Algorithmen und Datenstrukturen: Übung 5

Tanja Zast, Alexander Waldenmaier

10. Dezember 2020

Aufgabe 5.1

Die optimale Konstellation erhält man, indem man die Gegenstände 1, 2 und 4 mit Gesamtgewicht 8 und Gesamtwert 6 kombiniert.

i \ h	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1
2	0	1	1	2	3	3	3	3	3
3	0	1	1	2	3	3	3	4	4
4	0	1	1	2	3	4	4	4	6
5	0	1	1	2	3	4	4	4	(6)

Aufgabe 5.2

a) Es sind ingesamt 3 Editierungen notwendig, um von HOORAY auf HURRA zu kommen. Demnach beträgt die Länge der längsten gemeinsamen Teilfolge $6 - 3 = 3$. Die Editierdistanz zwischen den einzelnen Teilwörtern ist in der folgenden Tabelle dargestellt:

A	5	4	4	4	3	2	(3)
R	4	3	3	3	2	3	4
R	3	2	2	2	3	4	5
U	2	1	1	2	3	4	5
H	1	0	1	2	3	4	5
	0	1	2	3	4	5	6
		H	O	O	R	A	Y

b) Siehe Algorithm 1 „Längste gemeinsame Teilfolge“.

c) Die Tabelle hat stets die Größe $n \times m$, wobei n, m die Längen der Wörter s_1, s_2 sind, die jeweils um ein führendes Leerzeichen erweitert wurden. Da jede Tabellenzelle einmals mit einer Berechnung von konstanter Ausführungszeit bestimmt wird, beträgt ist die Laufzeit $\in \mathcal{O}(n \cdot m)$.

d) Bestenfalls sind beide Wörter identisch: $s_1 = s_2$ und $m = n$. In diesem Fall ist $lgT(s_1, s_2) = m$ und die rechte Seite der Ungleichung ergibt sich zu $m+n-2lgT(s_1, s_2) = m+n-2m = 0$. Da die Editierdistanz $x = 0$ beträgt herrscht Gleichheit der beiden Seiten.

Werden nun beliebig die Wörter verlängert ohne dabei $lgT(s_1, s_2)$ zu verändern, so vergrößert sich die rechte Seite exakt um die Anzahl der ergänzten Buchstaben. Die Editierdistanz vergrößert sich maximal um die Anzahl der ergänzten Buchstaben (alle Buchstaben werden einem Wort ergänzt) und minimal um die Hälfte der Anzahl (die Buchstaben werden gleichmäßig beiden Wörtern angefügt). Im Maximalfall herrscht erneut Gleichheit. Das exakt gleiche Szenario ergibt sich, wenn die Buchstaben am Anfang statt am Ende eingefügt werden.

Algorithm 1 Funktion: Längste gemeinsame Teilfolge

```
1:  procedure LGT(str1, str2)
2:      str1 ← " "+str1
3:      str2 ← " "+str2
4:      n ← length(str1)
5:      m ← length(str2)
6:
7:      Initialize mat[0...n−1, 0...m−1] = 0
8:
9:      for i ← 0 to n − 1 do
10:         for j ← 0 to m − 1 do
11:             if i == 0 or j == 0 then
12:                 Continue
13:             else if str1[i] == str2[j] then
14:                 mat[i, j] ← mat[i − 1, j − 1] + 1
15:             else
16:                 mat[i, j] ← max(mat[i − 1, j], mat[i, j − 1])
17:
18:     return mat[−1, −1]
```

Die Schreibweise $mat[-1,-1]$ referenziert das Matrixelement in der letzen Spalte und Zeile.

Daraus ergibt sich, dass genau dann Gleichheit herrscht, wenn eines der beiden Wörter vollständig und am Stück an irgend einer Stelle im anderen Wort enthalten ist. Dies gilt auch für den Fall, dass das eine Wort leer ist, da das leere Wort auch ein Unterwort eines Wortes mit Länge größer 0 ist. Für alle anderen Fälle ist die linke Seite kleiner als die rechte.

Aufgabe 5.3

a) In jedem Funktionsaufruf, egal ob $n = 1000000$ oder $n = 10$, wird $n \leq 1$ getestet - dieser eine Fall sollte ausgegrenzt werden. Außerdem wird jeder Funktionswert zweimal berechnet: Einmal, wenn er der $n-1$-te ist und einmal, wenn er der $\lfloor n/2 \rfloor$-te ist. Die Ausführungszeit ließe sich ca. verdoppeln, wenn bereits berechnete Funktionswerte wiederverwendet werden würden.

b) / c) Ja. Durch die Verwendung eines eindimensionalen Arrays mit Länge n lassen sich alle Funktionswerte abspeichern, die bereits bestimmt wurden, und zwar an der zugehörigen i-ten Stelle. Um $F(n)$ zu berechnen, legt man zunächst ein leeres Array der Länge n an und initialisiert das erste Element mit einer 1 (Basisfall). Dann iteriert man alle Zahlen von 2 bis n durch und bestimmt für jedes den Funktionswert. Die erforderlichen Werte $F(n-1)$ und $F(\lfloor n/2 \rfloor)$ sind bereits im Array vorhanden und können einfach ausgelesen werden.

