

Vorlesung

Effiziente Algorithmen und Komplexitätstheorie

Sommersemester 2008

Ingo Wegener

Was bisher geschah...

- starke Zusammenhangskomponenten ✓
- Matchings ✓
- Flussproblem ✓
- Amortisierte Analyse ✓

jetzt Textmustersuche

Textmustersuche

neues Thema Suche nach allen Vorkommen eines Muster $P \in \Sigma^*$ in einem Text $T \in \Sigma^*$.
 endliches Alphabet Σ , $n = |T|$, $m = |P|$
 heißt **String Matching**

Beispiel

P : O O O H

T : O O O O O O O O O O O O O O O

Beispiel

P : O O O H

T : O O O O H U H U D U D A I S T

Naiver Algorithmus

Algorithmus 6.2

1. For $i \in \{0, 1, 2, \dots, n - m\}$
2. $j := 1$; While $T[i + j] = P[i + j]$
3. $j = j + 1$; If $j > m$ Then Ausgabe i

Theorem 6.3

Algorithmus 6.2 löst das String-Matching-Problem in Zeit $\Theta((n - m + 1) \cdot m)$.

Beweis.

obere Schranke äußere Schleife über $n - m + 1$ Werte
 innere Schleife über $\leq m$ Werte
 alle Befehle Zeit $O(1)$

zusammen Zeit $O((n - m + 1) \cdot m)$ ✓

Beweis von Theorem 6.3

Wir haben obere Schranke $O((n - m + 1) \cdot m)$ durch
äußere Schleife über $n - m + 1$ Werte
innere Schleife über $\leq m$ Werte
alle Befehle Zeit $O(1)$

untere Schranke betrachte $T = 0^n$, $P = 0^m$
innere Schleife über m Werte

zusammen Zeit $\Omega((n - m + 1) \cdot m)$



Geht das nicht irgendwie schneller?

Beobachtungen zum naiven String Matching

Beobachtung typisch $T[i]$ mit jedem $P[j]$ vergleichen
 \rightsquigarrow „ $n \cdot m$ “

Beobachtung Algorithmus „fast gedächtnislos“

Kann man vielleicht geschickter vorgehen und jedes $T[i]$ weniger oft lesen?

Kommt man vielleicht mit einmaligem Lesen aus ohne zu großen Overhead?

Wege zu schnellerem String Matching

Erinnerung aus GTI/TIfAI
endliche Automaten (DFA/NFA)

Welche Laufzeit hat ein DFA?

klar immer Linearzeit

also DFA fürs String Matching
↔ String Matching in Linearzeit

Vorsicht nicht alle Probleme sind regulär

String Matching mit DFA

Idee für Muster P definiere festen DFA

Annahme DFA $A = (Q, \Sigma, q_0, \delta, F)$ ist ein passend definierter DFA
 $q \in F \Leftrightarrow P$ gefunden

Algorithmus 6.4

1. $q := q_0$
2. For $i \in \{1, 2, \dots, n\}$
2. $q := \delta(q, T[i])$
3. If $q \in F$ Then Ausgabe $i - m + 1$

klar Algorithmus 6.4 läuft in Zeit $\Theta(n)$

aber Gibt es sicher passenden DFA A ?

Der Weg zum DFA

Definition 6.1

Sei Σ ein endliches Alphabet, $x = x_1x_2 \dots x_k \in \Sigma^k$ und $y = y_1y_2 \dots y_l \in \Sigma^l$ Zeichenketten endlicher Länge über Σ . Für $1 \leq i \leq j \leq k$ schreiben wir $x[i..j]$ für die Zeichenkette $x_ix_{i+1} \dots x_j$ und $x[i]$ für x_i .

Wir schreiben $x \sqsubset y$, wenn $y[1..k] = x$ gilt und nennen x ein *Präfix* von y . Wir schreiben $x \sqsupset y$, wenn $y[l - k + 1..l] = x$ gilt und nennen x *Suffix* von y .

