Teil 9

String Matching

String Matching

In vielen Anwendungen ist es notwendig in einem Text aus Buchstaben einen bestimmten Teilstring zu suchen. Dieses Problem wollen wir nun näher betrachten.

Σ sei ein endliches Alphabet. Ein Text $T = T[1 \dots n]$ der Länge n über Σ ist eine Sequenz $(T[1], \dots, T[n])$ von Buchstaben aus Σ.

Je nach Rolle des Textes, reden wir auch von einem Pattern $P[1 \dots m]$.

Seien $T[1 \dots n]$ und $P[1 \dots m]$ Test und Pattern mit $m \geq n$. P tritt mit Shift s in T auf, wenn $0 \leq s \leq n - m$ und $T[s + 1 \dots s + m] = P[1 \dots m]$.

Tritt P mit Shift s auf, so sprechen wir von einem gültigen Shift s von P in T, während s sonst ungültig heißt.

String Matching

Problem (String Matching)

Gegeben: *Ein endliches Alphabet* Σ, *ein Text* $T[1 \ldots n]$ *der Länge* n *über* Σ *und ein Pattern* $P[1 \ldots m]$ *der Länge* $m \geq n$ *über* Σ.

Bestimme alle gültigen Shifts von P *in* T.

Beispiel

Text T

c	b	a	a	b	a	b	a	b	c	a	a	b	a	b

Pattern P $\xrightarrow{s=3}$

a	b	a	b

Notationen und Begriffe

Σ^* sei die Menge aller endlichen Strings (Zeichensequenzen) über dem Alphabet Σ.

Den leeren String bezeichnen wir mit ε.

Die Länge eines Strings $x = x_1 \dots x_n$ bezeichnen wir mit $|x|$.

Die Konkatenation xy zweier Strings $x = x_1 \dots x_n$ und $y = y_1 \dots y_m$ ist $xy := x_1 \dots x_n y_1 \dots y_m$.

Ein String x ist ein Präfix eines Strings y, wenn es einen String z gibt mit $y = xz$. Wir schreiben auch $x \vdash y$.

Ein String x ist ein Suffix von y, wenn es einen String z gibt mit $y = zx$. Wir schreiben auch $x \dashv y$.

Der leere String ε ist sowohl Suffix, als auch Präfix jedes beliebigen Strings.

Beispiel

$$ab \vdash abcca \qquad \text{und} \qquad cca \dashv abcca.$$

Notationen und Begriffe

Beobachtung

Für jeden Buchstaben $a \in \Sigma$ und Strings $x, y \in \Sigma^*$ gilt

$$x \vdash y \quad \Leftrightarrow \quad ax \vdash ay \qquad \text{und} \qquad x \dashv y \quad \Leftrightarrow \quad xa \dashv ya.$$

Lemma (Suffix–Überlappungslemma)

x, y und z seien Strings über Σ mit $x \dashv z$ und $y \dashv z$.

- *Falls $|x| \leq |y|$, gilt $x \dashv y$.*
- *Falls $|x| \geq |y|$, gilt $y \dashv y$.*
- *Falls $|x| = |y|$, gilt $x = y$.*

Beweis

Übungsaufgabe

Notationen und Begriffe

Für einen Text $T = T[1 \dots n]$ bezeichnen wir das Präfix mit k Zeichen als T_k, d.h.

$$T_k := T[1 \dots k].$$

Damit ergibt sich die folgende äquivalente Formulierung für das String Matching Problem:

Problem (String Matching)

Gegeben: *Ein endliches Alphabet* Σ*, ein Text* $T[1 \dots n]$ *der Länge* n *über* Σ *und ein Pattern* $P[1 \dots m]$ *der Länge* $m \geq n$ *über* Σ*.*

Bestimme alle s*, so dass* $P \dashv T_{s+m}$*.*

Eine grundlegende Operation

Algorithmus (Common Prefix)

Eingabe: *Zwei Strings X und Y gleicher Länge.*
Rückgabe: *Die Länge des längsten gemeinsamen Präfixes.*

$i := 0$
solange $i < |X|$ *und* $X[i+1] = Y[i+1]$ **tue**
    $i := i + 1$
Ende
Gebe i zurück

Beobachtung

Ist k die Länge des längsten gemeinsamen Präfix, so benötigt `CommonPrefix` Zeit $\Theta(k+1)$.

