

Vorlesung

Effiziente Algorithmen und Komplexitätstheorie

Sommersemester 2008

Ingo Wegener

Ein neuer Blickwinkel

wir haben Algorithmus von Knuth, Morris und Pratt
Laufzeit $\Theta(n + m)$, asymptotisch optimal

Kann man mehr verlangen für das String-Matching-Problem?

Erinnerung Worst Case

Kann man in günstigen Fällen schneller sein?

jetzt String Matching mit **sublinearer** Laufzeit
in günstigen Fällen

Idee Vergleiche Text und Muster **von rechts nach links**

Vergleiche von rechts nach links

T E R Z I E H U N G **S** W I S S E N S C H A F T

P I N F O R M A T I **K**

T E R Z I E H U N G **S** W I S S E N S C H A F T

P I N F O R M A T I **K**

T E R Z I E H U N G **S** W I S S E N S C H A F T

P I N F O R M A T I **K**

T E R Z I E H U N G **S** W I S S E N S C H A F T

P I N F O R M A T I **K**

T E R Z I E H U N G S W I S S E N S C H A F T

P I N F O R M A T I K

Beobachtung weil S in INFORMATIK nicht vorkommt
 Verschiebung um $|\text{INFORMATIK}| = 10$ möglich

also im Best Case Zeit $O(n/m + f(\text{Preprocessing}))$ möglich

Bad-Character-Verschiebung

allgemeiner Verschiebung beim „schlechten Buchstaben“

Einsicht bei $P[i] \neq T[j]$
 Verschiebung so, dass
 „rechtstes“ Vorkommen von $T[j]$ in P unter $T[j]$
 oder $P[1]$ rechts von $T[j]$ steht

Verschiebung um $i - \text{bad}(T[j])$ mit

Definition 6.19

Für Suchmuster $P \in \Sigma^*$ ist **Bad-Character-Verschiebung**

$\text{bad}: \Sigma \rightarrow \{0, 1, 2, \dots, m\}$ definiert durch

$$\text{bad}(a) := \begin{cases} 0 & \text{falls } a \notin P \\ \max\{i \mid P[i] = a\} & \text{sonst} \end{cases}$$

für alle $a \in \Sigma$.

Berechnung der Bad-Character-Verschiebung

Berechne $\text{bad}(a) := \begin{cases} 0 & \text{falls } a \notin P \\ \max\{i \mid P[i] = a\} & \text{sonst} \end{cases}$

1. For $a \in \Sigma$
 $b[a] := 0$
2. For $i \in \{1, 2, \dots, m\}$
 $b[P[i]] := i$

klar

- korrekt
- Laufzeit $\Theta(|\Sigma| + m)$

Anmerkung für sehr großes Σ
Wörterbuchansatz sinnvoll

Einsatz der Bad-Character-Verschiebung

klar toll, für Buchstaben $a \notin P$

Und allgemein?

Beobachtung Verschiebung $i - \text{bad}(a)$ kann negativ sein

Und dann?

Beobachtung Verschiebung um $\max\{i - \text{bad}(a), 1\}$ auch korrekt

Anmerkung bei Implementierung der Bad-Character-Verschiebung

- Zeit $\Theta(|\Sigma| + m + n/m)$ im Best Case
- empirisch oft sehr gute Laufzeit
- Worst Case $\Theta(n \cdot m)$

Verbesserung des Worst Case

wir haben dank Bad-Character-Verschiebung
sehr guten Best Case
schlechten Worst Case

Wie können wir den Worst Case verbessern?

einfache Idee weitere Verschiebung definieren
größte der berechneten Verschiebungen wählen

Wie kann eine andere Verschiebung aussehen?

klar wie bei Knuth, Morris, und Pratt, aber rückwärts

Die Good-Suffix-Verschiebung

Definition 6.20

Für ein Suchmuster $P \in \Sigma^m$ ist die *Good-Suffix-Verschiebung* $\text{good}: \{1, 2, \dots, m\} \rightarrow \{1, 2, \dots, m\}$ definiert durch

$$\text{good}(i) := \begin{cases} \min\{i - k \mid k < i, P[k] \neq P[i] \text{ und} \\ P[(k + 1) .. m - i + k] = P[(i + 1) .. m]\} & \text{falls } \exists k \\ \min\{m, m - k \mid P[1 .. k] \sqsupseteq P[1 .. m]\} & \text{sonst} \end{cases}$$

für alle $i \in \{1, 2, \dots, m\}$.