Idee merken, wie viele Zeichen $P[1..l]$ Suffix von $T[1..i]$ sind

klar P gefunden $\Leftrightarrow l = m$

Definiere $\sigma: \Sigma^* \rightarrow \{0, 1, \dots, m\}$
mit $\sigma(x) = \max\{l \mid P[1..l] \sqsupset x\}$

Ziel $\forall i: \delta(q_0, T[1..i]) = \sigma(T[1..i])$

Konstruktion des DFA

haben $\sigma(x) = \max\{l \mid P[1..k] \sqsupseteq x\}$

wollen $\forall i: \delta(q_0, T[1..i]) = \sigma(T[1..i])$

Zustände $Q = \{0, 1, \dots, m\}$

$q_0 = 0$

$F = \{m\}$

Zustandsübergänge $\delta(q, a) = \sigma(P[1..q]a)$

Beispiel $P = \text{OOOH}$ über $\Sigma = \{H, O, G\}$

δ	H	O	G
0	0	1	0
1	0	2	0
2	0	3	0
3	4	3	0
4	0	1	0

Beispiel für DFA-String-Matching mit $P = \text{OOOH}$

δ	H	O	G
0	0	1	0
1	0	2	0
2	0	3	0
3	4	3	0
4	0	1	0

O O O O H G O O O H



Q 4

Ausgabe 2, 7

Also String Matching in Zeit $O(n)$?

Rechenzeit String Matching mit DFAs

klar Algorithmus 6.4 hat Laufzeit $\Theta(n)$
wenn der DFA für P gegeben ist

Erinnerung Eingabe sind T und P

also Zeit zur Berechnung des DFA A für P mitzählen

Beobachtung für mehrfache Suchen mit P
nur einen DFA konstruieren

also neue Art der Kostenaufteilung
Gesamtlaufzeit = Preprocessing + Suche

DFA-Konstruktion – Offene Fragen

Beobachtung Zeit zur Berechnung von DFA A
von T unabhängig

also Gesamtlaufzeit $O(n) + O(f(m))$ erreichbar
genauer $O(n) + O(f(m, |\Sigma|))$

Problem $m = \Theta(n)$ möglich
nicht wirklich gewonnen, wenn f nicht klein ist

offene Frage Wie lange dauert die DFA-Konstruktion?

noch dringender Ist unsere DFA-Konstruktion überhaupt
korrekt?

Korrektheit der DFA-Konstruktion

jetzt kleine Schritte in Richtung Korrektheitsbeweis

Lemma 6.6

$\forall P \in \Sigma^*, x \in \Sigma^*, a \in \Sigma: \sigma_P(xa) \leq \sigma_P(x) + 1$

Beweis.

klar trivial für $\sigma_P(xa) = 0$ ✓

Voraussetzung $\sigma_P(xa) > 0$

klar $\sigma_P(xa) > 0 \Rightarrow P[1..\sigma_P(xa)] \sqsupseteq xa$

also $P[1..\sigma_P(xa) - 1] \sqsupseteq x$

also $\sigma_P(x) \geq \sigma_P(xa) - 1$

also $\sigma_P(xa) \leq \sigma_P(x) + 1$



Noch ein Blick auf die Suffixfunktion σ

Lemma 6.7

$$\forall P \in \Sigma^*, x \in \Sigma^*, a \in \Sigma:$$

$$(\sigma_P(x) = q) \Rightarrow (\sigma_P(xa) = \sigma_P(P[1..q]a))$$

Beweis.

klar $(\sigma_P(x) = q) \Rightarrow (P[1..q] \sqsupset x)$ also $P[1..q]a \sqsupset xa$ Sei $r := \sigma_P(xa)$ **damit** $r \leq q + 1$ (Lemma 6.6)also $|P[1..r]| \leq |P[1..q]a|$ klar $(\sigma_P(xa) = r) \Rightarrow (P[1..r] \sqsupset xa)$ zusammen $P[1..r] \sqsupset P[1..q]a$ also $\sigma_P(xa) = r \leq \sigma_P(P[1..q]a)$ aber auch $\sigma_P(P[1..q]a) \leq \sigma_P(xa)$ **weil** $P[1..q]a \sqsupset xa$ zusammen $\sigma_P(xa) = \sigma_P(P[1..q]a)$ 

Korrektheit des DFA

Theorem 6.8

Für den DFA A zu Muster P gilt $\delta(q_0, T[1..i]) = \sigma_P(T[1..i])$ für alle $T \in \Sigma^*$ und alle $i \in \{0, 1, \dots, |T|\}$.