Algorithmus (Naives String-Matching)

Eingabe: *Ein Test X und ein Pattern P.*
Ausgabe: *Eine Liste S aller gültigen Shifts von Pattern P.*

$S := \emptyset$
$n := |X|$
$m := |P|$
für $s := 0 \ldots n - m$ **tue**
  wenn $\texttt{CommonPrefix}(X[s+1 \ldots s+m], P) = m$ **dann** $S := (S, s)$
Ende

Das Pattern wird somit an dem zu überprüfenden Text entlanggeschoben und mit jedem Teilstring der Länge $m = |P|$ verglichen.

Naives String-Matching

Lemma

Das naive String-Matching benötigt Zeit $O((n-m+1)m)$. Diese Schranke wird im schlechtesten Fall erreicht.

Beweis.

Offensichtlich wird `CommonPrefix` für insgesamt $n-m+1$ verschiedene Teilstrings von T aufgerufen (alle Shifts zwischen 0 und $n-m$). Da `CommonPrefix` im schlechtesten Fall $O(m)$ seit benötigt, ergibt sich die Schranke $O((n-m+1)m)$.

Für die Striktheit der Schranke betrachte $T = a^n$ und $P = a^m$. □

Der Nachteil des naiven String-Matching

Naives String-Matching ist kein besonders guter Algorithmus zur Lösung des Problems.

Das wesentliche Problem besteht darin, dass bereits gewonnene Informationen wieder vergessen werden.

Beispiel

$T = aaababcdef$ und $P = aaab$.

Wir stellen fest, dass $s = 0$ ein gültiger Shift ist.

Aufgrund der Struktur des Patterns ist damit klar, dass $1, 2$ und 3 keine gültigen Shifts mehr sein können.

Dennoch werden sie im naiven String-Matching noch überprüft.

Teil 9: String Matching

String-Matching mit DFAs

Die Idee

Die Idee hinter diesem Ansatz, ist die Konstruktion eines DFA A_P für ein gegebenes Pattern P. der anschließend dazu genutzt werden kann einen beliebigen Text T nach dem Pattern zu durchsuchen.

D.h. unser Algorithmus besteht aus zwei Teilen:

- Einer Vorverarbeitung (Preprocessing) des Patterns, und
- einem Matching-Schritt, der einen Text durchsucht.

Wie wir sehen werden, hat dieser Ansatz nicht nur eine effiziente Laufzeit im Verhältnis zum naiven Ansatz, sondern die Vorverarbeitung muss nur einmal gemacht werden, unabhängig von der Anzahl der Texte, die nach dem Pattern durchsucht werden.

DFA

Zuerst wiederholen wir kurz das Konzept der DFAs.

Definition (DFA – Deterministischer Endlicher Automat)

Ein DFA $A = (Q, q_0, F, \Sigma, \delta)$ besteht aus

- einer endlichen Menge Q von Zuständen,
- einem Startzustand $q_0 \in Q$,
- einer Menge $F \subseteq Q$ von akzeptierenden Zuständen,
- einem Eingabealphabet Σ und
- einer Übergangsfunktion $\delta \colon Q \times \Sigma \to Q$.

DFAs

Die Funktionsweise eines endlichen Automaten A lässt sich folgender maßen beschreiben:

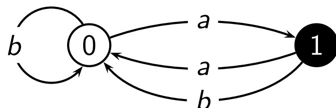
- A erhält einen String $X \in \Sigma^*$ als Eingabe.
- Zu jedem Zeitpunkt hat A einen aktuellen Zustand $q \in Q$.
- Zu Beginn ist A im Startzustand q_0.
- A liest die Zeichen der Eingabe sequentiell.
- Mit jedem gelesenen Zeichen verändert A (möglicherweise) seinen Zustand.
- Ist q der aktuelle Zustand und liest A x, so ist der Folgezustand $q' = \delta(q, x)$.
- Ist das Ende der Eingabe erreicht, so werden bezüglich des Endzustandes $\delta^*(X)$ zwei Fälle unterschieden:

$$\delta^*(X) \in F \Rightarrow \text{ A akzeptiert X}$$
$$\delta^*(X) \notin F \Rightarrow \text{ A verwirft X}$$

DFAs

$A = (\{0,1\}, 1, \{1\}, \{a,b\}, \delta)$ mit folgender Übergangsfunktion δ.

	a	b
0	1	0
1	0	0



Die Suffix-Funktion

Im Folgenden sei P ein festes Pattern der Länge m.