Berechnung der Good-Suffix-Verschiebung

Algorithmus 6.21

1. For $i \in \{1, 2, \dots, m\}$ $\text{good}(i) := m$
2. $\text{tmp}(m) := 1$
3. $i := m$
4. For $k \in \{m - 1, m - 2, \dots, 1\}$
5. While $P[i] \neq P[k]$
6. $\text{good}(i) := \min\{\text{good}(i), i - k\}$
7. If $i < m$ Then $i := i + \text{tmp}(i + 1)$ Else While-Schleife beenden.
8. If $P[i] = P[k]$
9. Then $\text{tmp}(k) := i - k; i := i - 1$
10. Else $\text{tmp}(k) := i - k + 1$
11. $i := i + 1; i' := 1$
12. While $i \leq m$
13. For $j \in \{i', i' + 1, \dots, i - 1\}$
14. $\text{good}(j) := \min\{\text{good}(j), i - 1\}$
15. $i' := i; i := i + \text{tmp}(i)$

Der Algorithmus von Boyer und Moore

Algorithmus 6.26

1. $i := 1$ { * Position von $P[1]$ in T * }
2. While $i \leq n - m + 1$
3. $j := m$ { * nächster Vergleichsversuch * }
4. While $P[j] = T[i + j - 1]$ und $j \geq 1$
5. $j := j - 1$
6. If $j > 0$ { * $j > 0 \Leftrightarrow$ keine Übereinstimmung * }
7. Then $i := i + \max\{j - \text{bad}(T[i + j - 1]), \text{good}(j)\}$
8. Else Ausgabe i ; $i := i + \text{good}(1)$

Beispiel für Algorithmus von Boyer und Moore

$$n = 18, m = 6$$

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

a	A	B	E	M	N	S	T	·
$\text{bad}(a)$	5	0	0	0	4	6	0	0

i	1	2	3	4	5	6
$\text{good}(i)$	6	6	6	6	6	1

Ausgabe 9

Laufzeit Preprocessing

Lemma 6.22

Für $P \in \Sigma^m$ können bad und good in Zeit $O(|\Sigma| + m)$ berechnet werden.

Beweis.

schon überlegt bad in Zeit $\Theta(|\Sigma| + m)$ ✓

für good

Laufzeit im Wesentlichen wie für Knuth, Morris und Pratt

in Zeilen 3–10 nur While-Schleife **kritisch**

Beobachtung i kann nur wachsen, wenn vorher kleiner geworden

Beobachtung fällt um $\leq m$ ✓

in Zeilen 11-15 i' wächst mit i ✓

also Zeit $O(m)$ zur good-Berechnung ✓

Korrektheit analog zur K./M./P. □

Semizyklische Suchmuster

leider semizyklische Suchmuster **problematisch**
 darum zunächst **ohne semizyklische Suchmuster**

Definition 6.23

$u \in \Sigma^*$ heißt **semizyklisch**

$:\Leftrightarrow \exists w \in \Sigma^*, v \in \Sigma^+, k \in \mathbb{N}: u = wv^k, v \neq w, w \sqsupset v$

Gilt zusätzlich $w = \varepsilon$, heißt u **zyklisch**.

Gilt $k > 1$, heißt u **echt semizyklisch** (bzw. **echt zyklisch**) in Bezug auf v .

$a \in \Sigma^*$ heißt **zyklische Verschiebung** von $b \in \Sigma^*$

$:\Leftrightarrow \exists x, y \in \Sigma^+ : a = xy, b = yx$

Über zyklische Zeichenketten

Lemma 6.24

Seien $x, y \in \Sigma^+$.

$(xy = yx) \Rightarrow (\exists z \in \Sigma^+ : x \text{ und } y \text{ sind zyklisch in Bezug auf } z)$

Beweis.

