DAP2 – Heimübung 7

Ausgabedatum: 26. 5. 17 — Abgabedatum: Fr. 2. 6. 17 (Di. 6. 6. für Gruppen 27–32) 12 Uhr

Schreiben Sie unbedingt immer Ihren **vollständigen Namen**, **Ihre Matrikelnummer** und **Ihre Gruppennummer** auf Ihre Abgaben!

Aufgabe 7.1 (5 Punkte): (Dynamische Programmierung)

Auf Präsenzblatt 4 haben Sie das Problem der *längsten absteigenden, zusammenhängenden Teilfolge* kennengelernt:

Gegeben sei ein Array $A[1..n]$ *ganzer Zahlen. Gesucht ist die Länge eines maximalen Teilarrays* $A[i..j]$ *mit* $1 \leq i \leq j \leq n$ *und der Eigenschaft* $A[i] > A[i+1] > \ldots > A[j]$.

a) Finden Sie eine rekursive Form für die Länge des maximalen absteigenden, zusammenhängenden Teilarrays, das in $A[j]$ endet. Beschreiben Sie Ihre rekursive Form mit eigenen Worten.

b) Geben Sie (in Pseudocode) einen auf dynamischer Programmierung beruhenden Algorithmus an, der die Länge des maximalen absteigenden, zusammenhängenden Teilarrays berechnet und zurückgibt.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Zeigen Sie die Korrektheit der in Teilaufgabe a) angegebenen rekursiven Form.

Lösung:

a) Sei $B[j]$ die Länge des maximalen absteigenden, zusammenhängenden Teilarrays, das in $A[j]$ endet. Für $j = 1$ besteht das Teilarray nur aus einem Element, sodass $B[1] = 1$ ist. Wenn $j > 1$ ist, kann das Teilarray, das in $A[j-1]$ endet, entweder fortgesetzt (wenn $A[j-1] > A[j]$) oder ein neues Teilarray angefangen werden. Deswegen gilt es folgende rekursive Beziehung:

$$B[j] = \begin{cases} 1 & \text{falls } j = 1 \\ B[j-1] + 1 & \text{falls } j > 1 \text{ und } A[j-1] > A[j] \\ 1 & \text{sonst} \end{cases} .$$

Die Länge des gesuchten Teilarrays ist $\max_{1 \leq i \leq n} B[i]$ (das wiederum dynamisch mit dem Algorithmus aus der Vorlesung bestimmt werden kann).

b) Auf Basis der rekursiven Definition ergibt sich das folgende dynamische Programm zur Berechnung von minimalen Kosten:

```
LaengsteTeilfolge(Array A):
1 n ← length(A)
2 B ← new Array [1..n]
3 B[1] ← 1
4 for i ← 2 to n do
5     if A[i − 1] > A[i] then
6         B[i] ← B[i − 1] + 1
7     else
8         B[i] ← 1
9 return max_{1≤i≤n}{B[i]}
```

c) Zeilen 1 und 3 benötigen konstante Laufzeit. Um ein Array der Länge n zu erzeugen (Zeile 3), benötigt man eine Laufzeit in $\mathcal{O}(n)$. Die Schleife in den Zeilen 4–8 benötigt $\mathcal{O}(n)$ Rechenschritte (n in Zeile 4, $n-1$ in Zeile 5 und je entweder Zeile 6 oder 8, also $3n-2$ Rechenschritte). Um das Maxiimum eines Arrays der Länge n zu finden, werden $\mathcal{O}(n)$ Rechenschritte benötigt. Die gesamte Laufzeit dieses Algorithmus liegt also in $\mathcal{O}(n)$.

d) **Beh.** $B[i]$ enthält die Länge des längsten absteigenden, zusammenhängenden Teilarrays, das in $A[i]$ endet.

I.A. Wenn $i = 1$ ist, enthält das Array $A[1..1]$ nur ein Element und die Länge des längsten Arrays ist 1. Somit gilt die Behauptung im I.A.

I.V. Für beliebiges aber festes i gelte die Behauptung.

I.S. Wenn $A[i] \leq A[i+1]$ ist, kann ein absteigendes Teilarray, das in $A[i+1]$ endet, nicht $A[i]$ enthalten, somit kann es nicht länger als 1 sein. D. h. der Wert ist korrekt für $B[i+1]$. Wenn $A[i] > A[i+1]$ ist, kann ein absteigendes Array, das in $A[i]$ endete, durch $A[i+1]$ fortgesetzt werden. Da nach der I. V. in $B[i]$ die Länge des längsten solchen Arrays gespeichert wurde, wird dieses Array durch $A[i+1]$ um ein Element größer und bildet das längste absteigende, zusammenhängende Array, das in $A[i+1]$ endet. Seine Länge ist $B[i] + 1$, was korrekt in berechnet wird.

Somit gilt die Behauptung für alle natürlichen Zahlen i. □

Die gesuchte maximale Länge ist das Maxiumum über die Längen all dieser absteigenden Teilarrays, die in einem i, $1 \leq i \leq n$, liegen.