Unser Ziel ist es zu diesem Pattern einen DFA zu konstruieren, der jedes Vorkommen von P in seiner Eingabe erkennt.

Dafür benötigen wir die Suffix-Funktion $\sigma(X)$ von P

$$\sigma(X) := \max\{k \mid P_k \sqsupset X\},$$

d.h. $\sigma(X)$ ist die Länge des längsten Präfix von P, das ein Suffix von X ist.

Die Suffix-Funktion

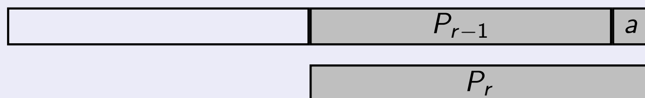
Lemma

Für jeden String X und jeden Buchstaben a gilt $\sigma(Xa) \leq \sigma(X) + 1$.

Beweis.

Falls $\sigma(Xa) = 0$, ist die Behauptung trivialwerweise wahr, denn $\sigma(X) \geq 0$.

Sei nun $r := \sigma(Xa) > 0$. Nach Definition ist somit P_r ein Suffix von Xa. Daher ist P_{r-1} ein Suffix von X (den letzten Buchstaben a weglassen).



Damit gilt $r - 1 \leq \sigma(X)$, da $\sigma(X)$ die längste Länge eines Präfixes von P ist, das auch Suffix von X ist.

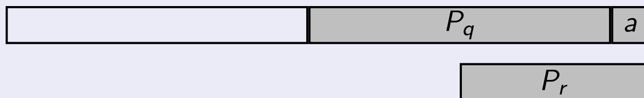
Damit gilt auch $\delta(Xa) = r \leq \delta(X) + 1$. □

Lemma

Für jeden String X und jeden Buchstaben a mit $q = \sigma(X)$, gilt $\sigma(Xa) = \sigma(P_q a)$.

Beweis

Wegen $\sigma(X) = q$ gilt $P_q \dashv X$. Damit gilt auch $P_q a \dashv Xa$.



Wegen des vorhergehenden Lemma gilt $r := \sigma(Xa) \leq \sigma(X) + 1 = q + 1$. Das Suffix–Überlappungslemma impliziert somit $P_r \dashv P_q a \dashv Xa$.

Aus $P_q a \dashv Xa$ folgt, dass jedes Suffix der Form P_k von $P_q a$ auch eines von Xa ist, und somit $\sigma(P_q a) \leq \sigma(Xa)$.

Gleichzeitig is P_r das längste Suffix von Xa der Form P_k. Da es auch ein Suffix von $P_q a$ ist, gilt $r = \sigma(Xa) \leq \sigma(P_q a)$. □

Der Pattern-Automat

Ziel ist es nun den Automaten A_P zum gegebenen Pattern P so zu konstruieren, dass sein Zustand nach dem Lesen der i ersten Zeichen von X_i genau $\sigma(T_i)$ ist.

Wir definieren den Automaten A_P zum Pattern P folgendermaßen:

- $Q := \{0, 1, \ldots, m\}$ mit $m = |P|$.
- 0 ist der Startzustand und m ist der einzige akzeptierende Zustand.
- Die Übergangsfunktion δ ist definiert als

$$\delta(q, a) = \sigma(P_q a).$$

Der Pattern-Automat

Satz

δ^* sei die iterierte Übergangsfunktion von A_P und $T[1 \dots n]$ sei der zu durchsuchende Text. Dann gilt für $i = 0, \dots, n$:

$$\delta^*(T_i) = \sigma(T_i).$$

Beweis

Wir führen eine Induktion über i durch.