Beweis. per vollständiger Induktion über i

Induktionsanfang $i = 0$ $\delta(q_0, \varepsilon) = q_0 = 0 = \sigma_P(\varepsilon)$ ✓

Induktionsschluss

Voraussetzung $\delta(q_0, T[1..i]) = \sigma_P(T[1..i])$

Definiere $a := T[i + 1]$, $q := \delta(q_0, T[1..i])$

$$\begin{aligned}
 \delta(q_0, T[1..i + 1]) &= \delta(q_0, T[1..i]a) = \delta(\delta(q_0, T[1..i]), a) \\
 &= \delta(q, a) = \sigma_P(P[1..q]a) \{ * \text{Definition von } \delta * \} \\
 &= \sigma_P(T[1..i]a) \{ * \text{Lemma 6.7} * \} \\
 &= \sigma_P(T[1..i + 1]) \quad \square
 \end{aligned}$$

Auf dem Weg zur Gesamtlaufzeit

also Vorgehen **korrekt** ✓
und **sehr effizient** nach **Preprocessing**

noch offen Laufzeit Preprocessing (**DFA-Berechnung**)

klar dazu **Algorithmus** zur DFA-Berechnung erforderlich

Erinnerung brauchen $\forall q \in Q, a \in \Sigma: \delta(q, a) = \sigma_P(P[1..q]a)$
mit $\sigma_P(P[1..q]a) = \max\{l \mid P[1..l] \sqsupseteq P[1..q]a\}$

Naive δ -Berechnung

Algorithmus 6.10

1. For $q \in \{0, 1, 2, \dots, m\}$
2. For $a \in \Sigma$
3. $l := \min\{m, q + 1\}$
4. Repeat
5. If $P[l] = a$ Then $\{ * \text{Ist } P[1..l-1] \sqsupseteq P[1..q]? * \}$
6. $i := l$; $\text{istSuffix} := \text{true}$
7. While $i > 1$ und $\text{istSuffix} = \text{true}$
8. $i := i - 1$
9. If $P[i] \neq P[q + l - 1 - i]$
- Then $\text{istSuffix} := \text{false}$
10. Else $\text{istSuffix} := \text{false}$
11. If $\text{istSuffix} = \text{false}$ Then $l := l - 1$
12. Until $l = 0$ oder $\text{istSuffix} = \text{true}$
13. $\delta(q, a) := l$
14. Ausgabe δ

Laufzeit von Algorithmus 6.10

Beobachtungen

- q -For-Schleife $m + 1$ Werte
- a -For-Schleife $|\Sigma|$ Werte
- Repeat-Schleife l mit $\leq m + 1$ Werten
- While-Schleife i mit $\leq m$ Werten

zusammen Laufzeit $O(|\Sigma| \cdot m^3)$

klar viel zu langsam

Anmerkung geht auch in Zeit $O(|\Sigma| \cdot m)$
(lösen wir aber lieber anders)

Beobachtung Zeit $\Omega(|\Sigma| \cdot m)$ nicht zu vermeiden

Korrektheit von Algorithmus 6.10

Beobachtung naive Implementierung offensichtlich korrekt

- Schleife über alle q und a
- maximal möglicher Wert von $\sigma_P(P[1..q]a)$ als Startwert für l
- direkter Test $P[1..l] \stackrel{?}{=} P[1..q]a$
- falls **ja**, $\delta(q, a)$ korrekt gesetzt
- falls **nein**, l um 1 verkleinert
- Abbruch bei 0 korrekt, weil $\varepsilon \sqsubset P[1..q]a$

zusammen

Theorem 6.11

Die naive δ -Berechnung berechnet $\delta: Q \times \Sigma \rightarrow Q$ für jedes $P \in \Sigma^m$ in Zeit $O(|\Sigma| \cdot m^3)$.

Rückblick String Matching mit DFA

Fazit

- + einmaliges Lesen von $T[i]$ reicht aus
- + nach Preprocessing sehr schnelle Suche
- + Preprocessing unabhängig von T , für viele T ausnutzbar
- Faktor $|\Sigma|$ für große Alphabete **völlig inakzeptabel**
Was ist mit UTF-32?
- + wichtige Struktureinsicht gewonnen

Gute Ideen weiterentwickelt

Wunsch **effizient** wie mit δ , aber **ohne** δ

konkreter berechne **Hilfsinformationen**,
so dass δ „on the fly“ **effizient** berechenbar

Was soll effizient bedeuten?