Behauptung zusätzlich $|x| = |y|$ impliziert $x = y$

denn $xy = yx \Rightarrow x[1] = y[1]$

also mit $x = x[1]x'$, $y = y[1]y'$ ist dann

$x'y' = y'x' \rightsquigarrow$ iterieren, bis $|x| = |y| = 1$ ✓

klar für $x = y$ ist $z = x = y$ passend ✓

jetzt vollständige Induktion über $|x| + |y|$

Beweis von Lemma 6.24

zu zeigen $(xy = yx)$
 $\Rightarrow (\exists z \in \Sigma^+ : x \text{ und } y \text{ sind zyklisch in Bezug auf } z)$

haben $|x| < |y|$ (sonst umbenennen oder fertig)

klar $xy = yx \Rightarrow x \sqsubset y$

also $\exists y' \in \Sigma^+ : y = xy'$

damit $xx y' = x y' x$

also $xy' = y'x$ mit $|x| + |y'| < |x| + |y|$

also $\exists z' : x = z'^i, y' = z'^j$

klar $x = z'^i, y = xy' = z'^i z'^j = z'^{i+j}$



Noch mehr über zyklische Zeichenketten

Lemma 6.25

Ist $v \in \Sigma^+$ echte zyklische Verschiebung von $w \in \Sigma^+$ und gilt $v = w$, so ist v zyklisch.

Beweis.

klar v echte zyklische Verschiebung von w
 $\Rightarrow \exists x, y \in \Sigma^+ : v = xy, w = yx$

also $v = w \Leftrightarrow xy = yx$

Lemma 6.24 $\exists z : x = z^i, y = z^j$

also $v = xy = z^i z^j = z^{i+j}$



Über den Algorithmus von Boyer und Moore

Theorem 6.27

Für nicht echt zyklische Suchmuster $P \in \Sigma^*$ löst der Algorithmus von Boyer und Moore das String-Matching-Problem in Zeit $O(|\Sigma| + m + n)$.

Beweis.

Beobachtung korrekt, weil Verschiebungen korrekt ✓

noch offen Laufzeit

klar Preprocessing in Zeit $O(|\Sigma| + m)$

zu zeigen Mustersuche in Zeit $O(n)$

Wo ist denn das Problem?

Beobachtung Vergleich von rechts nach links, Verschiebung evtl. klein
 $\rightsquigarrow T[i]$ kann mehrfach betrachtet werden

Analyse Laufzeit Boyer/Moore

amortisierte Analyse mit Potenzialfunktion

Potenzial $\Phi = 3s + u$ mit

u #nie gesehener Buchstaben in T

s #Buchstaben, über die P noch nicht verschoben ist

initial $u = n, s = n$

$\Phi = 4n > 0$ Nachher bedenken!

zunächst Analyse nur mit good ohne bad

Betrachtung einer Verschiebung

Betrachte Vergleich $P[j] \stackrel{?}{=} T[i+j]$ mit $P[j] \neq T[i+j]$
dabei t Buchstaben betrachtet (**also** $t = m - j$)

klar

- führt zu Good-Suffix-Verschiebung um $s \geq 1$
- hat reale Koste t
- liest τ bisher ungelesene Buchstaben (mit $0 \leq \tau \leq t$)

amortisierte Kosten

$$\begin{aligned}
 & t + \Phi_{\text{nachher}} - \Phi_{\text{vorher}} \\
 & = t + \underbrace{3(s' - s) + t' - \tau}_{\Phi_{\text{nachher}}} - \underbrace{(3s' + t')}_{\Phi_{\text{vorher}}} \\
 & = t - 3s - \tau \leq t - 3s
 \end{aligned}$$

Abschätzung amortisierter Kosten

Betrachte Vergleich $P[j] \stackrel{?}{=} T[i+j]$ mit $P[j] \neq T[i+j]$
 dabei t Buchstaben betrachtet (**also** $t = m - j$),
 führt zu Verschiebung um $s \geq 1$,
 liest τ bisher ungelesene Buchstaben

amortisierte Kosten $= t - 3s - \tau \leq t - 3s$

1. Fall $3s \geq t$

klar amortisierte Kosten ≤ 0 ✓

noch offen 2. Fall $3s < t$

Amortisierte Kosten für $t > 3s$

Betrachte Vergleich $P[j] \stackrel{?}{=} T[i+j]$ mit $P[j] \neq T[i+j]$
 dabei t Buchstaben betrachtet (also $t = m - j$),
 führt zu Verschiebung um $s \geq 1$,
 liest τ bisher ungelesene Buchstaben