Für $i = 0$ ist die Aussage offensichtlich wahr, denn $T_0 = \varepsilon$ und $\sigma(\varepsilon) = 0$.

Sei $q := \delta^*(T_i) = \sigma(T_i)$ und a das $(i+1)$-te Zeichen $T[i+1]$ von T.

Beweis (Fortsetzung)

Dann gilt

$$
\begin{aligned}
\delta^*(T_{i+1}) &= \delta^*(T_i a) \\
&= \delta(\delta^*(T_i), a) && \text{(nach Definition von } \delta^*\text{)} \\
&= \delta(q, a) && (q = \delta^*(T_i)) \\
&= \sigma(P_q a) && \text{(Definition von } \delta\text{)} \\
&= \sigma(T_i a) && \text{(nach obigem Lemma)} \\
&= \sigma(T_{i+1})
\end{aligned}
$$

Die Simulation des DFA

Um einen Text zu durchsuchen, müssen wir nun den DFA simulieren.

Algorithmus (DFA-Matcher)

Eingabe: *Die Übergangsfunktion δ, den Text T und die Länge m des Pattern*

Ausgabe: *Eine Liste S aller gültigen Shifts von P in T*

$S := \emptyset$
$n := |T|$
$q := 0$
für $i := 1 \ldots n$ **tue**
  $q := \delta(q, T[i])$
  wenn $q = m$ **dann** $S := (S, i - m)$
Ende

Die Simulation des DFA

Satz

Ist die Übergangsfunktion bekannt, so können alle gültigen Shifts von P in einem Text der Länge n in Zeit $O(n)$ berechnet werden.

Beweis.

Die Laufzeit der Simulation ist klar. Es muss noch die Korrektheit gezeigt werden.

Da der Automat den Zustand m genau dann annimmt, wenn $\sigma(T_i) = m$, gilt dann stets

$$m = \sigma(T_i) = \max\{k \mid P_k \dashv T_i\},$$

d.h. P ist ein Suffix von T_i.

Damit ist $i - m$ ein gültiger Shift von P in T.

Ist umgekehrt $i - m$ ein gültiger Shift, dann ist P ein Suffix von T_i und somit muss $\sigma(T_i) = m$ gelten und A_P muss den Zustand m annehmen. □

Berechnung der Übergangsfunktion

Der Vorverarbeitungsschritt des Pattern besteht nun darin, die Übergangsfunktion $\delta(q, a) = \sigma(P_q a)$ zu berechnen.

Algorithmus (COMPUTE-TRANSITION-FUNCTION)

Eingabe: *Das Pattern P*
Ausgabe: *Eine Tabelle δ, die die Übergangsfunktion beschreibt*

$m := |P|$
für $q := 0 \ldots m$ **tue**
  für alle $a \in \Sigma$ **tue**
    $k := \min(m + 1, q + 2)$
    wiederhole $k := k - 1$ **bis** $P_k \sqsupset P_q a$
    $\delta(q, a) := k$
  Ende
Ende

Berechnung der Übergangsfunktion

Satz

Die Berechnung der Übergangsfunktion eines Patterns P mit Länge m benötigt $O(m^3|\Sigma|)$ Zeit.

Beweis.

Die Berechnung der Tabelle erfolgt eintragsweise. Die beiden äußeren Schleifen durchlaufen systematisch alle Einträge.

die Zeilen innerhalb dieser Schleifen berechnen das maximale k, so dass $P_k \dashv P_q a$. Dabei beginnen sie bei dem maximal möglichen Wert $\min(m+1, q+2)$ und senken k, bis die Bedingung erfüllt ist.

Für jeden Zustand und jedes Zeichen wird die innerste Schleife zur Senkung von k höchstens $m+1$ Mal durchlaufen. Bei jedem Durchlauf benötigt der Test $P_k \dashv P_q a$ maximal Zeit $O(m)$.

