

## 7. Übungsblatt

zur Vorlesung

# Grundzüge der Informatik I

Abgabe über Ilias bis zum 24.5. 14:00 Uhr.  
Besprechung in Kalenderwoche 23.

### Aufgabe 1 *Dynamische Programmierung (3 + 2 + 2 + 2 + 2 Punkte)*

Bob hat Angst vorm Klimawandel und betrachtet die Höchsttemperaturen der letzten Jahre. Er möchte die steigende Tendenz der Temperaturen beobachten. Einige kalte Zeiten dazwischen interessieren nicht, da die Tendenz für Bob interessant ist.

Dafür notiert er die Temperaturen in zeitlicher Reihenfolge in einem Array  $A = [a_1, \dots, a_\ell]$ . Darin sucht er nach einer längsten aufsteigenden Teilsequenz. Dies ist eine Teilsequenz mit maximaler Länge, deren Elemente aufsteigend sortiert sind. Dabei muss die Sequenz nicht zusammenhängend sein.

- a) Es bezeichne  $T(i)$  die Länge der längsten aufsteigenden Teilfolge im Teilarray  $[a_1, \dots, a_i]$ ,  $1 \leq i \leq \ell$ . Geben Sie eine Rekursionsgleichung für  $T(i)$  an und erklären Sie diese.
- b) Entwerfen Sie basierend auf der Rekursionsgleichung aus a) einen Algorithmus in Pseudocode, der mit dynamischer Programmierung bei Eingabe eines Arrays  $A = [a_1, \dots, a_\ell]$  die Länge einer längsten aufsteigenden Teilsequenz ausgibt. Kommentieren Sie Ihren Algorithmus mit Bezug zu Ihrer Rekursionsgleichung aus a).
- c) Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Algorithmus.
- d) Beweisen Sie die Korrektheit Ihres Algorithmus mithilfe einer Schleifenvariante.
- e) Geben Sie nun einen Algorithmus in Pseudocode an, der die in b) berechnete Lösung rekonstruiert und eine längste aufsteigende Teilsequenz ausgibt. Wandeln Sie dafür Ihren Algorithmus aus b) ab, indem Sie statt der Länge die Teilsequenz zurück geben. Erklären oder kommentieren Sie Ihren Algorithmus.

Beispiellosung:

a)  $T(i) = \begin{cases} 1 & i=1 \\ 1 & i>1 \wedge \forall j, 1 \leq j < i : A[j] > A[i] \\ 1 + \max \{ T(j) \mid 1 \leq j < i \wedge A[j] < A[i] \} & \text{sonst} \end{cases}$

b) Langste aufsteigende Teilfolge ( $A, l$ )

(c) Laufzeit

1.  $T = \text{new array}[1..l]$

$\in O(l)$

2.  $T[1] = 1$

1

3. **for**  $i = 2$  to  $l$  do

// Tabelle füllen

l

4.  $T[i] = 1$

$l-1$

5. **for**  $j = 1$  to  $i-1$  do

$\sum_{i=2}^1 1$

6. if  $(A[j] < A[i]) \text{ and } T[i] < 1 + T[j]$

$\sum_{i=2}^1 2 \cdot (i-1)$

7. then  $T[i] = 1 + T[j]$

$\leq \sum_{i=2}^1 i-1$

8.  $\max = T[1]; \text{index} = 1$

// gesuchtes Maximum berechnen 2

9. **for**  $i = 2$  to  $l$  do

l

10. if  $(T[i] > \max)$  then  $\max = T[i]; \text{index} = i$

$\leq 3l$

11. Lösung = new array [1..max]; aktuell = index

// Teilsequenz finden  
(index  $\leq l$ )  $\in O(l)$

12. Lösung [max] = A[index]; k = max - 1

2

13. **for**  $j = \text{index} - 1$  down to 1 do

$O(l)$

14. if  $(T[j] = T[\text{aktuell}] - 1)$  then  $\text{Lösung}[k] = A[j]; \text{aktuell} = j; k -$

$O(l)$   
 $O(l)$

15. return Lösung

c) Laufzeit (vgl. zuletztreise ausgetauschte Laufzeiten):

Zeilen 1-4 und 8-15 in  $\mathcal{O}(l)$

Zeilen 5-7 # Reduzesschleife  $\leq 4 \cdot \sum_{i=2}^l i \leq 2(l+1) \cdot l \in \mathcal{O}(l^2)$

Insgesamt # Reduzesschleife  $\in \mathcal{O}(l^2)$

d) Korrektheit des Algorithmus

Behauptung: Bei Eingabe eines Arrays A der Länge l gibt

Langste aufsteigende Teilfolge (A, l) eine längste aufsteigende Teilfolge aus.