Aufgabe 7.2 (5 Punkte): (Dynamische Programmierung)

Alice ist inzwischen in die neue Wohnung eingezogen und räumt nun ihre Bücher in Regale ein. Auf einem zentral gelegenen Regalbrett der Breite B möchte sie möglichst viele Bücher unterbringen ohne dabei Genres und Buchreihen zu trennen. Sie besitzt n unterschiedliche Genres, wobei jedes Genre i aus a_i Büchern der (ganzzahligen) Gesamtbreite b_i besteht.

Sei $R(i, j)$ die maximale Anzahl an Büchern aus einer Auswahl aus den ersten i Genres, um das Regal bis zur Breite j zu füllen.

a) Geben die eine rekursive Form für die Berechnung von $R(i, j)$ an.

b) Geben Sie einen Algorithmus in Pseudocode an, der das Gesamtproblem basierend auf der rekursiven Beziehung mit dynamischer Programmierung löst.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Beweisen Sie die Korrektheit der in Teilaufgabe a) angegebenen rekursiven Form.

Lösung:

a) Wir können folgende rekursive Form angeben:

$$R(i,j) = \begin{cases} 0 & \text{falls } i = 0, j \geq 0 \\ R(i-1, j) & \text{falls } i > 0, j < b_i \\ \max\{R(i-1,j), R(i-1, j-b_i) + a_i\} & \text{falls } i > 0, j \geq b_i. \end{cases}$$

b) Der foldende Algorithmus erhält die oben definierten Anzahlen in Array $A = [a_1, \ldots, a_n]$, die Breiten in Array $C = [b_1, \ldots, b_n]$ und die Gesamtbreite B und gibt die maximale Anzahl $R(n, B)$ aus.

```
Buecher(Array A, Array C, int B):
1  n ← length(A)
2  R ← new Array [0..n][0..B]
3  for j ← 0 to B do
4      R[0][j] ← 0
5  for i ← 1 to n do
6      for j ← 0 to B do
7          if j < B[i] then
8              R[i][j] ← R[i − 1][j]
9          else
10             R[i][j] ← max{R[i − 1][j], R[i − 1][j − B[i]] + A[i]}
11 return R[n][B]
```

c) Die Laufzeit des Algorithmus hängt von den Eingabegrößen $n = \text{length}(A)$ und B ab. Zeilen 1 und 11 benötigen konstante Laufzeit. In Zeile 2 werden $(n + 1)(B + 1)$ Felder initialisiert, benötigt also eine Laufzeit in $\mathcal{O}(n \cdot B)$. Die Schleife in Zeile 3 und 4 läuft in $\mathcal{O}(B)$; die verschachtelte Schleife von Zeile 5 bis 10 läuft in $\mathcal{O}(n \cdot B)$. Insgesamt hat der Algorithmus also eine Laufzeit in $\mathcal{O}(n \cdot B)$.

d) Wir wollen nun zeigen, dass die in Aufgabenteil a) angegebene rekursive Form korrekt ist, also für alle i, $0 \leq i \leq n$, und alle j, $0 \leq j \leq B$, $R(i, j)$ die maximale Anzahl an Büchern enthält, die aus den ersten i Genres ausgewählt das Regal bis zur Breite j füllen.

Wir zeigen dies mittels Induktion über i.

Induktionsanfang Falls $i = 0$, also kein Genre vorhanden ist, kann man für jede Breite $j \geq 0$ keine Bücher ergänzen, also ist $R(0, j) = 0$.

Induktionsvoraussetzung Für ein festes $i_0 \geq 0$ und für alle $j \geq 0$ sei $R(i_0, j)$ korrekt

Induktionsschritt Wir betrachten $i = i_0 + 1 > 0$. Falls $j < b_i$ ist, passen die a_i Bücher von Genre i nicht mehr auf das Regal, also entspricht die optimale Auswahl der bisherigen Auswahl aus den vorherigen Genres: $R(i, j) = R(i - 1, j) = R(i_0, j)$. Diese ist nach Induktionsvoraussetzung korrekt. Dies schließt insbesondere auch den Spezialfall $j = 0$ ein. Falls nun $j \geq b_i$ ist, kann Genre i entwender mit auf das Regal gestellt werden oder nicht. Im ersten Fall wird Breite b_i benötigt und es kommen a_i Bücher dazu, es finden also bestenfalls $R(i - 1, j - b_i) + 1_i$ Bücher Platz. Im zweiten Fall werden nur Bücher aus den Genres 1 bis $i - 1$ hinzugefügt, also bestenfalls $R(i-1, j)$. Das Optimum ist also das Maximum aus beiden Fällen. Sowohl $R(i-1, j-b_i) = R(i_0, j - b_i)$ als auch $R(i - 1, j) = R(i_0, j)$ sind laut Induktionsvoraussetzung korrekt, da $j - b_i \geq 0$. $\square$