amortisierte Kosten $= t - 3s - \tau$

wollen zeigen $t - 3s - \tau \leq 0$

Sei $u \sqsupset P$ mit $|u| = s$

Sei $u = v^k$ mit $k \geq 1$

Erinnerung $t > 3s$

Beobachtung $P[(m - t + 1) .. m]$ semizyklisch in u
 also auch semizyklisch in v

Über semizyklische Suffixe

Betrachte Vergleich $P[j] \stackrel{?}{=} T[i+j]$ mit $P[j] \neq T[i+j]$
 dabei t Buchstaben betrachtet (**also** $t = m - j$),
 führt zu Verschiebung um $s \geq 1$,
 liest τ bisher ungelesene Buchstaben,
 hat **amortisierte Kosten** $t - 3s - \tau$;
Sei $u \sqsubset P$ mit $|u| = s$, $u = v^k$ mit $k \geq 1$,
 $P[(m - t + 1) .. m]$ semizyklisch in v

Behauptung in früheren Versuchen $P[m]$ nicht über $v[|v|]$ ($\forall v$)
sonst links von v Nichtübereinstimmung \rightsquigarrow Verschiebung
Kleinere Verschiebung?

Beobachtung bringt v übereinander
 $\rightsquigarrow v$ zyklisch (Lemma 6.25) **Widerspruch**

Gleiche Verschiebung?

klar nicht im **früheren** Versuch

Über das frühere Lesen von T

Behauptung in früheren Versuch $\leq |v|$ Vergleiche

sonst v in T gefunden

$\rightsquigarrow v$ zyklisch (wie eben) **Widerspruch**

Behauptung in früheren Versuchen $P[m]$ positioniert
entweder über rechtestem v oder über einem der linkensten

sonst betrachte $|v|$ Vergleiche

wie gehabt spätestens dann Nichtübereinstimmung

wie eben $\rightsquigarrow P[m]$ über $v[|v|]$ **Widerspruch**

zusammen in früheren Versuchen $\leq 3|v|$ Zeichen
im betrachteten Bereich gelesen

Amortisierte Analyse für $t > 3s$ zusammenfassen

Betrachte Vergleich $P[j] \stackrel{?}{=} T[i+j]$ mit $P[j] \neq T[i+j]$
 dabei t Buchstaben betrachtet (**also** $t = m - j$),
 führt zu Verschiebung um $s \geq 1$,
 liest τ bisher ungelesene Buchstaben,
 hat **amortisierte Kosten** $t - 3s - \tau$

gesehen $\tau \geq t - 3|v| \geq t - 3s$

also amortisierte Kosten

$$\leq t - 3s - \tau \leq t - 3s - (t - 3s) = 0 \checkmark$$

Erinnerung gilt nur **ohne Bad-Character-Verschiebung**

Was ändert sich mit?

klar Verschiebungen werden nur größer \checkmark

Beweis von Theorem 6.27

Wir haben amortisierte Kosten ≤ 0 für jeden Vergleich

Also fertig?

Erinnerung wir hatten

$$\text{initial } u = n, s = n$$

$$\Phi = 4n > 0 \quad \text{Nachher bedenken!}$$

$$\begin{aligned} \text{Erinnerung } \sum \text{amortisierte Kosten} \\ = (\sum \text{reale Kosten}) + \Phi(D_{\text{Ende}}) - \Phi(D_0) \end{aligned}$$

$$\text{Erinnerung hier } \Phi(D_0) = 4n, \Phi(D_{\text{Ende}}) \geq 0$$

$$\begin{aligned} \text{also } \sum \text{reale Kosten} \\ \leq (\sum \text{amortisierte Kosten}) + \Phi(D_0) \\ \leq 4n \quad \square \end{aligned}$$

Erweiterung für beliebige Suchmuster

klar Boyer/Moore für beliebige Suchmuster wünschenswert

Betrachte $P = wv^i, i \in \mathbb{N} \setminus \{1\}, w \sqsupset v, |w| < |v|$

Idee Suche nach $P' = wv$

Beobachtung i Vorkommen von $P' = wv$ im Abstand $|v|$
 $\rightsquigarrow P = wv^i$ gefunden

also Problem prinzipiell lösbar

Beobachtung $|P'| < |P| \rightsquigarrow n/m$ wächst

Theorem 6.28

Der Algorithmus von Boyer und Moore löst das String-Matching-Problem in Zeit $O(|\Sigma| + m + n)$.