Zeige Schleifeninvariante  $A_n$ : Zu Beginn des i. Dschlufs,  $1 \leq i \leq l$ , des For-Schleife in Zeile 3 enthalten  $T[i]$ ,  $1 \leq i \leq u$ , die Längen der längsten aufsteigenden Teilfolgen, die in  $i$  enden.

Beweis per vollständige Induktion über u.

Induktionsanfang: Vor dem 1. Dschlauf wird  $T[1]=1$  gesetzt. Für den Teilarray der Länge 1 ist die Länge der längsten aufsteigenden Teilfolge immer 1. Also gilt  $A_1(1)$ .

Induktionsannahme: Es gelte  $A_i(u)$  für ein  $u$ ,  $1 \leq u < l$ .

Induktionsabschluß: Zeige, dass daraus  $A_i(u+1)$  folgt.

Betrachte dazu u. Dschlauf, also  $i = u+1$ . ( $2 \leq i \leq l$ )

2.4.,  $T[u+1] = 1$

2.5-7.:  $T[u+1]^{(*)} = 1 + \max \{ T[p] \mid 1 \leq p \leq u \wedge A[p] > A[i] \}$ .

Nach Ind. Ann. ist  $T[p]$  jeweils die maximale Länge einer aufsteigenden Teilfolge, die in  $p$  endet. Deren Maximum ist also die maximale Länge einer Teilfolge, die vor  $u+1$  endet. Die nächst höhere Wert ist also die maximale Länge einer aufsteigenden Teilfolge, die in  $u+1$  endet.

$T[1, \dots, u]$  bleibt unverändert. Also folgt  $A_i(u+1)$ .

$\Rightarrow A_l(l)$ : nach Schleifenabschluß (2.3-7) enthalten alle  $T[i], 1 \leq i \leq l$ , die Längen der längsten aufst. Teilf., die in  $i$  enden.

Das Gesamtmaximum ist demnach  $\max \{ T(i) \mid 1 \leq i \leq l \}$ .

Dieses wird in Zeilen 8-10 berechnet und in max gespeichert. (\*\*)

In Zeilen 11-14 wird das Array T genutzt, um die explizite aufsteigende Teilsequenz der Länge max im Array Loesung abzuspeichern. (\*\*\*). Dieses wird in Zeile 15 ausgegeben.

Es können weitere Details gezeigt werden:

(\*) Zeige Schleifenuvariante A<sub>2</sub>: Zu Beginn des j. Ondelaufs,  $1 \leq j \leq i$ , ist  $T[i] = \lambda + \max \{ T[p] \mid 1 \leq p < j \wedge A[p] > A[i] \}$

Beweis per Induktion über j.

$\Rightarrow A_2(i)$ : Nach Schleifenaustritt (2.5-7) ist

$$T[i] = \lambda + \max \{ T[p] \mid 1 \leq p < i \wedge A[p] > A[i] \}.$$

(\*\*) Zeige weitere A<sub>3</sub>: Zu Beginn des n. Ondelaufs,  $1 \leq n \leq l$ , der For-Schleife in 2.9 gilt  $\max = \max \{ T[i] \mid 1 \leq i \leq n \}$

$\Rightarrow$  iuss. A<sub>3(l)</sub>  $\max = \max \{ T[i] \mid 1 \leq i \leq l \}$ , also die gesuchte maximale Länge.

(\*\*\*) Für die Ausgabe des Teilstückes zeige

A<sub>4</sub>: Zu Beginn des q. Ondelaufs,  $1 \leq q \leq \text{index}$ , der

For-Schleife in 2.13 enthält  $\text{Loesung}[k..max]$  die letzten  $q-1$  Elemente des gesuchten Teilstückes.

**Aufgabe 2** Dynamische Programmierung (3 + 2 + 2 + 2 Punkte)

Seien  $x = (x_1, \dots, x_m) \in \{0, 1\}^m$  und  $y = (y_1, \dots, y_n) \in \{0, 1\}^n$  zwei Bitstrings. Die *Editierdistanz* zwischen  $x$  und  $y$  sei definiert als die minimale Anzahl der Operationen, um  $x$  in  $y$  umzuwandeln: ein Bit einfügen, ein Bit löschen, eine 0 in eine 1 umwandeln oder umgekehrt eine 1 in eine 0 umwandeln.

