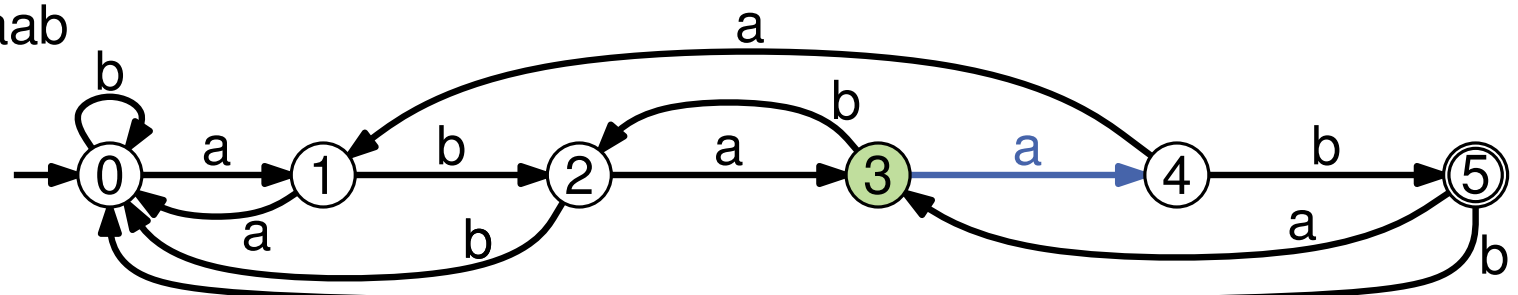
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \underline{b a b a a b} a a b b$

# Der String-Matching-Automat

## Definition: Suffixfunktion

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

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

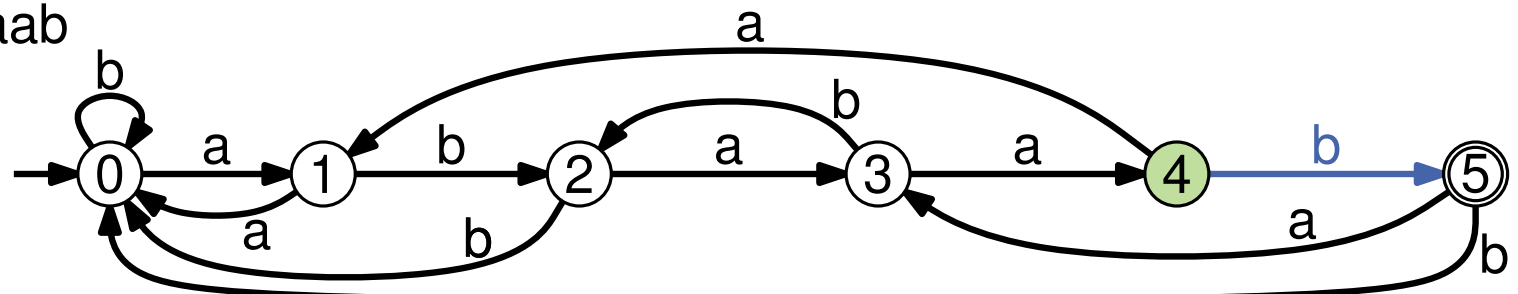
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \underline{babaab}aabb$



# Der String-Matching-Automat

## Definition: Suffixfunktion

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

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

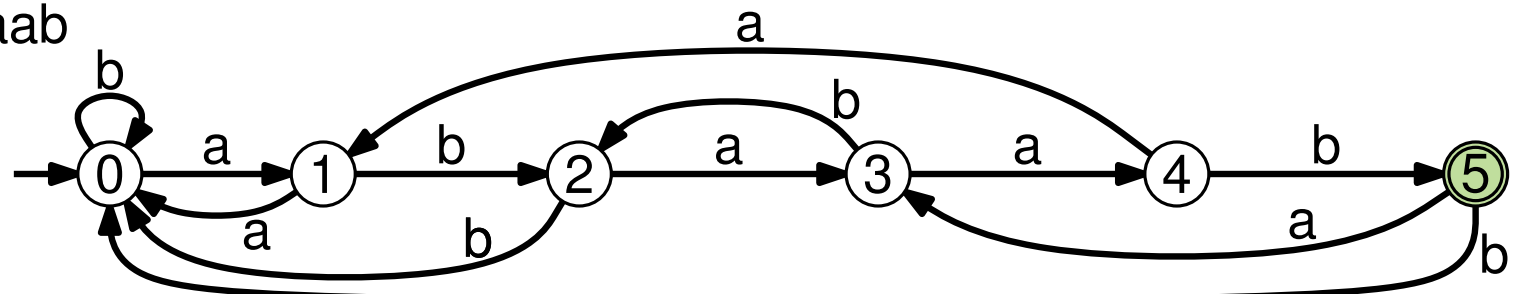
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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