Damit ergibt sich die Gesamtzeit als $O(m^3|\Sigma|)$. □

Berechnung der Übergangsfunktion

Bemerkung

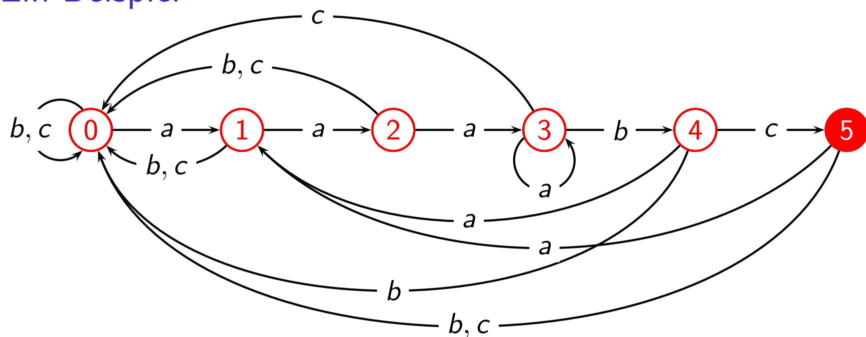
Durch eine geschicktere Berechnung der Übergangsfunktion, lässt sich die Zeit auf $O(m|\Sigma|)$ senken.

Ein Beispiel

$P = aaabc$

	a	b	c
0	1	0	0
1	2	0	0
2	3	0	0
3	3	4	0
4	1	0	5
5	1	0	0

Ein Beispiel



$T = aaaabcaaabcaaaba$

Gültige Verschiebungen: $s = 1$, $s = 6$

Teil 9: String Matching

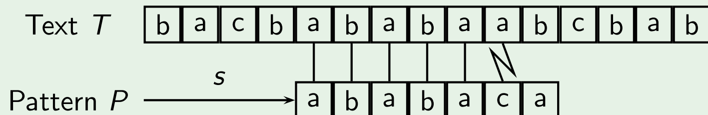
Der Algorithmus von Knuth-Morris-Pratt

Die Idee

Der Algorithmus von Knuth-Morris-Pratt (KMP-Algorithmus) verzichtet auf die vollständige Berechnung der Übergangsfunktion des oben vorgestellten DFA. Stattdessen wird eine Funktion π in linearer Zeit berechnet, die es erlaubt die Übergänge (amortisiert) effizient zu berechnen.

Um die Präfix-Funktion π zu konstruieren, betrachten wir ein Beispiel.

Beispiel



Die Präfix-Funktion

Wie man an dem Beispiel sieht, kann anhand der Struktur des Patterns schon auf die Gültigkeit bestimmter Shifts geschlossen werden.

Im Allgemeinen ist es daher nützlich Fragen der folgenden Form beantworteten zu können:

Frage

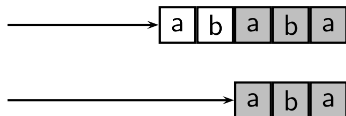
Das Präfix $P[1 \ldots q]$ eines Patterns stimme mit den Zeichen $T[s+1 \ldots s+q]$ eines Textes überein.
Welches ist das kleinste $s' > s$, so dass $P[1 \ldots k] = T[s'+1 \ldots s'+k]$ und $s'+k = s+q$?

Das so gewählte s' ist somit die kleinste Verschiebung, die nicht notwendigerweise ungültig ist. Dabei schließt die letzte Bedingung eindeutig zu kurze Übereinstimmungen aus.

Die Präfix-Funktion

Ist s' bekannt, so müssen die Shifts $s+1, \ldots, s+q-1$ nicht mehr überprüft werden. Außerdem ist bekannt, dass die ersten k Zeichen des Pattern ab Position $s'+1$ mit T übereinstimmen.

Betrachten wir die Situation genauer.



$T[s'+1 \ldots s'+k] = P_k$ ist ein Teil des bereits bekannten Textes $T[s+1 \ldots s+q] = P_q$. Damit ist P_k ein Suffix von P_q und es ergibt sich eine äquivalente Fragestellung.

Die Präfix-Funktion

Frage

Es sei ein Präfix P_q von P gegeben. Welches ist das größte $k < q$ mit $P_k \sqsupset P_q$?

Ist k die Antwort auf diese Frage, so ist $s' = s + (q - k)$ der nächste möglicherweise gültige Shift.

Die Antworten werden nun in der Präfix-Funktion (bzw. Tabelle) gespeichert.