im Idealfall in konstanter Zeit

fast ebenso gut in **amortisierter** konstanter Zeit

Betrachte $P[1..q] \sqsubseteq T[1..s + q]$ (**schon gesehen**)
 $P[q + 1] \not\sqsubseteq T[s + q + 1]$ (**aktueller Vergleich**)

naiver Algorithmus neuer Versuch $P[1] \stackrel{?}{=} T[s + 2]$

DFA Wissen über $T[s + 2..s + q]$ schon vorhanden
 \rightsquigarrow keine weiteren Tests

brauchen q', s' , so dass $T[s'..s' + q' - 1] = P[1..q']$

Einsicht s' allein aus P und Wissen „ $P[1..q']$ **passt**“ ableitbar

Zum Preprocessing

Definition 6.12

Für $P \in \Sigma^m$ ist die **Präfixfunktion**

$\pi_P: \{1, 2, \dots, m\} \rightarrow \{0, 1, \dots, m-1\}$ durch

$\pi_P(q) = \max\{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$ definiert.

in Worten $\pi_P(q)$ ist Länge des längsten Präfixes von P ,
der echter Suffix von $P[1..q]$ ist

Beispiel

P A N A N A S B A N A N E N

<i>q</i>	1	2	3	4	5	6	7	8	9	10	11	12	13
$\pi_P(q)$	0	0	1	2	3	0	0	1	2	3	4	0	0

Anwendung der Präfixfunktion

am Beispiel

<i>T</i>	O	W	A	N	A	N	A	S	B	A	N	A	N	A	S	S	E					
<i>P</i>			A	N	A	N	A	S	B	A	N	A	N	E								
<i>T</i>	O	W	A	N	A	N	A	S	B	A	N	A	N	A	S	S	E					
<i>P</i>										A	N	A	N	A	S	B	A	N	A	N	E	N

<i>q</i>	1	2	3	4	5	6	7	8	9	10	11	12	13
$\pi_P(q)$	0	0	1	2	3	0	0	1	2	3	4	0	0

allgemein nach $P[q + 1] \neq T[i]$
 ersetze $q := \pi_P[q]$
 „wiederhole“ $P[q + 1] \stackrel{?}{=} T[i]$

Algorithmus von Knuth, Morris und Pratt

Algorithmus 6.13

1. Berechne π_P .
2. $q := 0$ { * Länge des passenden Präfix von P * }
3. For $i \in \{1, 2, \dots, n\}$
4. While $q > 0$ und $P[q + 1] \neq T[i]$
5. $q := \pi_P[q]$
6. If $P[q + 1] = T[i]$ Then $q := q + 1$
7. If $q = m$ Then Ausgabe $i - m + 1$; $q := \pi_P[q]$

Berechnung der Präfixfunktion π_P

Algorithmus 6.13 (Teil 2)

1. $l := 0$
2. $\pi_P[1] := 0$
3. For $q \in \{2, 3, \dots, m\}$
4. While $l > 0$ und $P[l + 1] \neq P[q]$
5. $l := \pi_P[l]$
6. If $P[l + 1] = P[q]$ Then $l = l + 1$
7. $\pi_P[q] := l$
8. Ausgabe von π_P

Laufzeit der Berechnung der Präfixfunktion

Lemma 6.14

Die Berechnung von π_P erfolgt in Zeit $\Theta(m)$.

Beweis.

untere Schranke $\Omega(m)$ offensichtlich ✓
wegen q -For-Schleife über $m - 1$ Werte

obere Schranke **Problem** Laufzeit While-Schleife

triviale Abschätzung l wird in jedem Schritt kleiner
 $\rightsquigarrow O(m)$ für While-Schleife, $O(m^2)$ insgesamt

besser amortisierte Analyse mit Potenzialmethode

Amortisierte Analyse π_P -Berechnung

Definiere **Potenzial** Wert von l

Beobachtung initial $l = 0$ ✓

Beobachtung l wird kleiner nur in 5. $l := \pi_P(l)$

Beobachtung $\forall i: \pi_P(i) \geq 0$

also Φ nichtnegativ ✓

Beobachtung l wächst nur in 6. $l := l + 1$

also amortisierte Kosten je Zeile $O(1)$

Beobachtung amortisierte Rechenzeit While-Schleife 0 ✓

while echte Zeit 1 und Potenzialdifferent ≤ -1

also Gesamtrechenzeit $O(m)$



Laufzeit Algorithmus von Knuth, Morris und Pratt

Lemma 6.15

Der Algorithmus von Knuth, Morris und Pratt hat einschließlich Preprocessing eine Gesamtlaufzeit von $\Theta(n + m)$.

Beweis.