Hidden Markov Modelle

„Markov-Kette mit Ausgabe“

- endliche **Zustandsmenge** $Q = \{0, 1, \dots, |Q| - 1\}$
- **Transitionswahrscheinlichkeiten** $a_{i,j}$ für $i, j \in Q$ (natürlich $\forall i \in Q: \sum_{j \in Q} a_{i,j} = 1$)
- endliches **Alphabet** Σ
- **Ausgabewahrscheinlichkeiten** $e_i(b)$ für $i \in Q, b \in \Sigma$ (natürlich $\forall i \in Q: \sum_{b \in \Sigma} e_i(b) = 1$)

Markov-Kette

- startet (Zeitpunkt 0) im Zustand 0,
- vollzieht in jedem Schritt einen Zustandsübergang und gibt ein Zeichen aus.

Ausgabe **beobachtbar**, Zustände **verborgen**

Anwendungen von Hidden Markov Modellen

Was soll denn das?

klassisches Anwendung Spracherkennung

Aufnahme $\hat{=}$ Ausgabe
Gesprochenes $\hat{=}$ Zustandsfolge

inzwischen **viele verschiedene Anwendungen** z. B.

- Handschriftenerkennung
- Gestenerkennung
- Bilderkennung
- DNA-Sequenz-Erkennung
- Generkennung

Über Hidden Markov Modelle

Erinnerung

- endliche **Zustandsmenge** $Q = \{0, 1, \dots, |Q| - 1\}$
- **Transitionswahrscheinlichkeiten** $a_{i,j}$ für $i, j \in Q$ (mit $\forall i \in Q: \sum_{j \in Q} a_{i,j} = 1$)
- endliches **Alphabet** Σ
- **Ausgabewahrscheinlichkeiten** $e_i(b)$ für $i \in Q, b \in \Sigma$ (mit $\forall i \in Q: \sum_{b \in \Sigma} e_i(b) = 1$)

drei grundlegende Probleme

- **Lernen** Finde $a_{i,j}$ und $e_i(b)$ **schwierig**
- **Auswertung** Finde Prob (Ausgabefolge) **nicht schwierig**
- **Decodierung** Finde wahrscheinlichste Zustandsfolge **problematisch**

Dynamische Programmierung

Wichtige **Methode**, gesehen bei

- Rucksackproblem
 - opt. statische binäre Suchbäume
 - All Pairs Shortest Paths
 - global optimales Alignment
- } DAP 2
- CYK-Algorithmus
 - DFA \rightsquigarrow reg. Ausdruck
- } GTI/TifAI
- Algorithmen für HMM
 - weitere ...
- } EA

Jetzt noch ein Beispiel, hoffentlich interessant und spannend.

Das Decodierungsproblem

zur **Ausgabefolge** $x = (x_1, \dots, x_n) \in \Sigma^n$

finde

eine **Zustandsfolge** $\pi = (\pi_1, \dots, \pi_n) \in Q^n$,

die $\text{Prob}(\text{durchlaufe } \pi \mid \text{Ausgabe ist } x)$ **maximiert**

Notation $\text{Prob}(\pi \mid x) := \text{Prob}(\text{durchlaufe } \pi \mid \text{Ausgabe ist } x)$

nach **Definition** $\text{Prob}(\pi \mid x) = \frac{\text{Prob}(\pi \wedge x)}{\text{Prob}(x)}$

klar

$\text{Prob}(\pi \wedge x) = \text{Prob}(\text{durchlaufe } \pi \text{ und erzeuge dabei Ausgabe } x)$

auch klar $\text{Prob}(\pi \wedge x) = \prod_{i=1}^n a_{\pi_{i-1}, \pi_i} \cdot e_{\pi_i}(x_i)$ mit $\pi_0 = 0$

Beobachtung Es **genügt**, $\text{Prob}(\pi \wedge x)$ zu maximieren.

Der Viterbi-Algorithmus

Aufgabe

Maximiere $\text{Prob}(\pi \wedge x) = \prod_{i=1}^n a_{\pi_{i-1}, \pi_i} \cdot e_{\pi_i}(x_i)$

bei gegebenen $a_{q,r}$, $e_q(b)$, $x = (x_1, \dots, x_n)$

Definiere $v_k(i) :=$

$\max \{ \text{Prob}(\pi' \wedge x') \mid x' = (x_1, \dots, x_i) \in \Sigma^i, \pi' = (\pi_1, \dots, \pi_i = k) \in Q^i \}$

„Randfälle“ $v_k(0) = \begin{cases} 1 & \text{falls } k = 0 \\ 0 & \text{sonst} \end{cases}$

sonst $v_k(i) \underset{\text{Markov-Eigenschaft}}{=} e_k(x_i) \cdot \max \{ v_j(i-1) \cdot a_{j,k} \mid j \in Q \}$

Markov-Eigenschaft

Wir suchen $\max \{ v_k(n) \mid k \in Q \}$.

