Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithmen und Datenstrukturen
Sommersemester 2024
Korrekturanweisung Übungsblatt 11

Abgabe: Dienstag, 2. Juli, 2024, 10:00 Uhr

Aufgabe 1: Holzfäller *(8 Punkte)*

Gegeben ein Holzscheit der Länge n und ein Array p, welches die Verkaufsspreise von Holzsscheiten jeglicher länge speichert, d.h., für einen Scheit der Länge $i \in \{1, \dots, n\}$ ist der Verkauspreis $p[i]$. Ihre Aufgabe ist es, den *maximal erzielbaren Verkaufpreis* zu bestimmen, den man bekommt wenn man das gegebene Holzscheit bestmöglich zerschneiden und die Stücke gemäß den Preisen in p verkauft (wir zerschneiden nur in ganzzahlige Längen!). Zum Beispiel, wenn die Länge des gegebenen Holzscheits $n = 5$ ist und die Preise wie folgt angegeben sind, dann beträgt der maximal erzielbare Wert 26, indem man den Scheit in zwei Stücke der Länge 1 und ein Stück der Länge 3 zerschneidet.

$$p[1] = 5, \quad p[2] = 8, \quad p[3] = 16, \quad p[4] = 19, \quad p[5] = 25$$

a) Sei $OPT(n)$ dieser bestmögliche Verkaufspreis für ein Scheit der Länge $n \geq 0$. Geben Sie - ähnlich wie bei den Fibonacci Zahlen - eine rekursive Formel für $OPT(n)$ an. Vergessen Sie nicht einen Startwert zu definieren! *(4 Punkte)*

b) Geben Sie einen Algorithmus an der das obige Problem effizient löst. Was ist die Laufzeit ihres Algorithmus? *(4 Punkte)*

Musterlösung

a) Zunächst beachten wir, dass wir das Problem rekursiv lösen können. Sei $OPT(i)$ die optimale Lösung für einen Stab der Größe i:

$$\begin{aligned} OPT(0) &:= 0 \\ OPT(i) &:= \max_{j \in \{0,\dots,i-1\}} \{OPT(j) + p[i-j]\} \end{aligned}$$

b) Die Implementierung des Algorithmus funktioniert wie folgt: Wir initialisieren ein Array A der Größe $n+1$, das $OPT(i)$ an der Position i speichern soll (und somit für die *Memoization* verwednet werden kann). Zuerst setzten wir $A[0] = 0$, was uns konstante Zeit kostet. Um den Eintrag $A[i]$ zu berechnen, müssen wir jeden vorherigen Eintrag $0, \dots, i-1$ betrachten. Genauer gesagt müssen wir für jeden Eintag $0 \leq j \leq i-1$ den Wert $A[j]$ mit $p[i-j]$ addieren (das geht in konstanter Zeit) und das maximum über alle diese j bilden. Dies sind also i vergleiche und somit benötigt das berechnen von $A[i]$ gerade $\mathcal{O}(i)$ Zeitschritte. Die Gesamtlaufzeit um den kompletten Array A zu füllen ist also $\mathcal{O}(1) + \sum_{i=1}^{n} \mathcal{O}(i) = \mathcal{O}(n^2)$.

Aufgabe 2: Bitstrings ohne aufeinanderfolgende Einsen *(12 Punkte)*

Gegeben eine natürliche Zahl $n \geq 1$, möchte man die Anzahl der n-stelligen bitstrings berechnen, welche keine zwei aufeinanderfolgende Einsen enthalten (für $n = 3$ bspw. wäre diese Anzahl 5, da 000, 001, 010, 100, 101 genau die 3-stelligen bitstrings sind, welche keine zwei aufeinanderfolgende Einsen enthalten).

(a) Geben Sie einen Algorithmus an, welcher dieses Problem in $\mathcal{O}(n)$ Zeit löst. Erklären Sie die Laufzeit. *(7 Punkte)*

(b) Implementieren Sie Ihre Lösung. Sie können dazu die Vorlage `DP.py` benutzen. Wenden Sie Ihren Algorithmus auf die Werte 10, 20 und 50 an und schreiben Sie die Ergebnisse in Ihre Abgabe (oder in ihre `erfahrungen.txt`). *(5 Punkte)*

Musterlösung

(a) Sei $A(n, i)$ die Anzahl n-stelliger Bitstrings, welche keine zwei aufeinanderfolgende Einsen enthalten und auf i enden, für $i \in \{0, 1\}$. Wir haben $A(n, 0) = A(n-1, 0) + A(n-1, 1)$ sowie $A(n, 1) = A(n-1, 0)$. Es folgt $A(n, 0) = A(n-1, 0) + A(n-2, 0)$ mit den Basisfällen $A(1, 0) = 1$ und $A(2, 0) = 3$. Die rekursive Struktur ist also die gleiche wie bei den Fibonacci-Zahlen. Berechnung und Laufzeit geht daher analog zur Vorlesung (Woche 11, Folie 6). Die Anzahl n-stelliger Bitstrings ohne aufeinanderfolgende Einsen ist $A(n + 1, 0)$.

(b) Siehe `DP.py` (andere Implementierung als in (a) beschrieben). Die Werte für 10, 20 und 50 lauten 144, 17711 und 32951280099.