Lemma 6.14 Preprocessing in Zeit $\Theta(m)$

Beobachtung untere Schranke $\Omega(n)$ offensichtlich ✓
weil n -For-Schleife über n Werte

obere Schranke mit Potenzialmethode wie für Preprocessing
mit Potenzial q

Gesamtrechenzeit $\Theta(m + n)$



Auch korrekt?

Über die Präfixfunktion π_P

Lemma 6.16

Sei $P \in \Sigma^m$ Suchmuster. Definiere $\pi_P^{(0)}(q) := q$ und

$$\pi_P^{(i)}(q) := \pi_P\left(\pi_P^{(i-1)}(q)\right), \quad \pi_P^*(q) := \bigcup_{i=1}^{\infty} \left\{ \pi_P^{(i)}(q) \right\}.$$

$$\forall q \in \{1, 2, \dots, m\}: \pi_P^*(q) = \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$$

Beweis.

1. Teil $\pi_P^*(q) \subseteq \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$

also **zu zeigen** für $l \in \pi_P^*(q)$:

$$l \in \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$$

Betrachte $l \in \pi_P^*(q)$

klar $\exists i > 0: l = \pi_P^{(i)}(q)$

1. Fall $i = 1$ dann $l = \pi_P(q)$

also $P[1..l] \sqsupseteq P[1..q]$ nach Definition von π_P ✓

Beweis $\pi_P^*(q) \subseteq \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$

2. Fall $l = \pi_P^{(i)}$ mit $i > 1$

also $l = \pi_P \left(\pi_P^{(i-1)}(q) \right)$

Problem? Verlagerung von $P[1..q]$ nach $P[1..j]$ mit
 $j = \pi_P^{(i-1)}(q) < q$

Beobachtung $P[1..j] \sqsupseteq P[1..q]$
 also $P[1..j] = P[q - j + 1..q]$

also Fortsetzung in $P[1..j]$ liefert $P[1..l] \sqsupseteq P[1..q]$

also $\pi_P^*(q) \subseteq \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\} \checkmark$

Beweis $\pi_P^*(q) \supseteq \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$

2. Teil $\{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\} \subseteq \pi_P^*(q)$

Beweis durch **Widerspruch**

Annahme $\exists l < q: P[1..l] \sqsupseteq P[1..q] \text{ und } l \notin \pi_P^*(q)$

Betrachte l^* , größtes dieser l

Beobachtung $0 \leq \pi_P^{(i)}(q) \leq \max\{0, q - i\}$
also $0 \in \pi_P^{(*)}$ trivial

also $l^* > 0$

Sei i^* größtes i mit $\pi_P^{(i)}(q) \geq l^*$

Beobachtung Existenz **gesichert** durch $\pi_P^{(1)}(q)$

klar $\pi_P^{(i^*)}(q) = l^* \rightsquigarrow l^* \in \pi_P^*(q)$ **Widerspruch**

also $l^* < \pi_P^{(i^*)}(q)$

Beweis $\pi_P^*(q) \supseteq \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$

Betrachte $l^* > 0$, größtes $l < q$: $P[1..l] \sqsupseteq P[1..q]$ und $l \notin \pi_P^*(q)$
 i^* , größtes i mit $\pi_P^{(i)}(q) > l^*$

Betrachte $\pi_P^{(i^*+1)}(q)$

Beobachtung $\pi_P^{(i^*+1)}(q) \neq l^*$ wie vorhin

Beobachtung $\pi_P^{(i^*+1)}(q) < l^*$ wegen Definition von i^*

also $\pi_P^{(i^*+1)}(q) < l^* < \pi_P^{(i^*)}(q)$

aber Präfix „verpassen“ **nicht möglich**

also $\pi_P^*(q) = \{l \mid l < q \text{ und } P[1..l] \sqsupseteq P[1..q]\}$



Korrektheit π_P -Berechnung

Lemma 6.17

Der Algorithmus von Knuth, Morris und Pratt berechnet für jedes Suchmuster $P \in \Sigma^*$ die Präfixfunktion π_P korrekt.

Beweis.