Theorem 7.2 Das Decodierungsproblem kann in Zeit $O(n|Q|^2)$ auf Platz $O(n|Q|)$ gelöst werden. \square

Das Auswertungsproblem

Problem zu HMM und $x \in \Sigma^n$
bestimme $\text{Prob}(\sigma = x)$

Beobachtung $\text{Prob}(\sigma = x) = \sum_{k \in Q} \text{Prob}(q_n = k \wedge \sigma = x)$

Definiere $f_k(i) = \text{Prob}(q_i = k \wedge (\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i))$

Beobachtung $\text{Prob}(\sigma = x) = \sum_{k \in Q} f_k(n)$

Idee Berechne $f_k(n)$ mit dynamischer Programmierung.

„Randfall“ klar

$$f_k(0) = \begin{cases} 1 & \text{falls } k = 0 \\ 0 & \text{sonst} \end{cases}$$

Berechnung von $f_k(i)$

aus Definition

$$f_k(i+1) = \text{Prob}(q_{i+1} = k \wedge (\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1}))$$

wie oben

$$\begin{aligned} & \text{Prob}(q_{i+1} = k \wedge (\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1})) \\ &= \sum_{j \in Q} \text{Prob}(q_i = j \wedge q_{i+1} = k \wedge (\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1})) \end{aligned}$$

wichtig $\text{Prob}(A | B) = \text{Prob}(A \wedge B) / \text{Prob}(B)$
 $\Leftrightarrow \text{Prob}(A \wedge B) = \text{Prob}(A | B) \cdot \text{Prob}(B)$

klar **aufspalten** von $(\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1})$
 in $(\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i) \wedge \sigma_{i+1} = x_{i+1}$
 ändert nichts ✓

Einsicht für dynamische Programmierung

Wir haben

$$\begin{aligned} & \text{Prob}(q_{i+1} = k \wedge (\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1})) \\ &= \sum_{j \in Q} \text{Prob}(q_i = j \wedge q_{i+1} = k \wedge (\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1})) \end{aligned}$$

also

$$\begin{aligned} & \sum_{j \in Q} \text{Prob}(q_i = j \wedge q_{i+1} = k \wedge (\sigma_1, \dots, \sigma_{i+1}) = (x_1, \dots, x_{i+1})) \\ &= \sum_{j \in Q} \text{Prob}(q_{i+1} = k \wedge \sigma_{i+1} = x_{i+1} \mid q_i = j \wedge (\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i)) \\ & \quad \cdot \text{Prob}(q_i = j \wedge (\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i)) \end{aligned}$$

Beobachtung $\text{Prob}(q_i = j \wedge (\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i)) = f_j(i)$

also

$$\begin{aligned} & f_k(i+1) \\ &= \sum_{j \in Q} \text{Prob}(q_{i+1} = k \wedge \sigma_{i+1} = x_{i+1} \mid q_i = j \wedge (\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i)) \\ & \quad \cdot f_j(i) \end{aligned}$$

Über redundante Bedingungen

Wir haben

$$f_k(i+1) = \sum_{j \in Q} \text{Prob}(q_{i+1} = k \wedge \sigma_{i+1} = x_{i+1} \mid q_i = j \wedge (\sigma_1, \dots, \sigma_i) = (x_1, \dots, x_i)) \cdot f_j(i)$$

Einsicht

$$f_k(i+1) = \sum_{j \in Q} \text{Prob}(q_{i+1} = k \wedge \sigma_{i+1} = x_{i+1} \mid q_i = j) \cdot f_j(i)$$

wie vorhin

$$\begin{aligned} & \sum_{j \in Q} \text{Prob}(q_{i+1} = k \wedge \sigma_{i+1} = x_{i+1} \mid q_i = j) \cdot f_j(i) \\ &= \sum_{j \in Q} \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k \wedge q_i = j) \\ & \quad \cdot \text{Prob}(q_{i+1} = k \mid q_i = j) \cdot f_j(i) \end{aligned}$$