- a) Geben Sie eine Rekursionsgleichung  $E(i, j)$  zur Berechnung der Editierdistanz zwischen  $(x_1, \dots, x_i)$  und  $(y_1, \dots, y_j)$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , an und erklären Sie diese.

*Hinweis: Betrachten Sie die aktuellen Bits  $x_i$  und  $y_j$ .*

- b) Zeigen Sie die Korrektheit der Rekursionsgleichung aus a).
- c) Entwerfen Sie mithilfe der Rekursionsgleichung aus a) einen Algorithmus in Pseudocode, der mit dynamischer Programmierung bei Eingabe zweier Arrays  $X = [x_1, \dots, x_m]$  und  $Y = [y_1, \dots, y_n]$  mit Einträgen in  $\{0, 1\}$  die Editierdistanz zwischen den Bitstrings  $x$  und  $y$  ausgibt. Kommentieren Sie Ihren Algorithmus mit Bezug zu Ihrer Rekursionsgleichung aus a).
- d) Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Algorithmus.

### Aufgabe 3 Dynamische Programmierung (3 + 2 + 2 + 2 Punkte)

Seien  $x = (x_1, \dots, x_m) \in \{0, 1\}^m$  und  $y = (y_1, \dots, y_n) \in \{0, 1\}^n$  zwei Bitstrings. Die Editierdistanz zwischen  $x$  und  $y$  sei definiert als die minimale Anzahl der Operationen, um  $x$  in  $y$  umzuwandeln: ein Bit einfügen, ein Bit löschen, eine 0 in eine 1 umwandeln oder umgekehrt eine 1 in eine 0 umwandeln.

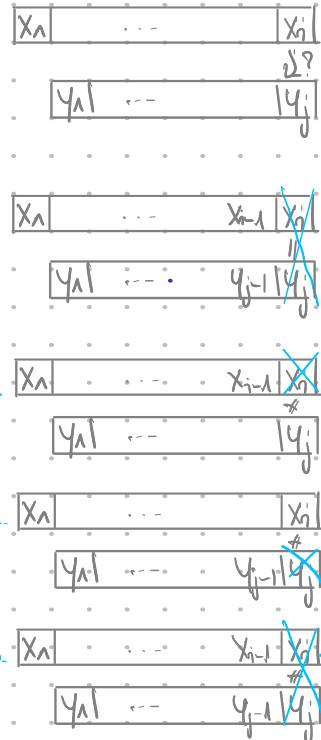
- a) Geben Sie eine Rekursionsgleichung  $E(i, j)$  zur Berechnung der Editierdistanz zwischen  $(x_1, \dots, x_i)$  und  $(y_1, \dots, y_j)$ ,  $1 \leq i \leq m$ ,  $1 \leq j \leq n$ , an.

Hinweis: Betrachten Sie die aktuellen Bits  $x_i$  und  $y_j$ .

- b) Zeigen Sie die Korrektheit der Rekursionsgleichung in a).

- c) Entwerfen Sie mithilfe der Rekursionsgleichung in a) einen Algorithmus in Pseudocode, der mit dynamischer Programmierung bei Eingabe zweier Arrays  $X = [x_1, \dots, x_m]$  und  $Y = [y_1, \dots, y_n]$  mit Einträgen in  $\{0, 1\}$  die Editierdistanz zwischen den Bitstrings  $x$  und  $y$  ausgibt.

- d) Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Algorithmus.



Beispiellösung:

$$a) E(i, j) = \begin{cases} j & \text{j hinzufügen} \\ i & \text{xi, ..., xj löschen} \\ E(i-1, j-1) & \text{xi = yj können gleich sein} \\ \min \{E(i-1, j) + 1, E(i, j-1) + 1, E(i-1, j-1) + 1\} & \begin{array}{l} 1. xi löschen \\ 2. yj hinzufügen \\ 3. xi zu yj umwandeln \end{array} \end{cases}$$

b) Behauptung:  $E(i, j)$  entspricht der Editierdistanz zwischen  $(x_1, \dots, x_i)$  und  $(y_1, \dots, y_j)$ ,  $0 \leq i \leq m$ ,  $0 \leq j \leq n$

Beweis per Induktion über  $k = i \cdot j$ ,  $0 \leq k \leq m \cdot n$

Induktionsanfang:  $k=0 \Rightarrow i=0$  oder  $j=0$ : • Die Editierdistanz zwischen dem leeren String und  $(y_1, \dots, y_j)$  ist  $j$ , da  $j$  Bits hinzugefügt werden müssen. • Um  $(x_1, \dots, x_i)$  in den leeren String umzuwandeln, müssen  $i$  Bits gelöscht werden. Also sind  $E(i, 0) = i$ ,  $E(0, j) = j$  korrekt.