Behauptung

$$\forall q \in \{1, 2, \dots, m\}: \pi_P(q) > 0 \Rightarrow \pi_P(q) - 1 \in \pi_P^*(q - 1)$$

Beobachtung $\pi_P(q) = r > 0$
also $r < q$ und $P[1..r] \sqsupseteq P[1..q]$

also $P[1..r - 1] \sqsupseteq P[1..q - 1]$

klar $r < q \Rightarrow r - 1 < q - 1$

also $\pi_P(q - 1) = r - 1 \in \pi_P^*(q - 1)$ (Lemma 6.16) ✓

Berechnung von π_P : Wachstum von l

Definiere $E_{q-1} := \{k \in \pi_P^*(q-1) \mid P[k+1] = P[q]\}$
für alle $q \in \{2, 3, \dots, m\}$

Beobachtung $E_{q-1} \subseteq \pi_P^*(q-1)$

Beobachtung

$E_{q-1} = \{k \mid k < q-1 \text{ und } P[1..k] \sqsupseteq P[1..q-1] \text{ und } P[k+1] = P[q]\}$

also $E_{q-1} = \{k \mid k < q-1 \text{ und } P[1..k+1] \sqsupseteq P[1..q]\}$

Behauptung $\pi_P(q) = \begin{cases} 0 & \text{falls } E_{q-1} = \emptyset \\ 1 + \max\{k \mid k \in E_{q-1}\} & \text{sonst} \end{cases}$

1. Fall $E_{q-1} = \emptyset$

dann $\nexists k: P[1..k+1] \sqsupseteq P[1..q]$

also $\pi_P(q) = 0$ ✓

Über E_{q-1}

Behauptung $\pi_P(q) = \begin{cases} 0 & \text{falls } E_{q-1} = \emptyset \\ 1 + \max\{k \mid k \in E_{q-1}\} & \text{sonst} \end{cases}$

2. Fall $E_{q-1} \neq \emptyset$

also $\forall k \in E_{q-1}: q > k + 1$
 $P[1..k+1] \sqsupset P[1..q]$

also $\pi_P(q) \geq 1 + \max\{k \mid k \in E_{q-1}\} > 0$

Sei $\pi_P(q) = 1 + r > 0$

dann $P[r+1] = P[q]$

Beobachtung $r \in \pi_P^*(q-1)$ (Lemma 6.16)

also $r \in E_{q-1}$

insbesondere $r \leq \max\{E_{q-1}\}$

zusammen $\pi_P(q) = 1 + \max\{k \mid k \in E_{q-1}\}$ ✓

Korrektheit der Berechnung von π_P

Beobachtung initial $l = \pi_P(q - 1)$

in Zeilen 4–6 l setzen, so dass $l = \pi_P(q)$
wegen Behauptung

Betrachte Schleifenende mit $l = 0$

Beobachtung je nach $P[1] \stackrel{?}{=} P[q]$
 l korrekt gesetzt

zusammen π_P korrekt berechnet



Über den Algorithmus von Knuth, Morris und Pratt

Theorem 6.18

Der Algorithmus von Knuth, Morris und Pratt löst das String-Matching-Problem in Zeit $\Theta(n + m)$.

Beweis.

Laufzeit

- Laufzeit $\Theta(m)$ für π_P -Berechnung (Lemma 6.14) ✓
- Laufzeit $\Theta(n)$ für Suche (Lemma 6.15) ✓

Korrektheit

- Korrektheit π_P -Berechnung (Lemma 6.17) ✓
- Korrektheit Suche **fehlt**

Behauptung Zeilen 4-6 liefern $q = \delta(q, T[i])$

Knuth-Morris-Pratt und String Matching mit DFA

Behauptung Zeilen 4-6 liefern $q = \delta(q, T[i])$

Erinnerung $\delta(q, T[i]) = \sigma_P(P[1..q]T[i])$ (Definition 6.6)

Erinnerung $\sigma_P(P[1..q]T[i]) = r \Rightarrow P[1..r] \sqsupseteq P[1..q]T[i]$
(Definition 6.5)

also $P[1..r-1] \sqsupseteq P[1..q]$
 $\delta(q, T[i]) \in \pi_P^*(q)$

Erinnerung $\pi_P^*(q)$ wird absteigend durchlaufen
(Beweis von Lemma 6.17)

also $q = \delta(q, T[i])$ nach Zeile 6

also Algorithmus von Knuth, Morris und Pratt **korrekt**
weil DFA-String-Matching **korrekt**



Beispiel für Algorithmus von Knuth, Morris und Pratt

T A N A N A M · B **A N A N A S** · T E E
P **A N A N A S**

q	1	2	3	4	5	6
$\pi_P(q)$	0	0	1	2	3	0

Ausgabe 9