Definition (Präfix-Funktion)

$P = P[1 \ldots m]$ sei ein Pattern. Die Präfix-Funktion $\pi\colon \{1, \ldots, m\} \to \{0, \ldots, m-1\}$ von P ist gegeben durch

$$\pi[q] := \max\{k\colon k < q \text{ und } P_k \sqsupset P_q\}.$$

Die Präfix-Funktion – Ein Beispiel

i	1	2	3	4	5	6	7	8	9	10
$P[i]$	a	b	a	b	a	b	a	b	c	a
$\pi[i]$	0	0	1	2	3	4	5	6	0	1

P_q | a | b | a | b | a | b | a | b |

P_k | a | b | a | b | a | b |

Die Präfix-Funktion

Beobachtung

- Ist P_k ein Suffix von P_q und P_l ein Suffix von P_k, so ist P_l ein Suffix von P_q, d.h.
$$P_l \dashv P_k \dashv P_q \;\Rightarrow\; P_l \dashv P_q.$$
- Sei $k = \pi[q]$, d.h. $P_k \dashv P_q$ und für alle l mit $P_l \dashv P_q$ gilt $l \leq k$. Dann ist jedes Suffix der Form P_l von P_q ein Suffix von P_k.
- Sei $\pi^*[q] := \{k \mid k < q \text{ und } P_k \dashv P_q\}$, d.h. $\pi^*[q]$ ist die Menge aller k, so dass P_k ein echtes Suffix von P_q ist.

Die Präfix-Funktion

Lemma

Sei P ein Pattern der Länge m und π seine Präfix-Funktion. Dann gilt:

$$k \in \pi^*[q] \;\Rightarrow\; \pi[k] \in \pi^*[q].$$

Beweis.

Sei $l = \pi[k]$ und $k \in \pi^*[q]$. Damit gilt $P_l \dashv P_k \dashv P_q$ und somit $P_l \dashv P_q$. □

$\pi^{(i)}[q]$ sei die i-te Iterierte von π, d.h. $\pi^{(0)}[q] = q$ und $\pi^{(i+1)}[q] = \pi\left[\pi^{(i)}[q]\right]$.

Korollar

$$\pi^{(i)}[q] \in \pi^*[q] \text{ für } i \geq 1.$$

Die Präfix-Funktion

Beobachtung

π ist streng monoton fallend, denn $\pi[q] < q$ laut Definition.

Lemma

Sei $\pi^*[q] = \{k_1 > k_2 > \cdots > k_t = 0\}$*. Dann gilt* $k_{i+1} = \pi[k_i]$.

Beweis

Für $i = 1$ *gilt nach Definition von* π *offensichtlich* $k_1 = \pi[q]$.

Wir nehmen an, die Behauptung gilt für ein $i + 1 > 1$ *nicht.*

Da aber $P_{k_{i+1}} \dashv P_q$ *und* $P_{k_i} \dashv P_q$ *gilt und* $k_{i+1} < k_i$*, muss nach dem Suffix-Überlappungslemma* $P_{k_{i+1}} \dashv P_{k_i}$ *gelten.*

Da $l = \pi[k_i] = \max\{k \mid k < k_i \text{ und } P_l \dashv P_{k_i}\}$*, muss* $k_{i+1} \leq l = \pi[k_i]$ *gelten.*

Die Präfix-Funktion

Beweis (Fortsetzung)

Gleichzeitig gilt $P_l \dashv P_{k_i} \dashv P_q$*, was* $l \in \pi^*[q]$ *impliziert.*

Damit muss aber $l = k_{i+1}$ *gelten, denn zwischen* k_i *und* k_{i+1} *liegt kein weiteres Element von* $\pi^*[q]$ *und* $k_{i+1} \leq l < k_i$*.* □

Korollar (Iterations-Lemma)

$$\{k \mid k < q \text{ und } P_k \dashv P_q\} = \pi^*[q] = \left\{\pi[q], \pi^{(2)}[q], \ldots, \pi^{(t)}[q]\right\},$$

wobei $\pi^{(t)}[q] = 0$*.*

Die Präfix-Funktion

Konsequenz

Die Iteration mittels π gibt uns eine einfache Möglichkeit alle Suffixe der Form P_k von P_q zu durchlaufen.

