DAP2 – Heimübung 8

Ausgabedatum: 2. 6. 17 — Abgabedatum: Fr. 9. 6. 17 (Mo. 12. 6. für Gruppen 27–32) 12 Uhr

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

Aufgabe 8.1 (5 Punkte): (Dynamische Programmierung)

In einer Uni im Ausland gelten folgende Regeln für das Programmierpraktikum: es gibt n Übungsblätter, und i-tes Übungsblatt ist $A[i]$ Punkte wert, $1 \leq i \leq n$. Wenn man aber die Punkte im Blatt i gekriegt hat, kann man weder im Blatt $i-1$ noch im Blatt $i+1$ Punkte kriegen. Alle Werte $A[i]$ sind positive ganze Zahlen.

Der Student Bob möchte wissen, wie viele Punkte kann er unten diesen Regeln sammeln. Z.B. wenn die Punktewerte aller Übungsblätter im Array $A = [3, 6, 5, 1, 1, 8, 4, 2]$ gegeben sind, ist es maximal möglich 18 Punkte zu sammeln, wenn man die Übungsblätter mit $[3, 5, 8, 2]$ Punkte bearbeiten würde.

a) Geben Sie eine Rekursionsgleichung für die maximale Anzahl der erreichbaren Punkte $B[i]$, wenn man nur die Übungsblätter $A[1..i]$ betrachtet.

b) Geben Sie dann (in Pseudocode) einen auf dynamischer Programmierung beruhenden Algorithmus an, der die maximal erreichbare Punktesumme des Arrays $A[1..n]$ bestimmt und zurückgibt.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Beweisen Sie die Korrektheit Ihres Algorithmus.

Lösung:

a) Wir bezeichnen mit $B(k)$ die maximale Summe der Punkte die erreicht werden können, wenn man die Übungsblätter $A[1..k]$ betrachten kann. Offensichtlich ist $B(0) = 0$ (die leere Menge der Übungsblätter hat keine Elemente, und somit keine Punkte) und $B(1) = A[1]$ (man wählt das einzige verfügbare Blatt). Für $n \geq 2$ gilt die folgende rekursive Definition:

$$B(k) = \max\{B(k-1), B(k-2) + A[k]\}$$

Für den Rekursionsschritt betrachten wir die beiden möglichen Fälle: das k-te Blatt mit $A[k]$ Punkte wird in $B(k)$ mit aufsummiert bzw. nicht. Wenn es mit aufsummiert wird, ist die optimale Lösung $B(k) = A[k] + B(k-2)$ (Beweis in 8.1.d). Wird $A[k]$ nicht mit aufsummiert, ist $B(k)$ gleich dem größeren Wert der beiden Werte $B(k-1)$ und $B(k-2)$. Da für jedes j gelten muss, dass $B(j-1) \leq B(j)$ ist (man kann nicht mit mehr Blatter weniger Punkte erreichen dürfen), brauchen wir in der rekursiven Formulierung nur $B(k-1)$ zu betrachten.

b) Im folgenden Algorithmus ist das Array A als Eingabeparameter gegeben.

MaxSumPunkte(Array A):
1 $B \leftarrow$ `new` $Array[0..Length(A)]$
2 $B[0] \leftarrow 0$
3 $B[1] \leftarrow A[1]$
4 **for** $i \leftarrow 2$ **to** $Length(A)$ **do**
5 $\quad B[i] \leftarrow$ `max` $\{B[i-1], B[i-2] + A[k]\}$ `/* berechnet das Maximum */`
6 **return** $B[Length(A)]$

c) Sei $|A| = n$. Die Initialisierung in der Zeile 1, die `for`-Schleife in den Zeilen 4 und 5 brauchen jeweils $O(n)$ Rechenschritte. Die Zeilen 2, 3 und 6 benötigen $O(1)$ Zeit. Insgesamt braucht der Algorithmus $O(n)$ Rechenschritte.

d) Wir beweisen die Korrektheit der rekursiven Form aus 8.1.a. Das in Teilaufgabe 8.1.b angegebene dynamische Programm ist nur eine direkte Umsetzung der rekursiven Form.

Wir müssen zeigen, dass für alle k mit $0 \leq k \leq n$, $B(k)$ die maximale Summe der erreichbaren Punkte der ÜbungsblÄtter $A[1..k]$ ist. Wir zeigen dies mittels Induktion nach k.

Induktionsanfang Nach Definition ist $B[0] = 0$ und $B[1]$ ist $A[1]$, was die einzige Möglichkeiten sind – keine Punkte und die einzige mögliche Punkte. Somit gilt die Behauptung für Induktionsanfang.

Induktionsvoraussetzung Es gelte die Induktionsbehauptung für alle $0 \leq i \leq k-1$, für beliebiges aber festes k.