Induktionsannahme: für ein  $k$ ,  $0 \leq k < m \cdot n$ , gelte: Für alle  $l$ ,  $0 \leq l \leq k$ , und für alle  $i, j$ ,  $0 \leq i \leq l$ ,  $0 \leq j \leq l$ , mit  $l = i \cdot j$  sei  $E(i, j)$  die Editierdistanz zwischen  $(x_1, \dots, x_i)$  und  $(y_1, \dots, y_j)$ .

Induktionsabschluß:  $\forall k+1$ : Beobachte alle  $i, j$  mit  $k+1 = i \cdot j$ .  $i > 0, j > 0$

Fall 1:  $x_i \neq y_j$ . Ann.  $E(i, j) < \min \{E(i-1, j) + 1, E(i, j-1) + 1, E(i-1, j-1) + 1\}$

- Lösche  $x_i$ . Dann kann in  $E(i,j)-1 < z-1$  Schritte  $(x_1, \dots, x_{i-1})$  in  $(y_1, \dots, y_j)$  umgewandelt werden.  $\hookrightarrow$  zu Judann, dass  $z-1 \leq E(i-1,j)$
- Edidistanz zwischen  $(x_1, \dots, x_{i-1})$  und  $(y_1, \dots, y_j)$  ist  $(i-1); j \leq k$
- $(x_1, \dots, x_i)$  aus  $(y_1, \dots, y_j)$  durch Hinzufügen von  $y_j$ .  
 $\Rightarrow$  in  $E(i,j)-1 < z-1 \leq E(i,j-1)$  Schritte  $(x_1, \dots, x_i)$  in  $(y_1, \dots, y_{j-1})$  umwandeln.  $\hookrightarrow$  zu Judann für  $E(i-1,j-1), i \cdot (j-1) \leq k$ .
- $x_i$  in  $y_j$  umwandeln.  $\hookrightarrow$  zu Judann für  $E(i-1,j-1), (i-1)j-1 \leq k$ .  
 $\Rightarrow$  insg. Ann. was falsch und  $E(i,j) \geq$  wiu. Also  $E(i,j) =$  wiu.

Fall 2:  $x_i = y_j$ . Ann.  $E(i,j) < E(i-1,j-1)$ .

- Da  $x_i = y_j$  ist, benötigt man die gleiche Anzahl Schritte, um  $(x_1, \dots, x_{i-1})$  in  $(y_1, \dots, y_{j-1})$  umzuwandeln. Laut Judann ist  $E(i-1,j-1)$  bereits das Minimum.  $\hookrightarrow$
  - $x_i$  zu löschen oder  $y_j$  hinzuzufügen kostet wiederum genauso viele Schritte.
- $\Rightarrow E(i,j) = E(i-1,j-1)$ .
- $\Rightarrow E(i,j)$  insgesamt korrekte Edidistanz.

### c) Editiedistanz ( $X, Y$ )

1.  $E = \text{new array } [0..m][0..n]$   $\mathcal{O}(m \cdot n)$

2. **for**  $i = 0$  to  $n$  **do**  $\mathcal{O}(n)$

3.    $E[0][j] = j$

4. **for**  $i = 1$  to  $m$  **do**  $\mathcal{O}(m)$

5.    $E[i][0] = i$

6.   **for**  $j = 1$  to  $n$  **do**  $\mathcal{O}(m \cdot n)$

7.     **if** ( $X[i] = X[j]$ )

8.       **then**  $E[i][j] = E[i-1][j-1]$

9.     **else**  $E[i][j] = \min \{ E[i-1][j] + 1, E[i][j-1] + 1, E[i-1][j-1] + 1 \}$

10.   **return**  $E[m][n]$   $\mathcal{O}(1)$

---

dann wird durch  
verschachtelte  
For-Schleife 2. 6-9

Laufzeit in  $\mathcal{O}(m \cdot n)$