Tapfer weiter rechnen

Wir haben

$$\begin{aligned} & \sum_{j \in Q} \text{Prob}(q_{i+1} = k \wedge \sigma_{i+1} = x_{i+1} \mid q_i = j) \cdot f_j(i) \\ &= \sum_{j \in Q} \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k \wedge q_i = j) \\ & \quad \cdot \text{Prob}(q_{i+1} = k \mid q_i = j) \cdot f_j(i) \end{aligned}$$

Einsicht

$$\begin{aligned} & \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k \wedge q_i = j) = \\ & \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k) \end{aligned}$$

also

$$\begin{aligned} & f_k(i+1) \\ &= \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k) \cdot \sum_{j \in Q} \text{Prob}(q_{i+1} = k \mid q_i = j) \cdot f_j(i) \end{aligned}$$

Zum Abschluss bringen

Wir haben

$$f_k(i+1) = \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k) \cdot \sum_{j \in Q} \text{Prob}(q_{i+1} = k \mid q_i = j) \cdot f_j(i)$$

Beobachtung $\text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = k) = e_k(x_{i+1})$

Beobachtung $\text{Prob}(q_{i+1} = k \mid q_i = j) = a_{j,k}$

also $f_k(i+1) = e_k(x_{i+1}) \cdot \sum_{j \in Q} a_{j,k} \cdot f_j(i)$

Theorem 7.3

Der Forward-Algorithmus löst das Auswertungsproblem in Zeit $O(|Q|^2 \cdot n)$ auf Platz $O(|Q| \cdot n)$.

Über wahrscheinlichste Zustände

Erinnerung Viterbi-Algorithmus berechnet zu HMM und $x \in \Sigma^n$ wahrscheinlichste Zustandsfolge $\pi = (\pi_1, \dots, \pi_n)$, die bei Erzeugung von x durchlaufen wird

Wie finden wir wahrscheinlichsten Zustand q_t für $0 < t \leq n$ bei Ausgabe x ?

Beobachtung denkbar bei fester Ausgabe x
zum Zeitpunkt t Zustand $i \in Q$ max. wahrscheinlich
zum Zeitpunkt $t + 1$ Zustand $j \in Q$ max. wahrscheinlich
aber $a_{i,j} = 0$

Aufgabe zu HMM, $x \in \Sigma^n$ und $t \in \{1, 2, \dots, n - 1\}$
finde $\pi_t \in Q$, so dass $\text{Prob}(q_t = \pi_t \mid \sigma = x)$ maximal

Wahrscheinlichste Zustände berechnen

Aufgabe zu HMM, $x \in \Sigma^n$ und $t \in \{1, 2, \dots, n-1\}$
finde $\pi_t \in Q$, so dass $\text{Prob}(q_t = \pi_t \mid \sigma = x)$ maximal

klar $\text{Prob}(q_t = \pi_t \mid \sigma = x) = \frac{\text{Prob}(q_t = \pi_t \wedge \sigma = x)}{\text{Prob}(\sigma = x)}$

Erinnerung $\text{Prob}(\sigma = x)$ schon berechnet ✓
beim Auswertungsproblem

Zeitpunkt t als Zäsur

$$\begin{aligned} & \text{Prob}(q_t = \pi_t \wedge \sigma = x) \\ &= \text{Prob}((\sigma_{t+1}, \dots, \sigma_n) = (x_{t+1}, \dots, x_n) \wedge q_t = \pi_t \\ & \quad \wedge (\sigma_1, \dots, \sigma_t) = (x_1, \dots, x_t)) \end{aligned}$$

Bedingung statt Schnitt

Wir haben

$$\begin{aligned} & \text{Prob}(q_t = \pi_t \wedge \sigma = x) \\ &= \text{Prob}((\sigma_{t+1}, \dots, \sigma_n) = (x_{t+1}, \dots, x_n) \wedge q_t = \pi_t \\ & \quad \wedge (\sigma_1, \dots, \sigma_t) = (x_1, \dots, x_t)) \end{aligned}$$