Insbesondere ...

- *... müssen für die Suffixe von P_q nur die Werte $\pi[q']$ für $q' = 0, \ldots, q$ bekannt sein.*
- *... gilt $\pi[0] = 0$.*

Frage

Wie kann dies genutzt werden, um π effizient berechnen zu können?

Um dies zu beantworten, führen wir $\pi[q]$ auf $\pi[q-1]$ zurück.

Die Präfix-Funktion

Beobachtung

- Ist P_k ein Suffix von P_q, so ist P_{k-1} ein Suffix von P_{q-1}, d.h.

$$P_k \dashv P_q \;\Rightarrow\; P_{k-1} \dashv P_{q-1}.$$

- Ist P_{k-1} ein Suffix von P_{q-1} und gilt $P[k] = P[q]$, so ist P_k ein Suffix von P_q, d.h.

$$P_{k-1} \dashv P_{q-1} \text{ und } P[k] = P[q] \;\Rightarrow\; P_k \dashv P_q.$$

Korollar

$$P_k \dashv P_q \;\Leftrightarrow\; P_{k-1} \dashv P_{q-1} \textit{ und } P[k] = P[q].$$

Die Präfix-Funktion

Das letzte Lemma gibt uns eine einfache Möglichkeit die Menge $\pi^*[q]$ aus der Menge $\pi^*[q-1]$ zu berechnen.

Wir müssen lediglich für alle $k \in \pi^*[q-1]$ überprüfen, ob $P[k+1] = P[q]$ gilt. Ist dies der Fall, so gehört $k+1$ zu $\pi^*[q]$, sonst nicht.

Tatsächlich benötigen wir aber lediglich das Maximum der Menge $\pi^*[q]$. Für dies gilt

$$\begin{aligned}
\pi^*[q] &= \max\{k \mid k < q \text{ und } P_k \dashv P_q\} \\
&= \max\{k \mid k < q \text{ und } k-1 \in \pi^*[q-1] \text{ und } P[k] = P[q]\} \\
&= 1 + \max\{k \mid k \in \pi^*[q-1] \text{ und } P[k] = P[q]\}\,.
\end{aligned}$$

Elemente $k \in \pi^*[q-1]$ in fallender Folge durchlaufen und überprüfen, ob $P[k+1] = P[q]$ gilt.

Die Präfix-Funktion

Das Durchlaufen von $\pi^*[q-1]$ in fallender Folge ist aber Dank des Iterations-Lemma einfach, wenn die Werte $\pi[1], \ldots, \pi[q-1]$ bereits bekannt sind.

Wir müssen lediglich, beginnend mit $k_1 = \pi[q-1]$, die Werte $k_i = \pi[k_{i-1}]$ durchlaufen.

Sobald wir auf ein k_i mit $P[k_i + 1] = P[q]$ stoßen, haben wir den korrekten Wert von $\pi[q]$ gefunden.

Dies führt zu folgendem Algorithmus für die Berechnung der Präfix-Funktion.

Die Präfix-Funktion

Algorithmus (Präfix-Funktion)

Eingabe: *Ein Pattern* $P = P[1 \dots m]$
Ausgabe: *Die Präfix-Funktion* π *für* P *als Tabelle*

$m := |P|$
$\pi[1] := 0$
$k := 0$
für $q := 2 \dots m$ **tue**
    solange $k > 0$ *und* $P[k+1] \neq P[q]$ **tue** $k := \pi[k]$
    wenn $P[k+1] = P[q]$ **dann** $k := k+1$
    $\pi[q] := k$
Ende

Die Laufzeit der Berechnung der Präfix-Funktion

Um die Laufzeit der Berechnung der Präfix-Funktion zu ermitteln, führen wir eine amortisierte Analyse mit Hilfe der Potentialmethode durch.

Das Potential des Algorithmus ist der aktuelle Wert der Variable k.