Induktionsschritt Angenommen, $\max\{B(k-1), B(k-2)+A[k]\}$ ist nicht die maximale Summe der möglichen Punkte aus Blätter $A[1..k]$. Es gibt dann eine natürliche Zahl v, sodass $B(k) = v > \max\{B(k-1), B(k-2) + A[k]\}$. Ist $A[k]$ als Summand in dieser maximalen Summe v enthalten, muss $v > B(k-2) + A[k]$ gelten. Dann ist $v - A[k] > B(k-2)$ und $v - A[k]$ ist die Summe der Punkte wenn die Blätter $A[1..k-2]$ verfügbar sind. Dies steht im Widerspruch zur Induktionsvoraussetzung, nach der $B(k-2)$ die maximale Summe aus $A[1..k-2]$ ist.

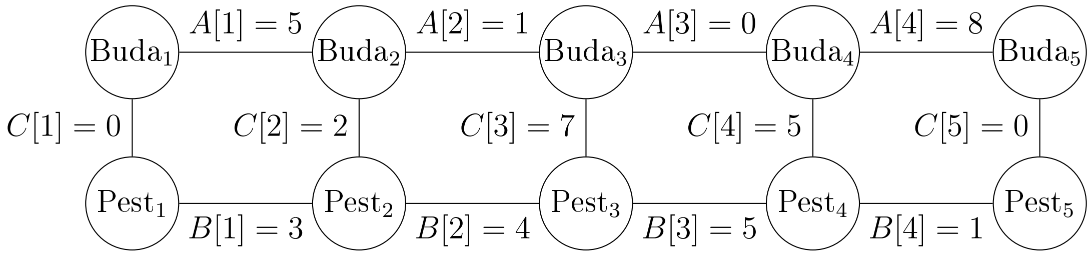
Kommt $A[k]$ nicht als Summand in der maximalen Summe v vor, so gilt $v > B(k-1)$ und v ist die Summe der Punkte aus der Blätter $A[1..k-1]$. Dies steht im Widerspruch zur Optimalität von $B(k-1)$, die in der Induktionsvoraussetzung angenommen wurde. Damit ist die Korrektheit der rekursiven Form, und folglich unseres Algorithmus für alle natürlichen Zahlen n bewiesen.

$\square$

Aufgabe 8.2 (5 Punkte): (Dynamische Programmierung)

In Budapest gibt es n Brücken am Donau. Alice möchte die Strecke zwischen Brücken 1 und n am Ufer entlang spazieren, wobei sie zwischen zwei nacheinanderliegenden Brücken entweder am rechten Ufer (in Buda) oder am linken Ufer (in Pest) spazieren kann. Während sie zwischen Brücken i und $i+1$ spaziert, $1 \leq i < n$, kriegt sie Brezeln, und zwar $A[i]$ Brezeln in Buda, und $B[i]$ Brezeln in Pest. Wenn sie Donau über i-te Brücke überqueren möchte, muss sie $C[i]$ Brezeln abgeben, wobei sie nie weniger als 0 Brezeln haben kann.

Alice beginnt mit 0 Brezeln in Buda bei der Brücke 1. Alle Werte $A[i]$ und $B[i]$ $(1 \leq i \leq n-1)$, sowie $C[i]$ $(1 \leq i \leq n)$ sind nicht negativ, und es gilt $C[1] = C[n] = 0$. Alice darf nicht in Gegenrichtung laufen (z.B. von Brücke $i+1$ nach Brücke i) und will die Anzahl der gesammelten Brezeln bei der n-ten Brücke in Buda maximieren.



In diesem Beispiel wäre es optimal, bei den Brücken 2 und 4 das Ufer zu wechseln. In diesem Fall kommt man mit $15 (= 5 - 2 + 4 + 5 - 5 + 8)$ Brezeln am Ziel in Buda an.

a) Geben Sie die Rekursionsgleichungen für die maximalen Anzahlen $p(i)$ und $q(i)$ von Brezeln an, die Alice bei Erreichen der i-ten Brücke in Buda bzw. Pest sammeln kann.

b) Geben Sie einen Algorithmus in Pseudocode an, der auf dem Prinzip der dynamischen Programmierung beruht und die maximale Anzahl gesammelter Brezeln beim Erreichen der Brücke n in Buda zurückgibt.

c) Analysieren Sie die Laufzeit Ihres Algorithmus.

d) Beweisen Sie die Korrektheit Ihrer Rekursionsgleichungen aus Teilaufgabe a) mittels Induktion.

Lösung:

a) Die rekursive Formen für $p(i)$ bzw. $q(i)$ werden aufeinander referenzieren. Die Abbruchbedingungen sind offensichtlich $p(1) = 0$ und $q(1) = 0$, da man bei der ersten Brücke in Buda keinen Brezel hat, bzw. mit $C[1] = 0$ ohne Abgaben die erste Brücke nach Pest überqueren kann.