Durch dieses Vorgehen kann der Funktionswert $F(n)$ mit $n-1$ Multiplikationen $(3 \cdot F(n-1))$ und $n-1$ Additionen $(3 \cdot F(n-1) + F(\lfloor n/2 \rfloor))$ durchgeführt werden. Damit ist die Funktion $\in \mathcal{O}(n)$. Der Algorithmus ist in Algorithm 2 „F(n)“ dargestellt.

Algorithm 2 Funktion: F(n)

```
1:  procedure F(n)
2:      Declare arr[0...n − 1]
3:      arr[0] ← 1
4:      for i ← 1 to n − 1 do
5:          arr[i] ← 3 ∗ arr[i − 1] + arr[floor(i/2)]
6:
7:      return arr[−1]
```

floor(x) rundet x auf die nächste Ganzzahl ab. Die Schreibweise $arr[-1]$ referenziert das letzte Arrayelement.

Aufgabe 5.4

a) Gesucht ist eine Darstellung einer beliebigen Potenzemenge S in Form einer einzelnen Zahl $\in \{0, \ldots, 2^{n-1} - 1$. Die naheliegendste Möglichkeit dafür ist, jedem Element der Grundmenge Ω eine Zweierpotenz zuzuweisen, bzw. eine Stelle in der Binärdarstellung der Zahl. Beispielsweise könnten den Elementen $\{2, 3, 4, 5, 6, \ldots\}$ jeweils die Werte $\{2^0, 2^1, 2^2, 2^3, 2^4, \ldots\}$ zugewiesen werden. Die gesuchte Zahl ergibt sich aus der Addition der zugehörigen Zweierpotenzen aller vorhanden Elemente in S.

b) Unter Verwendung der Konvention aus a):

- $i \in S \Leftrightarrow C(S) \geq 2^{i-2}$
- $C(S') = C(S) - 2^{i-2}$

Aufgabe 5.5

a) Es gilt:

$$\begin{aligned} \sigma(i-1) &= |a_{i-1} - a_1| + |a_{i-1} - a_2| + \ldots + |a_{i-1} - a_{i-1}| + |a_{i-1} - a_i| + \ldots + |a_{i-1} - a_n| \\ \sigma(i) &= \underbrace{|a_i - a_1| + |a_i - a_2| + \ldots + |a_i - a_{i-1}|}_{\text{Werden je größer um } a_i - a_{i-1}} + \underbrace{|a_i - a_i| + \ldots + |a_i - a_n|}_{\text{Werden je kleiner um } a_i - a_{i-1}} \end{aligned}$$

Die ersten $i-1$ Terme sind bei $\sigma(i)$ je um den Betrag $\Delta a = a_i - a_{i-1}$ größer als die zugehörigen Terme von $\sigma(i-1)$, da das Argument der Beträge stets größer Null war (dies ergibt sich aus der steigenden Sortierung der Zahlen) und nun um genau die Differenz Δa größer wurde. Bei diesen Termen wird die Null im Argument noch nicht überschritten. Erst im i-ten Term wird das Argument gerade 0, weshalb dieser und alle darauffolgenden Terme sich jeweils um Δa verringern. Folglich gilt:

$$\begin{aligned} \sigma(i) &= \sigma(i-1) + (a_i - a_{i-1}) \cdot ((i-1) - (n - (i-1))) \\ &= \sigma(i-1) + (a_i - a_{i-1}) \cdot (2(i-1) - n) \end{aligned}$$

b)

$$\sigma_{min} = \sigma(i_{min}) \text{ mit } \begin{cases} i_{min} = (n+1)/2 & n \text{ ungerade} \\ i_{min} \in \{\frac{n}{2}, \frac{n}{2} + 1\} & n \text{ gerade} \end{cases}$$

Das erste Element der Folge habe den Funktionswert $\sigma(1)$. Für jedes darauffolgende Element gibt es zuächst mehr Terme die kleiner werden, als Terme, die größer werden. Da sie sich jeweils um den gleichen Betrag vergrößern bzw. verkleinern, sinkt $\sigma(i)$ für zunehmende i. Dieses Verhalten stoppt, sobald die „Mitte“ der Folge erreicht ist. Ab diesem Punkt gibt es mehr Terme die größer werden, als solche, die kleiner werden. Folglich steigt $\sigma(i)$ dann wieder, bis das Ende der Folge erreicht ist.

c) Abgabe in DOMjudge. Teamname: „test“