- Zu Beginn ist somit das Potential 0, denn k wird auf 0 initialisiert.
- Da stets $k \geq 0$ gilt, ist das Potential nie negativ.
- Die Iteration $k := \pi[k]$ senkt das Potential, da π streng monoton fällt.
- Das Potential steigt nur um 1, wenn $P[k+1] = P[q]$ erreicht wird. Dies geschieht höchstens einmal pro Durchlauf der äußersten Schleife.
- Da zu Beginn $0 = k < q = 1$ galt und k höchstens für jede Erhöhung von q um eins erhöht wird, gilt stets $k < q$.

Die Laufzeit der Berechnung der Präfix-Funktion

Die amortisierten Kosten für einen Durchlauf er äußersten Schleife sind somit:

- 1 tatsächliche Laufzeit für das Erhöhen von k und das anschließende Speichern,
- l tatsächliche Laufzeit, falls der Wert von k insgesamt l Mal gesenkt wurde,
- Eine Senkung des Potentials um mindestens l durch die Iteration,
- Eine Erhöhung des Potentials um höchstens 1.

Insgesamt ergeben sich die amortisierten Kosten eines Schleifendurchlaufs somit als

$$1 + l - l + 1 = 2 = \Theta(1).$$

Da die Schleife genau m Mal durchlaufen wird, ergeben sich die amortisierten Kosten insgesamt als $\Theta(m)$.

Das Matching

Frage

Wie kann nun die Präfix-Funktion für das Matching verwendet werden?

Die Idee ist, dass man einfach das Matching des DFA-Matchers so umschreibt, dass der Folgezustand nicht direkt aus der Übergangstabelle δ gelesen wird, sondern mittels der Funktion π berechnet wird.

Es gilt

$$\begin{aligned}\delta(q,a) &= \sigma(P_q a) = \max\{k \mid P_k \sqsupset P_q a\} \\ &= \max\{k \mid P_{k-1} \sqsupset P_q \text{ und } P[k] = a\} \\ &= \max\{k \mid k-1 \in \pi^*[q] \text{ und } P[k] = a\}\end{aligned}$$

Damit haben wir eine ähnliche Situation, wie bei der Berechnung von π.

Das Matching

$$\delta(q, a) = \max\{k \mid k-1 \in \pi^*[q] \text{ und } P[k] = a\}.$$

Um $\delta(q, a)$ zu berechnen, müssen wir $\pi^*[q]$ in fallender Folge durchsuchen, bis wir ein $k \in \pi^*[q]$ finden mit $P[k+1] = a$.

Kennen wir dieses k, so ist $k+1$ der nächste Zustand des DFA.

Algorithmus (KMP-Matching)

Eingabe: *Ein Text T, ein Pattern P und die Präfix-Funktion π*
Ausgabe: *Eine Liste S aller gültigen Shifts von P in T*

$S := \emptyset$
$n := |T|, m := |P|$
$q := 0$
für $i := 1 \ldots n$ **tue**
  wiederhole $q := \pi[q]$ **bis** $q = 0$ *oder* $P[q+1] = T[i]$
  wenn $P[q+1] = T[i]$ **dann** $q := q + 1$
  wenn $q = m$ **dann** $S := (S, i - m)$
Ende

Laufzeit des KMP-Matching

Die Analyse der Laufzeit des KMP-Matchers verläuft analog zu der der Berechnung der Präfix-Funktion. Diesmal wird lediglich der Wert von q als Potential gewählt.

Mit den gleichen Argumenten wie vorher, erhält man, dass für jedes Zeichen des Textes T amortisierte Kosten der Höhe $\Theta(1)$ anfallen, so dass der gesamte Algorithmus amortisierte Kosten $\Theta(n)$ hat.

Satz

Mittels des KMP-Matchers kann eine Text T der Länge n in Zeit $\Theta(n)$ nach allen Vorkommen eines Patterns P der Länge m durchsucht werden. Dazu wird zusätzlich $\Theta(m)$ Zeit für die Vorverarbeitung benötigt.

Übersicht

Algorithmus	Vorverarbeitung	Matching
Naiv	-	$O((n-m+1)m)$
DFA	$O(m\|\Sigma\|)^*$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$

* Die Reduktion der Laufzeit für die Vorverarbeitung kann mittels der Präfix-Funktion erfolgen. Erst wird sie in Zeit $\Theta(m)$ berechnet, anschließend δ in Zeit $O(m|\Sigma|)$.