**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \mathbf{b a b a a b a a b b}$



## Definition: Suffixfunktion

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

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

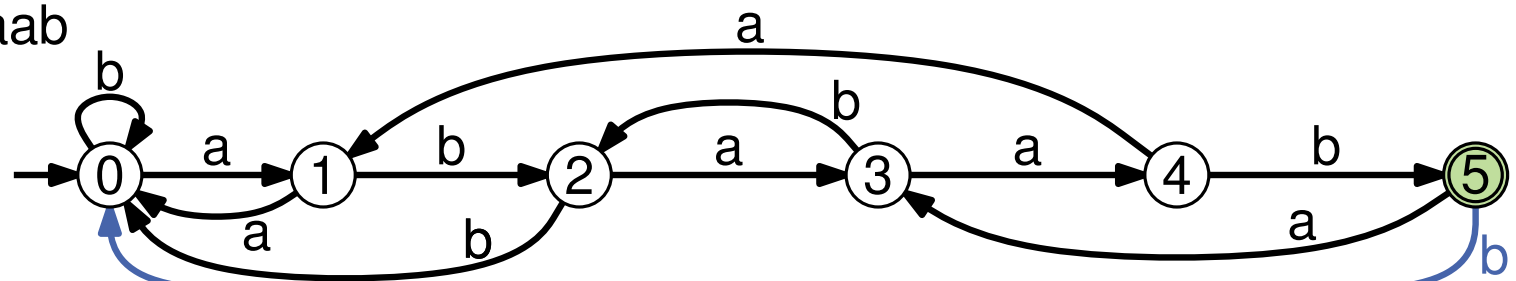
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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

**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \mathbf{b} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{a} \mathbf{b} \mathbf{b}$

## Definition: Suffixfunktion

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

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

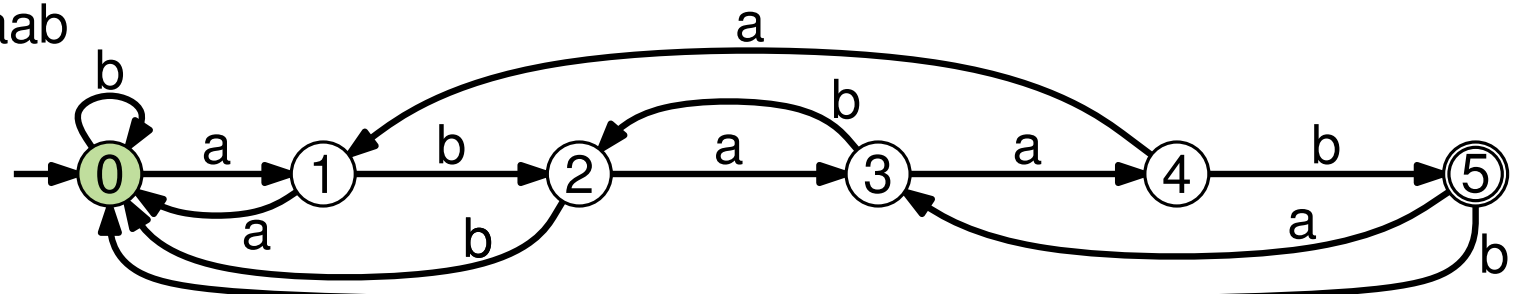
$\Rightarrow$  Das Wort  $w$  hat  $P$  genau dann als Suffix, wenn  $\text{suf}_P(w) = m$ .

## Definition: String-Matching-Automat

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


**Beispiel:**  $P = abaab$

$\mathcal{A}_P$ :



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

Ausführung für  $T = \mathbf{b a b a a b a a b b}$



# String-Matching-Automat

ÜBERGANGSFUNKTION( $P, \Sigma$ )

$m \leftarrow |P|$

**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**

$k \leftarrow k - 1$

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

ENDLICHER-AUTOMAT-MATCHER( $T, \delta, m$ )

$n \leftarrow |T|$

$q \leftarrow q_0$

**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“

ÜBERGANGSFUNKTION( $P, \Sigma$ )

$O(m \cdot |\Sigma| \cdot m^2)$

$m \leftarrow |P|$

**for**  $q = 0$  **to**  $m$  **do**

**for**  $a \in \Sigma$  **do**

$O(|\Sigma| \cdot m^2)$

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

$O(m^2)$

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

$k \leftarrow k - 1$

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

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

ENDLICHER-AUTOMAT-MATCHER( $T, \delta, m$ )

$O(n)$

$n \leftarrow |T|$

$q \leftarrow q_0$

**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“

**Laufzeit:**  $O(n)$

## Lemma: Anhängen eines Zeichens

Für jedes  $w \in \Sigma^*$  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  $\mathcal{C}(T[1 \dots i]) = \text{suf}_P(T[1 \dots i])$ .

## 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.

## Lemma: Anhängen eines Zeichens

Für jedes  $w \in \Sigma^*$  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  $\mathcal{C}(T[1 \dots i]) = \text{suf}_P(T[1 \dots i])$ .

## 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. → Gut für Suche eines Musters in mehreren Texten.
- Nächste Vorlesung: Suche in einem Text nach verschiedenen Mustern.