Für die weiteren Brücken $(i > 1)$, kann die i-te Brücke in Buda entweder von $i-1$-te Brücke durch Buda, oder über der i-ten Brücke aus Pest erreicht werden. Ähnliches gilt für die i-te Brücke in Pest. Dabei müssen wir darauf achten, dass man keinen Kreis in rekursivem Form baut (z.B. $p(i)$ referenziert auf $q(i)$, und $q(i)$ auf $p(i)$). Deswegen haben wir folgende Rekursiven formen:

$$p[i] = \begin{cases} 0 & \text{falls } i = 1 \\ \max\{p(i-1) + A[i-1], q(i-1) + B[i-1] - C[i]\} & \text{falls } i > 1 \end{cases}$$

$$q[i] = \begin{cases} 0 & \text{falls } i = 1 \\ \max\{q(i-1) + B[i-1], p(i) - C[i]\} & \text{falls } j > 1 \text{ oder} \\ \max\{q(i-1) + B[i-1], p(i-1) + A[i-1] - C[i]\} & \text{falls } j > 1 \end{cases}$$

b) In Pseudocode erhalten wir folgenden Algorithmus. Wir geben den größeren Wert von $P[n]$ und $Q[n]$ zurück, da man kostenlos aus Pest über die n-te Brücke nach Buda kommen kann ($C[n] = 0$).

```
Brezeln (Array A[1..n−1], Array B[1..n−1], Array C[1..n]):
1. P ← new Array [1..n]
2. Q ← new Array [1..n]
3. P[1] ← 0
4. Q[1] ← 0
5. for i ← 2 to n do
6.     P[i] ← max{P[i−1] + A[i−1], Q[i−1] + B[i−1] − C[i]}
7.     Q[i] ← max{Q[i−1] + B[i−1], P[i] − C[i]}
8. return max{P[n], Q[n]}
```

c) Die Laufzeit $T(n)$ des Algorithmus ist $O(n)$, weil wir die Zeit $O(n)$ für die Zeilen 1, 2, 5, 6 und 7 benötigen, und die restliche Zeilen (3, 4 und 7) konstante Zeit benötigen.

d) Behauptung: $p(i)$ und $q(i)$ enthalten die maximale Anzahl der Brezeln beim Erreichen der i-ten Brücke in Buda bzw. Pest, für alle $i \in \mathbb{N}$.

IA. Für $i = 1$ ist $p(1) = 0$ korrekt, da Alice am Anfang in Buda ist, und somit keine Brezeln hat, und $q(1) = 0$ also korrekt, da sie kostenlos die erste Brücke überqueren kann ($C[1] = 0$) und damit höchstens 0 Brezeln in Pest am Anfang haben kann.

IV. Seien $p(i-1)$ und $q(i-1)$ maximale Anzahl der Brezeln bei der $i-1$-ten Brücke, für beliebiges aber festes i.

IS. Wir zeigen zuerst die Korrektheit von $p(i)$. Nehmen wir an, es gäbe eine bessere Summe $u > p(i) = \max\{p(i-1) + A[i-1], q(i-1) + B[i-1] - C[i]\}$, sodass Alice u Brezeln bei der i-ten Brücke in Buda haben kann. Die i-te Brücke in Buda für u kann entweder durch Buda von $i-1$-ter Brücke, oder aus Pest über i-ten Brücke erreicht werden (es ist nicht erlaubt, in Gegenrichtung zu laufen).

Fall 1: Sie kam durch Buda. Dann ist $u > p(i-1) + A[i-1]$, und also $u - A[i-1] > p(i-1)$. $u - A[i-1]$ ist die Anzahl der Brezeln, die Alice in Buda bei der $i-1$-ten Brücke haben müsste. Das widerspricht der I.V. für $p(i-1)$.

Fall 2: Sie kam aus Pest. Dann ist $u > q(i-1) + B[i-1] - C[i]$. $u + C[i]$ wäre die Anzahl der Brezeln bei der i-ten Brücke in Pest, $u + C[i] - B[i-1]$ die Anzahl bei der $i-1$-ten Brücke in Pest, und diese Anzahl $u + C[i] - B[i-1] > q(i-1)$. Das widerspricht der I.V. für $q(i-1)$.

Somit ist auch $p(i)$ die maximale Anzahl der Brezeln in Buda bei der i-ten Brücke. Jetzt zeigen wir die Optimalität von $q(i)$. Nehmen wir an, sie ist nicht optimal und es gäbe eine bessere $w > q(i) = \max\{q(i-1) + B[i-1], p(i) - C[i]\}$. Diesen Wert w kann entweder aus Buda über i-ten Brücke, oder durch Pest von $i-1$-ter Brücke erreicht werden.

Fall 1: Sie kam aus Buda. Dann ist $w > p(i) - C[i]$, und also $w + C[i] > p(i)$. $w + C[i]$ wäre denn die Anzahl der Brezeln in Buda bei der i-ten Brücke, was nicht sein kann, weil wir gerade bewiesen haben, dass $p(i)$ maximal ist.

Fall 2: Sie kam durch Pest. Dann ist $w > q(i-1) + B[i-1]$. $w - B[i-1]$ wäre die Anzahl der Brezeln bei der $i-1$-ten Brücke in Pest, was widerspricht der I.V. für $q(i-1)$.

Somit ist also $q(i)$ optimal, und die Behauptung gilt für beide p und q für alle $i \in \mathbb{N}$.