Präsenzübung 10

Besprechung: 14.01.19 – 18.01.19

Aufgabe 1: Dynamische Programmierung (keine Punkte)

Rufen wir uns die Vorlesungsinhalte zu dynamischer Programmierung in Erinnerung. Dynamische Programmierung basiert darauf, für die gesuchte Gesamtlösung eine *rekursive* Zerlegung in stets kleinere Teilprobleme zu identifizieren.
Leider liefern viele rekursive Zerlegungen nur unwesentlich kleinere Teilprobleme (z.B. Größe nur um 1 verringert). Glücklicherweise kommt es dabei aber häufig vor, dass viele Teilprobleme letztlich auf Lösungen *derselben* kleineren Teilprobleme basieren. Das Gesamtproblem besteht sozusagen aus "überlappenden Teilproblemen". Genau an dieser Stelle setzt dynamisches Programmieren an: "*Vermeide das erneute Lösen von bereits gelösten Teilproblemen*!". Dafür gibt es im Wesentlichen zwei Varianten:

Bottom-Up: Finde eine Reihenfolge aller Teilprobleme (üblicherweise von klein nach groß), so dass zum Zeitpunkt der Berechnung eines Teilproblems alle dafür benötigten Teilprobleme bereits zuvor gelöst wurden. Berechne alle Teilprobleme in genau dieser Reihenfolge und speichere dabei ihr jeweiliges Ergebnis ab (z.B. in einem Array).

Top-Down (durch "Memoization"): Führe die Rekursion direkt auf dem Gesamtproblem aus, z.B. sei dies eine Funktion mit der Eingabe P. Zu Beginn der Funktion überprüfe nun zunächst, ob sie mit derselben Eingabe schon mal aufgerufen und das jeweilige Ergebnis gespeichert wurde Wenn ja, dann gebe das gespeicherte Ergebnis für Eingabe P zurück. Wenn nein, dann berechne das Ergebnis (rekursiv), aber speichere es zum Ende des Funktionsaufrufs als Rückgabewert für die Eingabe P, so dass es bei jedem späteren Aufruf der Funktion mit derselben Eingabe P nicht erneut berechnet werden muss.

Betrachten Sie erneut die rekursive Struktur der Fibonacci-Zahlen ($F_0 := 0, F_1 := 1, F_n := F_{n-1} + F_{n-2}$), und deren direkte Umsetzung als rekursiver Programmcode:

```
function FIB(n)
    if n ≤ 1 then
        return n
    else
        return FIB(n − 1) + FIB(n − 2)
    end if
end function
```

a) Warum ist diese Umsetzung extrem ineffizient? Zeigen Sie, dass sie $\Omega(2^{n/2})$ viele Additionen benötigt.

b) Geben Sie Pseudocode an, um dieselbe Rekursion mittels dynamischer Programmierung ("Bottom-Up") zu berechnen. Wieviele Additionen werden nun berechnet?

c) Wie (b), nun aber mittels "Top-Down".

Aufgabe 2: Dynamisches Programmieren (keine Punkte)

Sei $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{R}^n$ eine beliebige Zahlenfolge. Eine *Teilfolge* von $\mathbf{s}$ besteht aus einer beliebigen Auswahl der Elemente von $\mathbf{s}$ unter Beibehaltung ihrer Reihenfolge. Wir interessieren uns für die Länge $L(\mathbf{s})$ einer längsten monoton wachsenden Teilfolge in $\mathbf{s}$. Zum Beispiel ist dies in $\mathbf{s} = (5, 3, 4, 1, 7, 6, 7)$ die Länge $L(\mathbf{s}) = 4$, aufgrund der Teilfolge $(3, 4, 7, 7)$.

a) Finden Sie eine rekursive Berechnung von $L(\mathbf{s})$. Tipp: Definieren Sie eine Hilfsgröße $l(k)$ – die längste Länge einer Teilfolge von $\mathbf{s}$, welche monoton steigt und *exakt an Position k endet*.

b) Leiten Sie aus (a) mittels dynamischer Programmierung einen Algorithmus ab, der Ihnen $L(\mathbf{s})$ in Laufzeit $\mathcal{O}(n^2)$ liefert.

c) Erweitern Sie Ihren Algorithmus so, dass er zudem die zugehörige Teilfolge ausgibt.

Aufgabe 3: Dynamische Programmierung (keine Punkte)

Sie sind WerbebannerbeauftragterIn einer Werbeagentur und haben die Aufgabe auf einer Autobahnstrecke von m km an n möglichen Stellen $x_1, \ldots, x_n \in [0, m]$ Banner zu platzieren. Für eine Werbung an der Stelle x_i erhalten Sie Einnahmen r_i. Auf deutschen Autobahnen werden Werbebanner aber nur zugelassen, wenn sie einen Mindestabstand von 10 km zueinander haben. Sie wollen nun die Werbebanner so platzieren, dass Sie nicht gegen das Gesetz verstoßen, aber Ihre Einnahmen maximieren.

Beispiel: Sie betrachten einen 90 km langen Straßenabschnitt und 4 mögliche Positionen für Werbebanner an den Stellen

$$\{x_1, x_2, x_3, x_4\} = \{20, 25, 40, 42\},$$

die folgende Einnahmen bringen:

$$\{r_1, r_2, r_3, r_4\} = \{5, 6, 5, 1\}.$$

Es ist also $m = 90$ und $n = 4$. Da die Stellen x_1 und x_2 zu nah liegen und ebenso die Stellen x_3 und x_4, sind die optimalen Stellen x_2 und x_3 mit Gesamteinnahmen von 11.

Geben Sie einen Algorithmus in Pseudocode an, der Instanzen $(x_1, \ldots, x_n, r_1, \ldots r_n)$ erhält und das Problem **mittels Dynamischer Programmierung** löst. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus.