Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. Fabian Kuhn
P. Bamberger, Y. Maus, P. Schneider

Informatik 2 - Sommersemester 2018
Musterlösung Übungsblatt 10

Abgabe: Freitag, 20. Juli, 14:00 Uhr

Aufgabe 1: Editierdistanz *(13 Punkte)*

Seien $A = a_1 \dots a_n, B = b_1 \dots b_m$ zwei Wörter. Für $k \leqslant n, \ell \leqslant m$ seien $A_k = a_1 \dots a_k, B_\ell = b_1 \dots b_\ell$ die Präfixe von A und B. Sei wie in der Vorlesung $D_{k,\ell} := D(A_k, B_\ell)$ die Editierdistanz zwischen den beiden Präfixen A_k, B_ℓ. Es gilt $D(A, B) = D_{n,m}$.

(a) Betrachten Sie die Wörter $A = ananas$ und $B = bananen$. Berechnen Sie *alle* $D_{k,\ell}$ bezüglich einheitlichen Kosten von 1 für jede Operation (löschen, einfügen, ersetzen) für $1 \leqslant k \leqslant n$, $1 \leqslant \ell \leqslant m$ mittels dem rekursiven Ansatz und Memoisation, indem Sie wie in der Vorlesung eine entsprechende Tabelle (bottom-up) ausfüllen. Markieren Sie $D(A, B)$. *(3 Punkte)*

(b) Wiederholen Sie (a) mit erhöhten Kosten 2 für das Ersetzen. *(3 Punkte)*

(c) Sei 3 die Kosten ein Zeichen aus $\{e, n, s\}$ mit einem anderen aus dieser Menge zu ersetzen. Ersetzen-Operationen in denen mindestens ein anderes Zeichen $\notin \{e, n, s\}$ involviert ist, kosten 1. Einfügen und Löschen kostet 2. *Beweisen oder widerlegen* Sie die Korrektheit des dynamischen Programmieransatzes zur Berechnung von $D(A, B)$ für beliebige Wörter A, B *für diese Kosten.* *(5 Punkte)*

(d) Das Löschen eines Zeichens ist aus ungeklärten technischen Gründen nicht mehr möglich. Passen Sie die Rekursionsgleichung für $D_{k,\ell}$ an, um die Editierkosten für die Überführung eines gegebenen Wortes A in ein Wort B ohne Löschen zu berechnen. Gehen Sie davon aus, dass $|A| \leqslant |B|$ ist und das die Kosten für das Einfügen und Ersetzen 1 betragen. *(2 Punkte)*

Musterlösung

(a)

$D_{i,j}$	ε	b	a	n	a	n	e	n
ε	0	1	2	3	4	5	6	7
a	1	1	1	2	3	4	5	6
n	2	2	2	1	2	3	4	5
a	3	3	2	2	1	2	3	4
n	4	4	3	2	2	1	2	3
a	5	5	4	3	2	2	2	3
s	6	6	5	4	3	3	3	3

(b)

$D_{i,j}$	ε	b	a	n	a	n	e	n
ε	0	1	2	3	4	5	6	7
a	1	2	1	2	3	4	5	6
n	2	3	2	1	2	3	4	5
a	3	4	3	2	1	2	3	4
n	4	5	4	3	2	1	2	3
a	5	6	5	4	3	2	3	4
s	6	7	6	5	4	3	4	5

(c) Der Ansatz des dynamischen Programmierens funktioniert nicht für die gegebenen Kosten. Wir haben in der Vorlesung eine Dreiecksungleichung gefordert, welche besagt dass die Kosten einer Operation immer kleiner sein müssen als eine Abfolge von mindestens 2 Operationen die das gleiche bewirken. Diese Ungleichung ist für die gegebenen Kosten offensichtlich verletzt, denn es kann günstiger sein, ein Zeichen zweimal zu ersetzen.

In unserem Beispiel mit den beiden Wörtern *ananas* und *bananen* können wir ein *b* für Kosten 2 einfügen: *bananas*, dann können wir *a* für Kosten 1 mit e ersetzen: *bananes*. Jetzt machen wir einen Umweg. Wir ersetzen *s* zuerst mit *a* und danach mit *n* für Gesamtkosten 2 und erhalten: *bananen*. Insgesamt haben wir Kosten von 5. Wenn der dynamische Programmieransatz korrekt ist, darf er höchstens Editierdistanz 5 für die beiden Eingabewörter liefern. Um zu zeigen, dass das scheitert, exerzieren wir das Verfahren nochmals durch:

$D_{i,j}$	ε	b	a	n	a	n	e	n
ε	0	2	4	6	8	10	12	14
a	2	1	2	4	6	8	10	12
n	4	3	2	2	4	6	8	10
a	6	5	3	3	2	4	6	8
n	8	7	5	3	4	2	4	6
a	10	9	7	5	3	4	3	5
s	12	11	9	7	5	4	5	6 ↯

(d) Wir setzen die Kosten für das Löschen auf ∞ und erhalten die folgende Rekursionsgleichung:

$$D_{i,j} = \min\left(D_{i-1,j-1} + \mathbb{1}_{i=j}, D_{i,j-1} + 1\right)$$

Dabei ist $\mathbb{1}_{i=j} := 1$, falls $i = j$ und $\mathbb{1}_{i=j} := 0$, sonst.

Aufgabe 2: String Matching *(7 Punkte)*

Gegeben sei das Muster $P = ABABAA$ und der Text $T = BBABAABABAABABABAABA$.

(a) Berechnen Sie das Array S des Knuth-Morris-Pratt Algorithmus'. *(3 Punkte)*

(b) Finden Sie mithilfe des Knuth-Morris-Pratt Algorithmus alle Vorkommen von P in T. Die Schritte des Algorithmus sollen klar erkennbar sein (analog zum Beispiel aus der Vorlesung). *(4 Punkte)*

Musterlösung

(a) $S = [-1, 0, 0, 1, 2, 3, 1]$

(b)
```
B B A B A A B A B A A B A B A B A A B A
A B A B A A
  A B A B A A
    A B A B A A
      A B A B A A
        A B A B A A
          A B A B A A                   m a t c h
                A B A B A A
                  A B A B A A           m a t c h
                        A B A
```

Aufgabe 3: Approximate String Matching *(10 Punkte)*

Implementieren Sie den Algorithmus *Approximate String Matching* aus der Vorlesung. Dieser erhält einen *Suchtext* T und ein *Muster* P als Eingabe und sucht einen Teilstring von T der die Editierdistanz zu P minimiert. Gehen Sie von Kosten 1 für alle Operationen aus. Führen Sie Ihren Algorithmus

mit dem Suchtext `text.txt` und den Mustern aus die in `patterns.txt` gegeben sind (getrennt durch Zeilenumbrüche). Ihr Algorithmus soll für *jedes* Muster, den Substring des Suchtextes für welches die Editierdistanz D minimiert wird in eine Datei `result.txt` schreiben (getrennt durch Zeilenumbrüche).

Hinweis: $E(i, b)$ entspricht der minimalen Editierdistanz zwischen P_i und einem $T_{r,b}$ (vergleiche Vorlesung). Wir empfehlen eine $|P| \times |T|$-Tabelle $E(i, b)$ bottom-up zu füllen. Die minimale Distanz zwischen Muster P und einem Teilstring des Suchtextes T ist dann $\min_{0 \leqslant b \leqslant |T|} E(|P|, b)$.