Prob(A ∧ B) = Prob(A | B) · Prob(B) liefert

$$\begin{aligned} & \text{Prob}(q_t = \pi_t \wedge \sigma = x) \\ &= \text{Prob}((\sigma_{t+1}, \dots, \sigma_n) = (x_{t+1}, \dots, x_n) \mid q_t = \pi_t \\ & \quad \wedge (\sigma_1, \dots, \sigma_t) = (x_1, \dots, x_t)) \\ & \quad \cdot \text{Prob}(q_t = \pi_t \wedge (\sigma_1, \dots, \sigma_t) = (x_1, \dots, x_t)) \end{aligned}$$

Beobachtung

$$\text{Prob}(q_t = \pi_t \wedge (\sigma_1, \dots, \sigma_t) = (x_1, \dots, x_t)) = f_{\pi_t}(t)$$

zusätzliche redundante Bedingung fallen lassen liefert

$$\begin{aligned} & \text{Prob}(q_t = \pi_t \wedge \sigma = x) \\ &= \text{Prob}((\sigma_{t+1}, \dots, \sigma_n) = (x_{t+1}, \dots, x_n) \mid q_t = \pi_t) \cdot f_{\pi_t}(t) \end{aligned}$$

Bekannte Überlegungen neu gefasst

Wir haben

$$\begin{aligned} & \text{Prob}(q_t = \pi_t \wedge \sigma = x) \\ &= \text{Prob}((\sigma_{t+1}, \dots, \sigma_n) = (x_{t+1}, \dots, x_n) \mid q_t = \pi_t) \cdot f_{\pi_t}(t) \end{aligned}$$

Erinnerung $f_{\pi_t}(t)$ beim Auswertungsproblem berechnet ✓

Definiere

$$b_k(i) := \text{Prob}((\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_i = k)$$

damit $\text{Prob}(q_t = \pi_t \wedge \sigma = x) = b_{\pi_t}(t) \cdot f_{\pi_t}(t)$

jetzt $b_k(i)$ analog zum Forward-Algorithmus berechnen

konkret

$$\begin{aligned} b_k(i) &= \text{Prob}((\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_i = k) \\ &= \sum_{j \in Q} \text{Prob}(q_{i+1} = j \wedge (\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_i = k) \end{aligned}$$

Wieder tapfer rechnen

Wir haben

$$\begin{aligned}
 b_k(i) &= \text{Prob}((\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_i = k) \\
 &= \sum_{j \in Q} \text{Prob}(q_{i+1} = j \wedge (\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_i = k)
 \end{aligned}$$

$\text{Prob}(A \wedge B) = \text{Prob}(A \mid B) \cdot \text{Prob}(B)$ liefert

$$\begin{aligned}
 &\sum_{j \in Q} \text{Prob}(q_{i+1} = j \wedge (\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_i = k) \\
 &= \sum_{j \in Q} \text{Prob}((\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_{i+1} = j \wedge q_i = k) \\
 &\quad \cdot \text{Prob}(q_{i+1} = j \mid q_i = k)
 \end{aligned}$$

scharfes Hinsehen liefert

$$b_k(i) = \sum_{j \in Q} \text{Prob}((\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_{i+1} = j) \cdot a_{k,j}$$

Geeignete Abspaltung

Wir haben

$$b_k(i) = \sum_{j \in Q} \text{Prob}((\sigma_{i+1}, \dots, \sigma_n) = (x_{i+1}, \dots, x_n) \mid q_{i+1} = j) \cdot a_{k,j}$$

Abspalten von x_{i+1} und $\text{Prob}(A \wedge B) = \dots$ liefern

$$b_k(i) = \sum_{j \in Q} a_{k,j} \cdot \text{Prob}(\sigma_{i+1} = x_{i+1} \mid q_{i+1} = j) \\ \cdot \text{Prob}((\sigma_{i+2}, \dots, \sigma_n) = (x_{i+2}, \dots, x_n) \mid \sigma_{i+1} = x_{i+1} \wedge q_{i+1} = j)$$

scharfes Hinsehen liefert

$$b_k(i) = \sum_{j \in Q} a_{k,j} \cdot e_j(x_{i+1}) \cdot b_j(i+1)$$

Theorem 7.4

Mit dem Backward-Algorithmus kann man für ein HMM und eine Ausgabefolge $x \in \Sigma^n$ für jeden Zeitpunkt $0 < t \leq n$ einen wahrscheinlichsten Zustand berechnen in Zeit $O(|Q|^2 \cdot n)$ auf Platz $O(|Q| \cdot n)$.